

Deep Neural Networks

Convolutional Networks III

Bhiksha Raj

Fall 2021

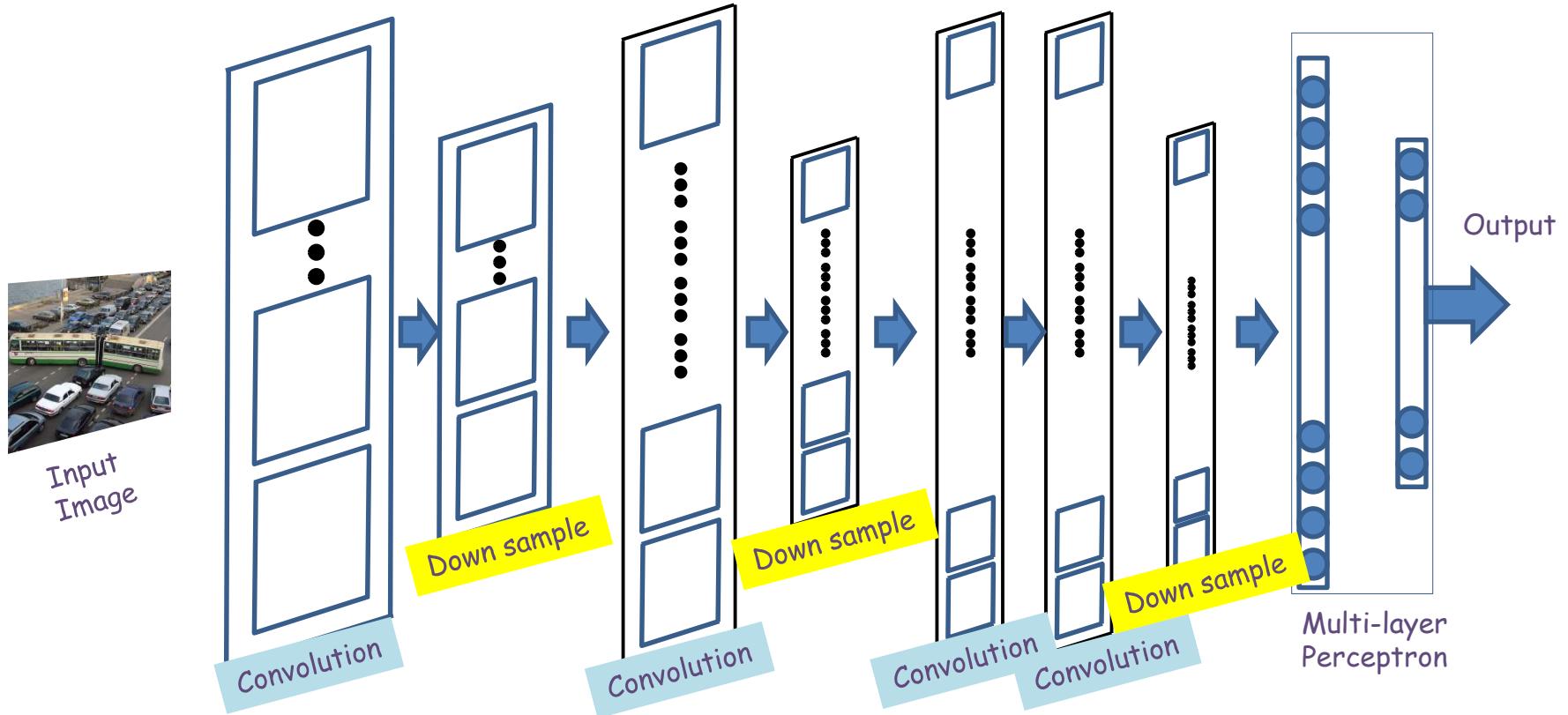
Outline

- Quick recap
- Back propagation through a CNN
- Modifications: Transposition, scaling, rotation and deformation invariance
- Segmentation and localization
- Some success stories
- Some advanced architectures
 - Resnet
 - Densenet
 - Transformers and self similarity

Story so far

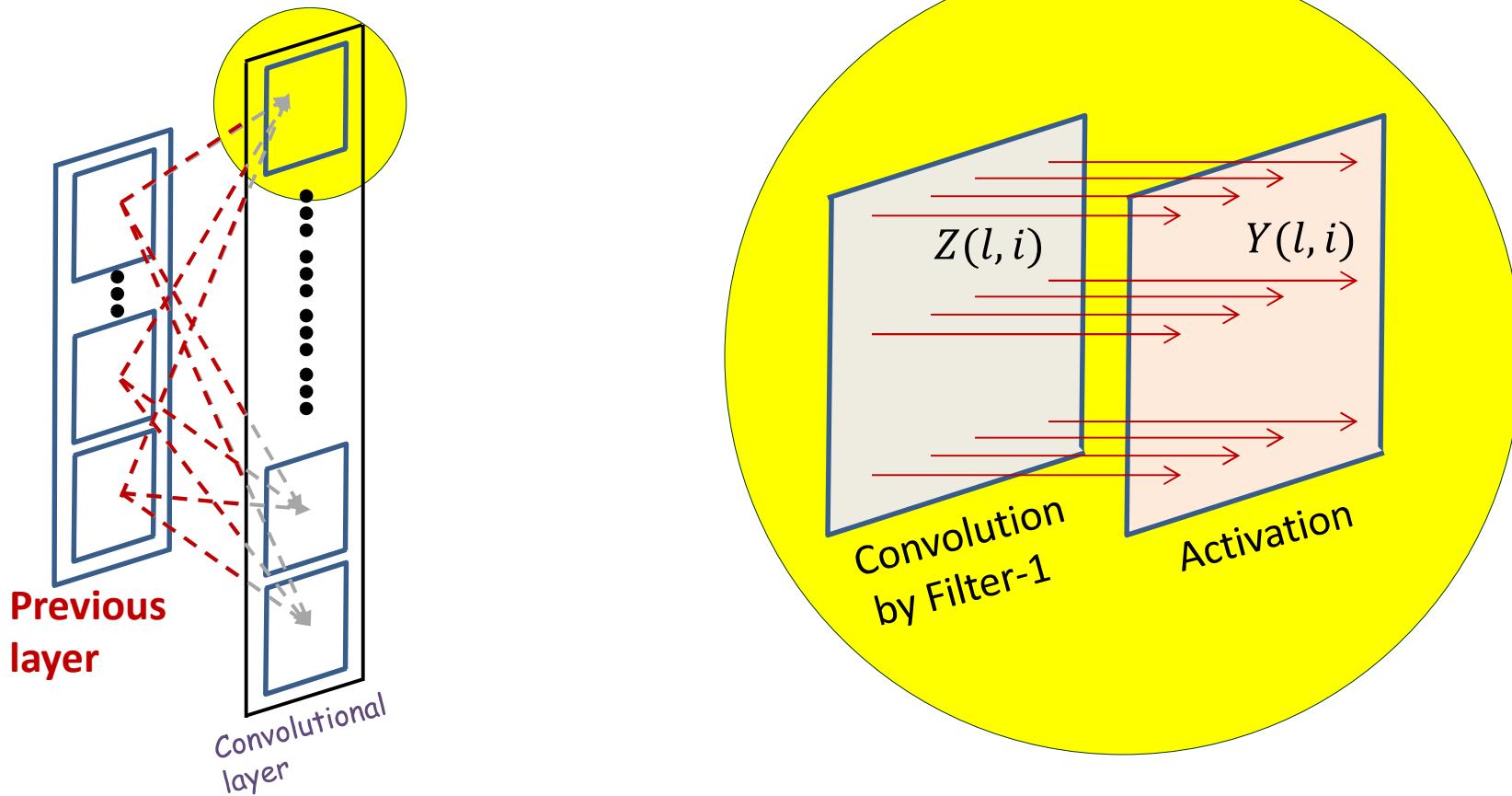
- Pattern classification tasks such as “does this picture contain a cat”, or “does this recording include HELLO” are best performed by scanning for the target pattern
- Scanning an input with a network and combining the outcomes is equivalent to scanning with individual neurons hierarchically
 - First level neurons scan the input
 - Higher-level neurons scan the “maps” formed by lower-level neurons
 - A final “decision” unit or layer makes the final decision
 - Deformations in the input can be handled by “pooling”
- For 2-D (or higher-dimensional) scans, the structure is called a convnet
- For 1-D scan along time, it is called a Time-delay neural network

Recap: The general architecture of a convolutional neural network



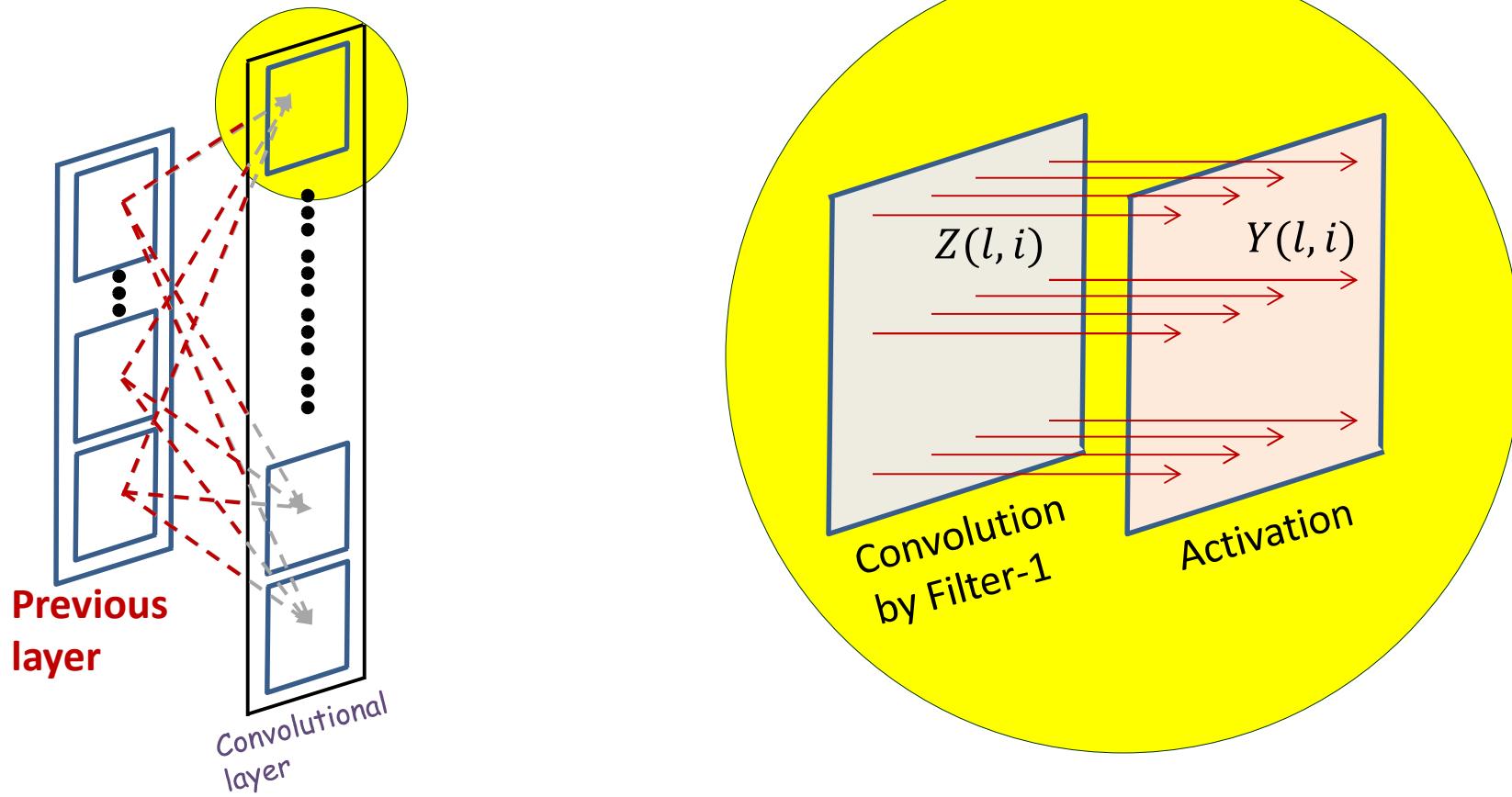
- A convolutional neural network comprises of “convolutional” and optional “downsampling” layers
- Followed by an MLP with one or more layers

Recap: A convolutional layer



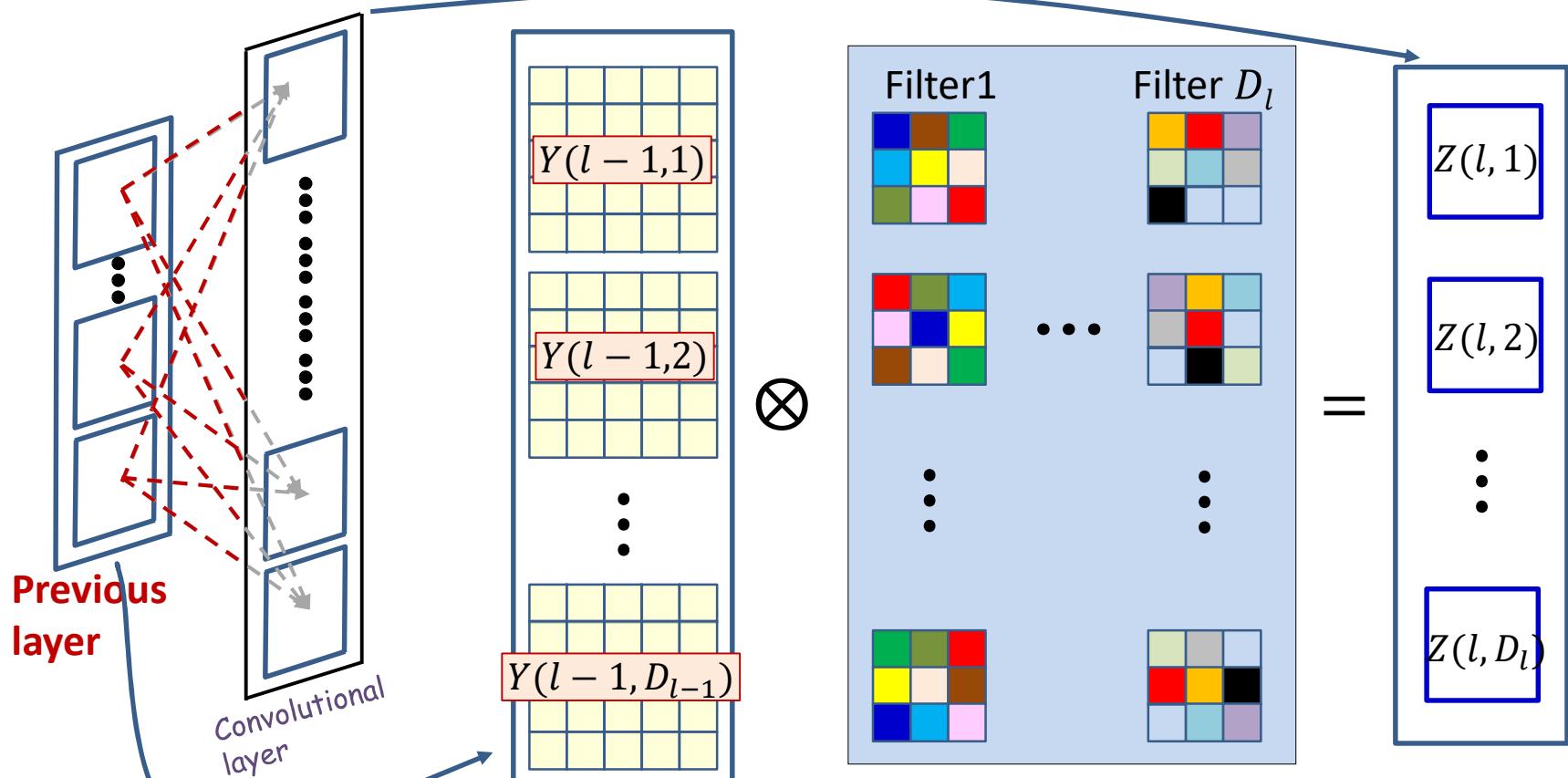
- The computation of each output map has two stages
 - Computing an *affine* map, by *convolution* over maps in the previous layer
 - Each affine map has, associated with it, a **learnable filter**
 - An *activation* that operates *point-wise* on the output of the convolution

Recap: A convolutional layer



- The computation of each output map has two stages
 - Computing an *affine* map, by *convolution* over maps in the previous layer
 - Each affine map has, associated with it, a **learnable filter**
 - An *activation* that operates *point-wise* on the output of the convolution

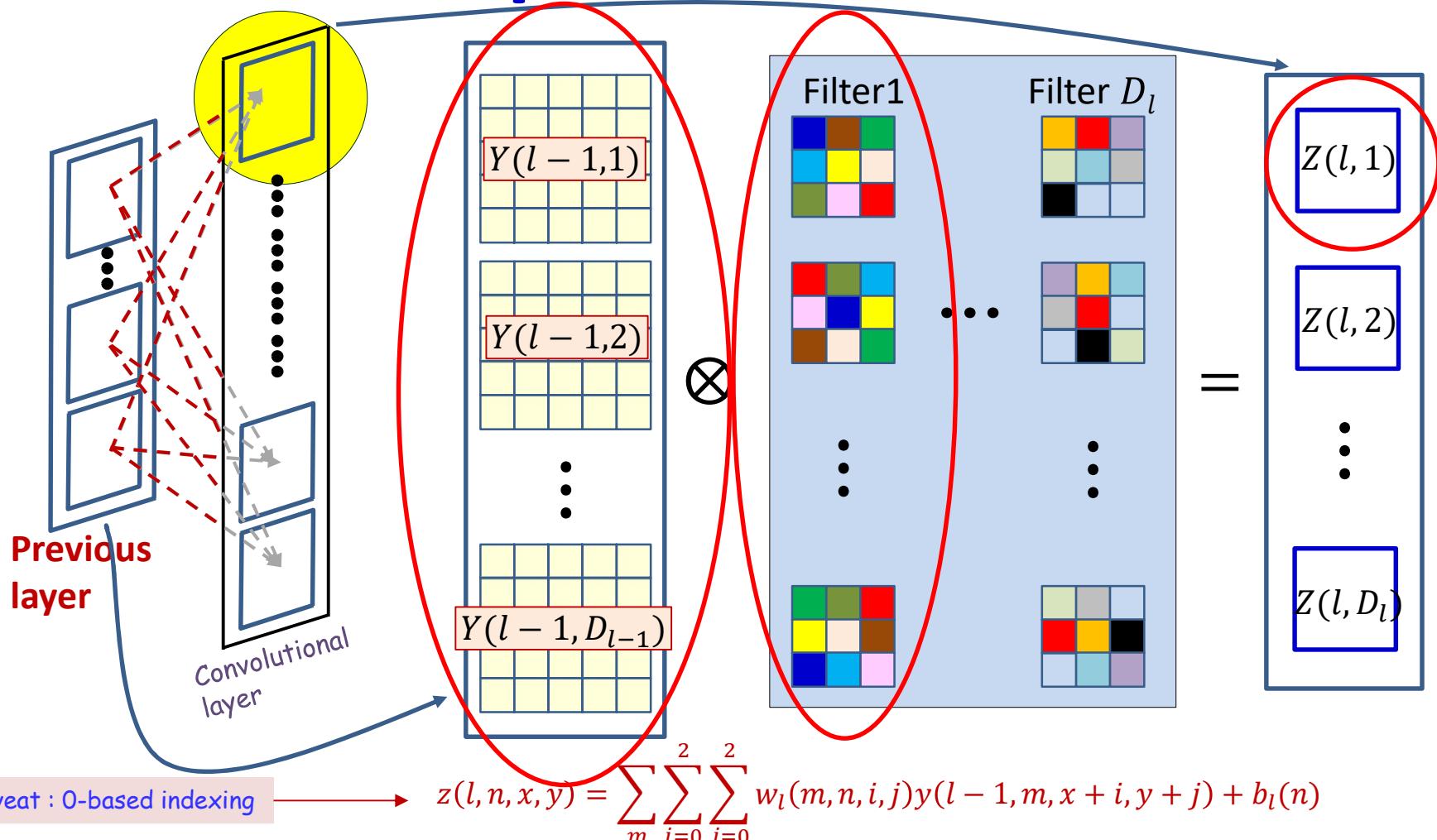
Recap: Convolution



Caveat : 0-based indexing $\longrightarrow z(l, n, x, y) = \sum_m^2 \sum_{i=0}^2 \sum_{j=0}^2 w_l(m, n, i, j)y(l - 1, m, x + i, y + j) + b_l(n)$

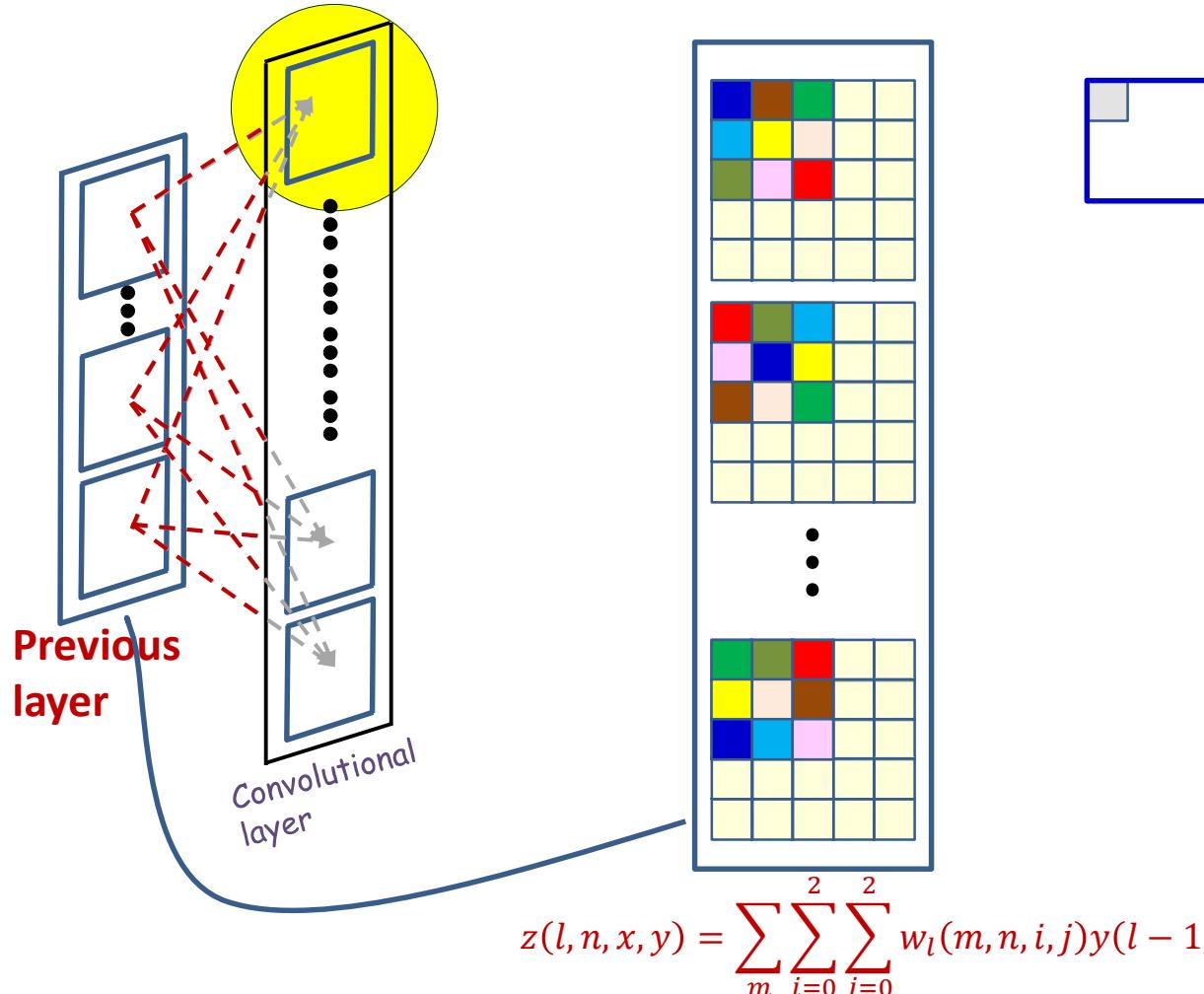
- Each affine output map is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
size of the filter x no. of maps in previous layer

Recap: Convolution



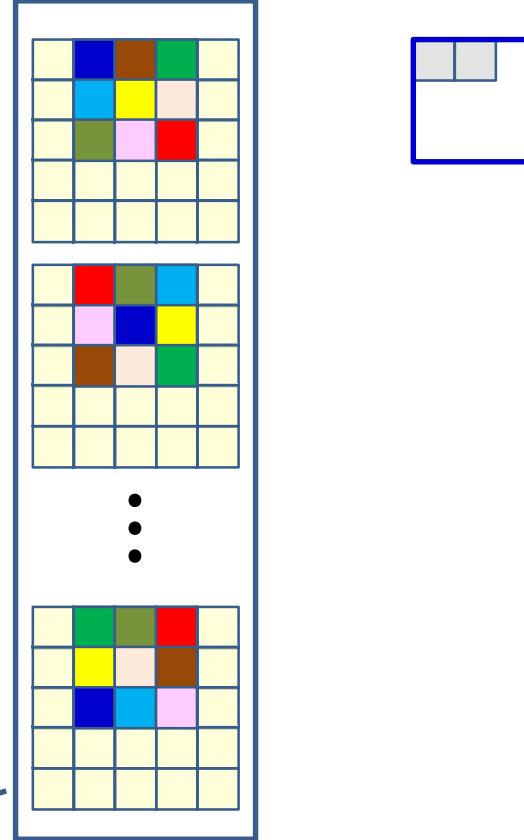
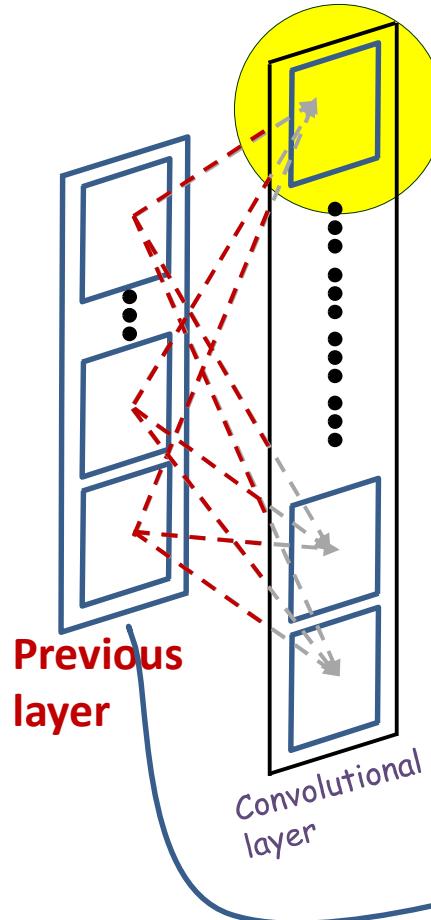
- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
size of the filter x no. of maps in previous layer

Recap: Convolution



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

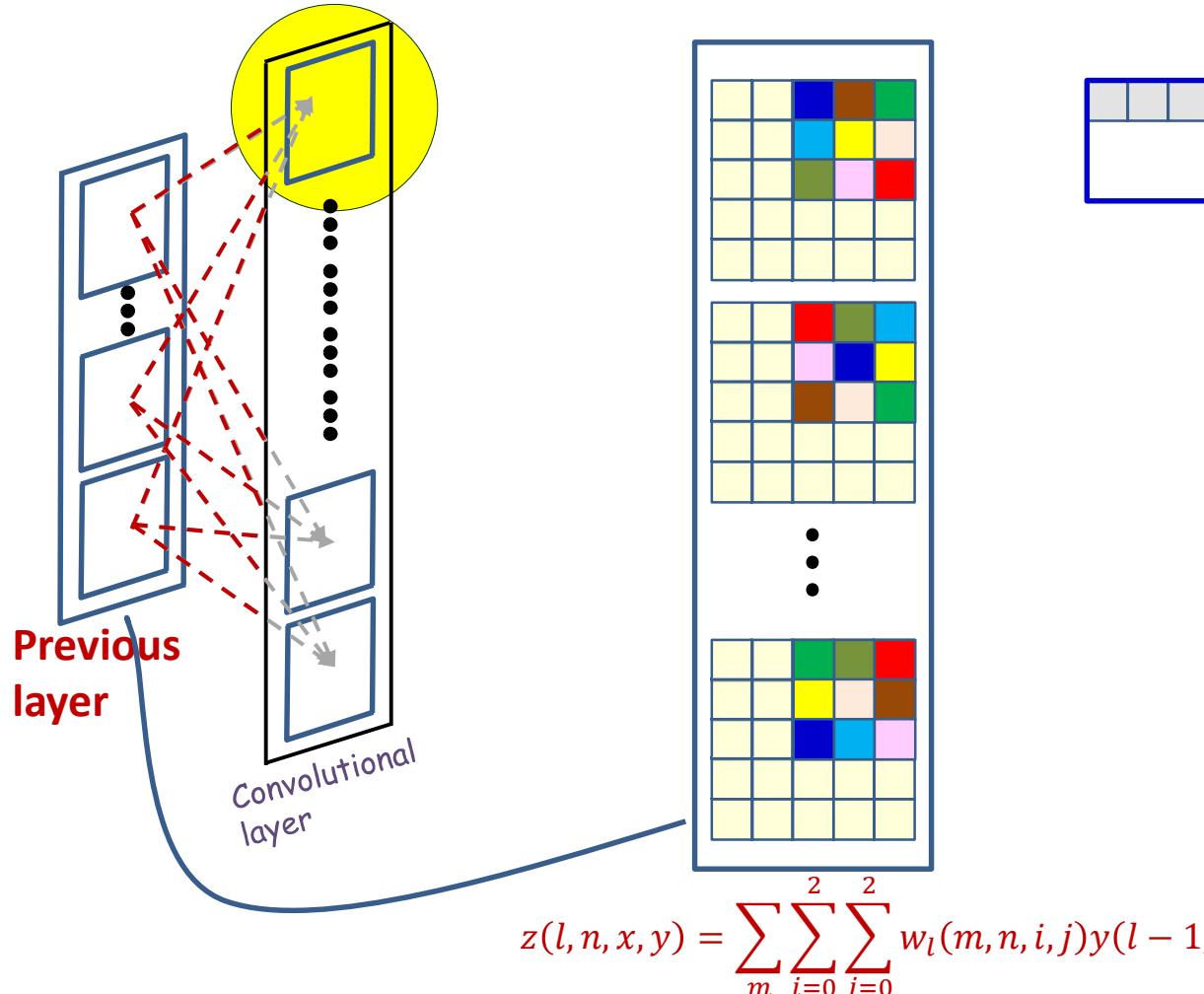
Recap: Convolution



$$z(l, n, x, y) = \sum_m \sum_{i=0}^2 \sum_{j=0}^2 w_l(m, n, i, j) y(l-1, m, x+i, y+j) + b_l(n)$$

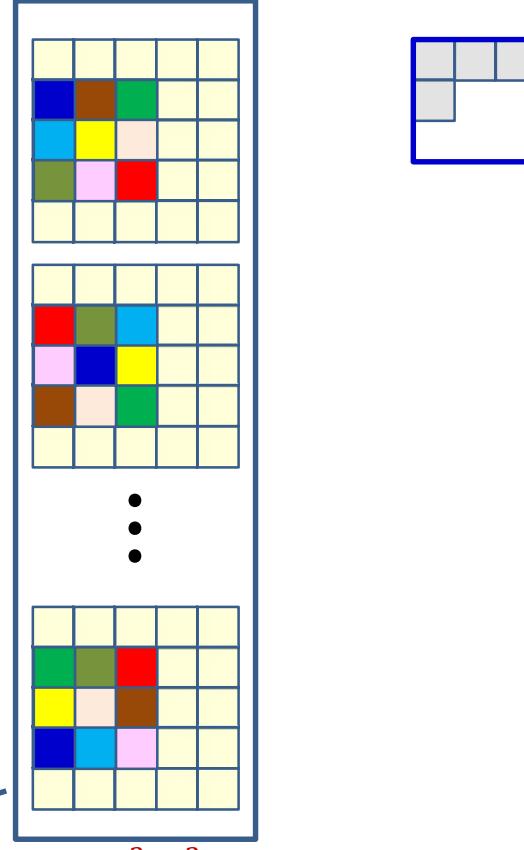
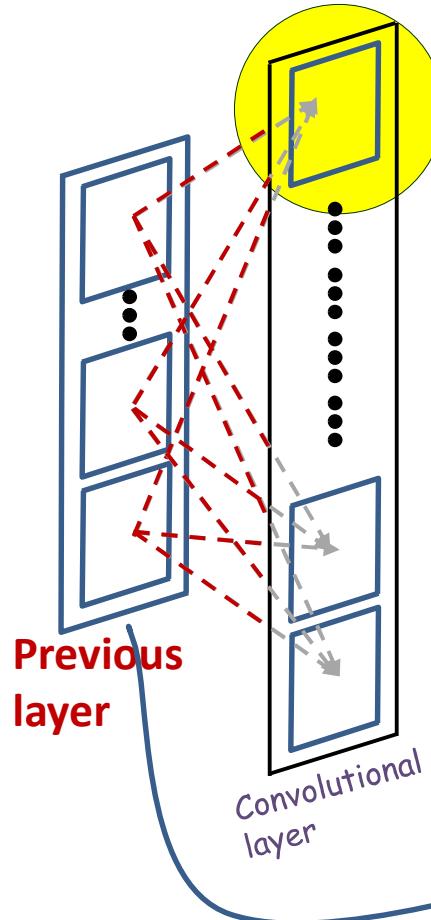
- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
size of the filter x no. of maps in previous layer

Recap: Convolution



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

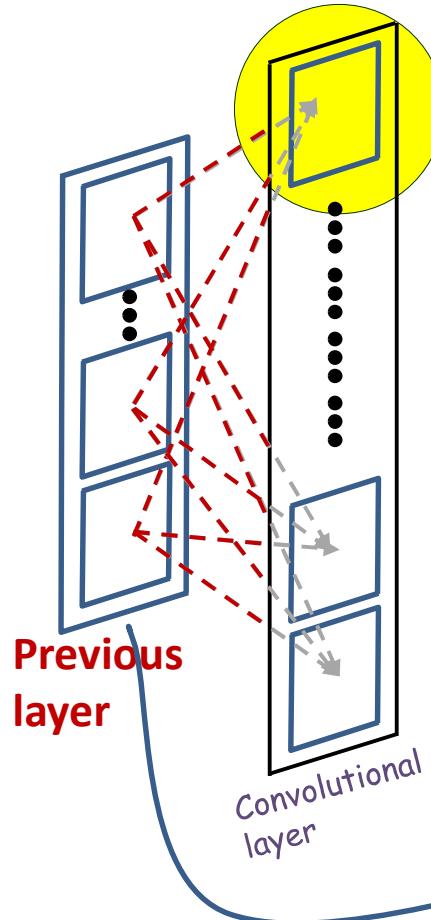
Recap: Convolution



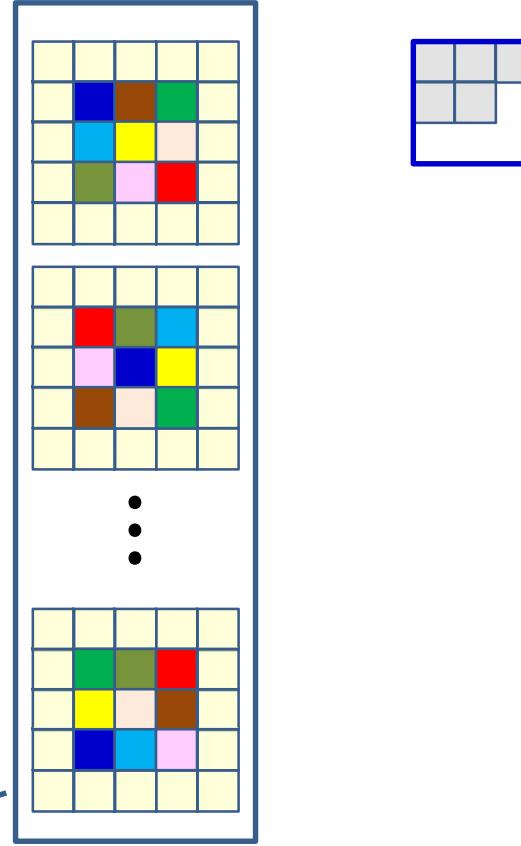
$$z(l, n, x, y) = \sum_m \sum_{i=0}^2 \sum_{j=0}^2 w_l(m, n, i, j)y(l - 1, m, x + i, y + j) + b_l(n)$$

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

Recap: Convolution

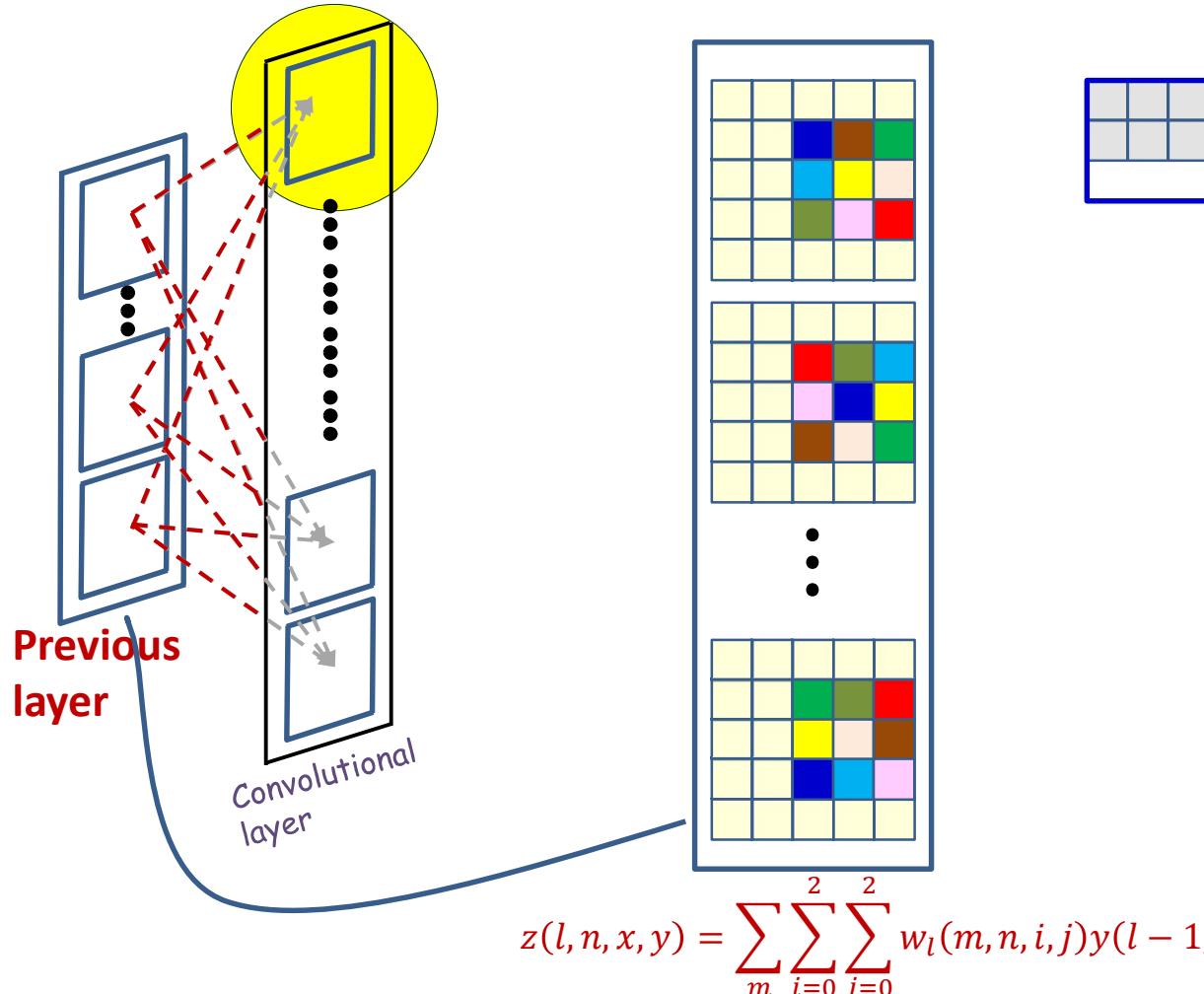


$$z(l, n, x, y) = \sum_m \sum_{i=0}^2 \sum_{j=0}^2 w_l(m, n, i, j)y(l-1, m, x+i, y+j) + b_l(n)$$



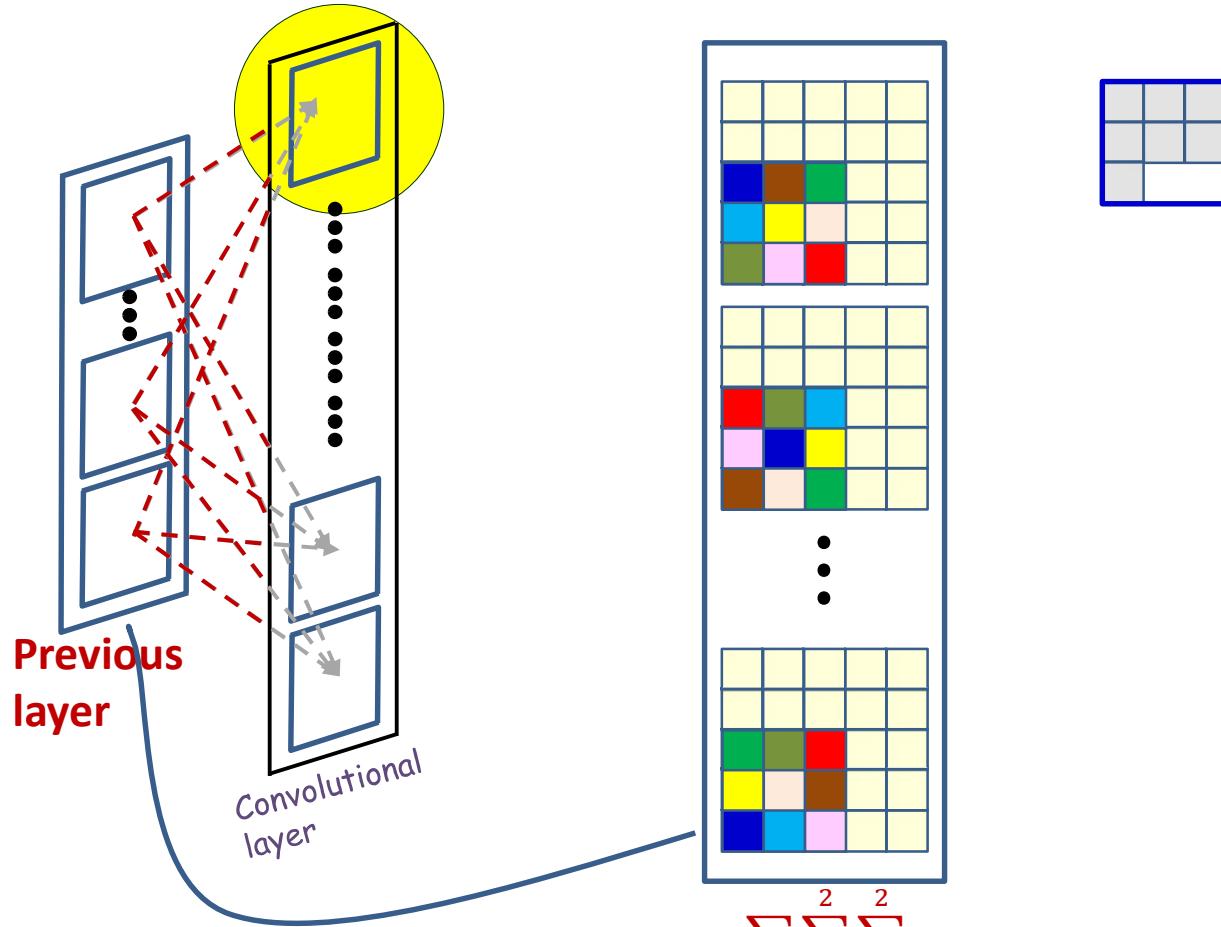
- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
size of the filter x no. of maps in previous layer

Recap: Convolution



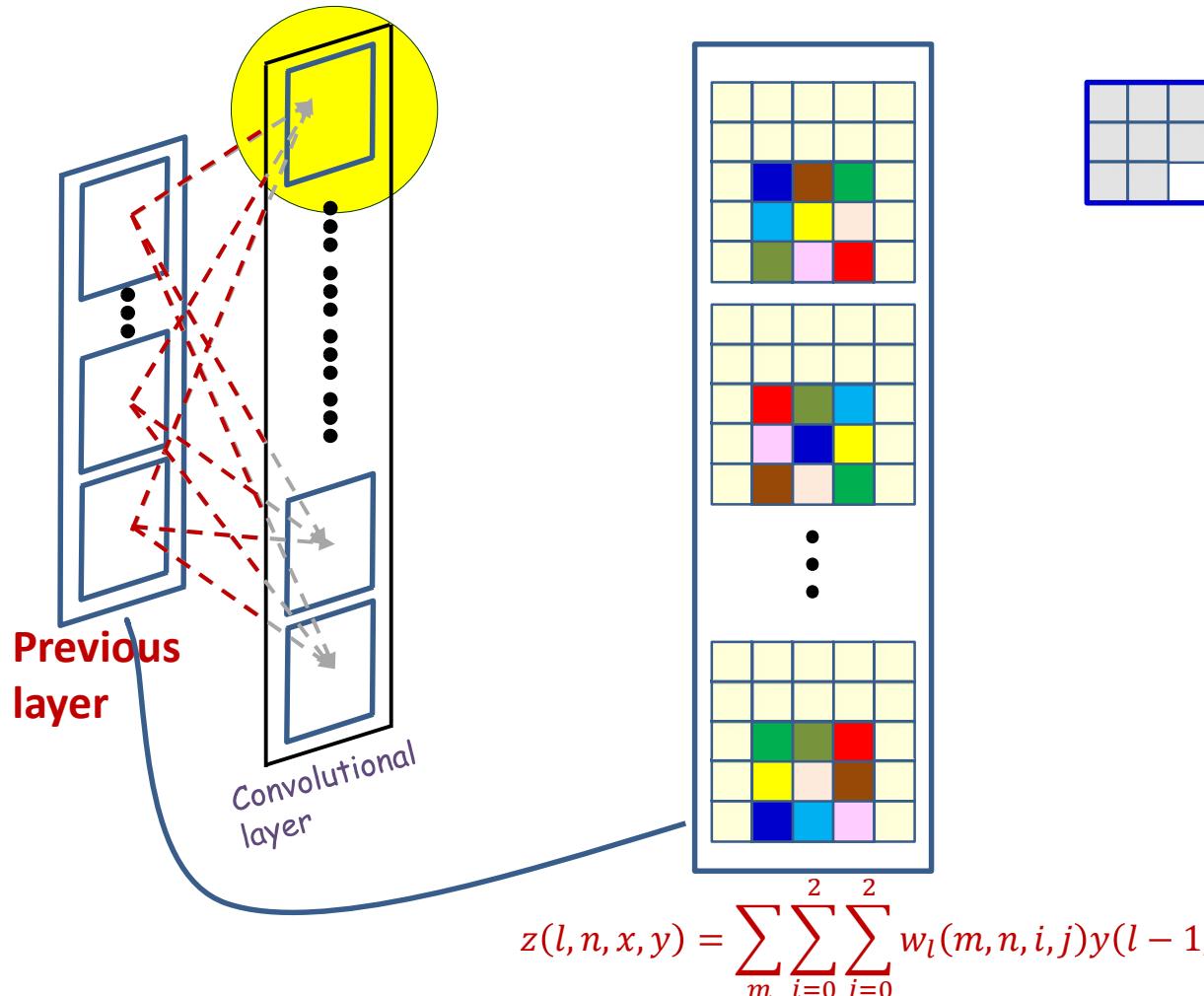
- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
size of the filter x no. of maps in previous layer

Recap: Convolution



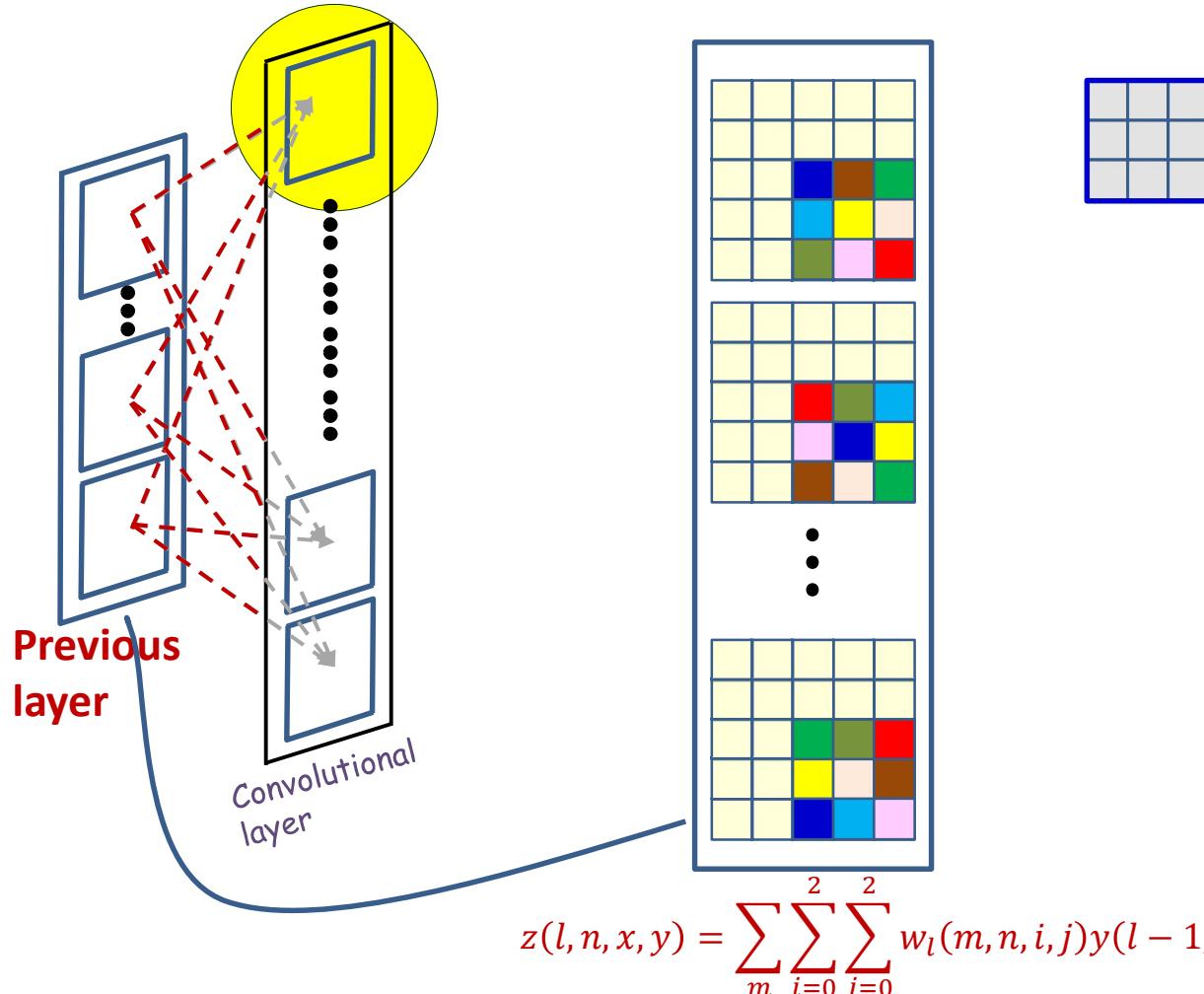
- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
size of the filter x no. of maps in previous layer

Recap: Convolution



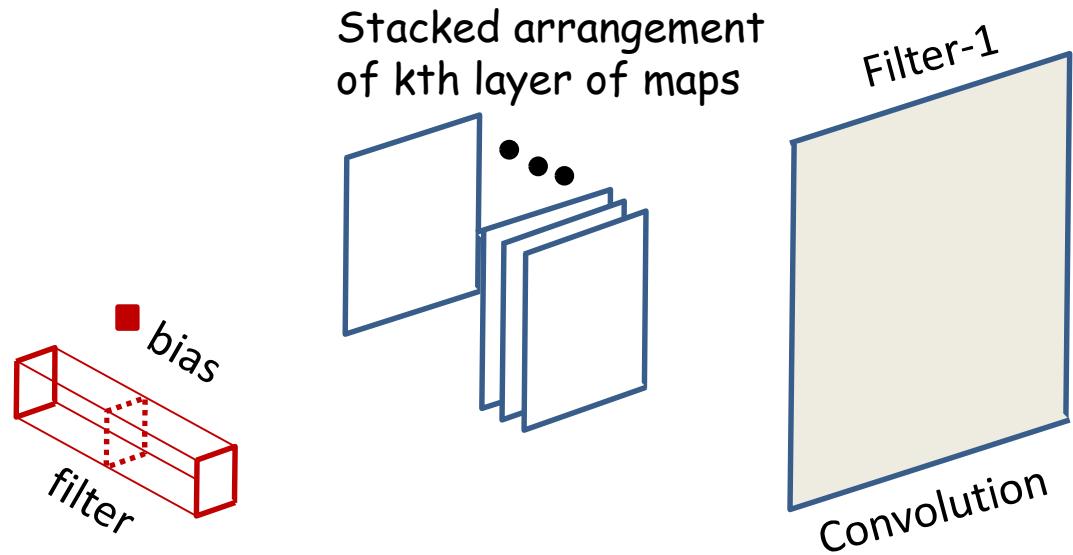
- Each affine output is computed from multiple input maps simultaneously
- There are as many weights (for each output map) as
size of the filter x no. of maps in previous layer

Recap: Convolution



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

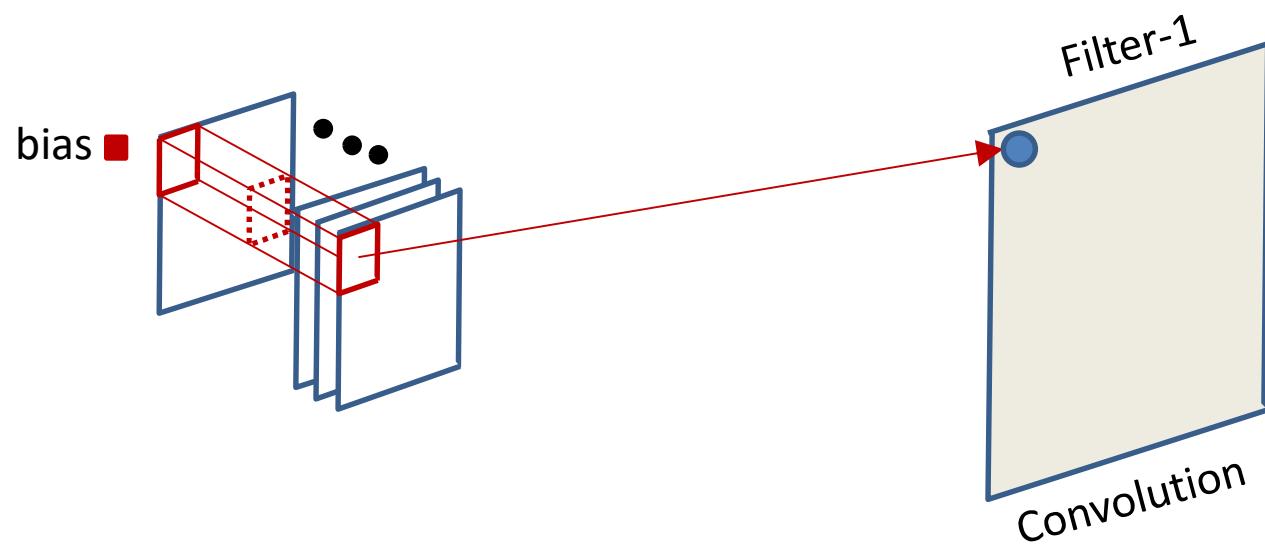
Recap: A cube visualization



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

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

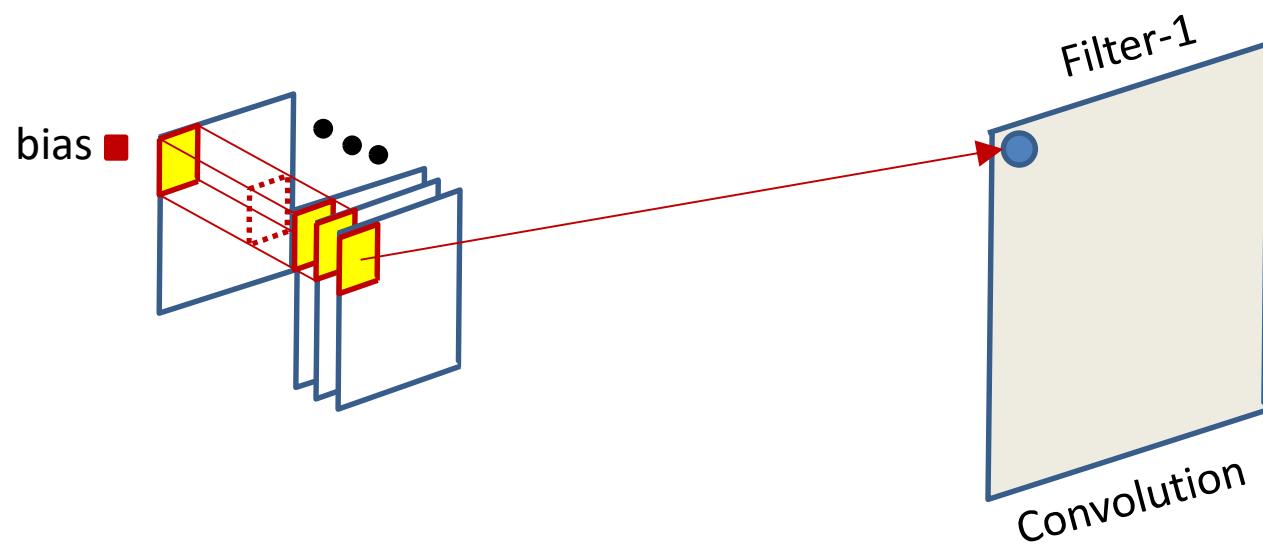
Recap: A cube visualization



$$z(l, n, x, y) = \sum_m \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} w_l(m, n, i, j) y(l - 1, m, x + i, y + j) + b_l(n)$$

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

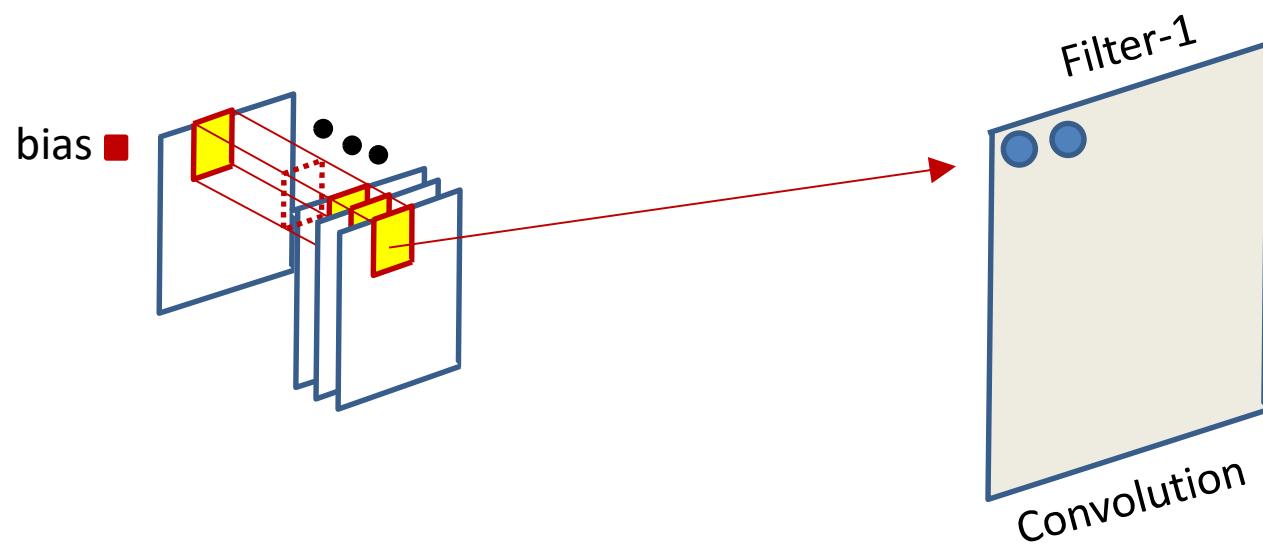
Recap: A cube visualization



$$z(l, n, x, y) = \sum_m \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} w_l(m, n, i, j) y(l - 1, m, x + i, y + j) + b_l(n)$$

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

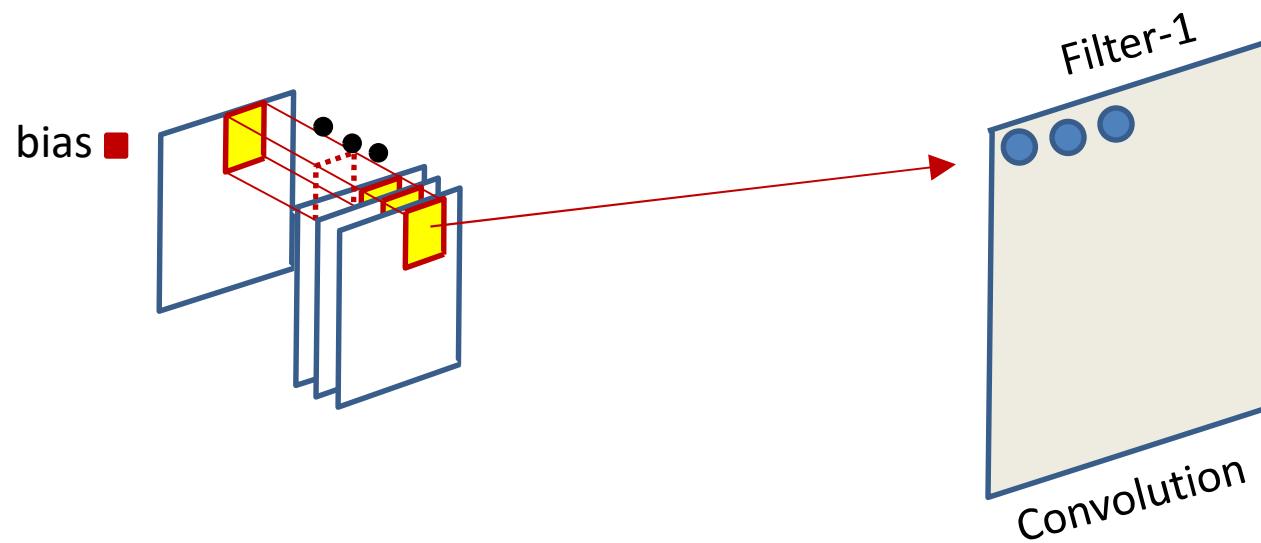
Recap: A cube visualization



$$z(l, n, x, y) = \sum_m \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} w_l(m, n, i, j) y(l - 1, m, x + i, y + j) + b_l(n)$$

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

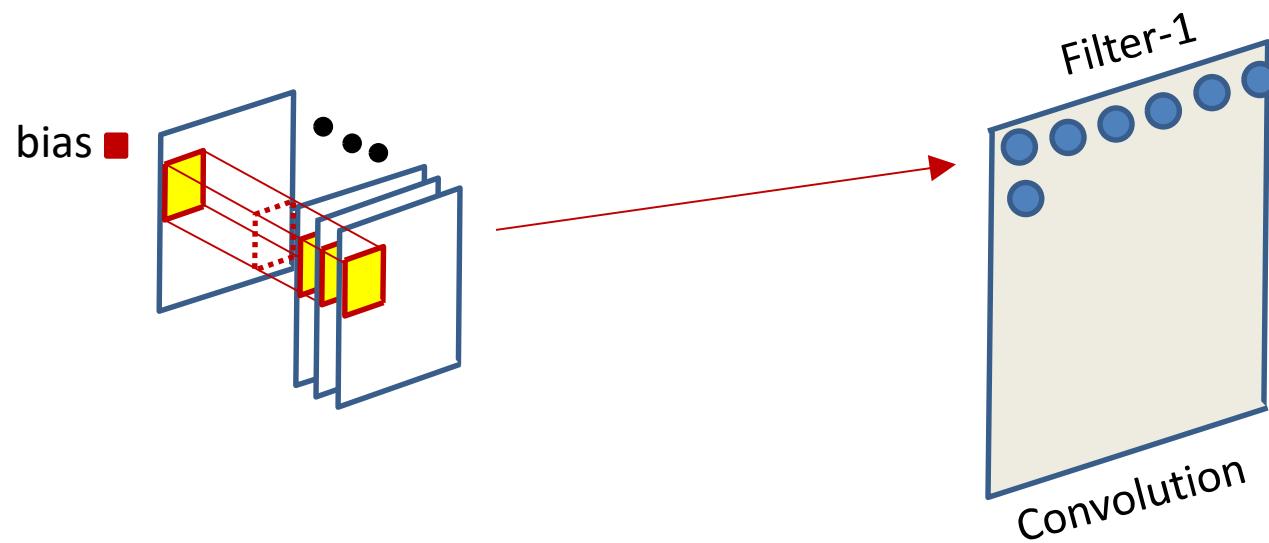
Recap: A cube visualization



$$z(l, n, x, y) = \sum_m \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} w_l(m, n, i, j) y(l - 1, m, x + i, y + j) + b_l(n)$$

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

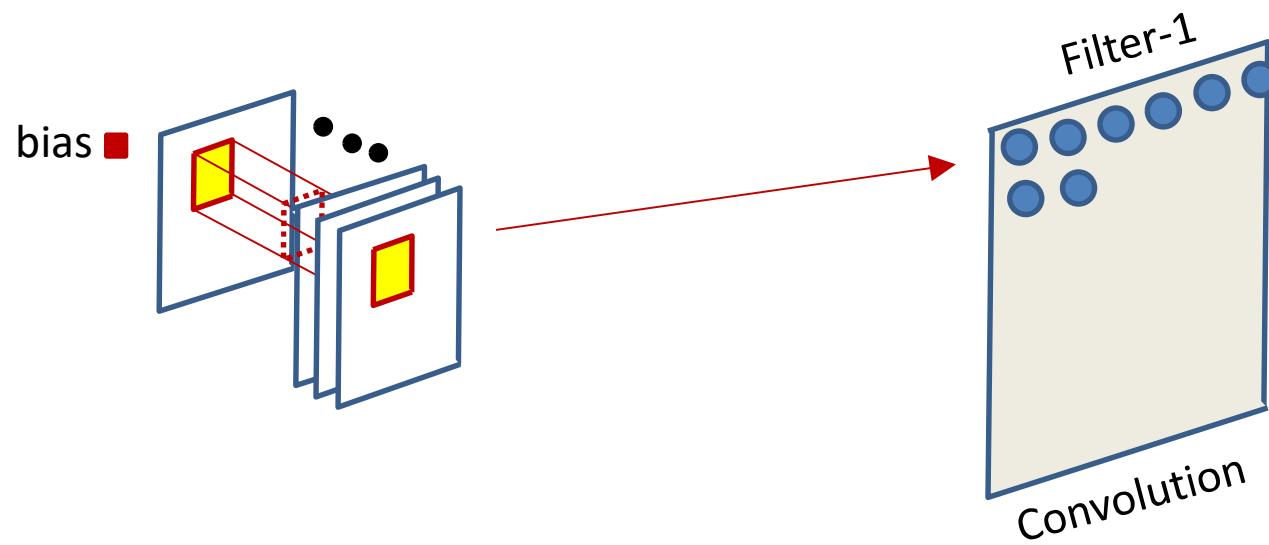
Recap: A cube visualization



$$z(l, n, x, y) = \sum_m \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} w_l(m, n, i, j) y(l - 1, m, x + i, y + j) + b_l(n)$$

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

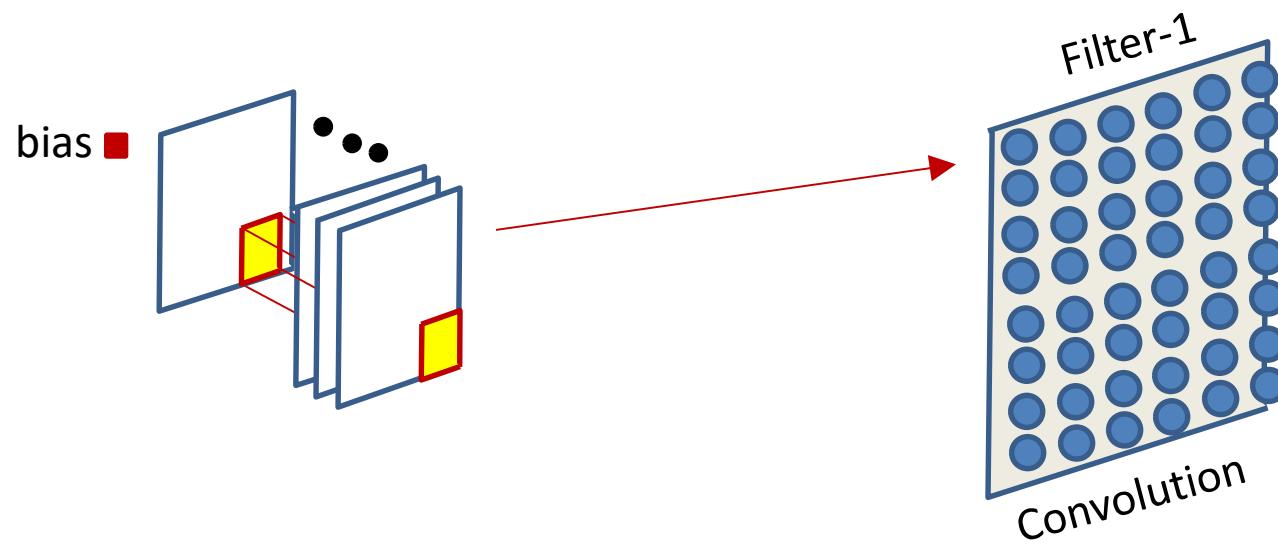
Recap: A cube visualization



$$z(l, n, x, y) = \sum_m \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} w_l(m, n, i, j) y(l - 1, m, x + i, y + j) + b_l(n)$$

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

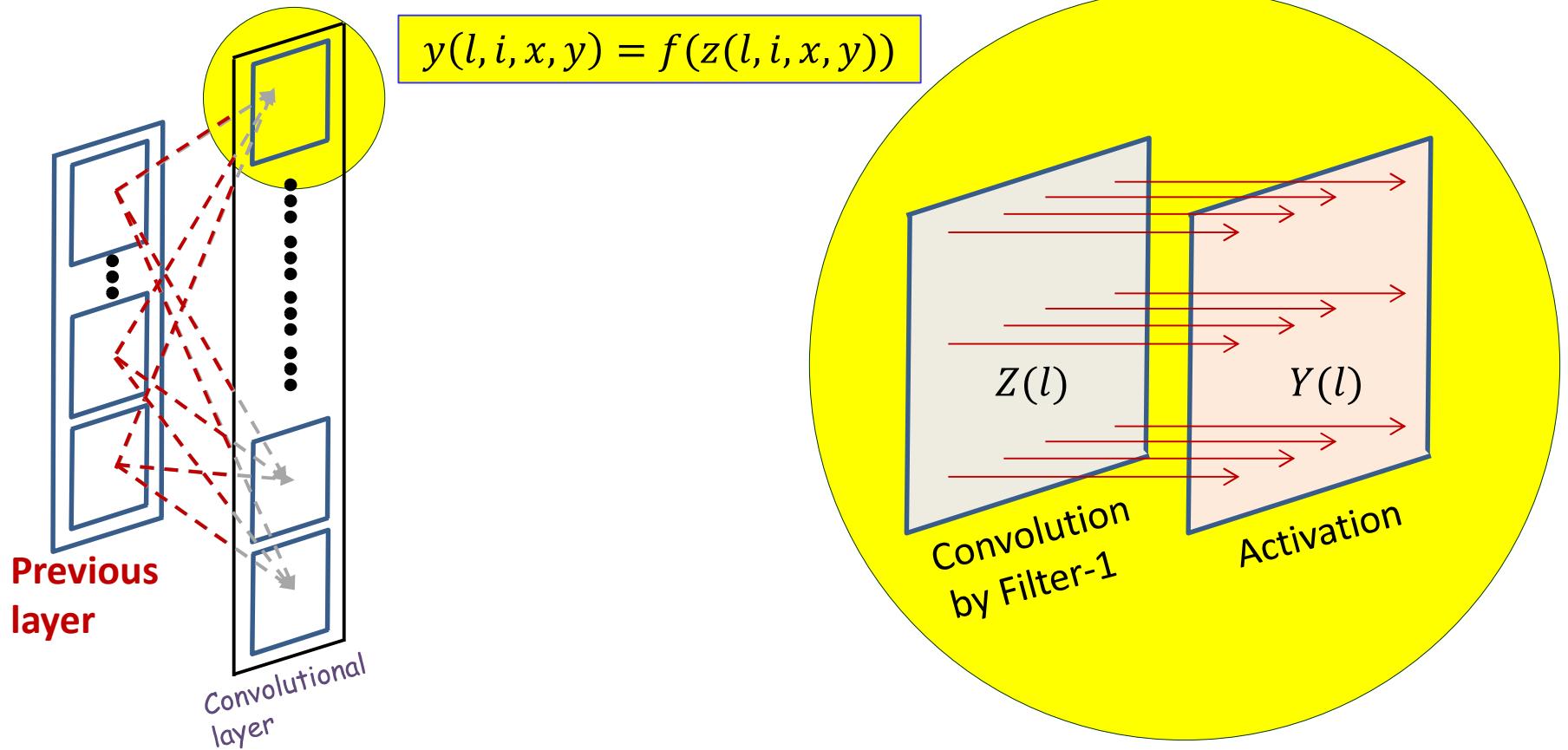
Recap: A cube visualization



$$z(l, n, x, y) = \sum_m \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} w_l(m, n, i, j) y(l - 1, m, x + i, y + j) + b_l(n)$$

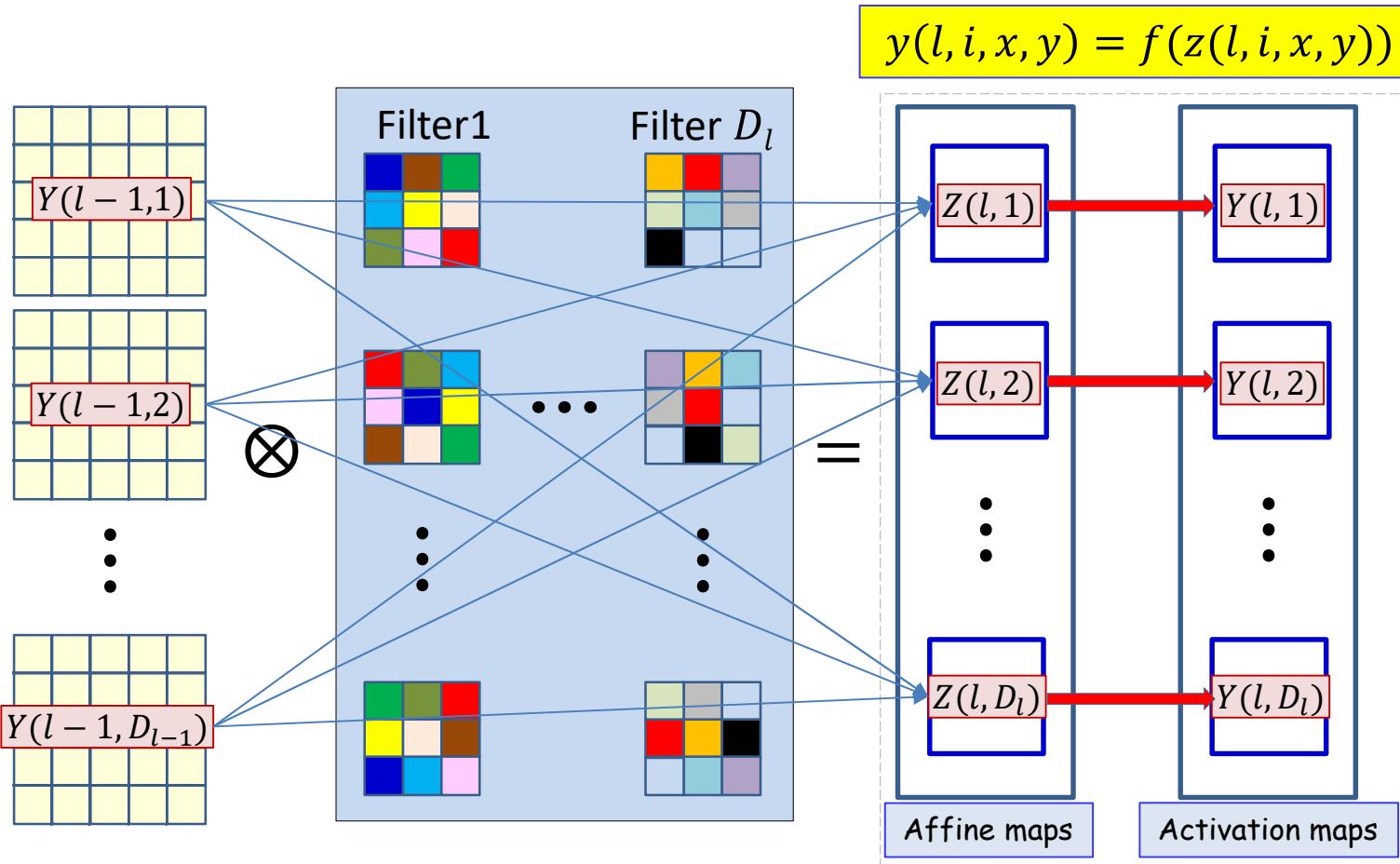
- The computation of the convolutive map at any location *sums* the convolutive outputs *at all planes*

Recap: A convolutional layer



- The computation of each output map has two stages
 - Computing an *affine* map, 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

Convolution layer: A more explicit illustration



- Input maps $Y(l - 1, *)$ are convolved with several filters to generate the affine maps $Z(l, *)$
 - Each filter consists of a set of square patterns of weights, with one set for each map in $Y(l - 1, *)$
 - We get one affine map per filter
- A *point-wise* activation function $f(z)$ is applied to each map in $Z(l, *)$ to produce the activation maps $Y(l, *)$

Pseudocode: Vector notation

The weight $W(l, j)$ is a 3D $D_{l-1} \times K_l \times K_l$ tensor

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

Pseudocode has 1-based indexing

Pseudocode: Vector notation

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

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

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

Poll 1

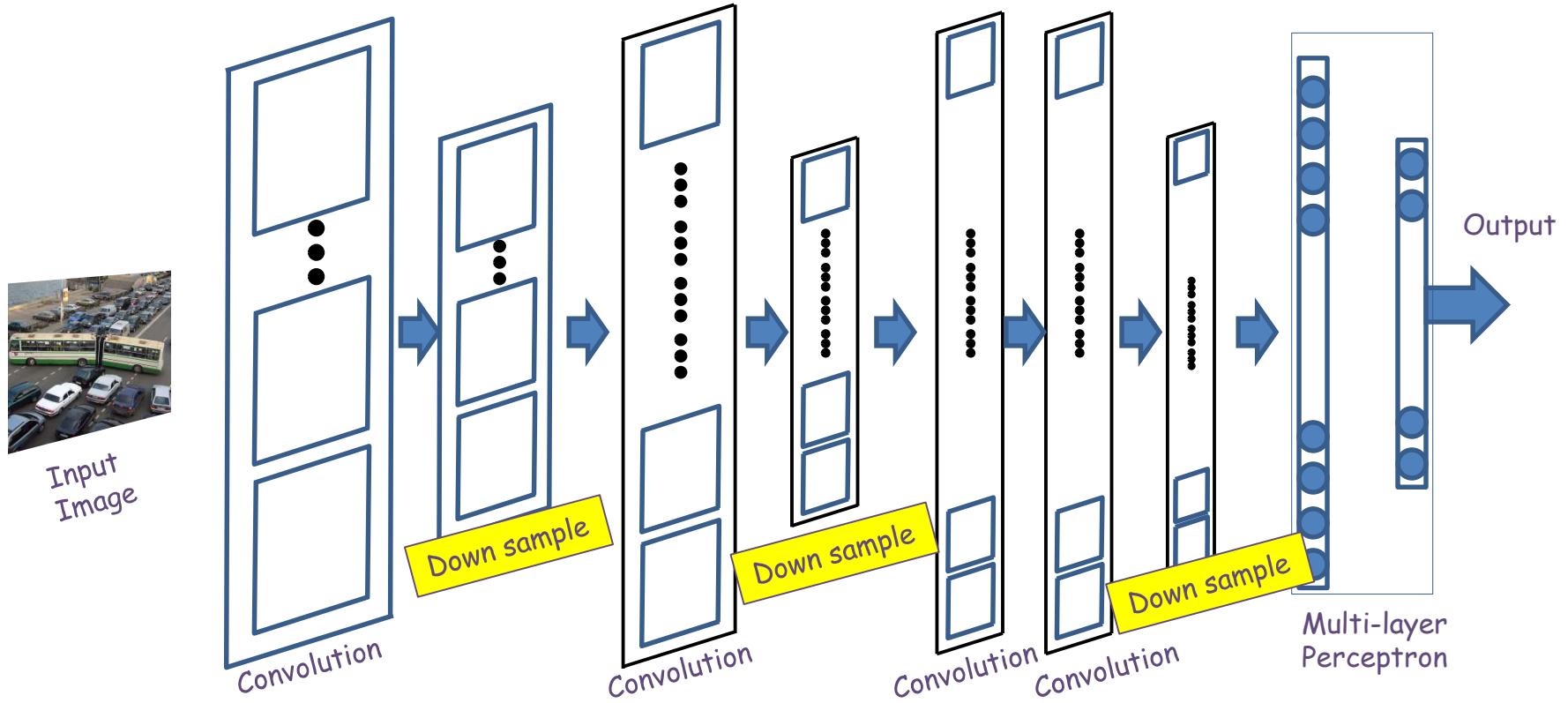
- @885

Poll 1

Select all true statements about a convolution layer.

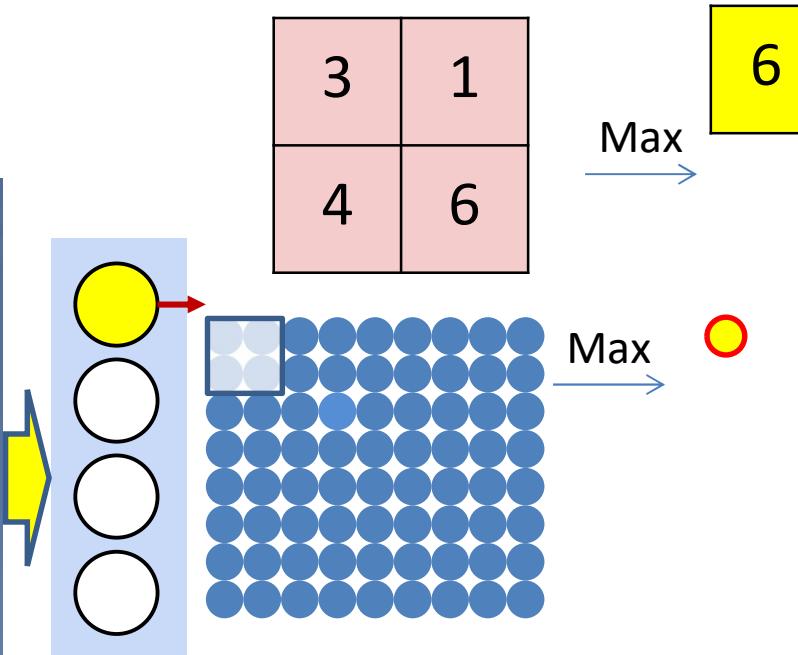
- **The number of “planes” in any filter equals the number of input maps (output maps from the previous layer)**
- The number of “planes” in any filter equals the number of output maps (affine maps output by the layer)
- The number of filters equals the number of input maps
- **The number of filters equals the number of output maps**

Downsampling/Pooling



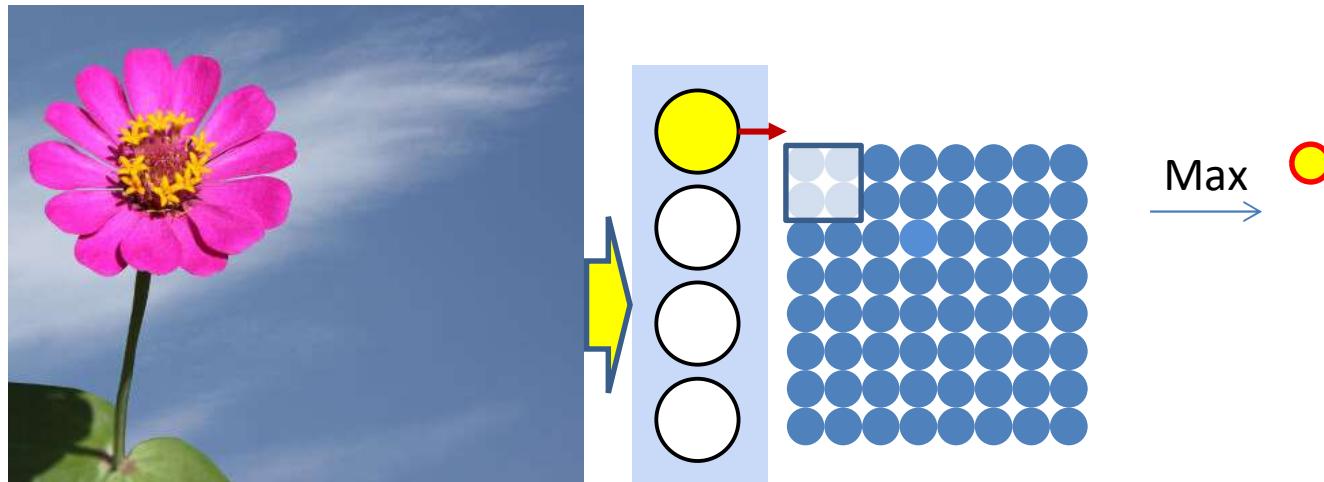
- Convolutional (and activation) layers are followed intermittently by “downsampling” (or “pooling”) layers
 - Often, they alternate with convolution, though this is not necessary

Recall: Max pooling



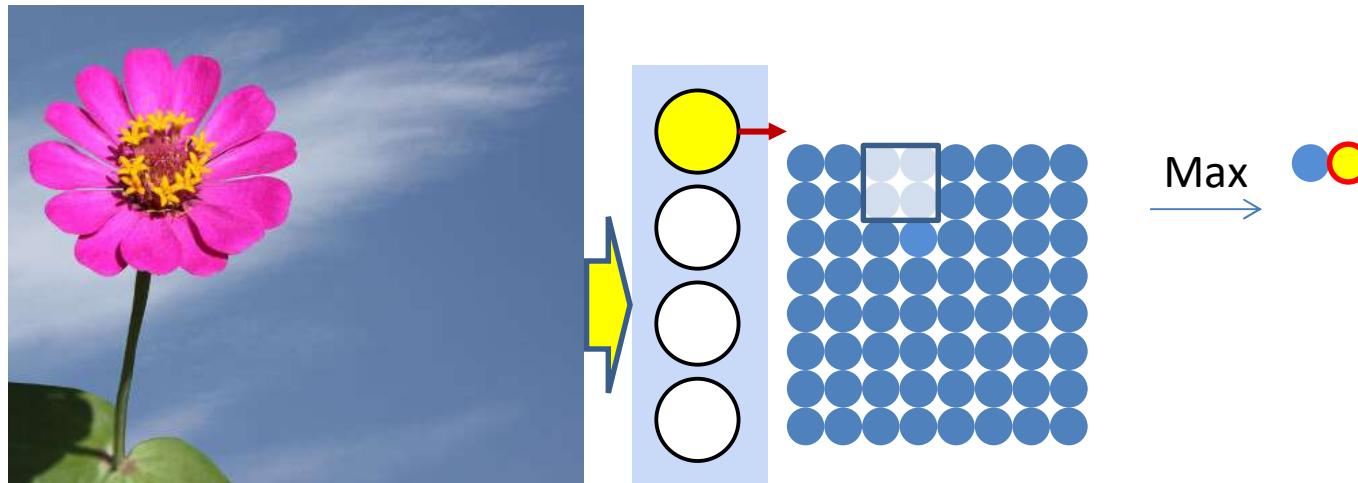
- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

Pooling and downsampling



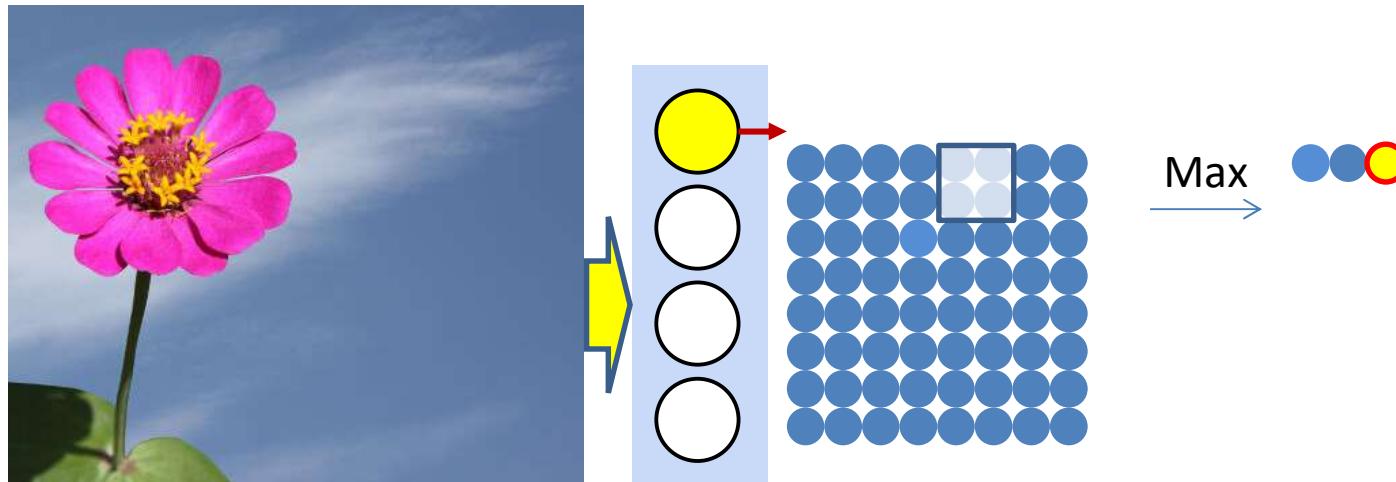
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



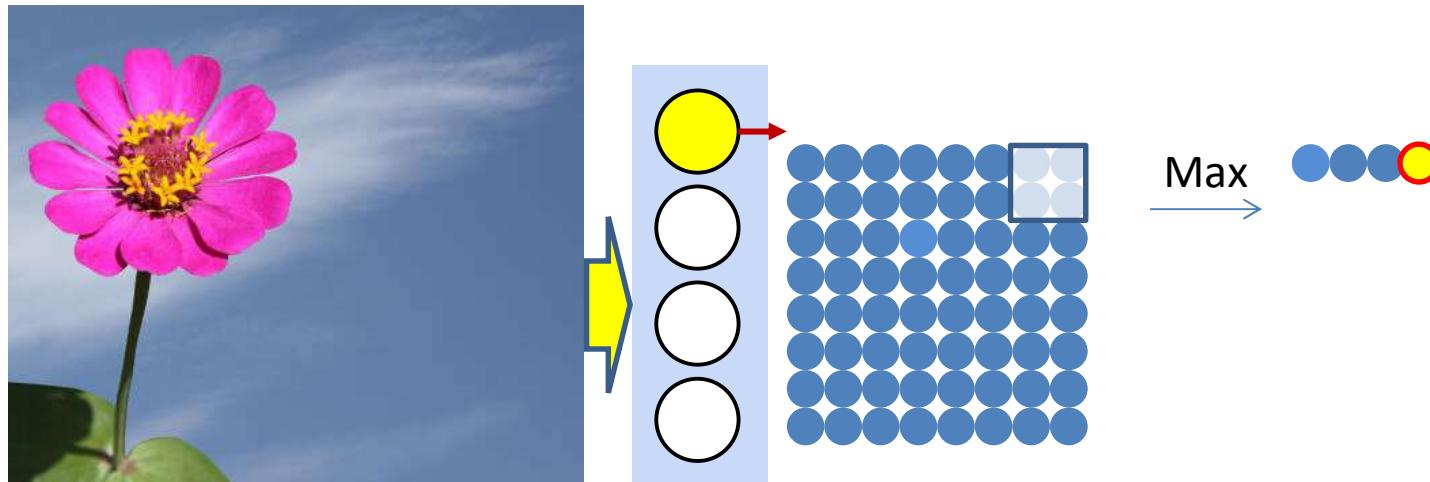
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



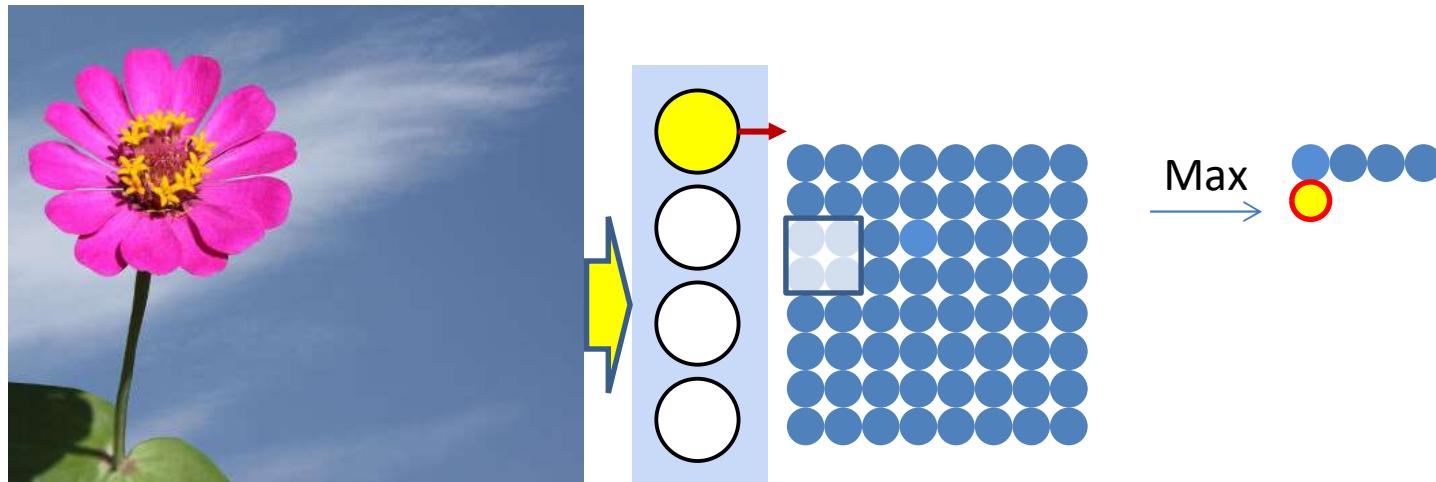
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



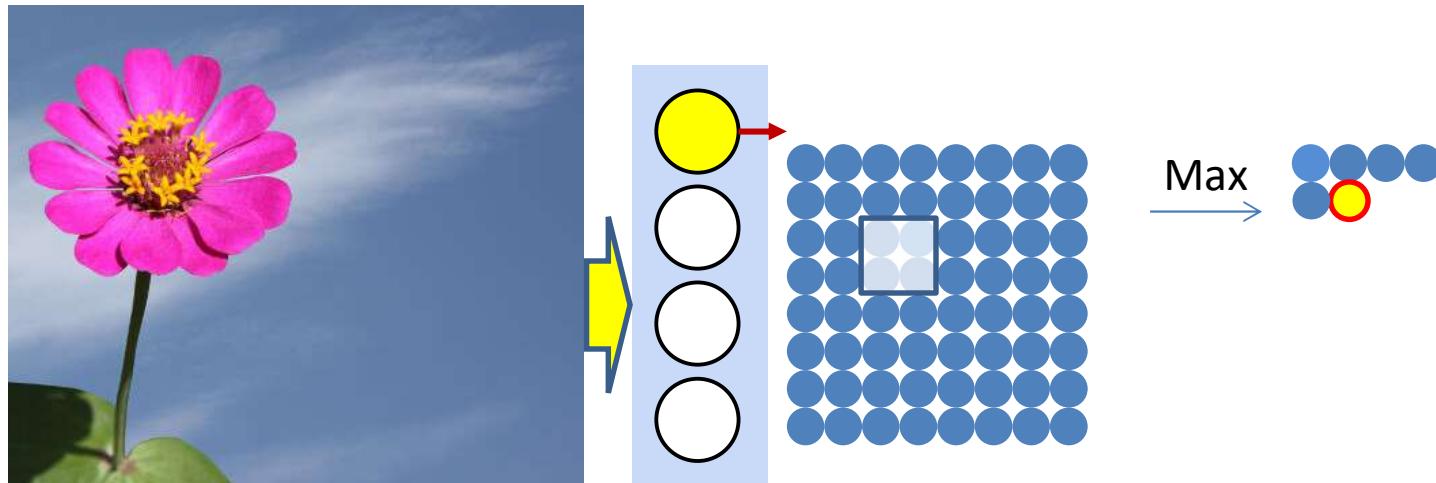
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



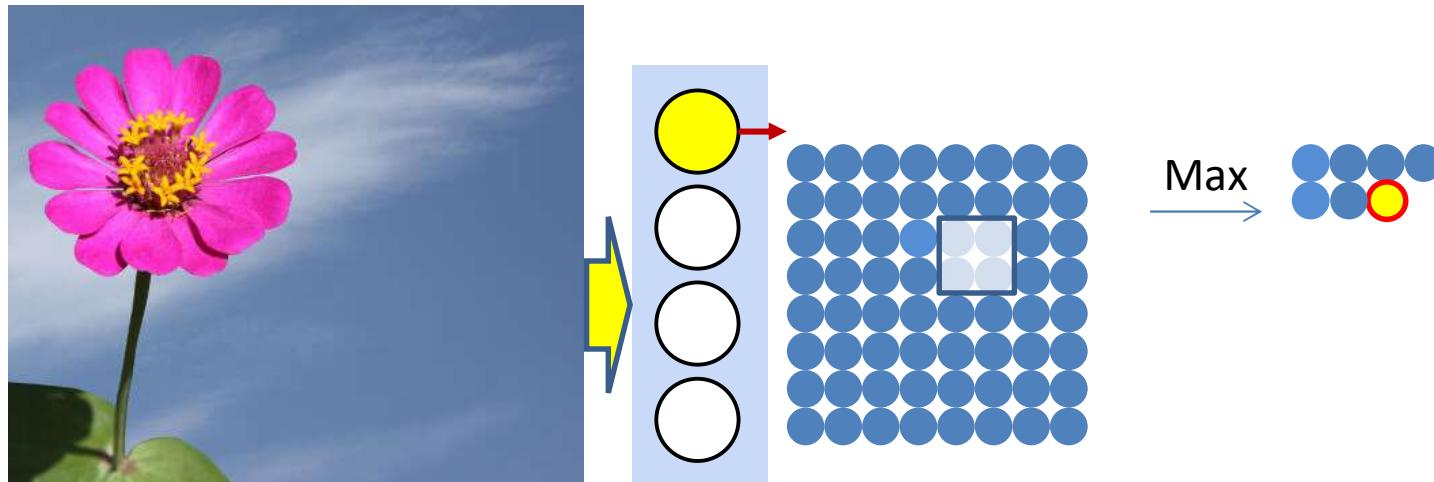
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



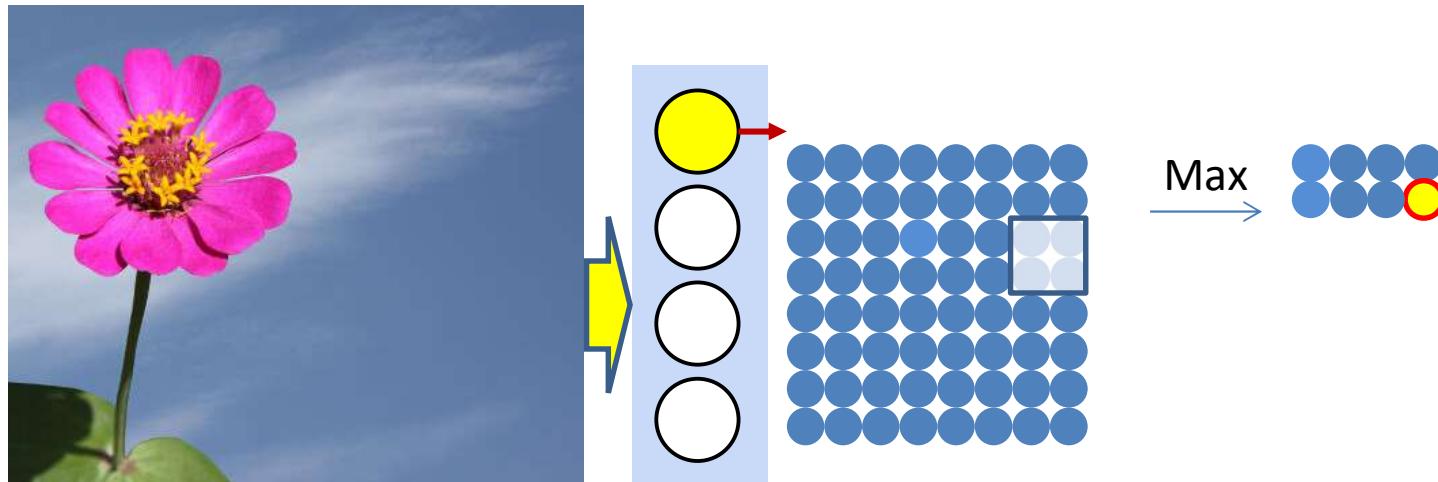
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



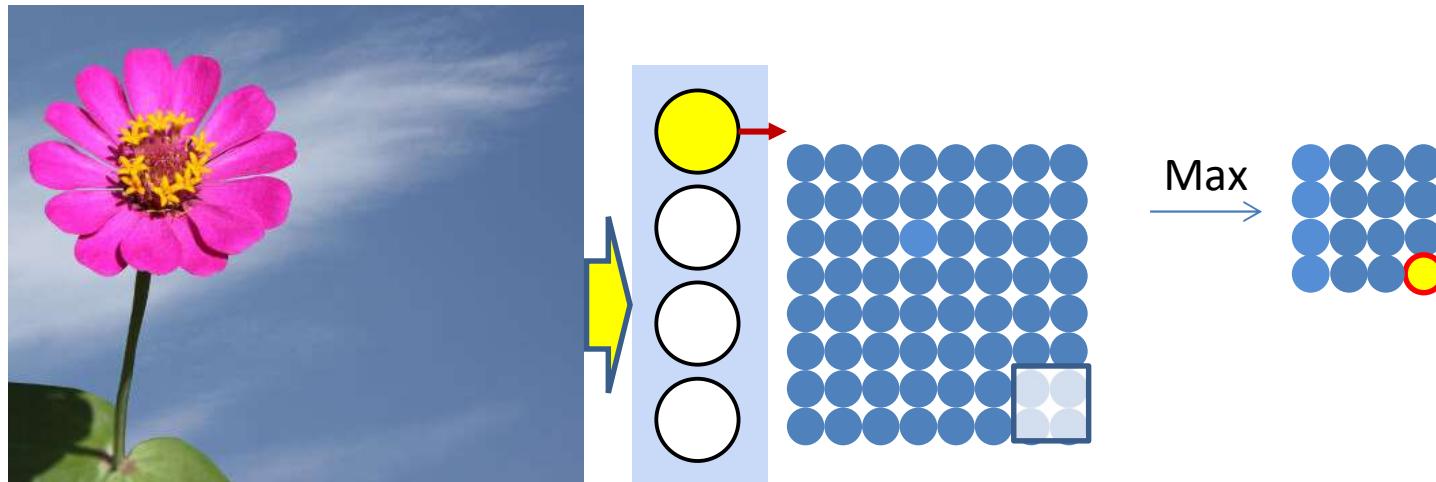
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



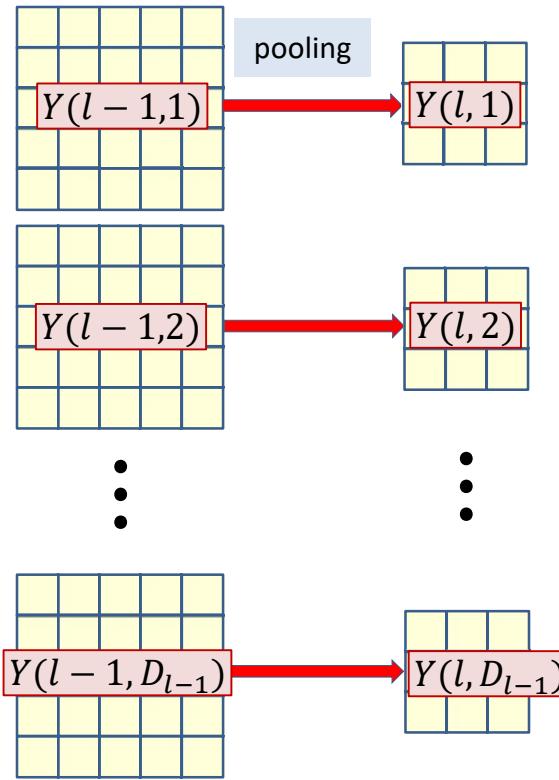
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Recap: Pooling and downsampling layer



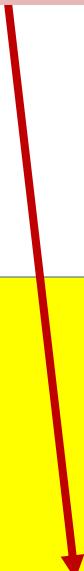
- Input maps $Y(l - 1, *)$ are operated on individually by pooling operations to produce the pooled maps $Y(l, *)$
 - Pooling is performed with stride > 1 resulting in downsampling
 - Output maps are smaller than input maps

Recap: Max Pooling layer at layer l

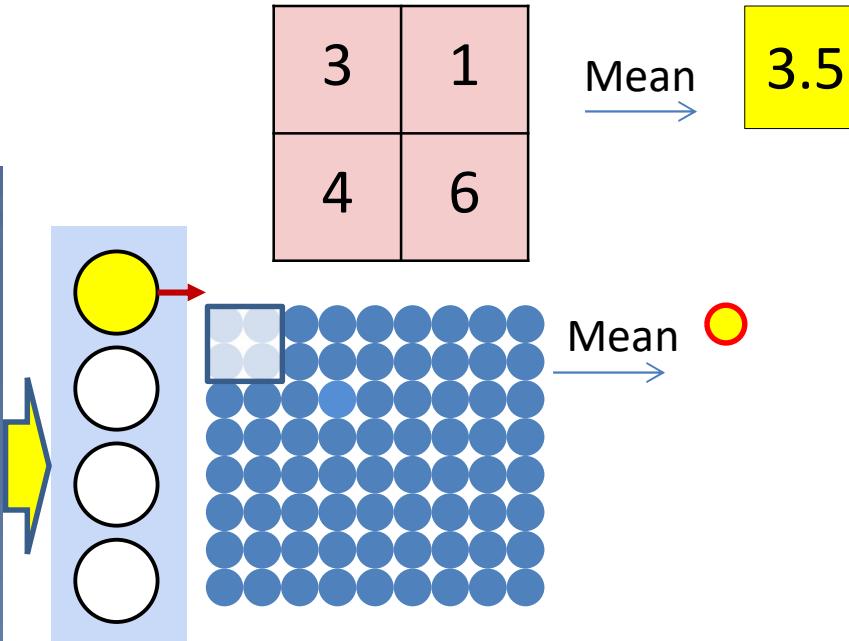
- a) Performed separately for every map (j).
*) Not combining multiple maps within a single max operation.
- b) Keeping track of location of max

Max pooling

```
for j = 1:D1
    m = 1
    for x = 1:stride(l):Wl-1-Kl+1
        n = 1
        for y = 1:stride(l):Hl-1-Kl+1
            pidx(l,j,m,n) = maxidx(Y(l-1,j,x:x+Kl-1,y:y+Kl-1))
            u(l,j,m,n) = Y(l-1,j,pidx(l,j,m,n))
            n = n+1
        m = m+1
```



Recall: Mean pooling



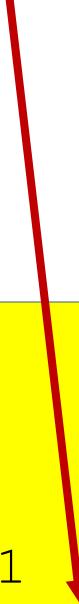
- Mean pooling computes the *mean* of the window of values
 - As opposed to the max of max pooling
- Scanning with strides is otherwise identical to max pooling

Recap: Mean Pooling layer at layer l

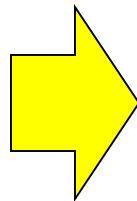
a) Performed separately for every map (j)

Mean pooling

```
for j = 1:D1
    m = 1
    for x = 1:stride(l):Wl-1-Kl+1
        n = 1
        for y = 1:stride(l):Hl-1-Kl+1
            u(l,j,m,n) = mean(Y(l-1,j,x:x+Kl-1,y:y+Kl-1))
            n = n+1
        m = m+1
```

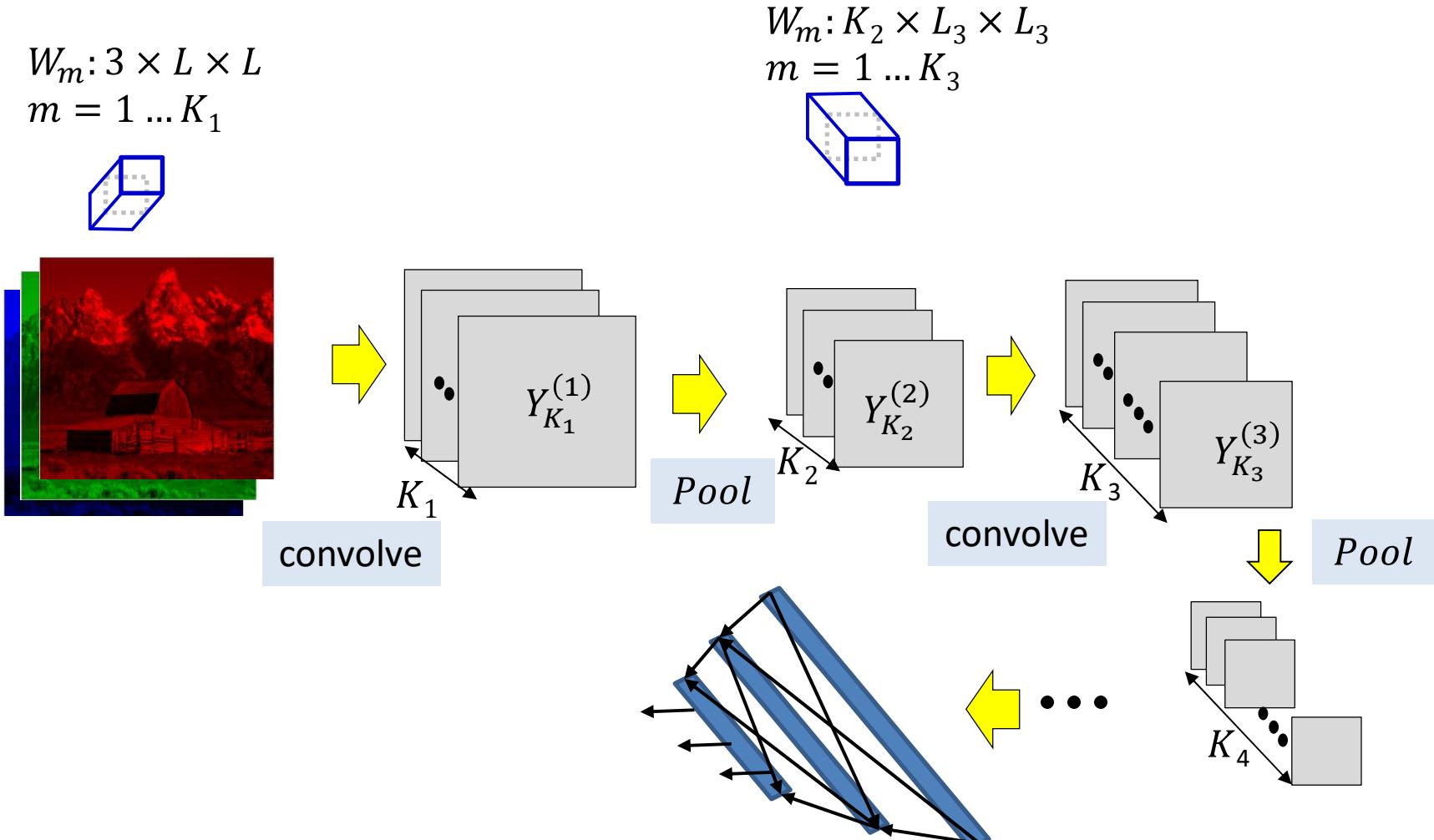


Recap: A CNN, end-to-end



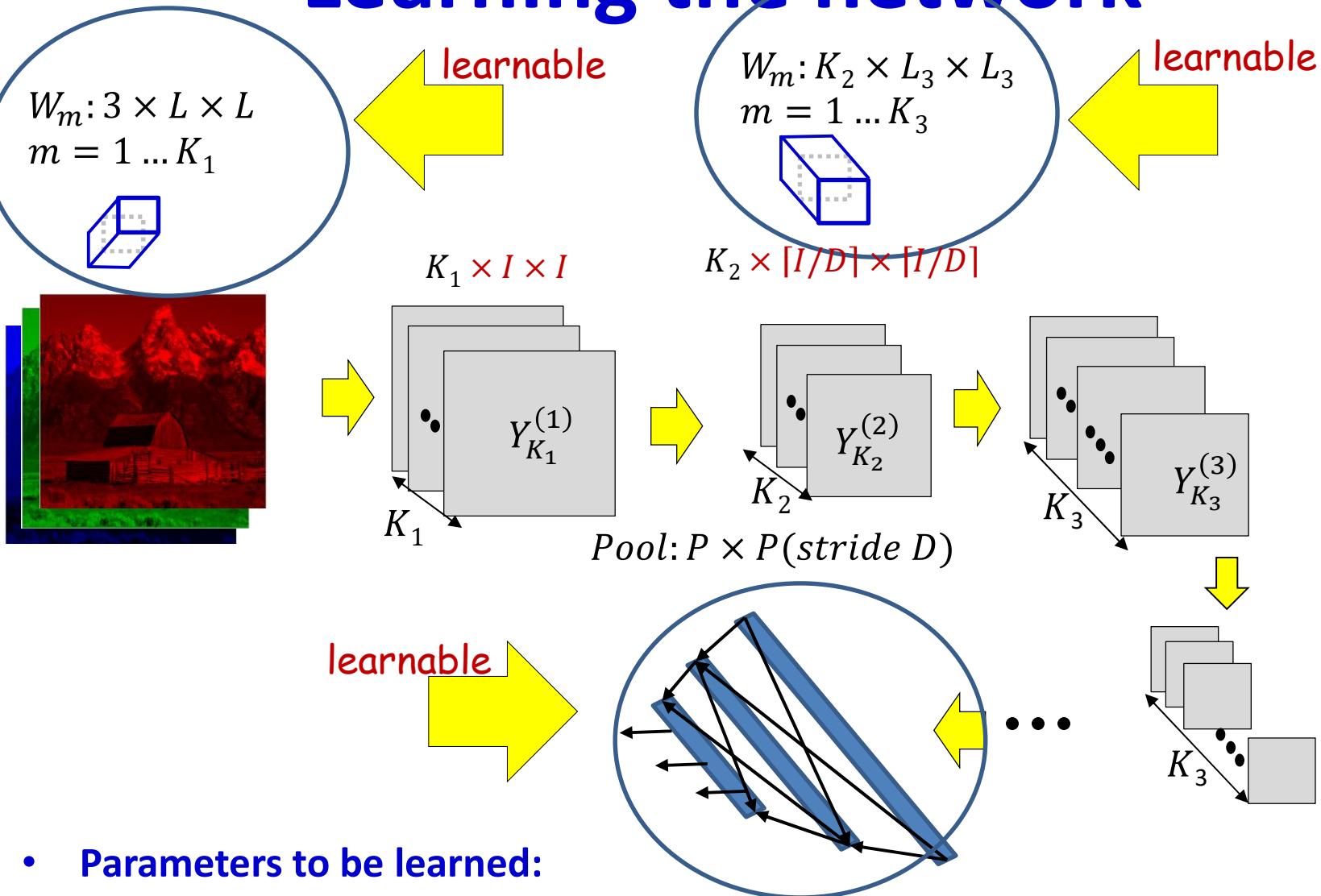
- Typical image classification task
 - Assuming maxpooling..
- Input: RGB images
 - Will assume color to be generic

Recap: A CNN, end-to-end



- Several convolutional and pooling layers.
- The output of the last layer is “flattened” and passed through an MLP

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

Recap: Learning the CNN

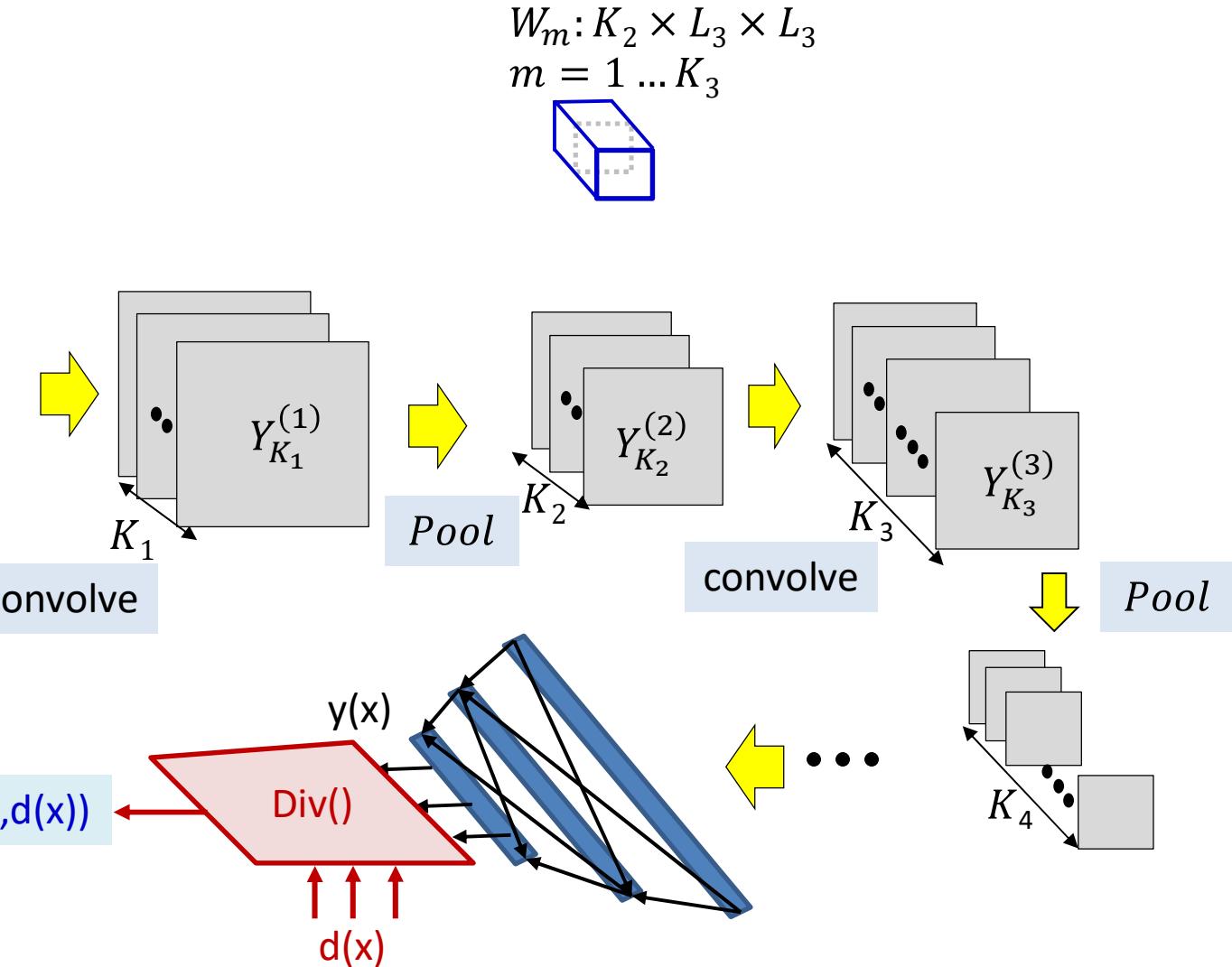
- Training is as in the case of the regular MLP
 - The *only* difference is in the *structure* of the network
- **Training examples of (Image, class) are provided**
- **Define a loss:**
 - Define a divergence between the desired output and true output of the network in response to any input
 - The loss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

Defining the loss

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



Input: x



- The loss for a single instance

Recap: Problem Setup

- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), \dots, (X_T, d_T)$
- The divergence on the i^{th} instance is $\text{div}(Y_i, d_i)$
- The aggregate Loss

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

- Minimize \textit{Loss} w.r.t $\{W_m, b_m\}$
 - Using gradient descent

Recap: The derivative

Total training loss:

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

- Computing the derivative

Total derivative:

$$\frac{dLoss}{dw} = \frac{1}{T} \sum_i \frac{dDiv(Y_i, d_i)}{dw}$$

Recap: The derivative

Total training loss:

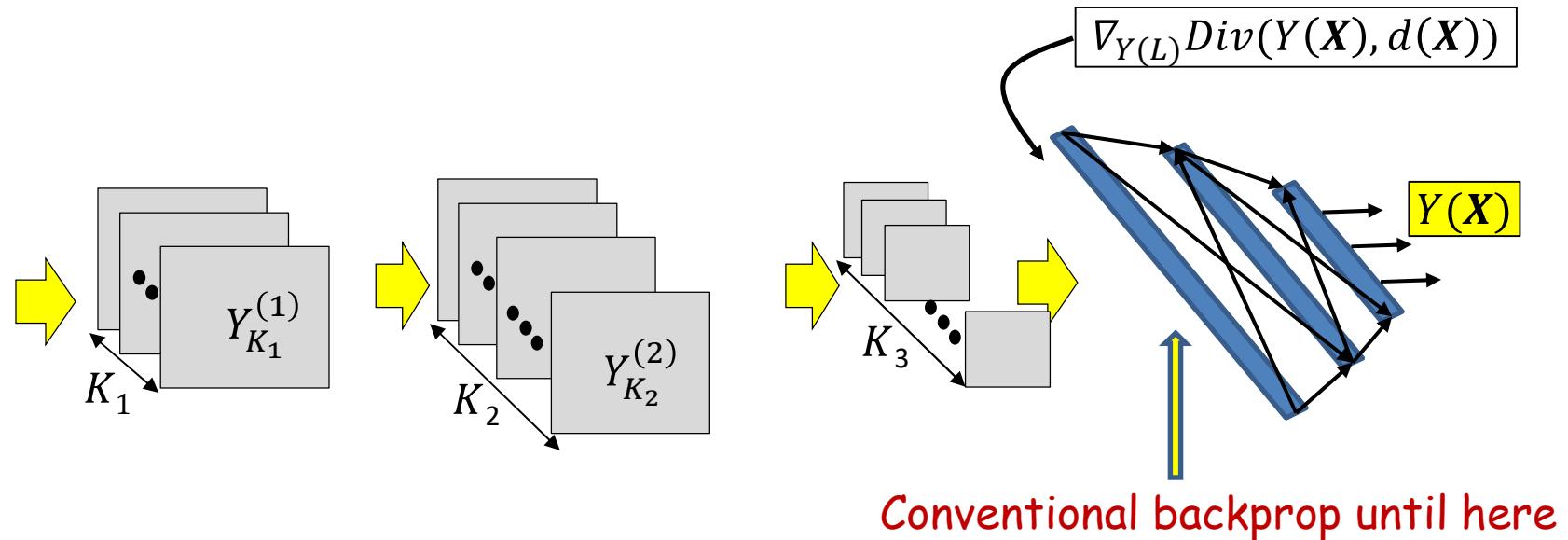
$$Loss = \frac{1}{T} \sum_i Div(Y_i, d_i)$$

- Computing the derivative

Total derivative:

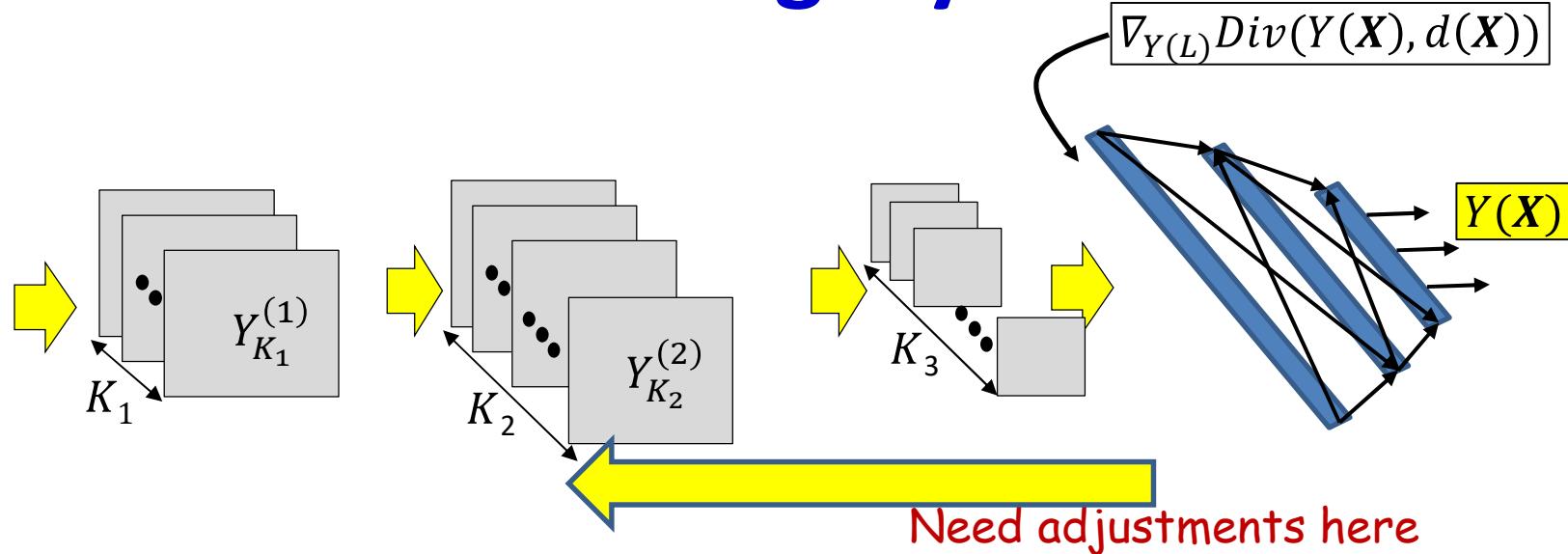
$$\frac{dLoss}{dw} = \frac{1}{T} \sum_i \frac{dDiv(Y_i, d_i)}{dw}$$

Backpropagation: Final flat layers



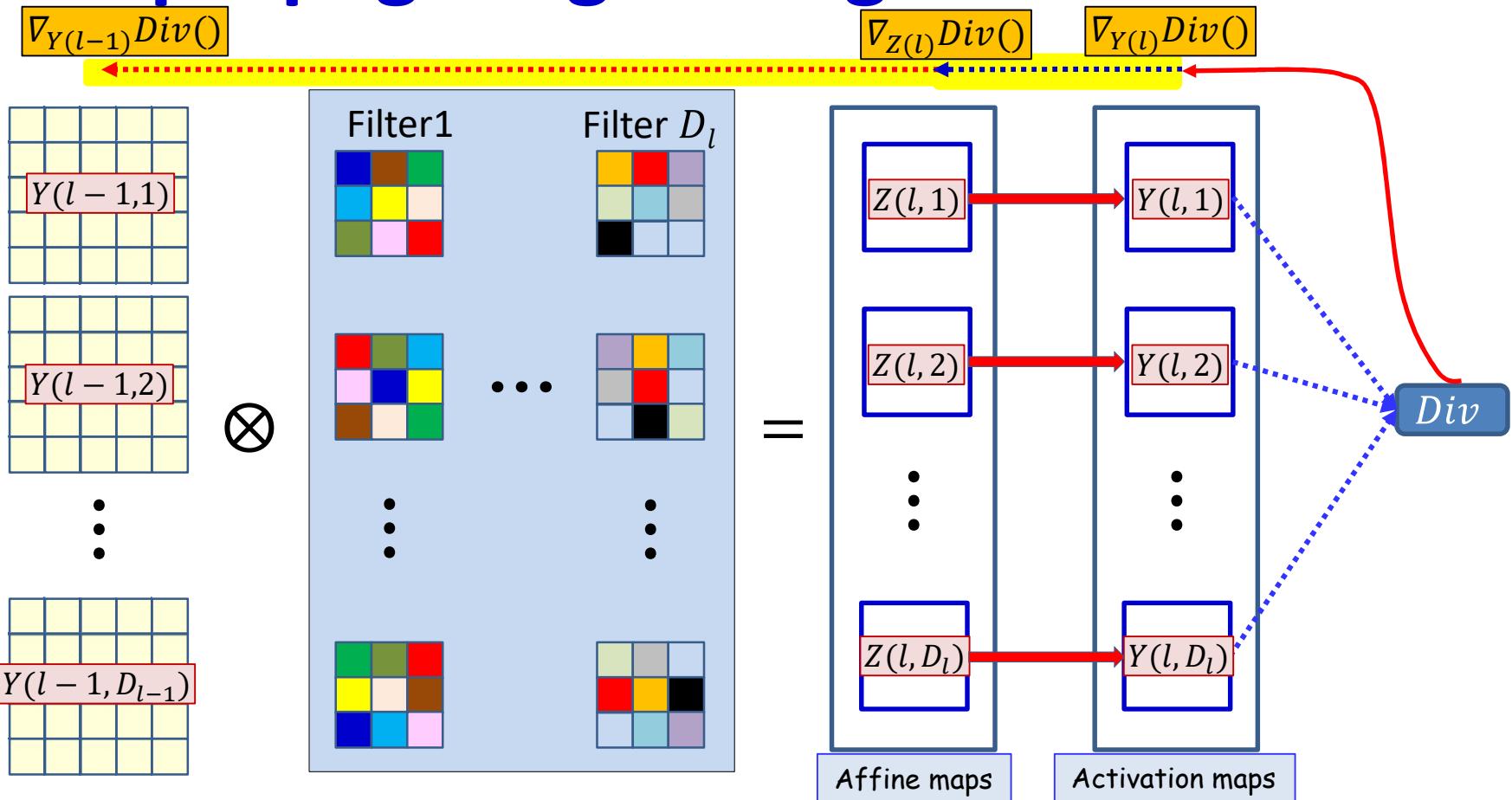
- For each training instance: First, a forward pass through the net
- Then the backpropagation of the derivative of the divergence
- Backpropagation continues in the usual manner until the computation of the derivative of the divergence w.r.t the inputs to the first “flat” layer
 - Important to recall: the first flat layer is only the “unrolling” of the maps from the final convolutional layer

Backpropagation: Convolutional and Pooling layers



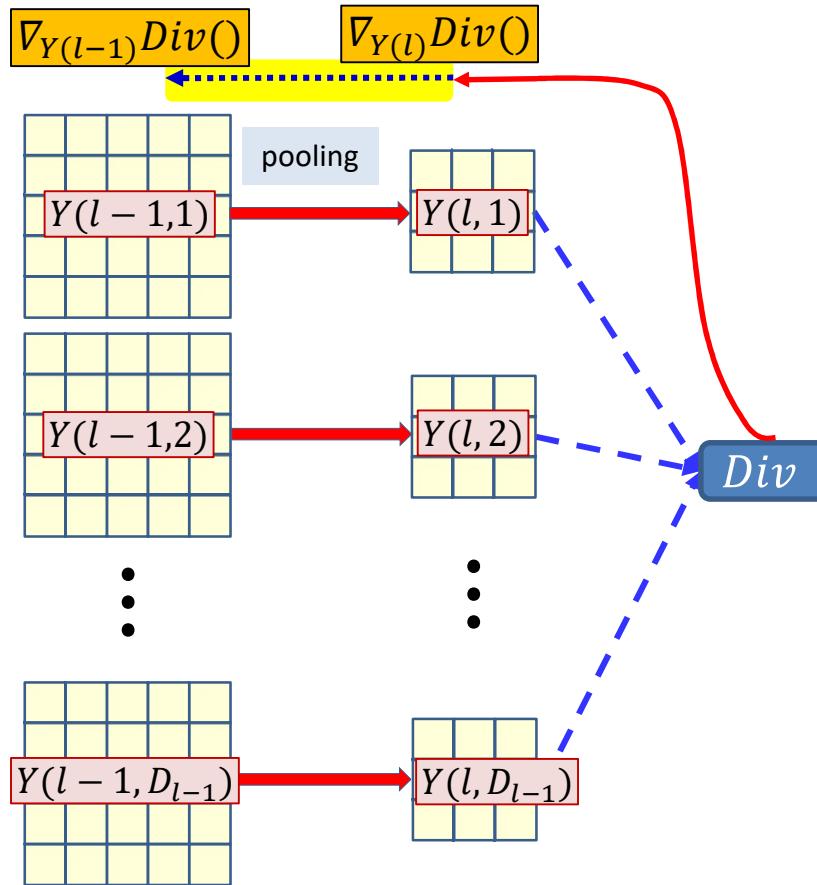
- Backpropagation from the flat MLP requires special consideration of
 - The shared computation in the convolution layers
 - The pooling layers (particularly maxout)

Backpropagating through the convolution



- **Convolution layers:**
- We already have the derivative w.r.t (all the elements of) activation map $Y(l,*)$
 - Having backpropagated it from the divergence
- We must backpropagate it through the activation to compute the derivative w.r.t. $Z(l,*)$ and further back to compute the derivative w.r.t the filters and $Y(l-1,*)$

Backprop: Pooling and D/S layer



- **Pooling and downsampling layers:**
- We already have the derivative w.r.t $Y(l,*)$
 - Having backpropagated it from the divergence
- We must compute the derivative w.r.t $Y(l-1,*)$

Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
 - Obtained as a result of backpropagating through the flat MLP
- **Required:**
 - **For convolutional layers:**
 - How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
 - How to compute the derivative w.r.t. $Y(l - 1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
 - **For pooling layers:**
 - How to compute the derivative w.r.t. $Y(l - 1)$ given derivatives w.r.t. $Y(l)$

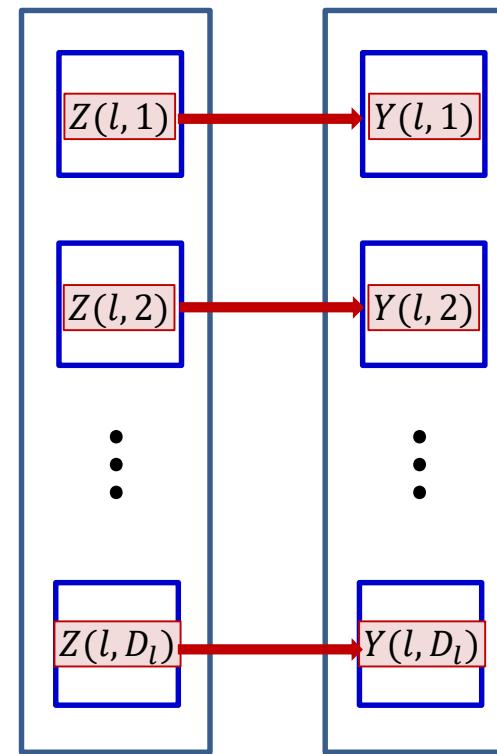
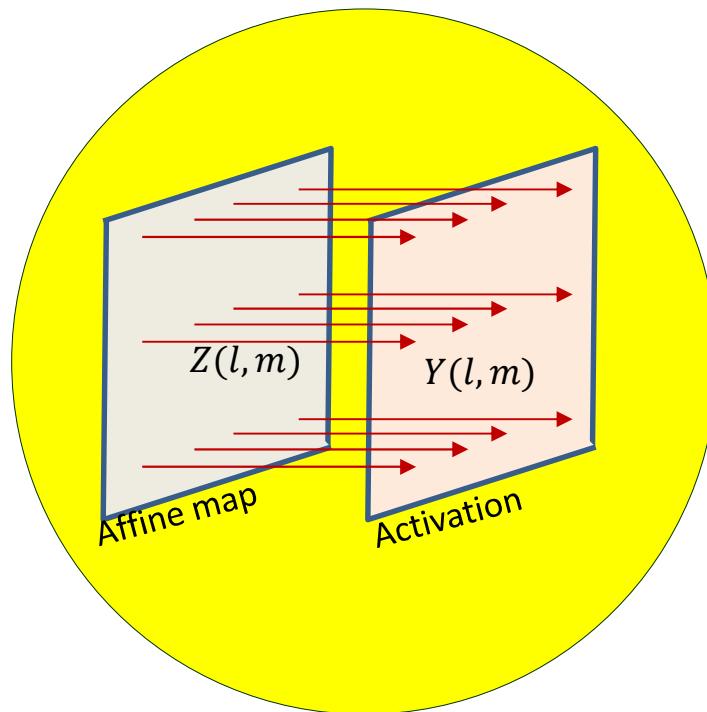
Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
 - Obtained as a result of backpropagating through the flat MLP
- **Required:**
 - **For convolutional layers:**
 - How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
 - How to compute the derivative w.r.t. $Y(l - 1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
 - **For pooling layers:**
 - How to compute the derivative w.r.t. $Y(l - 1)$ given derivatives w.r.t. $Y(l)$

Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
 - Obtained as a result of backpropagating through the flat MLP
- **Required:**
 - **For convolutional layers:**
 - How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
 - How to compute the derivative w.r.t. $Y(l - 1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
 - **For pooling layers:**
 - How to compute the derivative w.r.t. $Y(l - 1)$ given derivatives w.r.t. $Y(l)$

Backpropagating through the activation

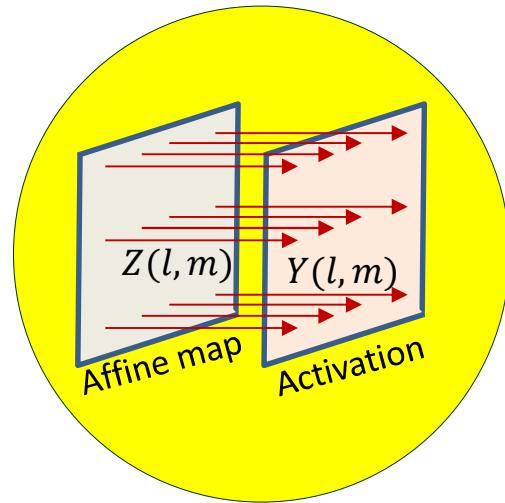


- **Forward computation:** The activation maps are obtained by point-wise application of the activation function to the affine maps

$$y(l, m, x, y) = f(z(l, m, x, y))$$

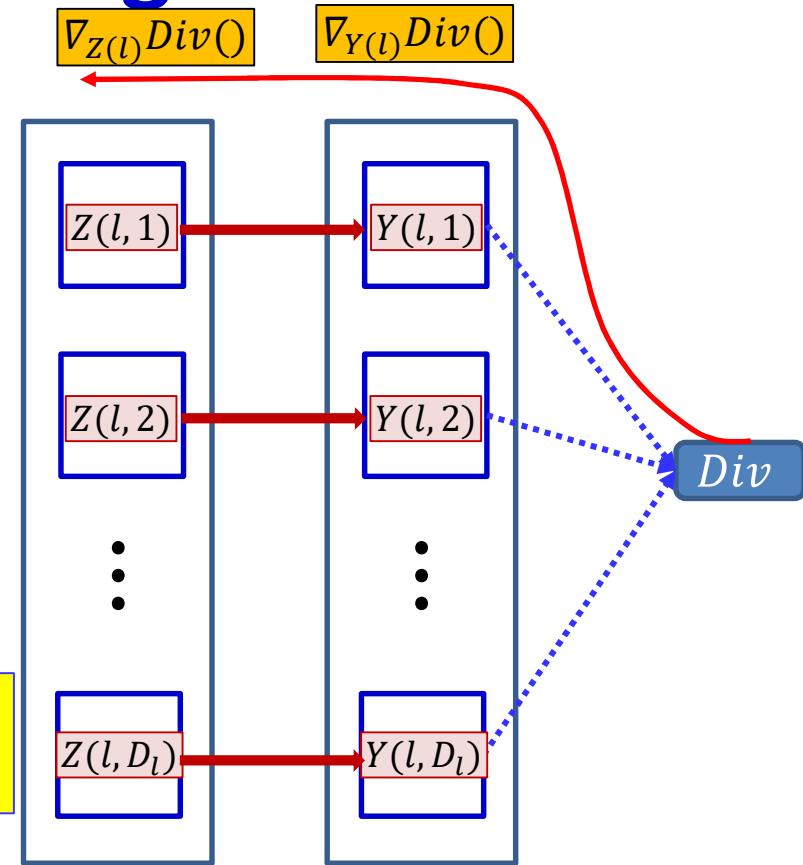
- The affine map entries $z(l, m, x, y)$ have already been computed via convolutions over the previous layer

Backpropagating through the activation



$$y(l, m, x, y) = f(z(l, m, x, y))$$

$$\frac{d\text{Div}}{dz(l, m, x, y)} = \frac{d\text{Div}}{d y(l, m, x, y)} f'(z(l, m, x, y))$$



- **Backward computation:** For every map $Y(l, m)$ for every position (x, y) , we already have the derivative of the divergence w.r.t. $y(l, m, x, y)$
 - Obtained via backpropagation
- We obtain the derivatives of the divergence w.r.t. $z(l, m, x, y)$ using the chain rule:

$$\frac{d\text{Div}}{dz(l, m, x, y)} = \frac{d\text{Div}}{d y(l, m, x, y)} f'(z(l, m, x, y))$$

- Simple component-wise computation

Backpropagation: Convolutional and Pooling layers

- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
 - Obtained as a result of backpropagating through the flat MLP
- **Required:**
 - **For convolutional layers:**
 - ✓ How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
 - How to compute the derivative w.r.t. $Y(l - 1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
 - **For pooling layers:**
 - How to compute the derivative w.r.t. $Y(l - 1)$ given derivatives w.r.t. $Y(l)$

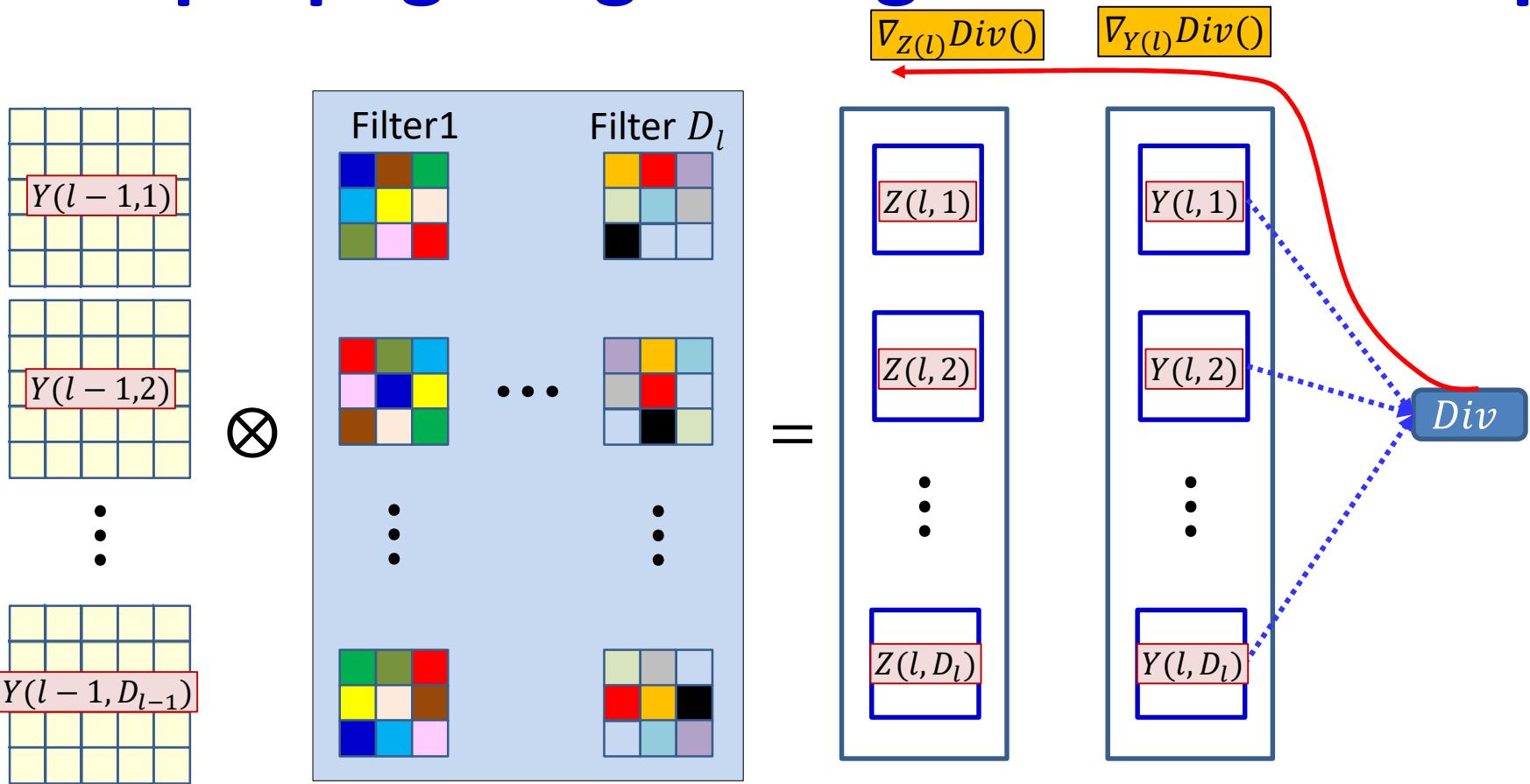
Backpropagating through affine map

- Forward affine computation:
 - Compute affine maps $z(l, n, x, y)$ from previous layer maps $y(l - 1, m, x, y)$ and filters $w_l(m, n, x, y)$
- Backpropagation: Given $\frac{dDiv}{dz(l,n,x,y)}$
 - Compute derivative w.r.t. $y(l - 1, m, x, y)$
 - Compute derivative w.r.t. $w_l(m, n, x, y)$

Backpropagating through affine map

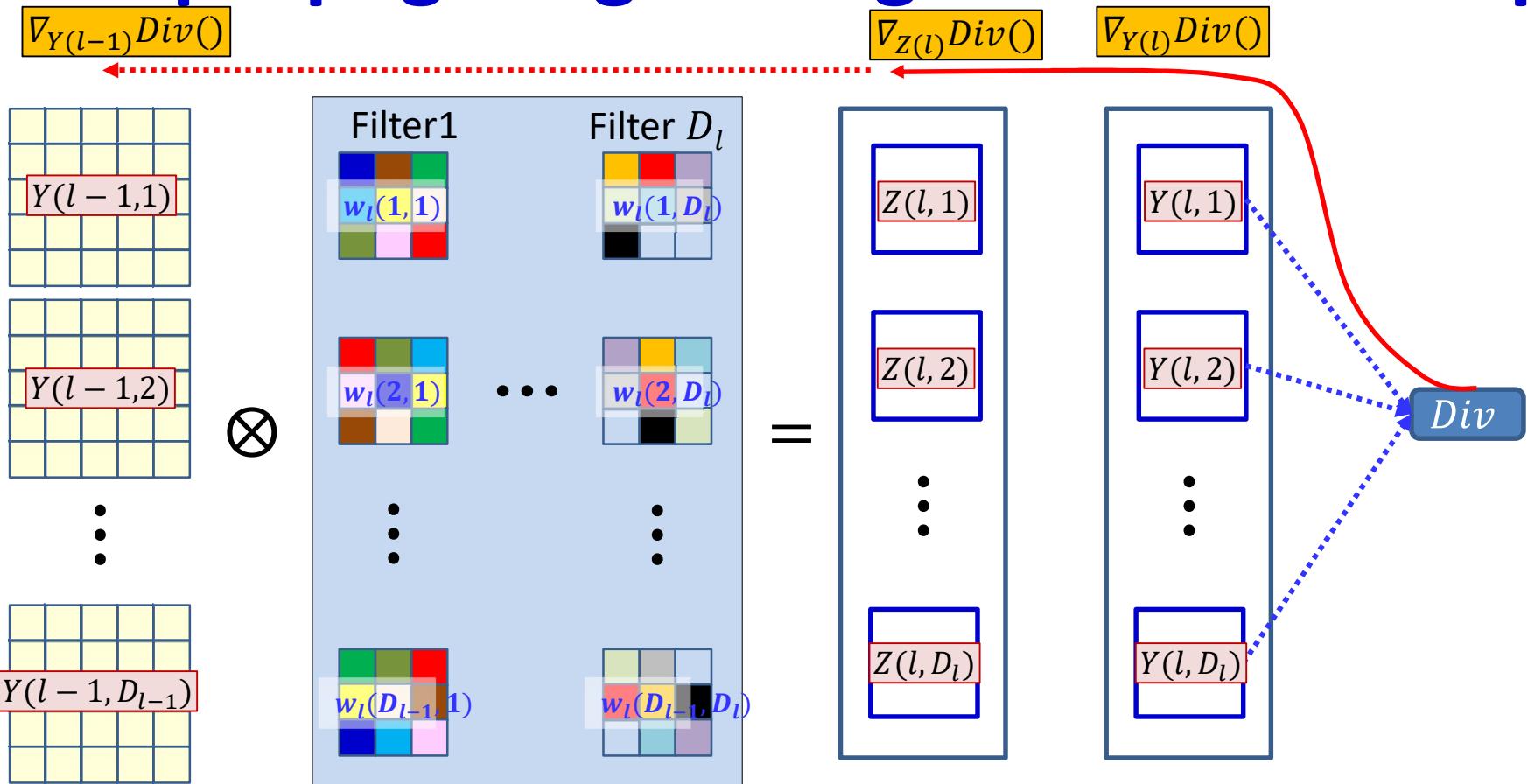
- Forward affine computation:
 - Compute affine maps $z(l, n, x, y)$ from previous layer maps $y(l - 1, m, x, y)$ and filters $w_l(m, n, x, y)$
- Backpropagation: Given $\frac{dDiv}{dz(l,n,x,y)}$
 - Compute derivative w.r.t. $y(l - 1, m, x, y)$
 - Compute derivative w.r.t. $w_l(m, n, x, y)$

Backpropagating through the affine map



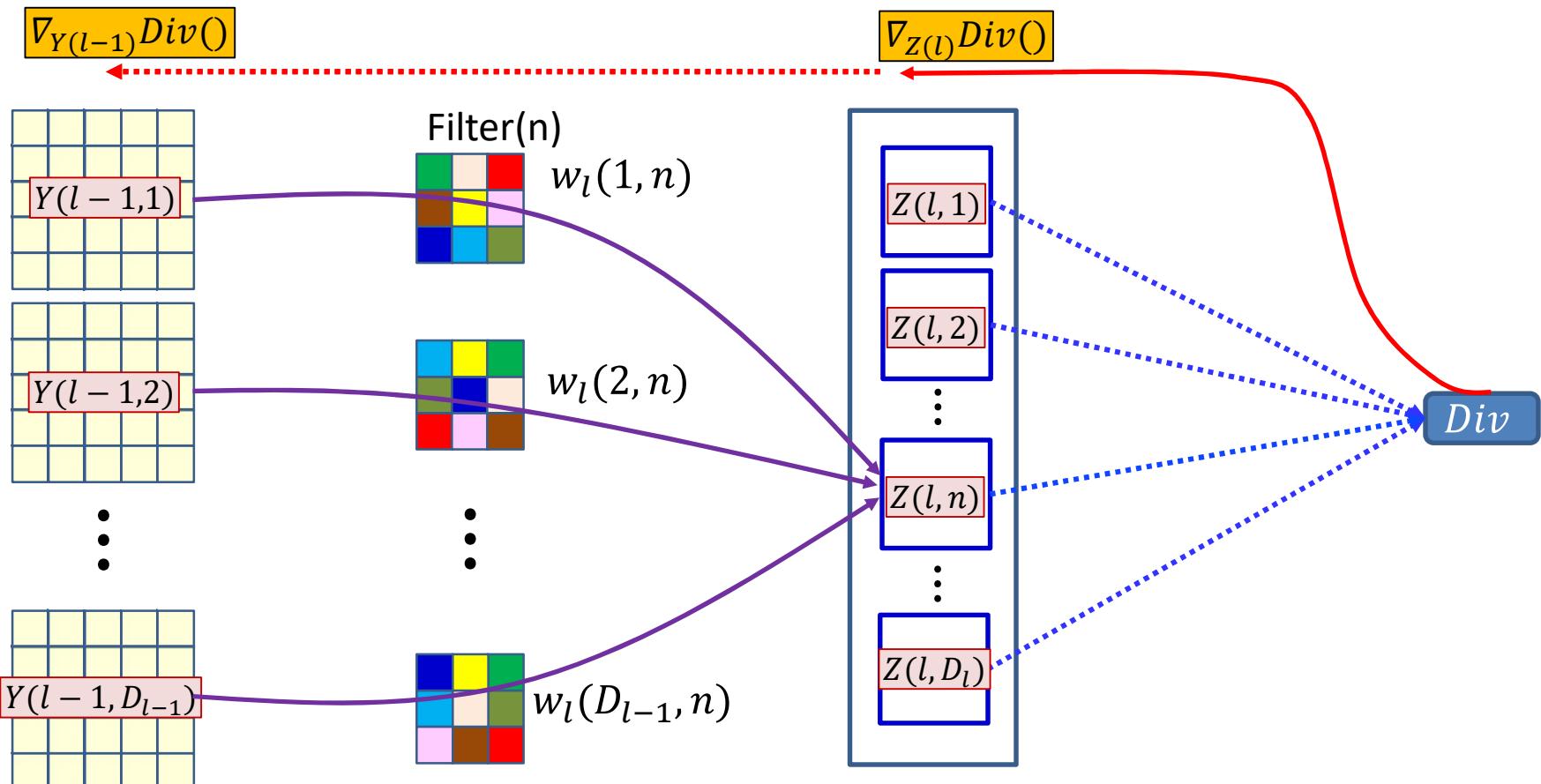
- We already have the derivative w.r.t $Z(l, *)$
 - Having backpropagated it past $Y(l, *)$

Backpropagating through the affine map



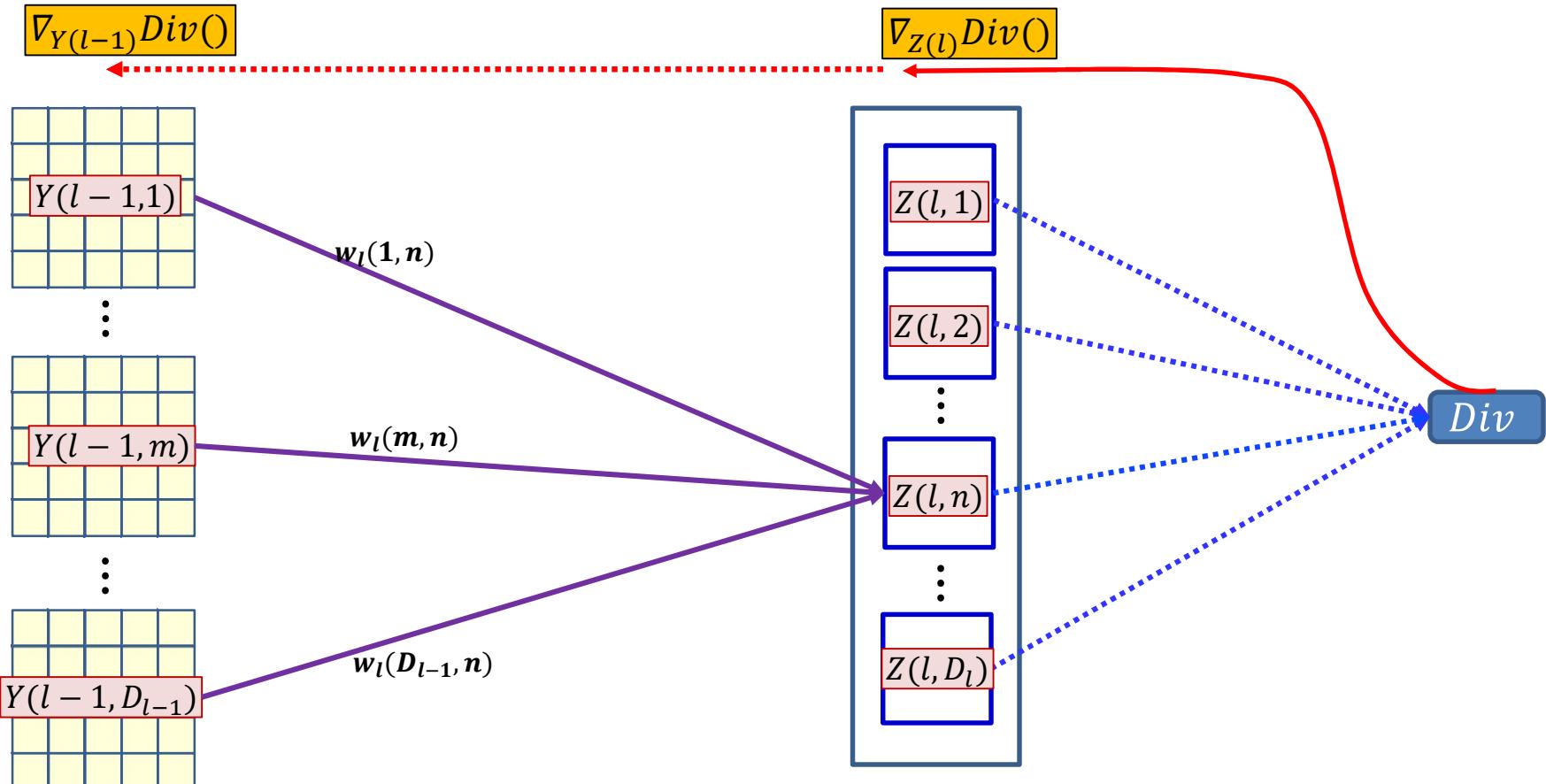
- We already have the derivative w.r.t $Z(l, \cdot)$
 - Having backpropagated it past $Y(l, \cdot)$
- We must compute the derivative w.r.t $Y(l-1, \cdot)$

Dependency between $Z(l,n)$ and $Y(l-1,*)$



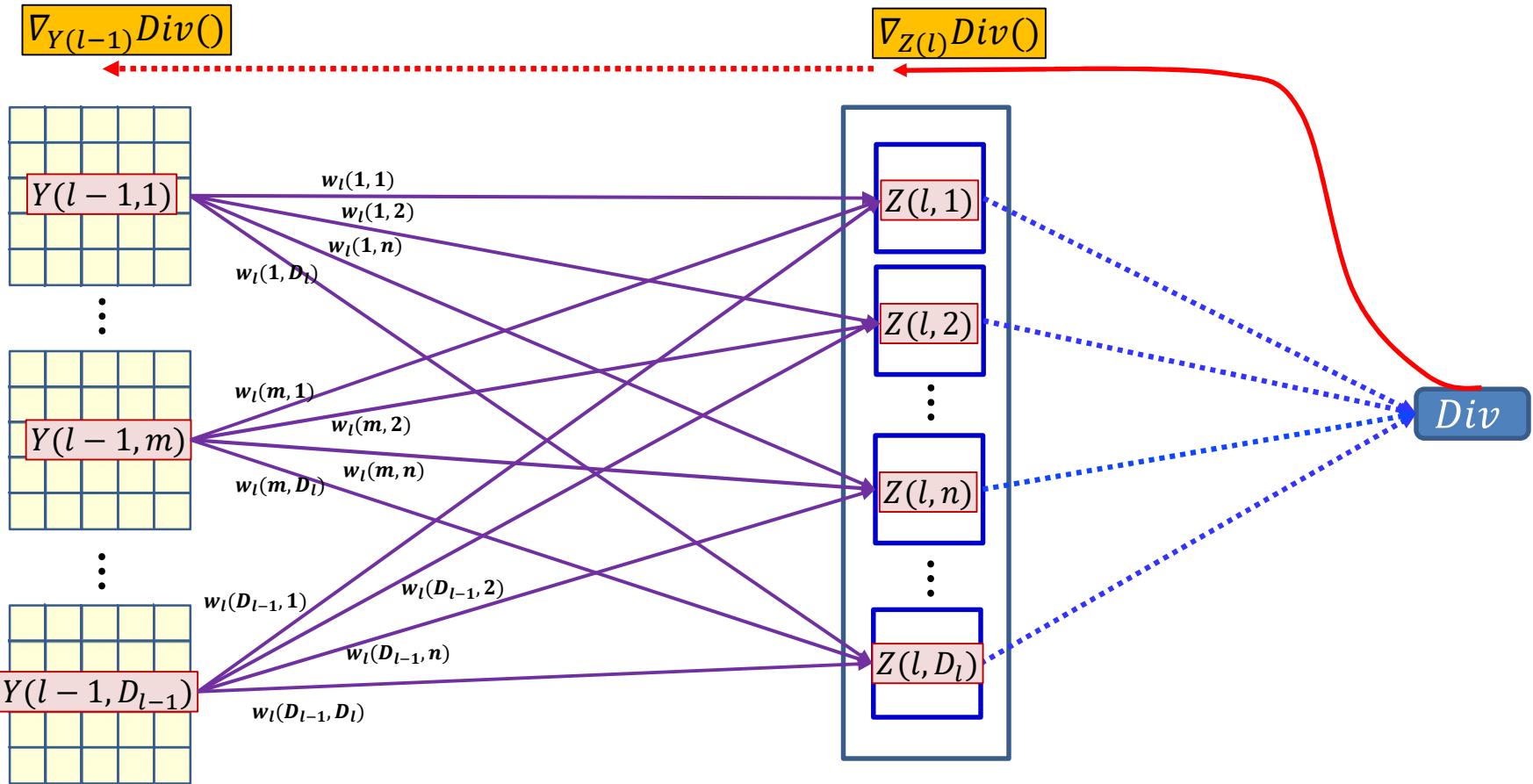
- Each $Y(l - 1, m)$ map influences $Z(l, n)$ through the m th “plane” of the n th filter $w_l(m, n)$

Dependency between $Z(l,n)$ and $Y(l-1,*)$



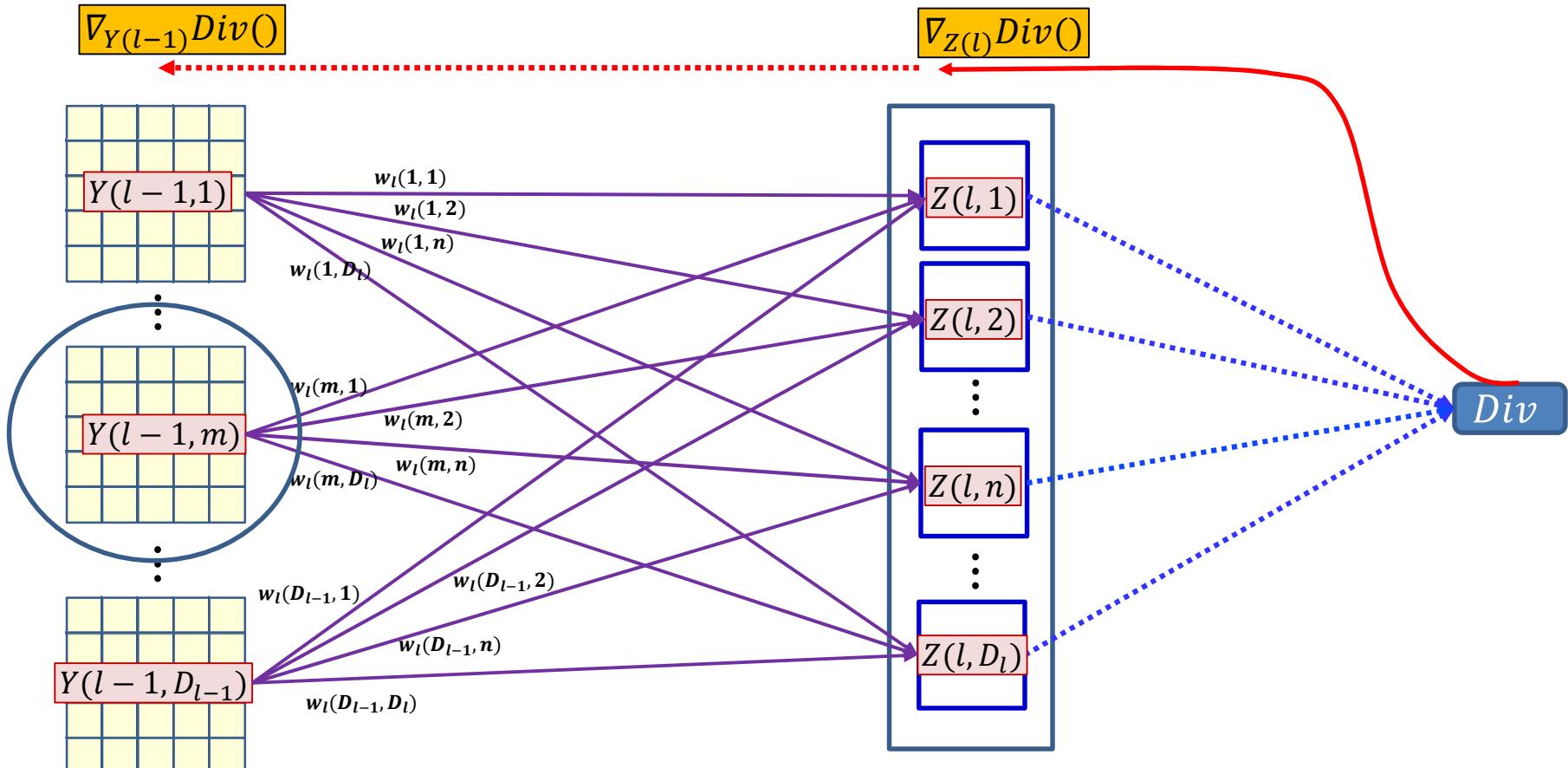
- Each $Y(l - 1, m)$ map influences $Z(l, n)$ through the m th “plane” of the n th filter $w_l(m, n)$

Dependency between $Z(l, *)$ and $Y(l-1, *)$



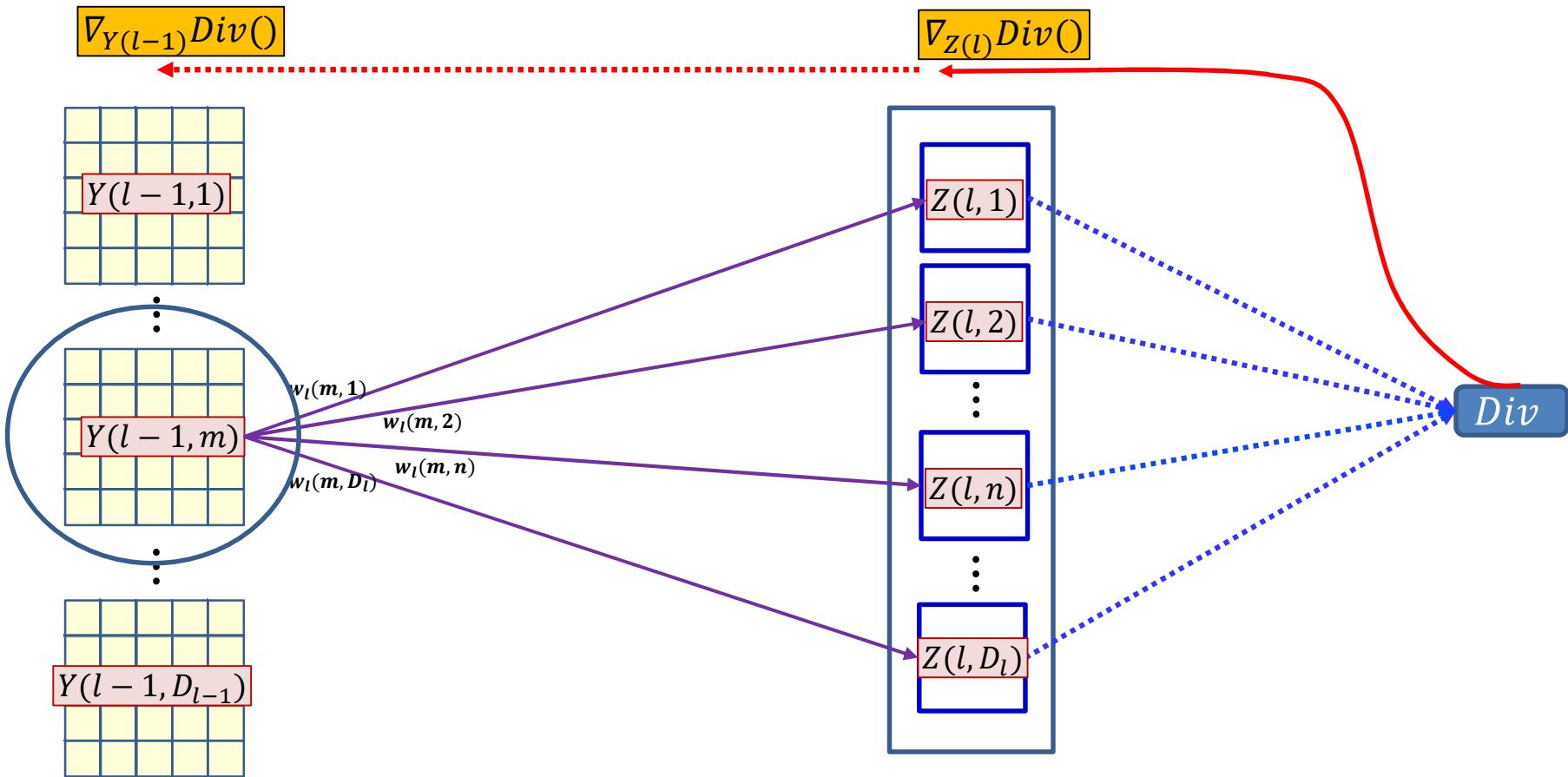
- Each $Y(l - 1, m)$ map influences $Z(l, n)$ through the m th “plane” of the n th filter $w_l(m, n)$

Dependency between $Z(l, *)$ and $Y(l-1, *)$



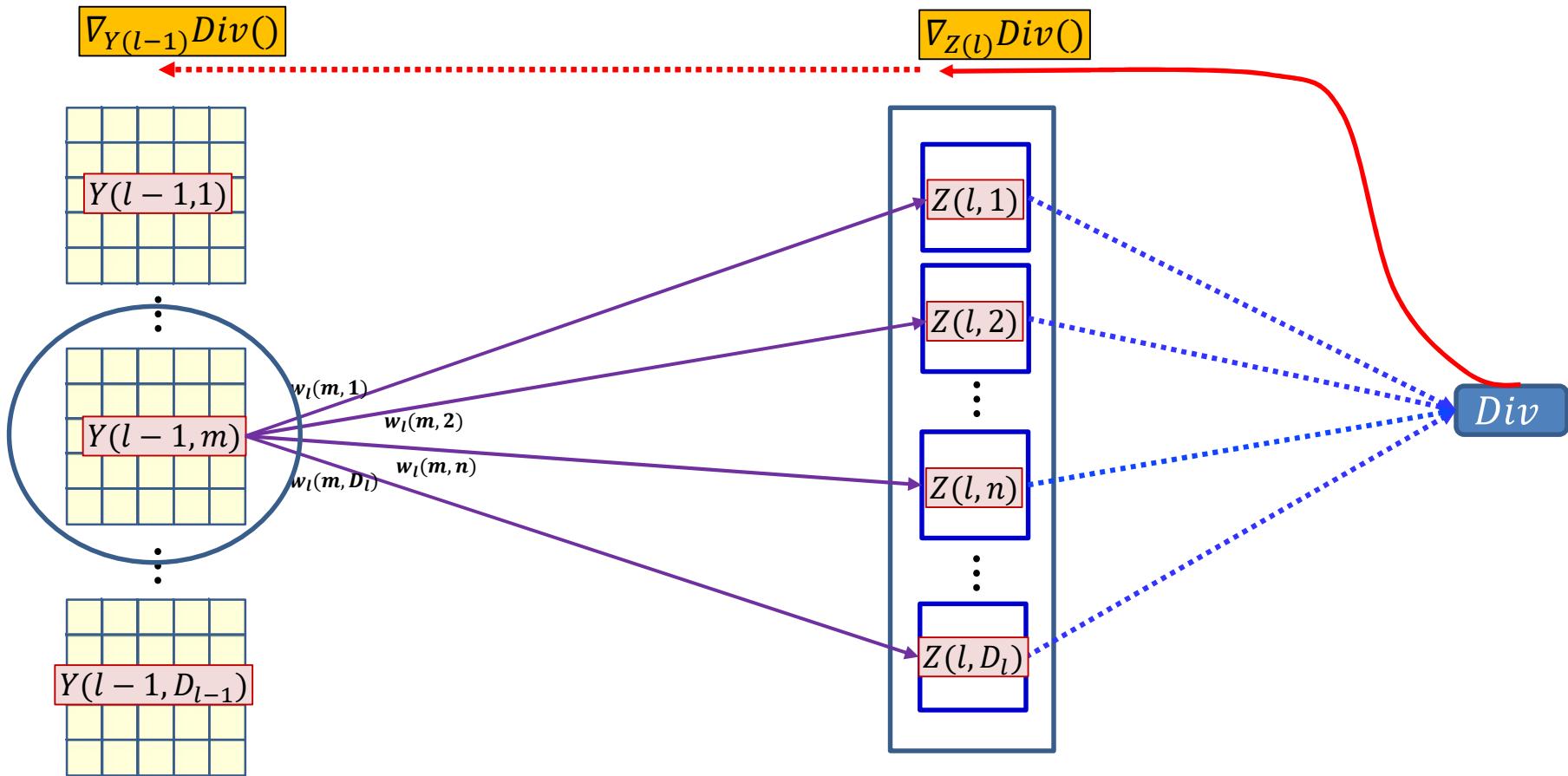
- Each $Y(l - 1, m)$ map influences $Z(l, n)$ through the m th “plane” of the n th filter $w_l(m, n)$

Dependency diagram for a single map



- Each $Y(l - 1, m)$ map influences $Z(l, n)$ through the m th “plane” of the n th filter $w_l(m, n)$
- $Y(l - 1, m, *, *)$ influences the divergence through all $Z(l, n, *, *)$ maps

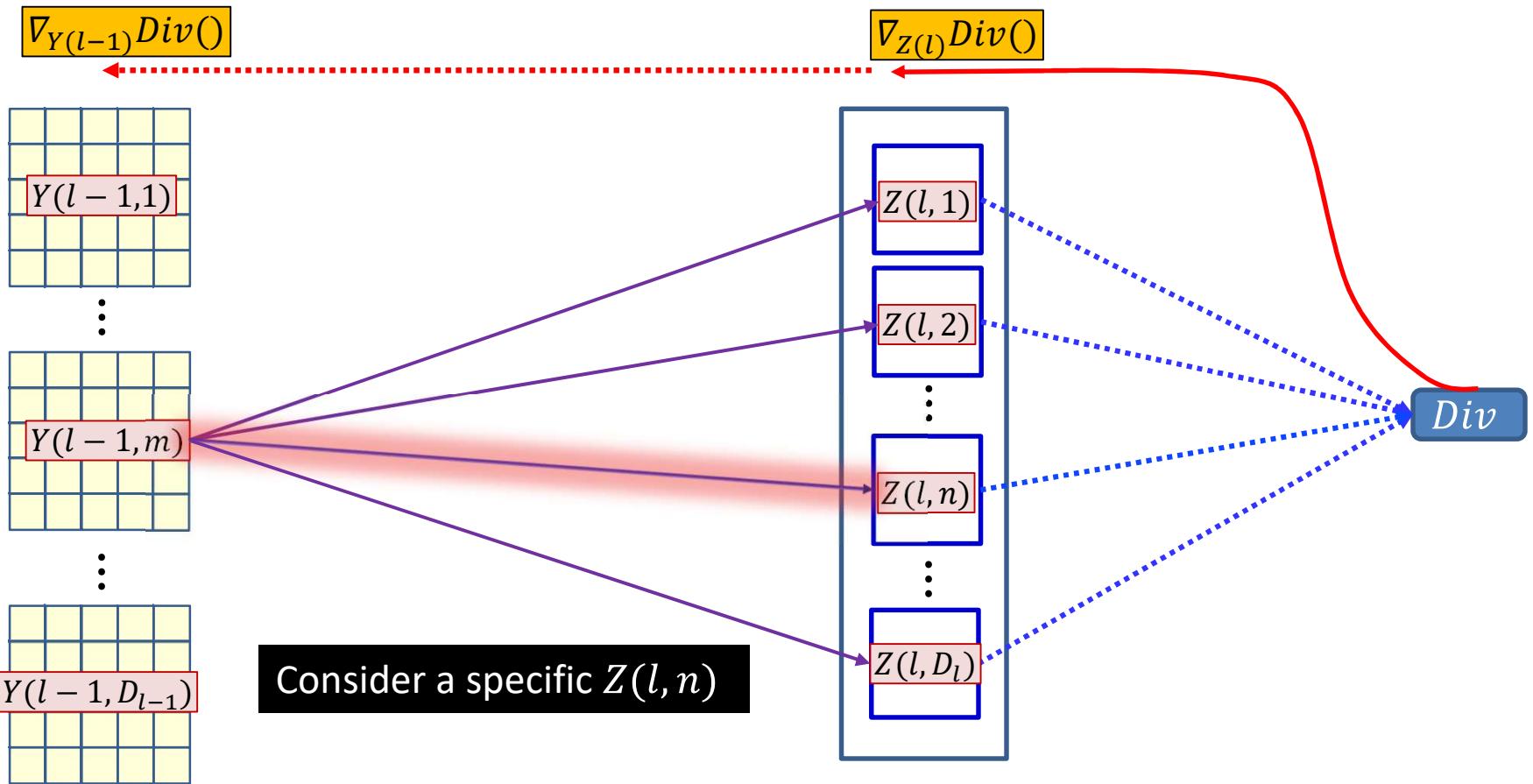
Dependency diagram for a single map



$$\nabla_{Y(l-1, m)} Div(.) = \sum_n \nabla_{Z(l, n)} Div(.) \underbrace{\nabla_{Y(l-1, m)} Z(l, n)}$$

- Need to compute $\nabla_{Y(l-1, m)} Z(l, n)$, the derivative of $Z(l, n)$ w.r.t. $Y(l - 1, m)$ to complete the computation of the formula

Dependency diagram for a single map



$$\nabla_{Y(l-1,m)} \text{Div}(\cdot) = \sum_n \nabla_{Z(l,n)} \text{Div}(\cdot) \underbrace{\nabla_{Y(l-1,m)} Z(l, n)}$$

- Need to compute $\nabla_{Y(l-1,m)} Z(l, n)$, the derivative of $Z(l, n)$ w.r.t. $Y(l-1, m)$ to complete the computation of the formula

BP: Convolutional layer

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

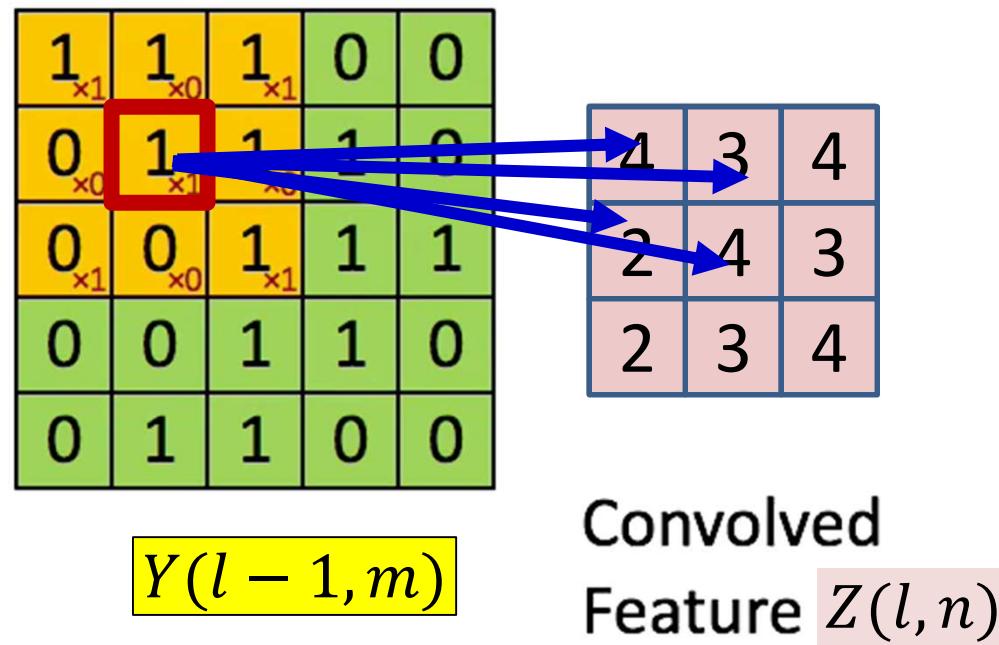
$$Y(l - 1, m)$$

4		

Convolved
Feature $Z(l, n)$

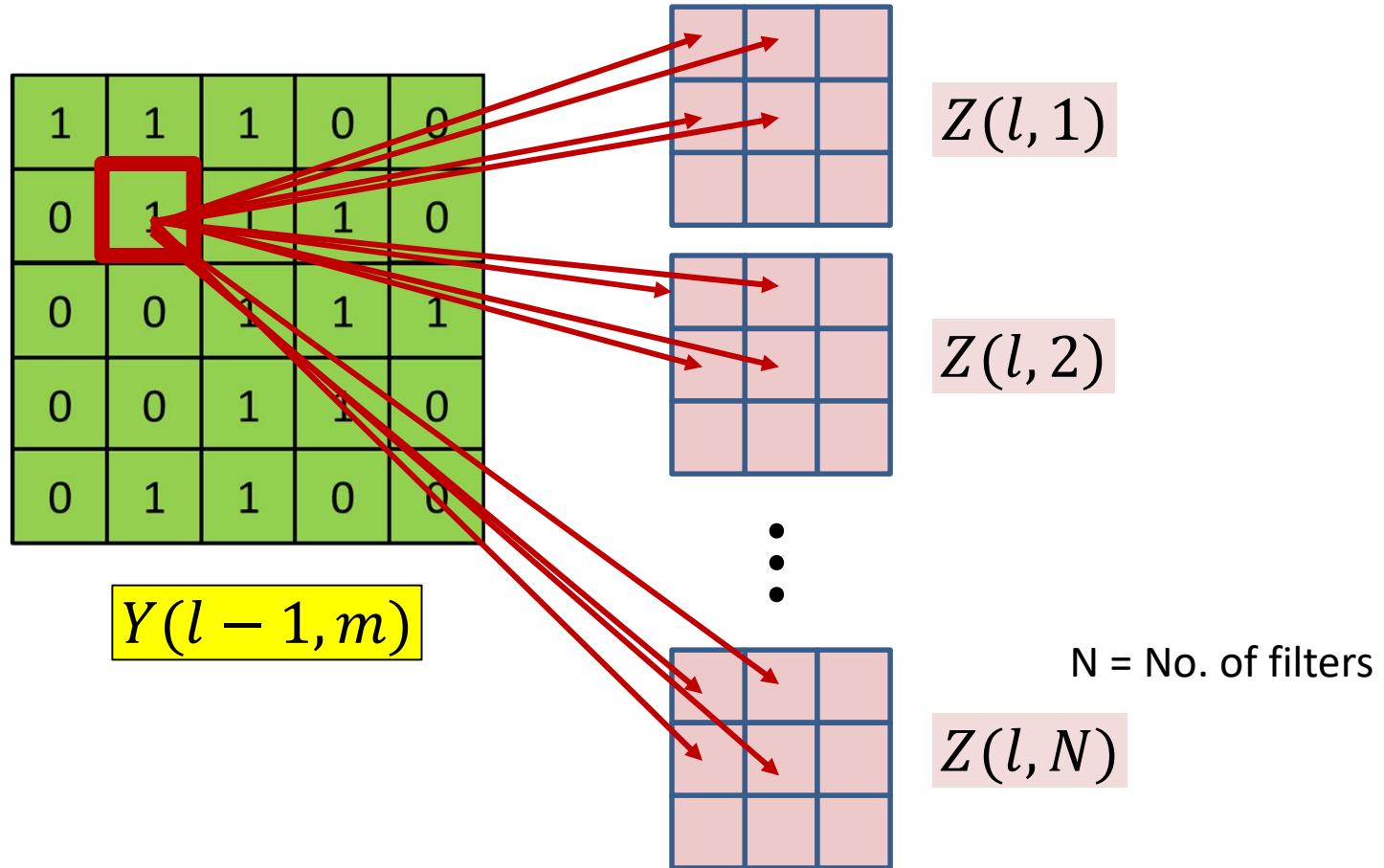
- Each $Y(l - 1, m, x, y)$ affects several $z(l, n, x', y')$ terms

BP: Convolutional layer



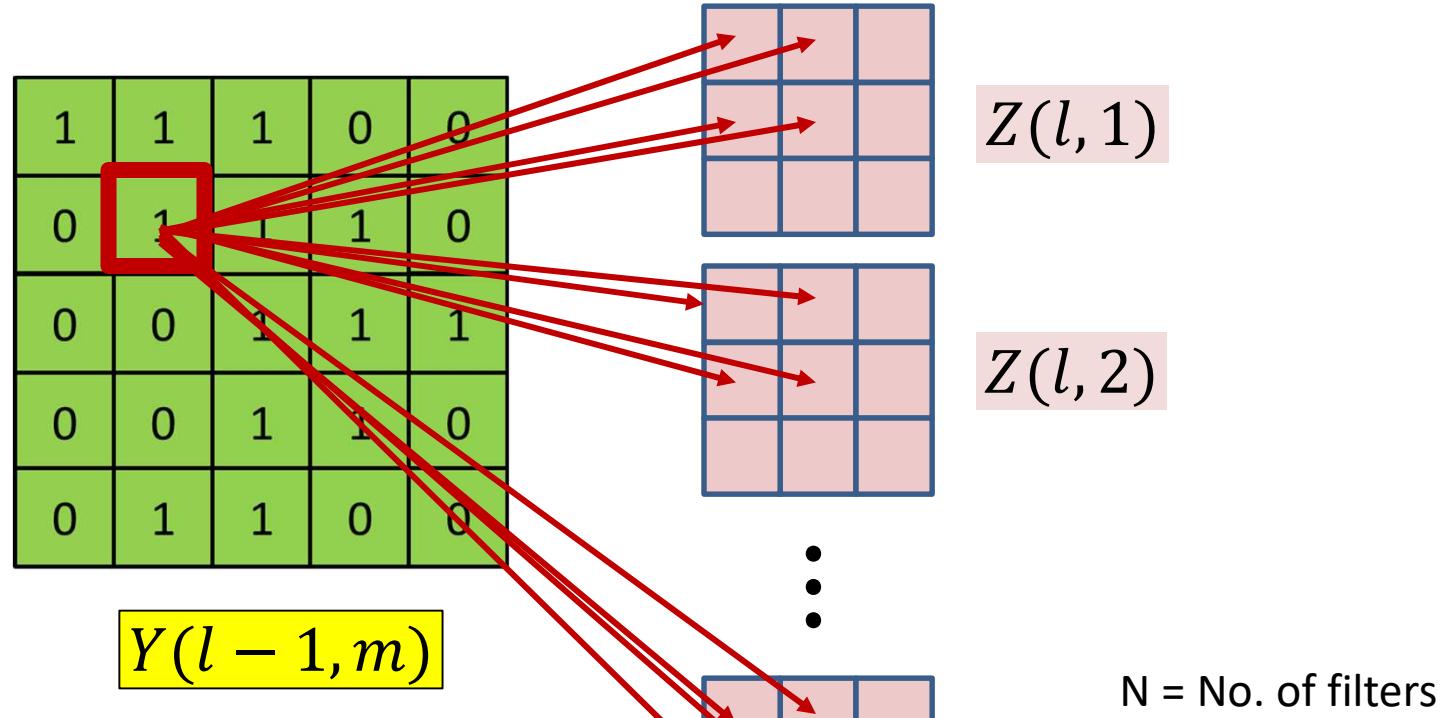
- Each $Y(l - 1, m, x, y)$ affects several $z(l, n, x', y')$ terms

BP: Convolutional layer



- Each $Y(l - 1, m, x, y)$ affects several $z(l, n, x', y')$ terms
 - Affects terms in *all* l^{th} layer Z maps

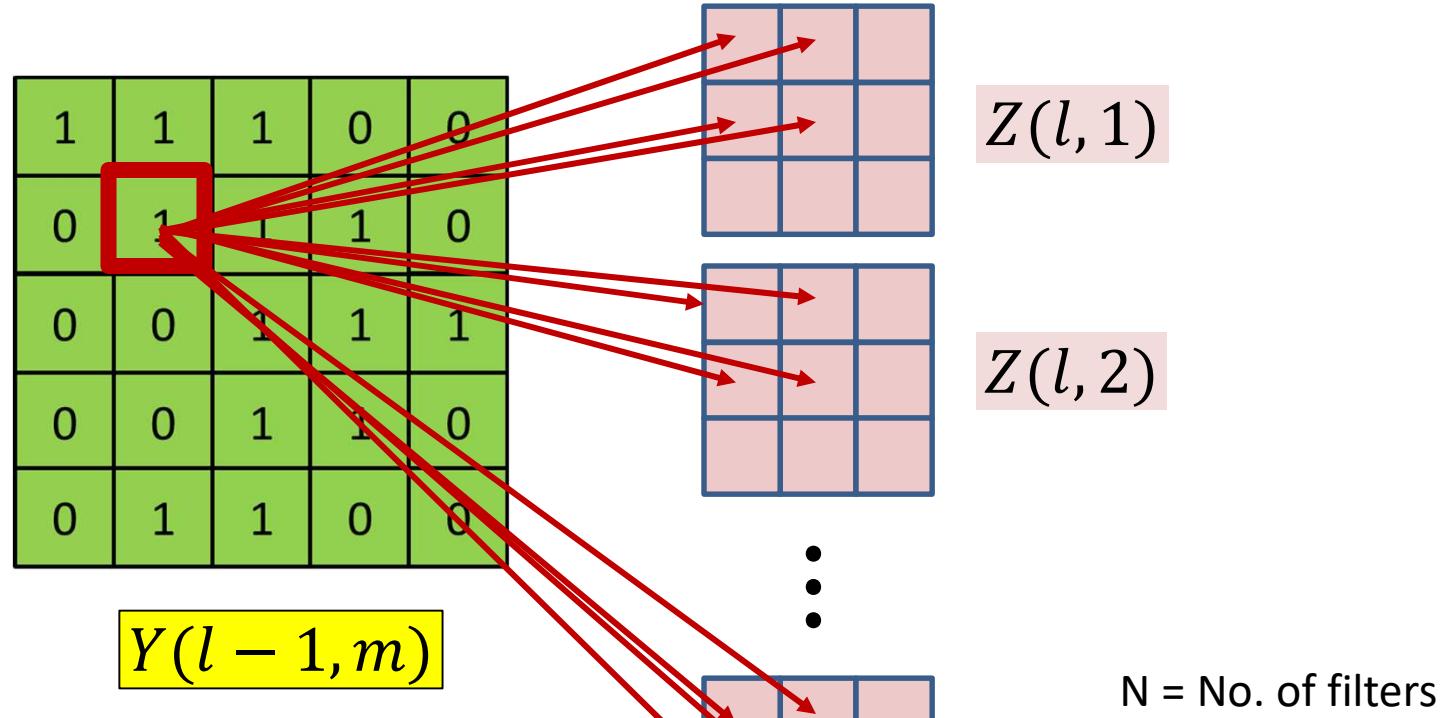
BP: Convolutional layer



Summing over all Z maps

$$\frac{dDiv}{dY(l - 1, m, x, y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} \frac{dz(l, n, x', y')}{dY(l - 1, m, x, y)}$$

BP: Convolutional layer

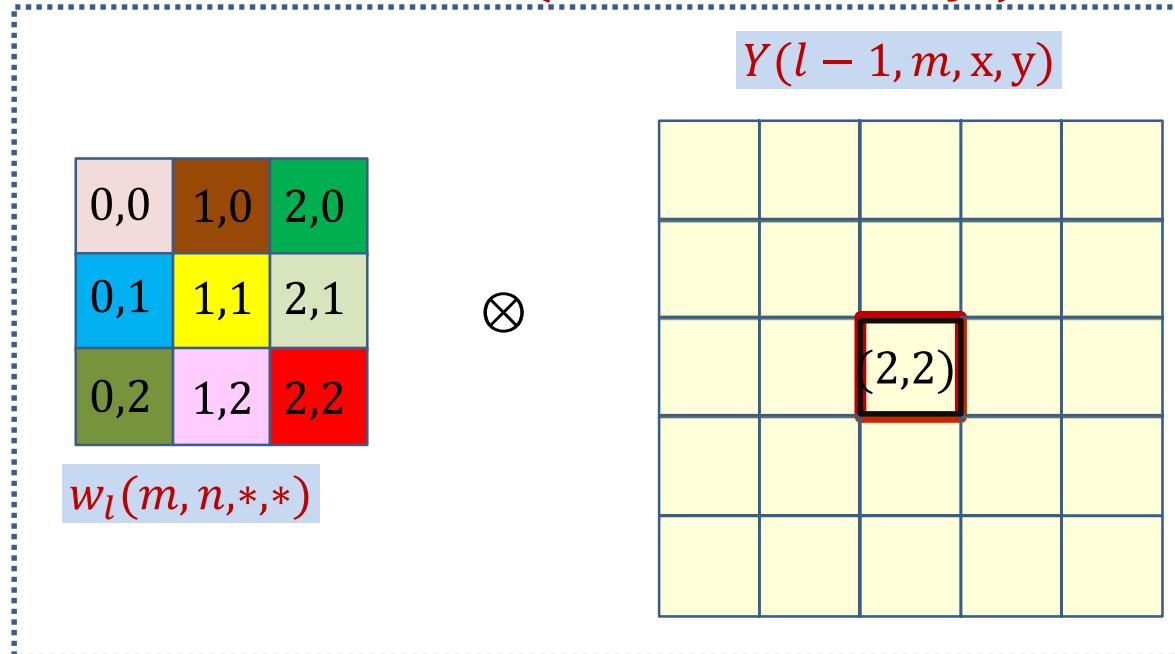


Summing over all Z maps

What is this?

$$\frac{dDiv}{dY(l - 1, m, x, y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} \frac{dz(l, n, x', y')}{dY(l - 1, m, x, y)}$$

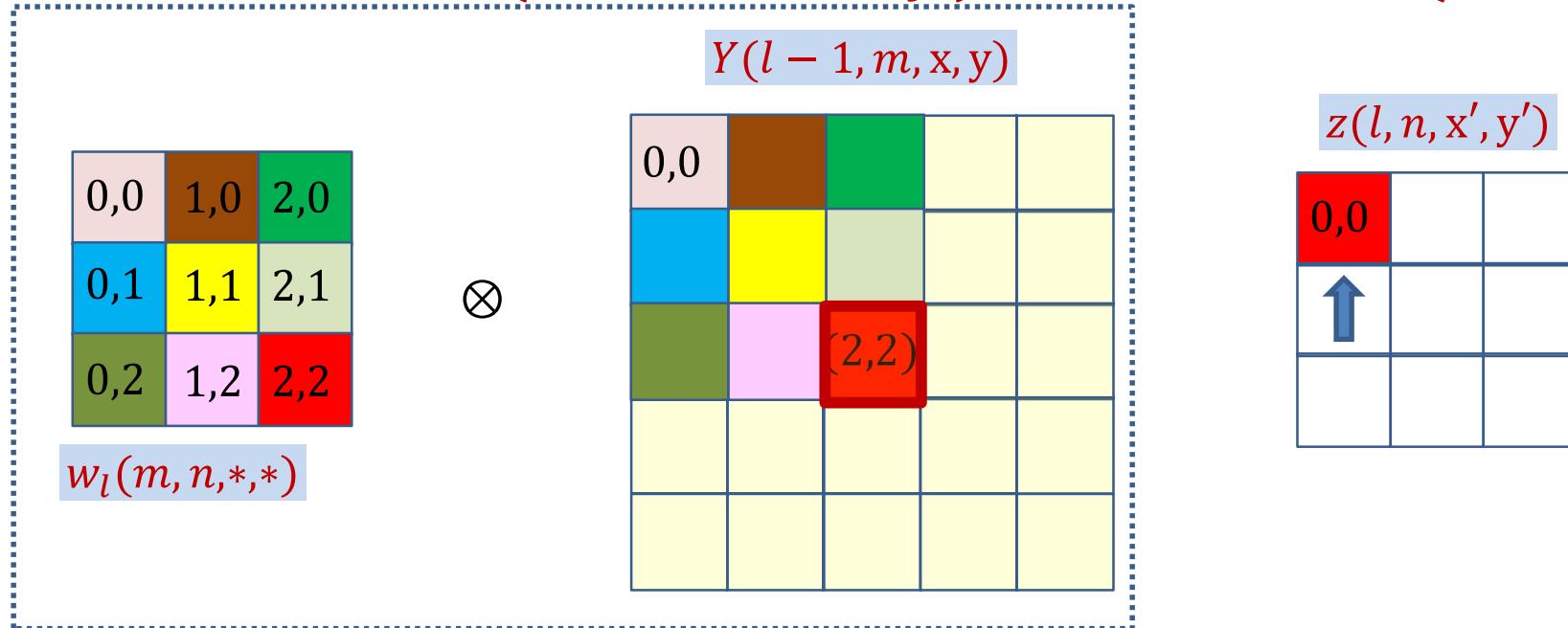
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



Assuming indexing
begins at 0

- Compute how *each* x, y in Y influences various locations of z

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

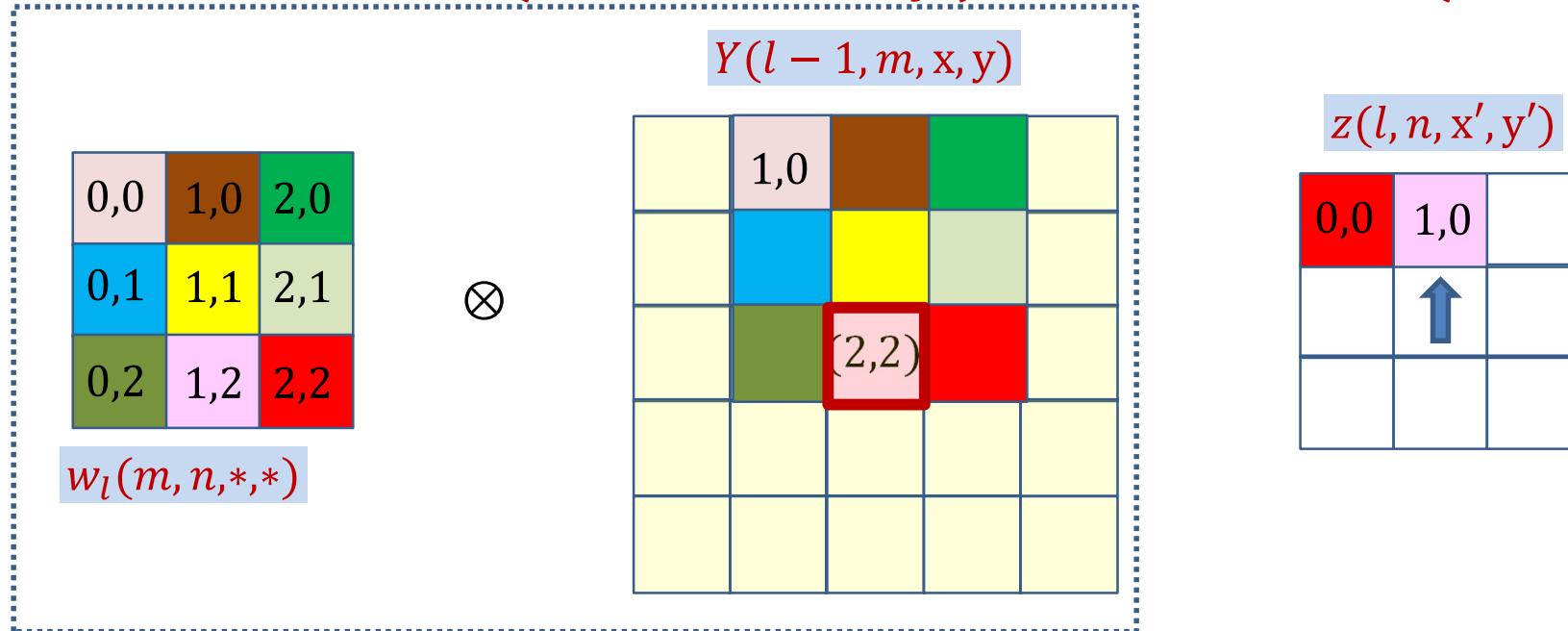


$$z(l, n, 0,0) += Y(l - 1, m, 2,2) w_l(m, n, 2,2)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2,2) w_l(m, n, 2 - x', 2 - y')$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

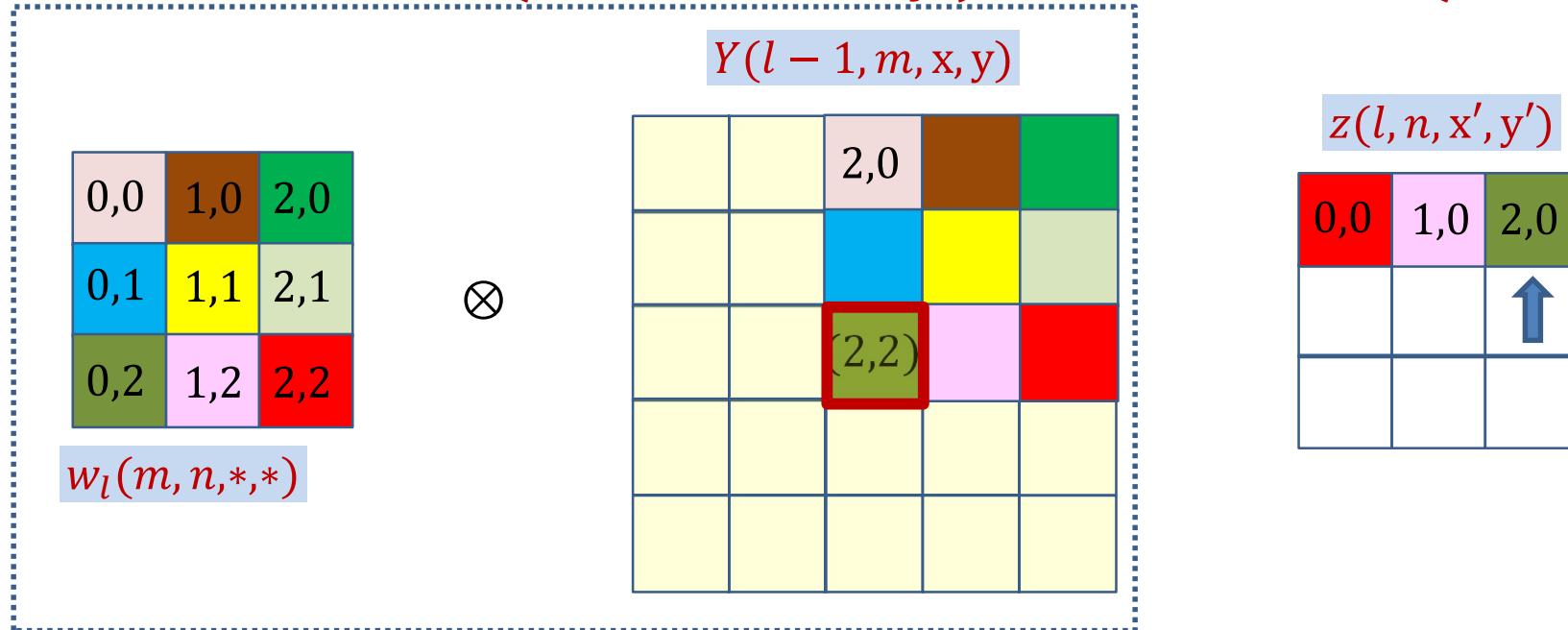


$$z(l, n, 1, 0) += Y(l - 1, m, 2, 2) w_l(m, n, 1, 2)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2, 2) w_l(m, n, 2 - x', 2 - y')$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

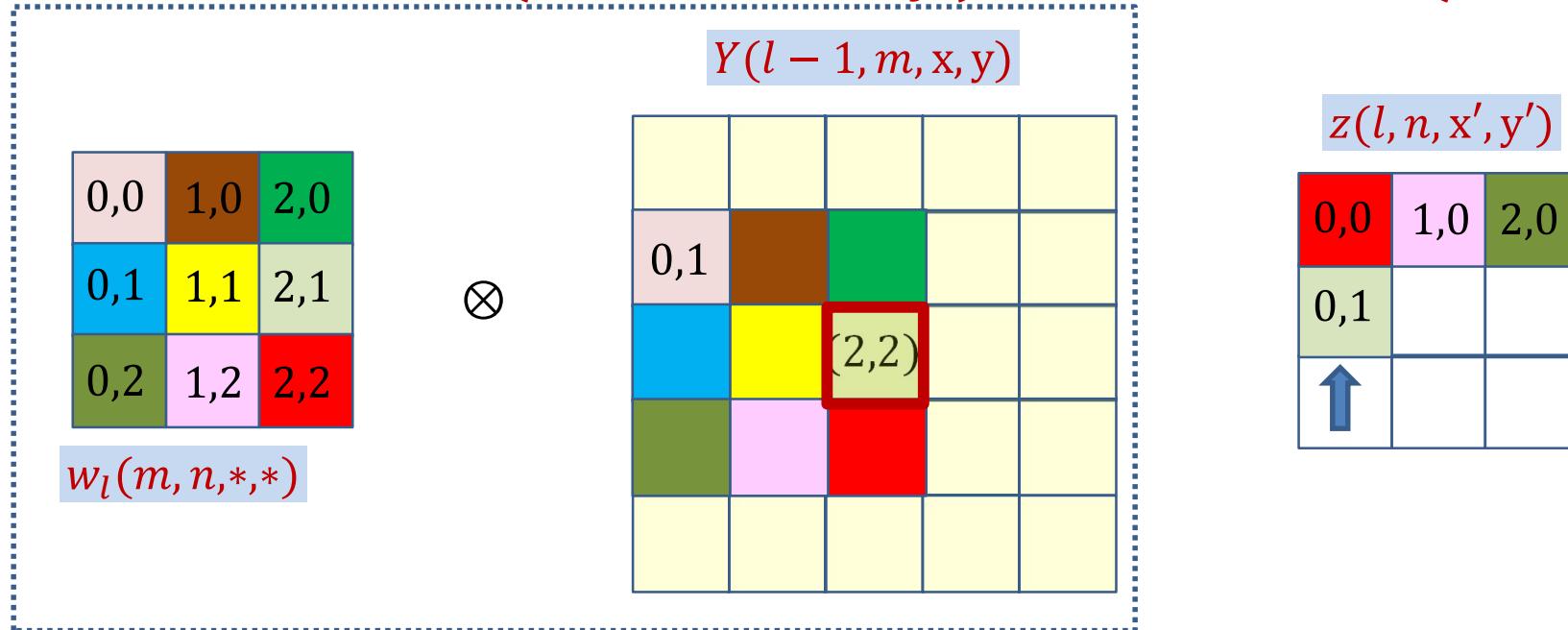


$$z(l, n, 2,0) += Y(l - 1, m, 2,2) w_l(m, n, 0,2)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2,2) w_l(m, n, 2 - x', 2 - y')$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

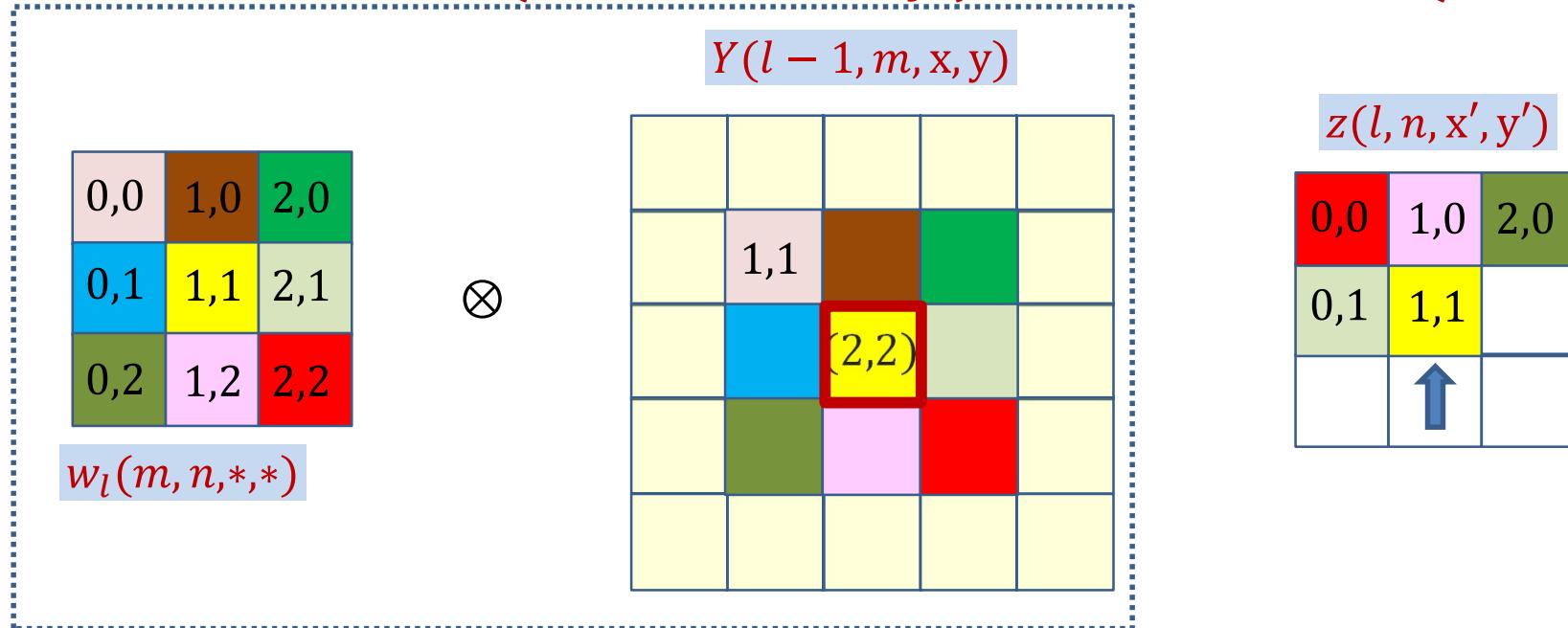


$$z(l, n, 0,1) += Y(l - 1, m, 2,2) w_l(m, n, 2,1)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2,2) w_l(m, n, 2 - x', 2 - y')$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

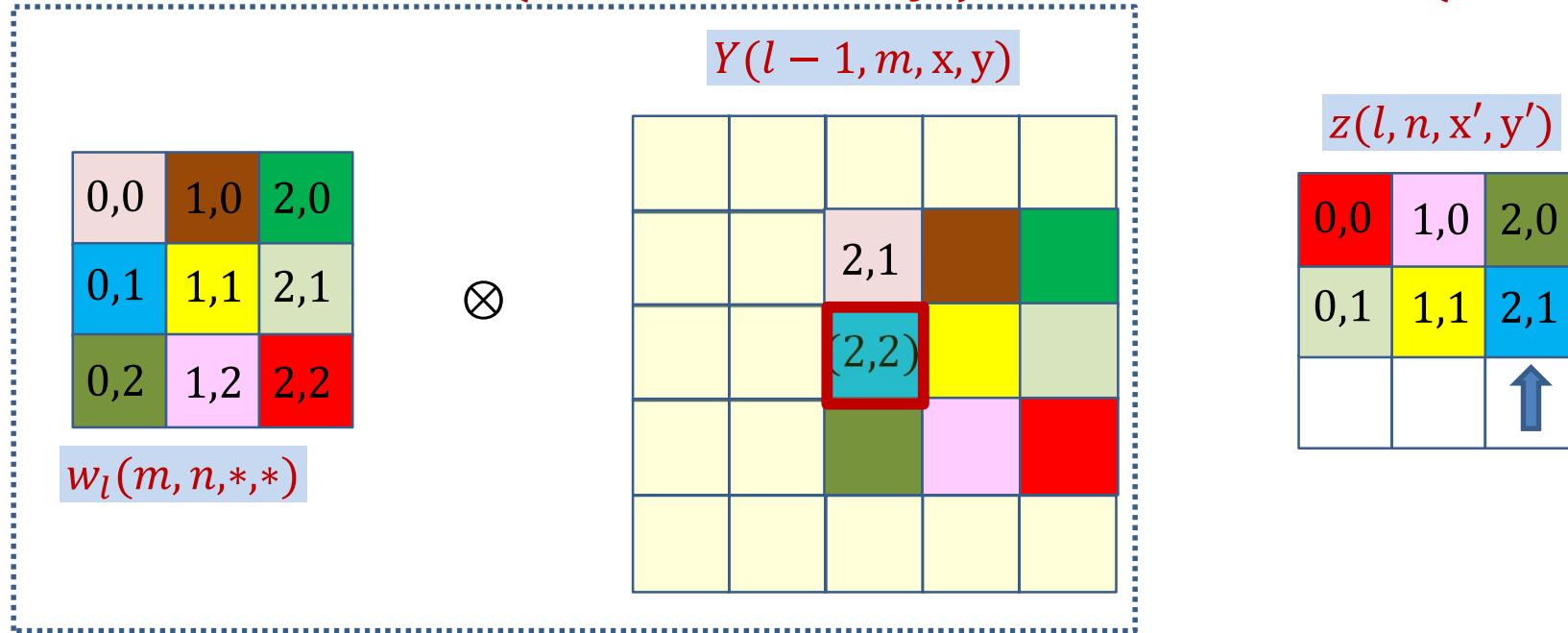


$$z(l, n, 1,1) += Y(l - 1, m, 2,2) w_l(m, n, 1,1)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2,2) w_l(m, n, 2 - x', 2 - y')$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

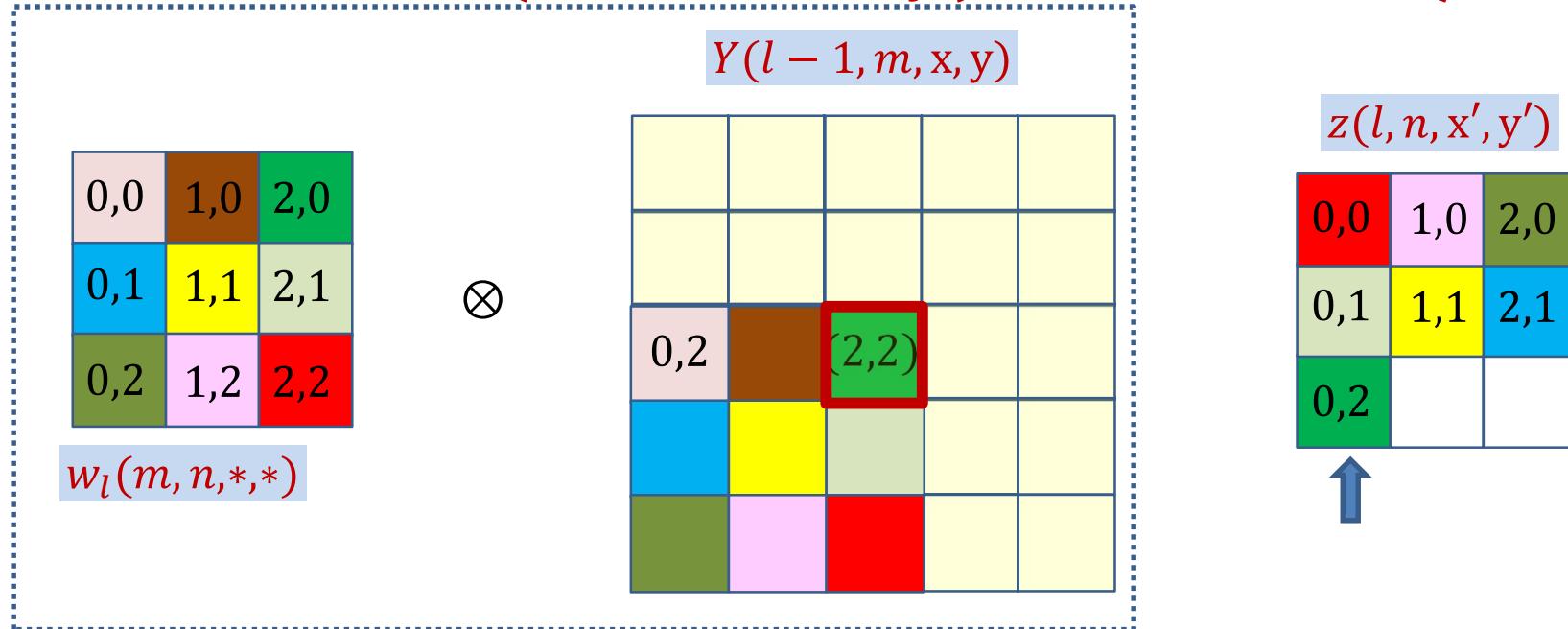


$$z(l, n, 2, 1) += Y(l - 1, m, 2, 2) w_l(m, n, 0, 1)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2, 2) w_l(m, n, 2 - x', 2 - y')$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

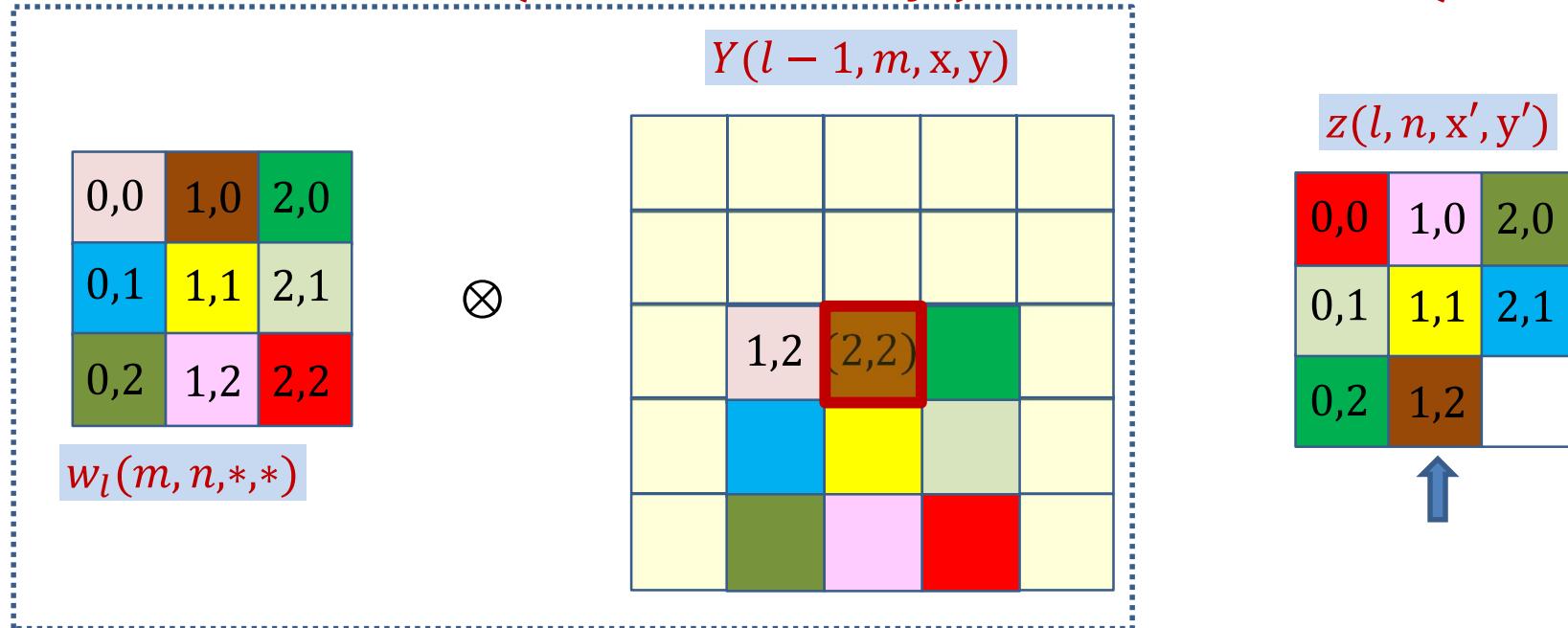


$$z(l, n, 0,2) += Y(l - 1, m, 2,2) w_l(m, n, 2,0)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2,2) w_l(m, n, 2 - x', 2 - y')$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

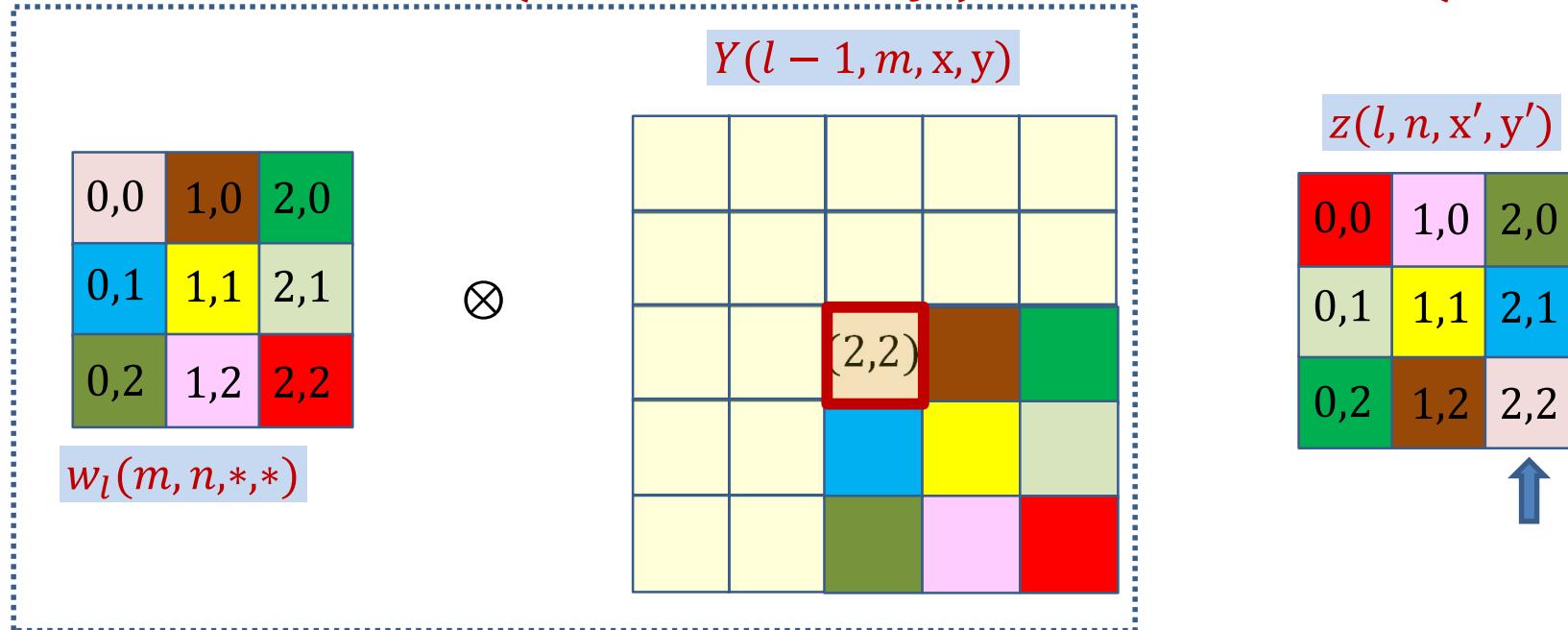


$$z(l, n, 1,2) += Y(l - 1, m, 2,2) w_l(m, n, 2,1)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2,2) w_l(m, n, 2 - x', 2 - y')$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

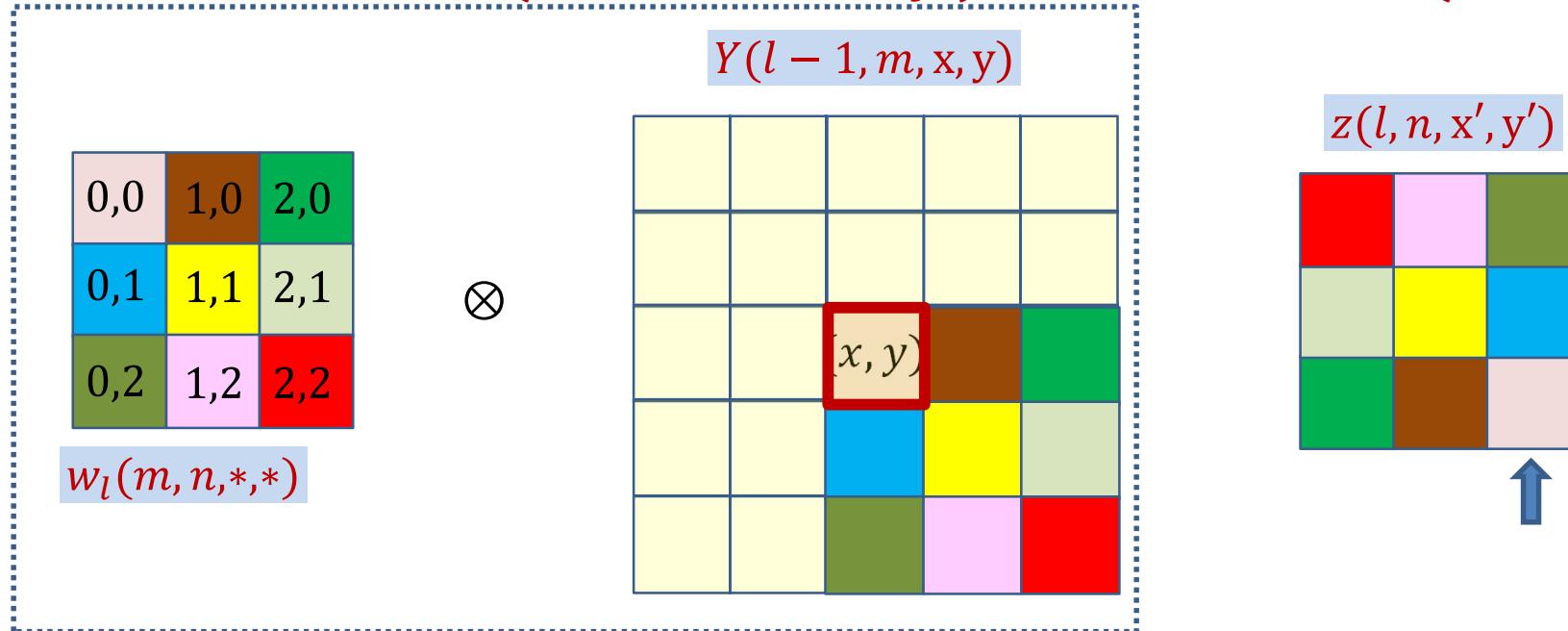


$$z(l, n, 2, 2) += Y(l - 1, m, 2, 2) w_l(m, n, 0, 0)$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

$$z(l, n, x', y') += Y(l - 1, m, 2, 2) w_l(m, n, 2 - x', 2 - y')$$

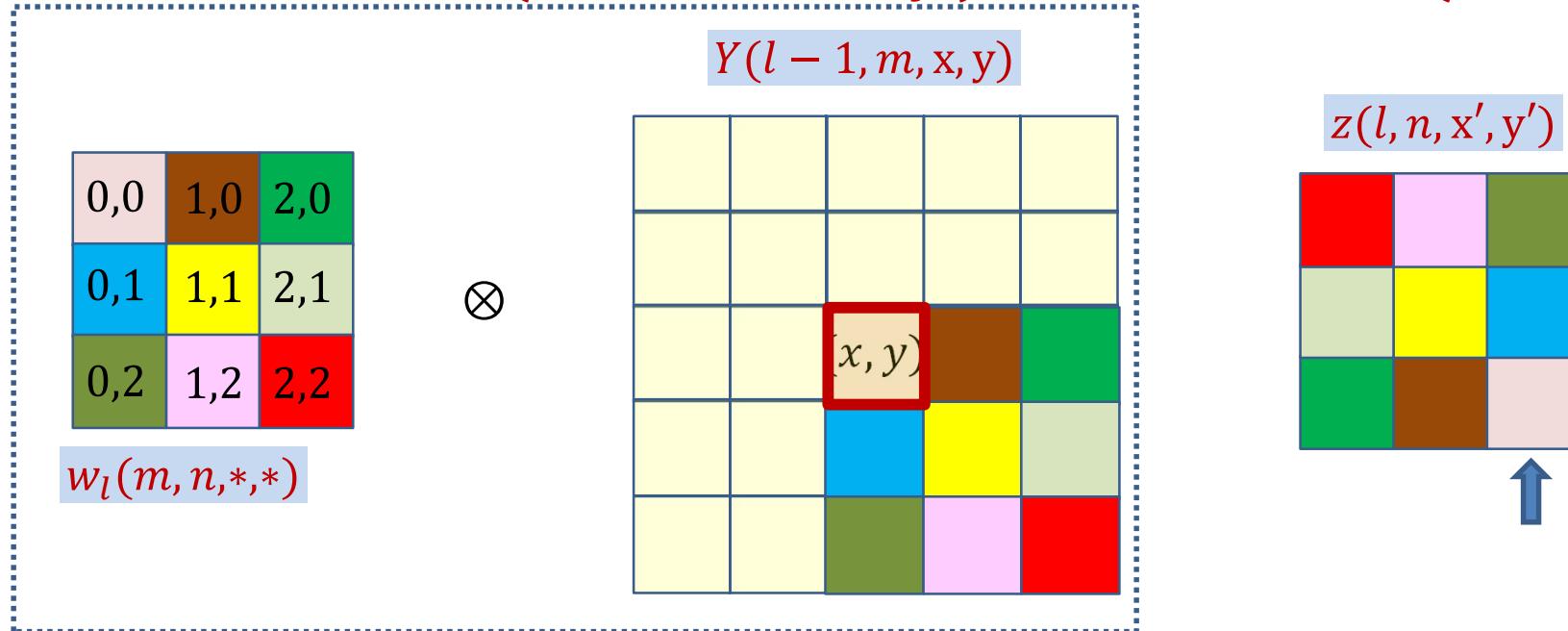
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, x', y') += Y(l - 1, m, x, y) w_l(m, n, x - x', y - y')$$

- **Note:** The coordinates of $z(l, n)$ and $w_l(m, n)$ sum to the coordinates of $Y(l - 1, m)$

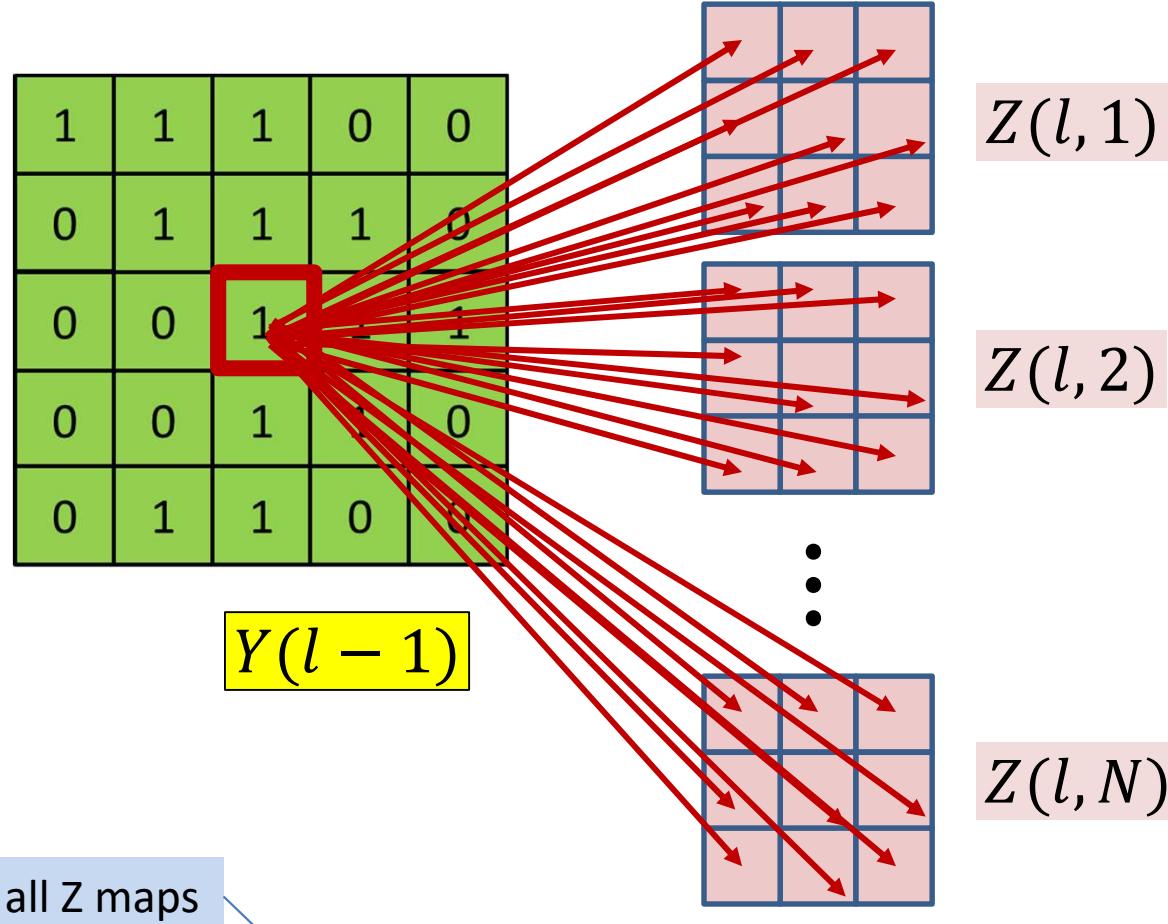
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, x', y') += Y(l - 1, m, x, y) w_l(m, n, x - x', y - y')$$

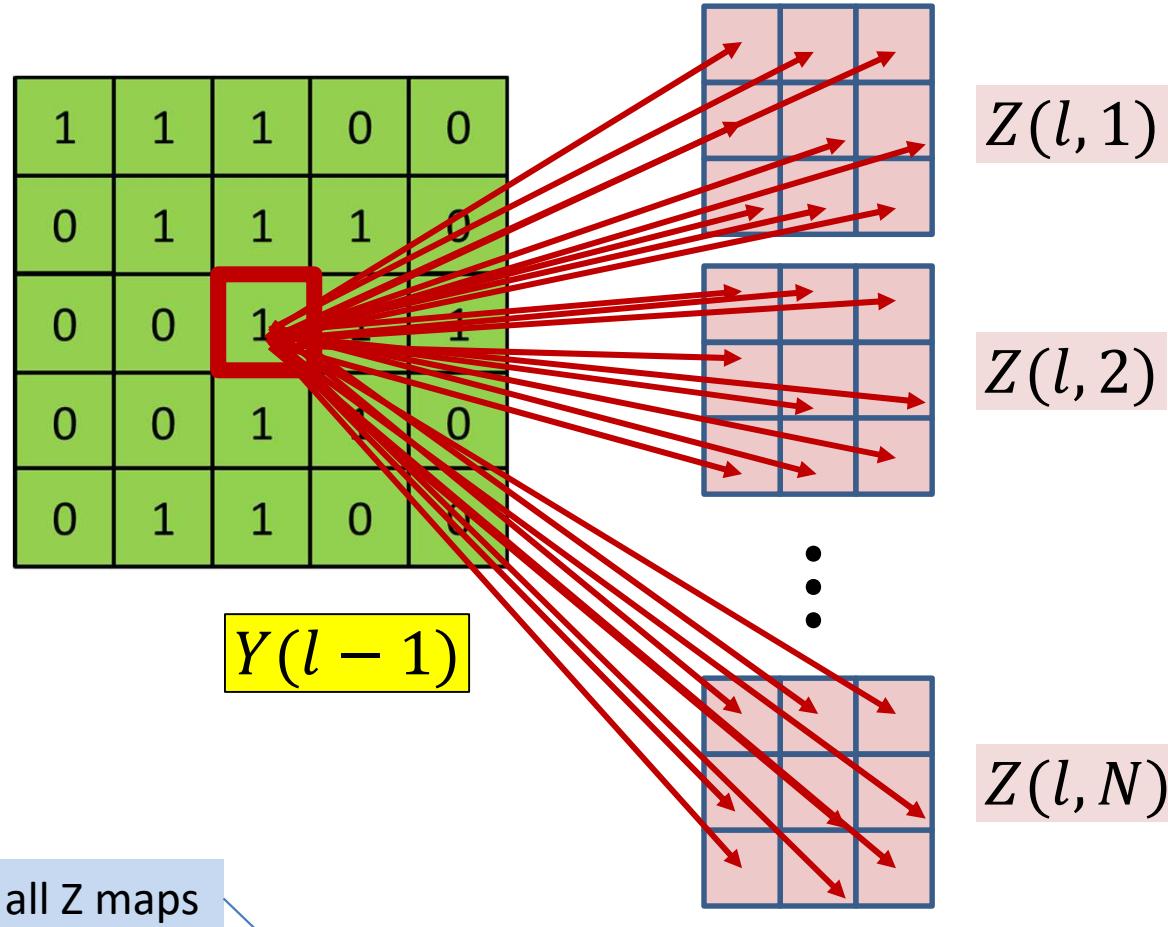
$$\frac{dz(l, n, x', y')}{dY(l - 1, m, x, y)} = w_l(m, n, x - x', y - y')$$

BP: Convolutional layer



$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} \frac{dz(l, n, x', y')}{dY(l-1, m, x, y)}$$

BP: Convolutional layer



$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

Poll 2

- @886, 887

Poll 2

In order to compute the derivative at a single affine element $Y(l,m,x,y)$, we must consider the contributions of *every* position of *every* affine map at the next layer: True or false

- **True**
- False

The derivative for an single affine element $Y(l,m,x,y)$ will require summing over every position of every Z map in the next layer: True or false

- **True**
- False

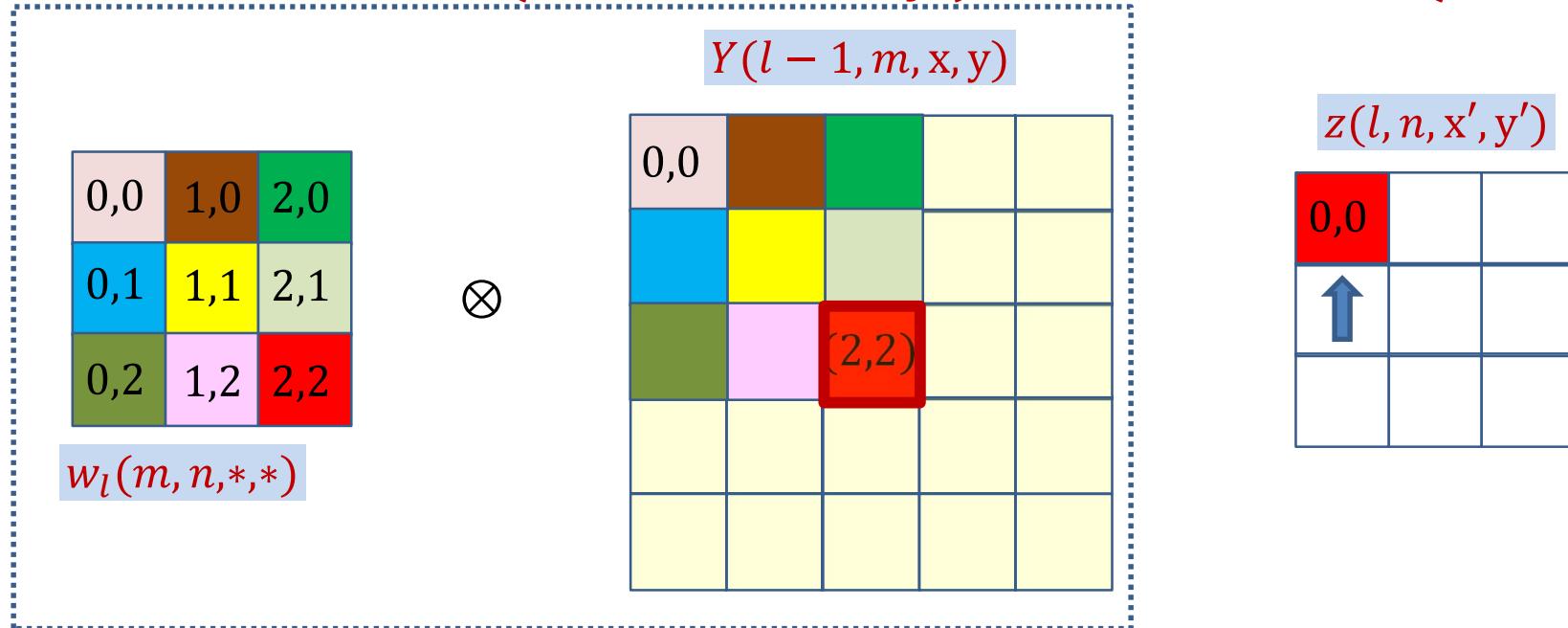
Computing derivative for $Y(l - 1, m, *, *)$

- The derivatives for every element of every map in $Y(l - 1)$ by direct implementation of the formula:

$$\frac{dDiv}{dY(l - 1, m, x, y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

- But this is actually a convolution!
 - Let's see how

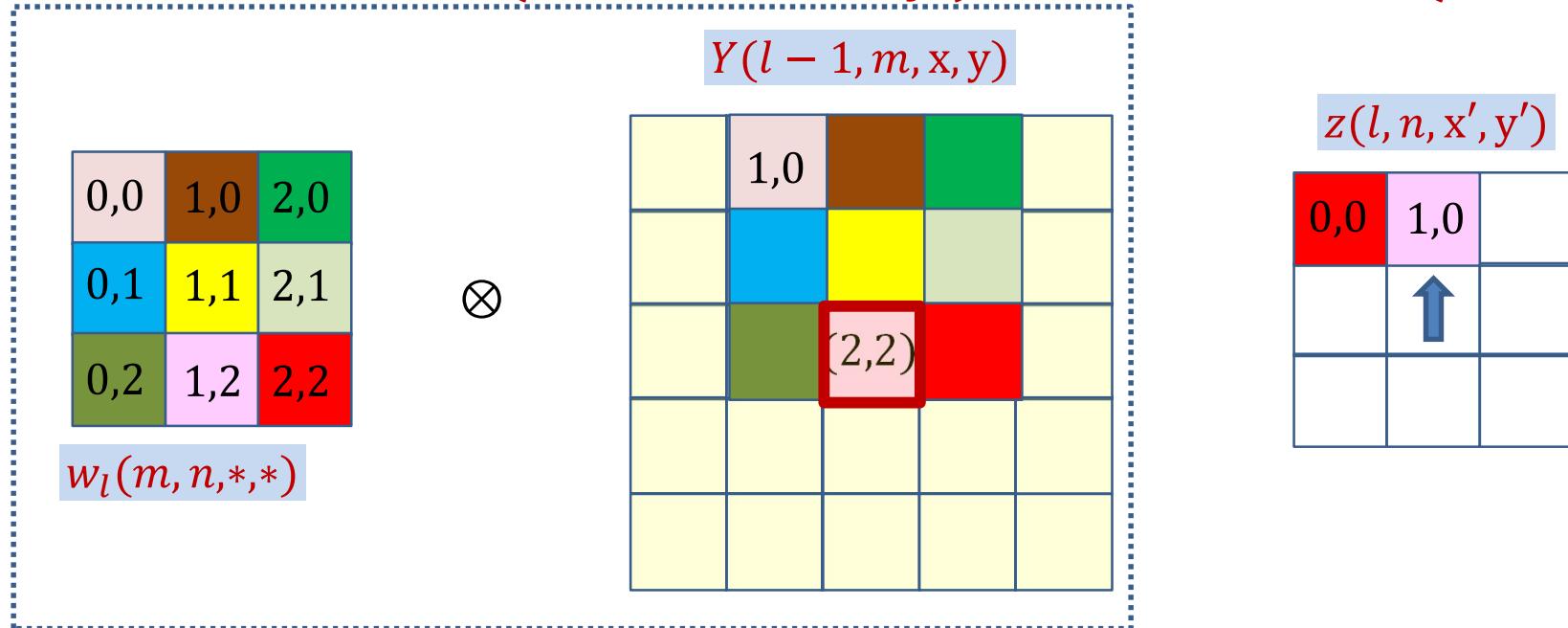
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 0, 0) += Y(l - 1, m, 2, 2) w_l(m, n, 2, 2)$$

$$\frac{dDiv}{dY(l - 1, m, 2, 2)} += \frac{dDiv}{dz(l, n, 0, 0)} w_l(m, n, 2, 2)$$

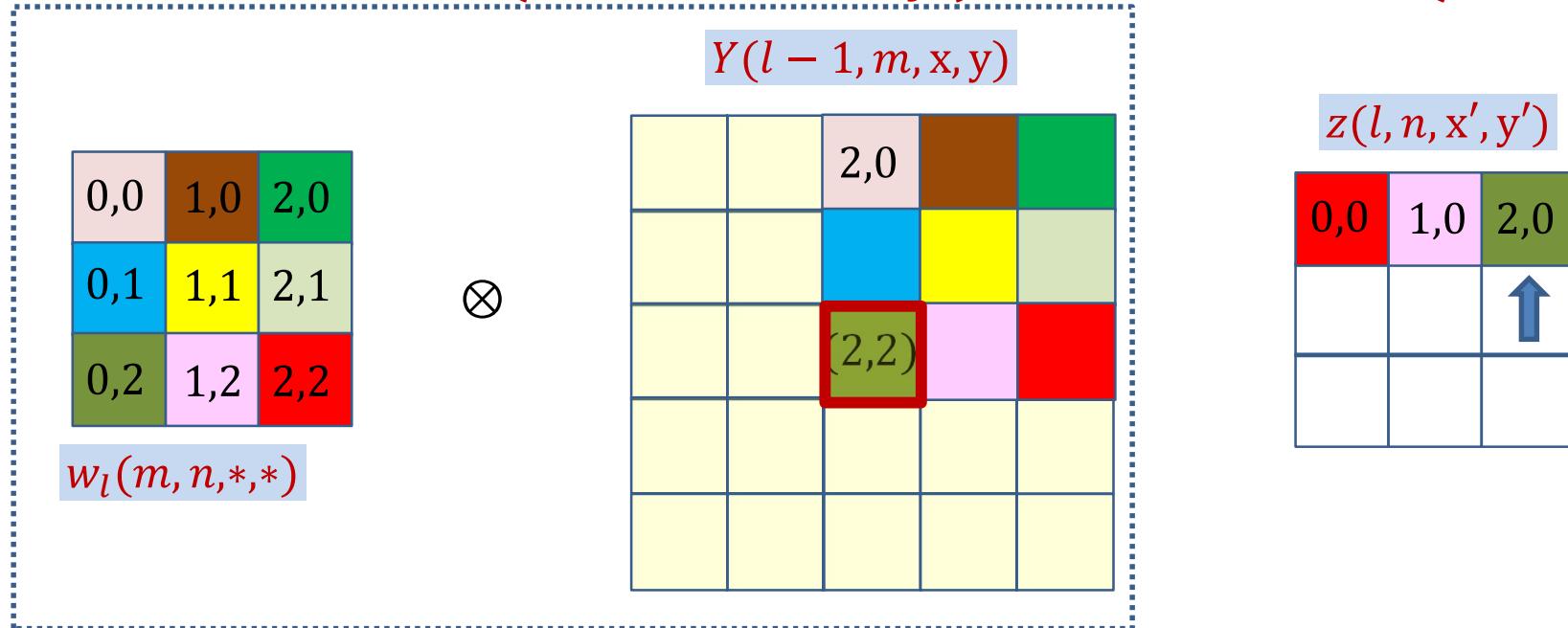
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 1,0) += Y(l - 1, m, 2,2) w_l(m, n, 1,2)$$

$$\frac{dDiv}{dY(l - 1, m, 2,2)} += \frac{dDiv}{dz(l, n, 1,0)} w_l(m, n, 1,2)$$

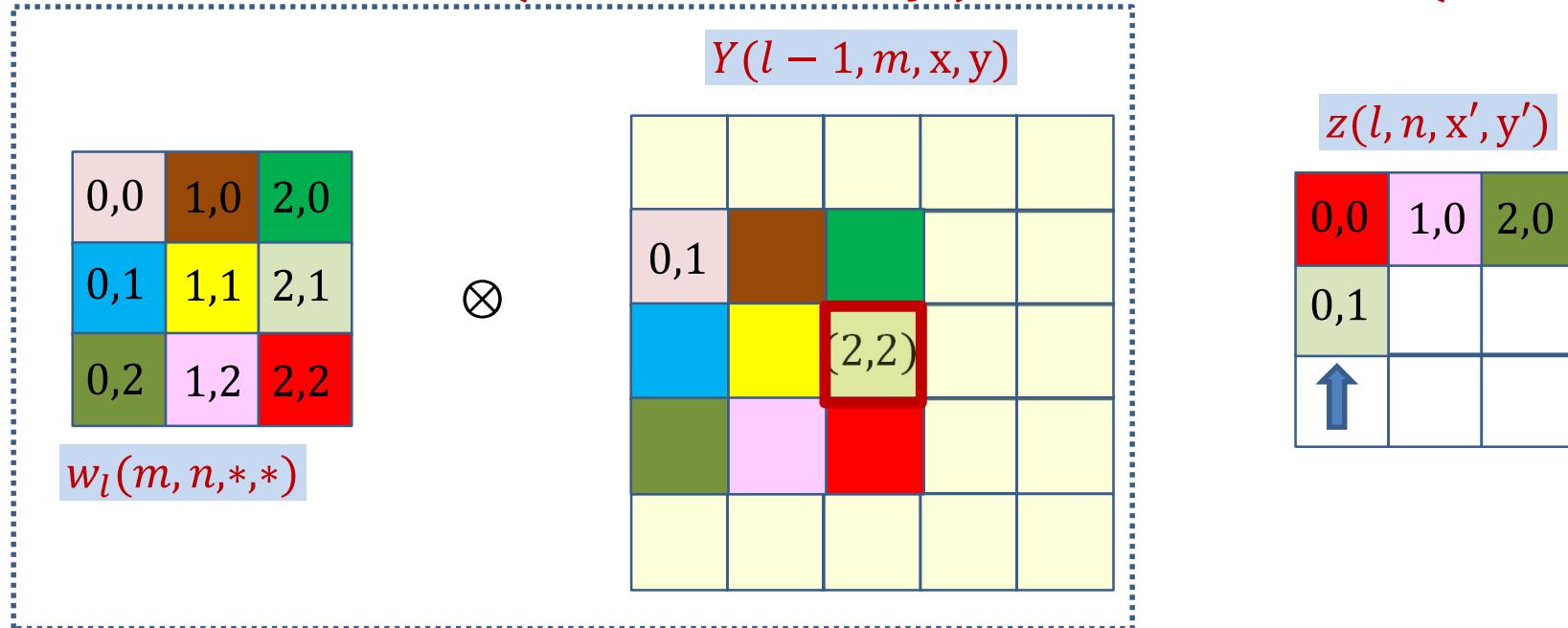
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 2,0) += Y(l - 1, m, 2,2)w_l(m, n, 0,2)$$

$$\frac{dDiv}{dY(l - 1, m, 2,2)} += \frac{dDiv}{dz(l, n, 2,0)} w_l(m, n, 0,2)$$

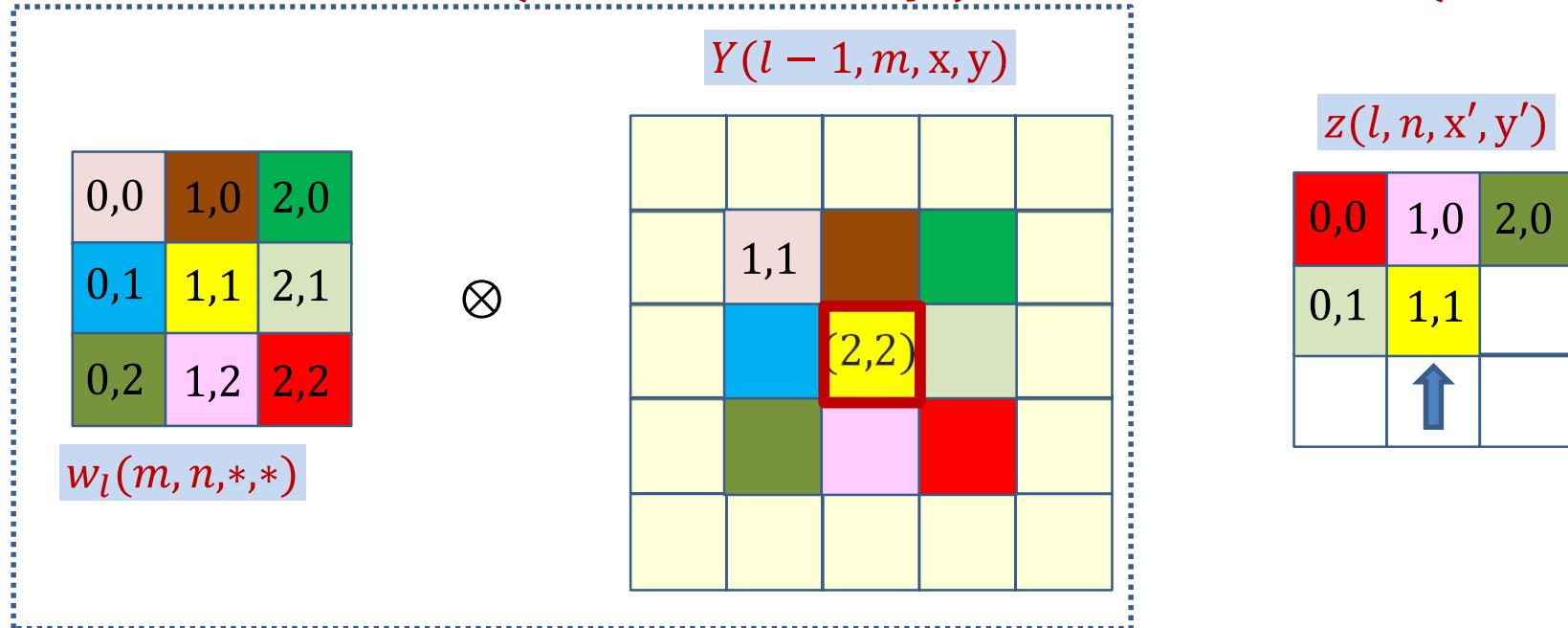
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 0, 1) += Y(l - 1, m, 2, 2) w_l(m, n, 2, 1)$$

$$\frac{dDiv}{dY(l - 1, m, 2, 2)} += \frac{dDiv}{dz(l, n, 0, 1)} w_l(m, n, 2, 1)$$

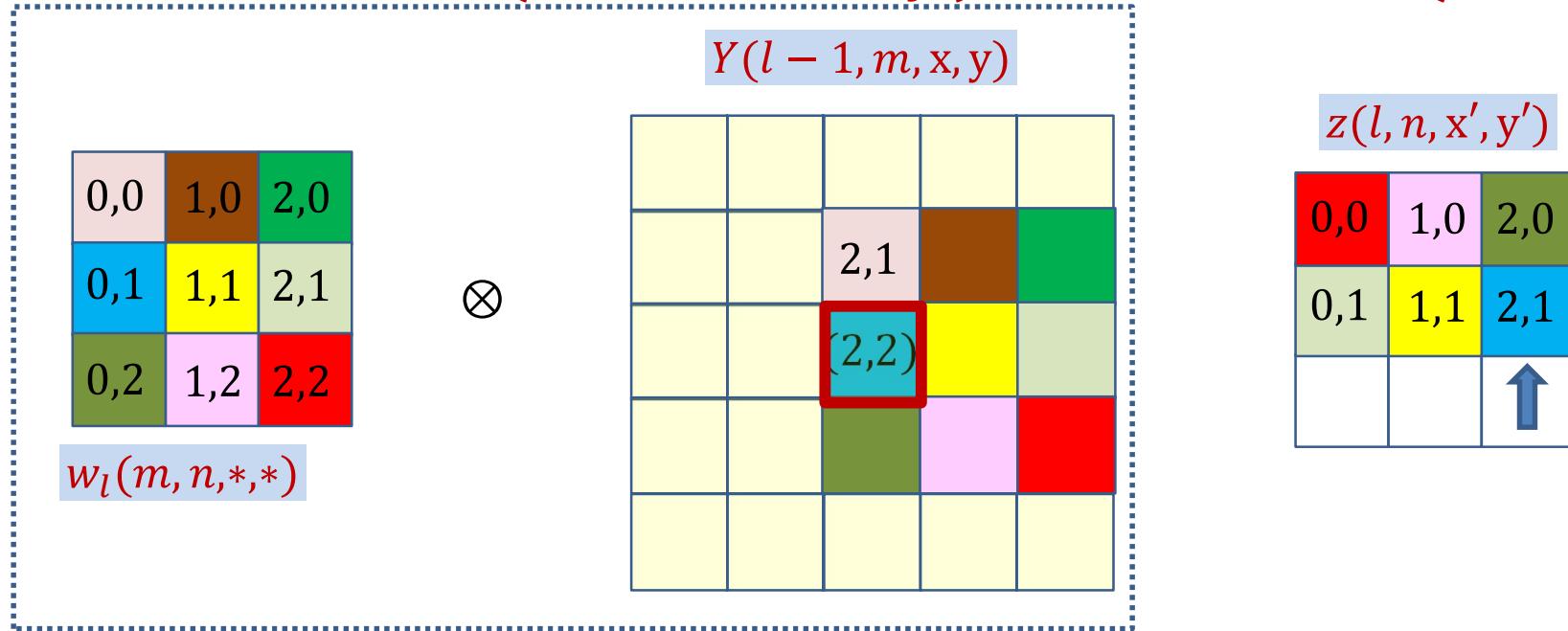
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 1,1) += Y(l - 1, m, 2,2) w_l(m, n, 1,1)$$

$$\frac{dDiv}{dY(l - 1, m, 2,2)} += \frac{dDiv}{dz(l, n, 1,1)} w_l(m, n, 1,1)$$

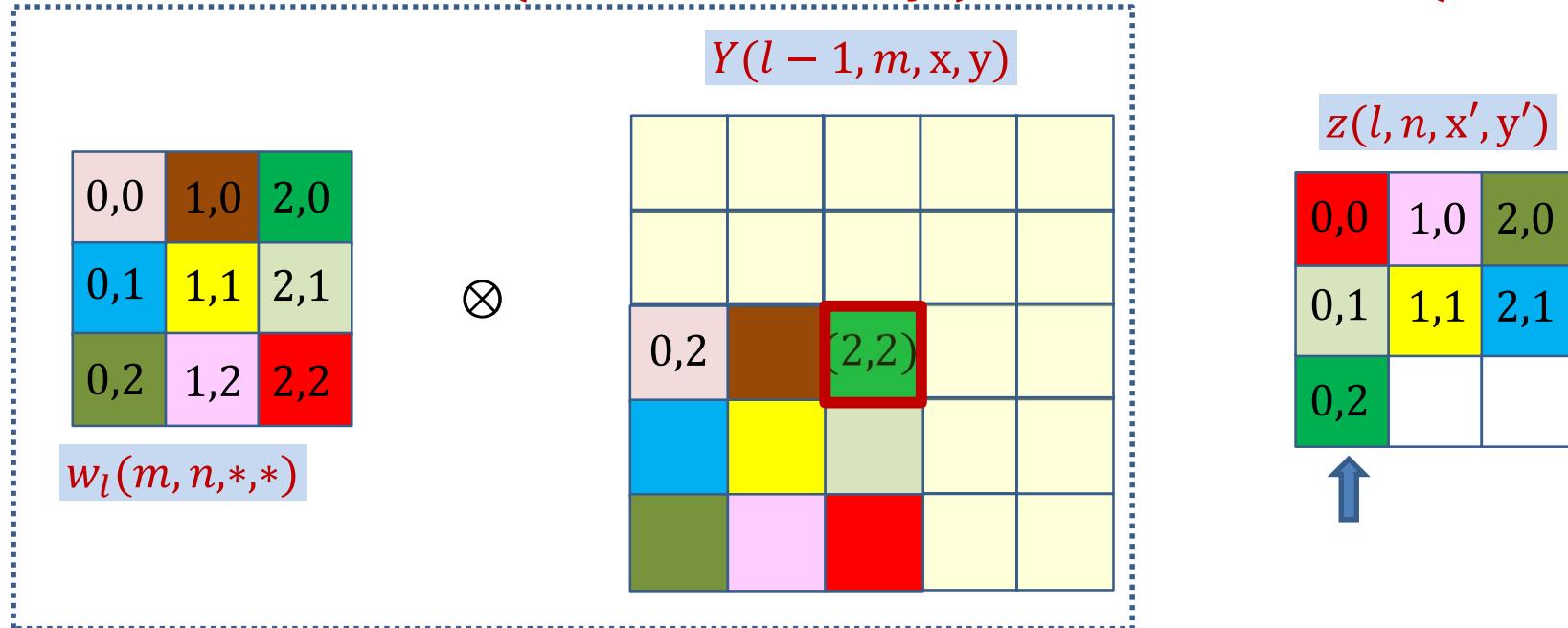
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 2, 1) += Y(l - 1, m, 2, 2) w_l(m, n, 0, 1)$$

$$\frac{dDiv}{dY(l - 1, m, 2, 2)} += \frac{dDiv}{dz(l, n, 2, 1)} w_l(m, n, 0, 1)$$

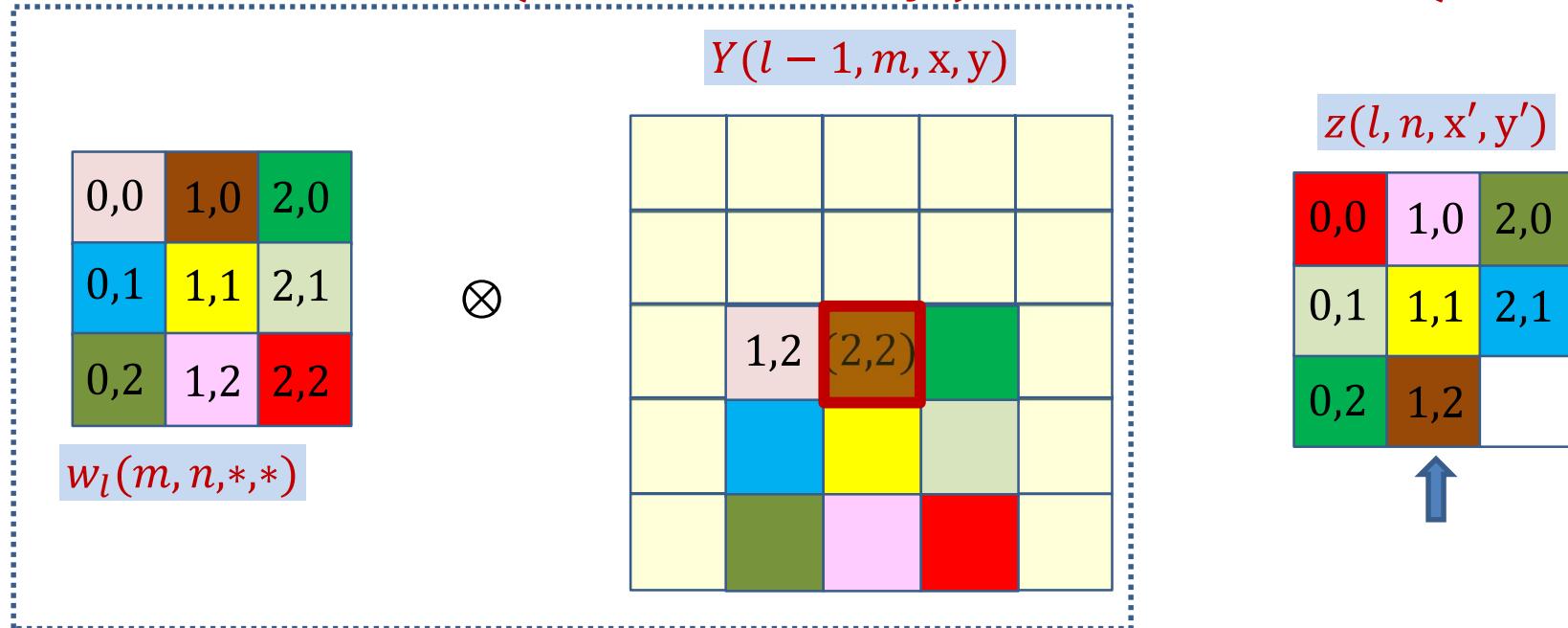
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 0, 2) += Y(l - 1, m, 2, 2) w_l(m, n, 2, 0)$$

$$\frac{dDiv}{dY(l - 1, m, 2, 2)} += \frac{dDiv}{dz(l, n, 0, 2)} w_l(m, n, 2, 0)$$

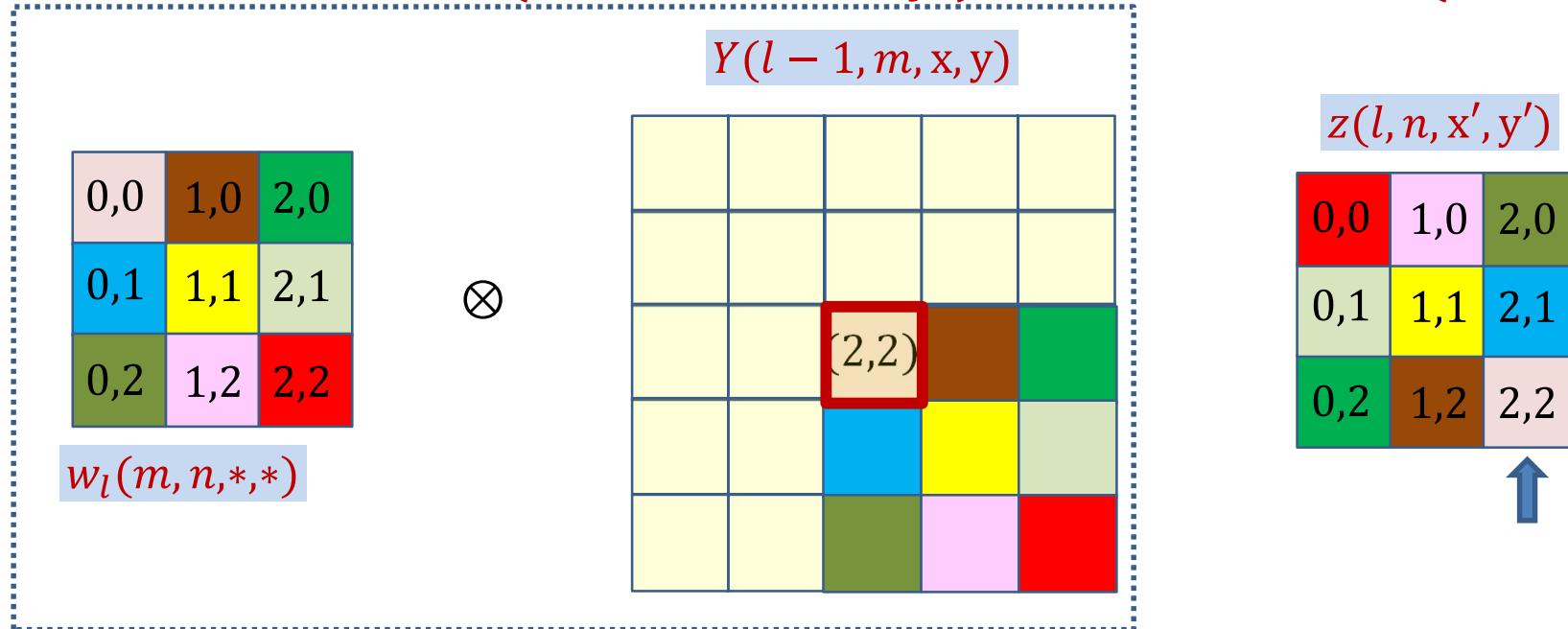
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 1,2) += Y(l - 1, m, 2,2) w_l(m, n, 2,1)$$

$$\frac{dDiv}{dY(l - 1, m, 2,2)} += \frac{dDiv}{dz(l, n, 1,2)} w_l(m, n, 1,0)$$

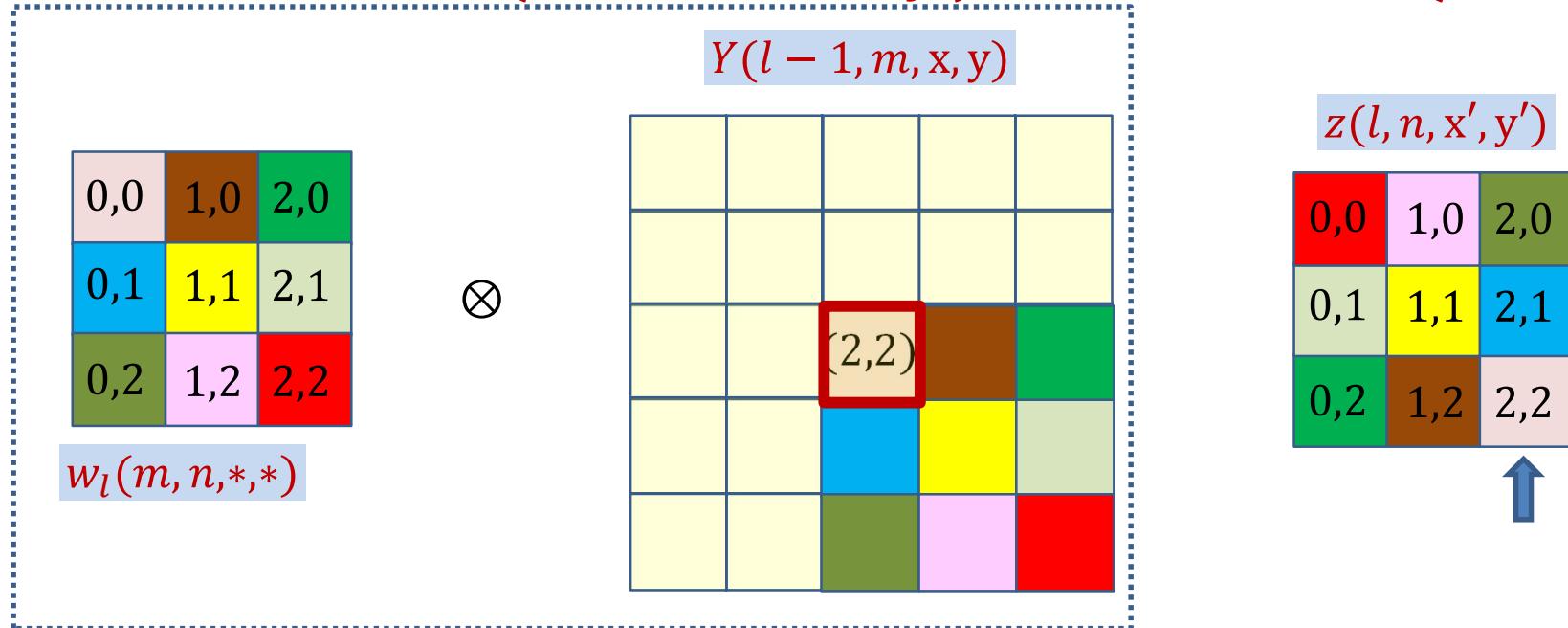
How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$



$$z(l, n, 2,2) += Y(l - 1, m, 2,2) w_l(m, n, 0,0)$$

$$\frac{dDiv}{dY(l - 1, m, 2,2)} += \frac{dDiv}{dz(l, n, 2,2)} w_l(m, n, 0,0)$$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

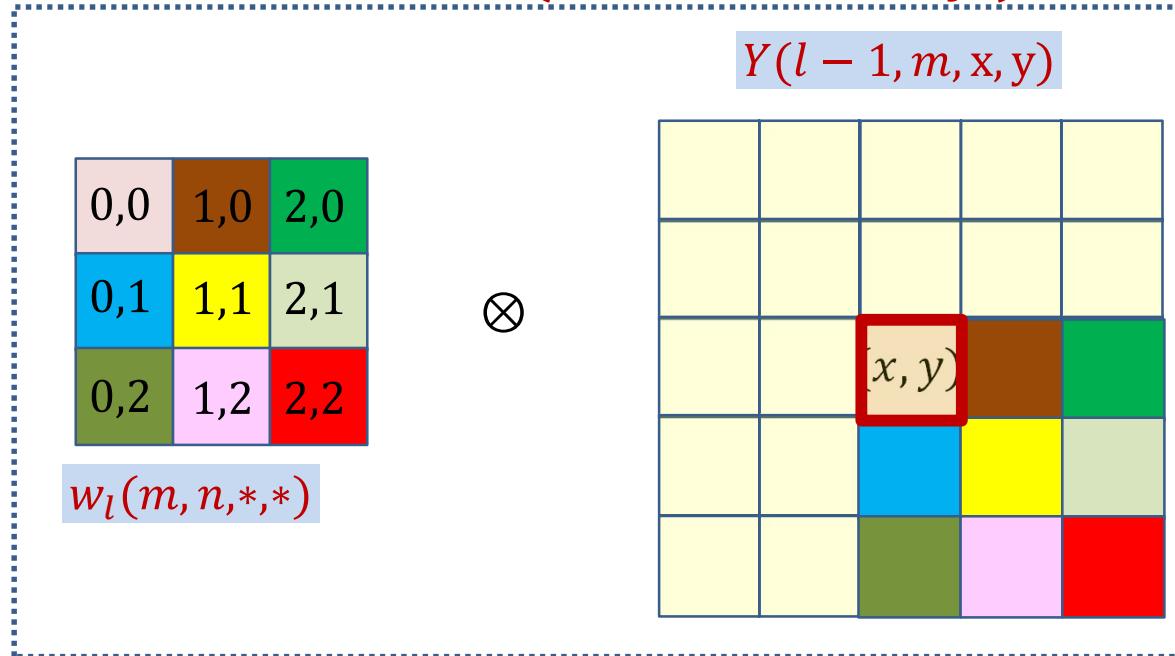


$$z(l, n, x', y') += Y(l - 1, m, 2, 2) w_l(m, n, 2 - x', 2 - y')$$

$$\frac{dDiv}{dY(l - 1, m, 2, 2)} += \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, 2 - x', 2 - y')$$

- The derivative at $Y(l - 1, m, 2, 2)$ is the sum of component-wise product of the filter elements and the elements of the derivative at $z(l, m, \dots)$

How a single $Y(l - 1, m, x, y)$ influences $z(l, n, x', y')$

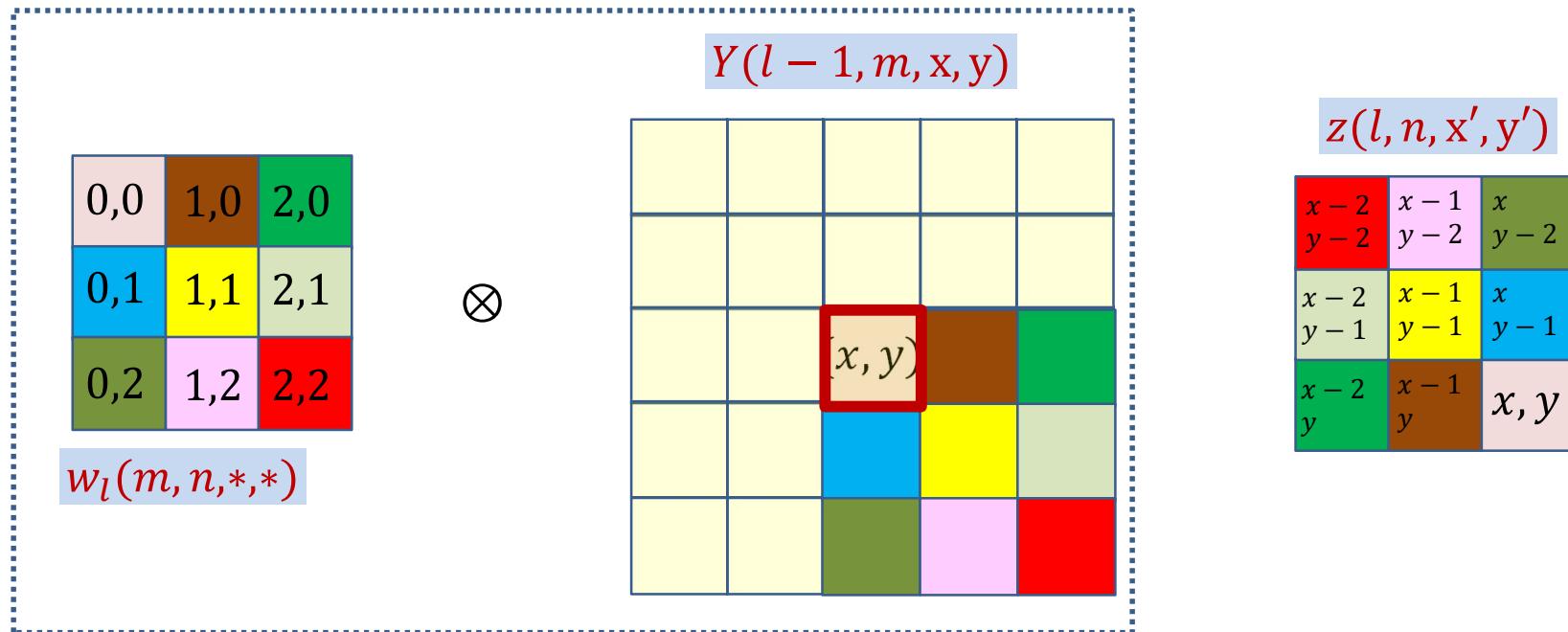


$$z(l, n, x', y') += Y(l - 1, m, x, y) w_l(m, n, x - x', y - y')$$

$$\frac{dDiv}{dY(l - 1, m, x, y)} += \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

- The derivative at $Y(l - 1, m, x, y)$ is the sum of component-wise product of the filter elements and the elements of the derivative at $z(l, m, \dots)$

Derivative at $Y(l - 1, m, x, y)$ from a single $Z(l, n)$ map



$$z(l, n, x', y') += Y(l - 1, m, x, y) w_l(m, n, x - x', y - y')$$

$$\frac{dDiv}{dY(l - 1, m, x, y)} += \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

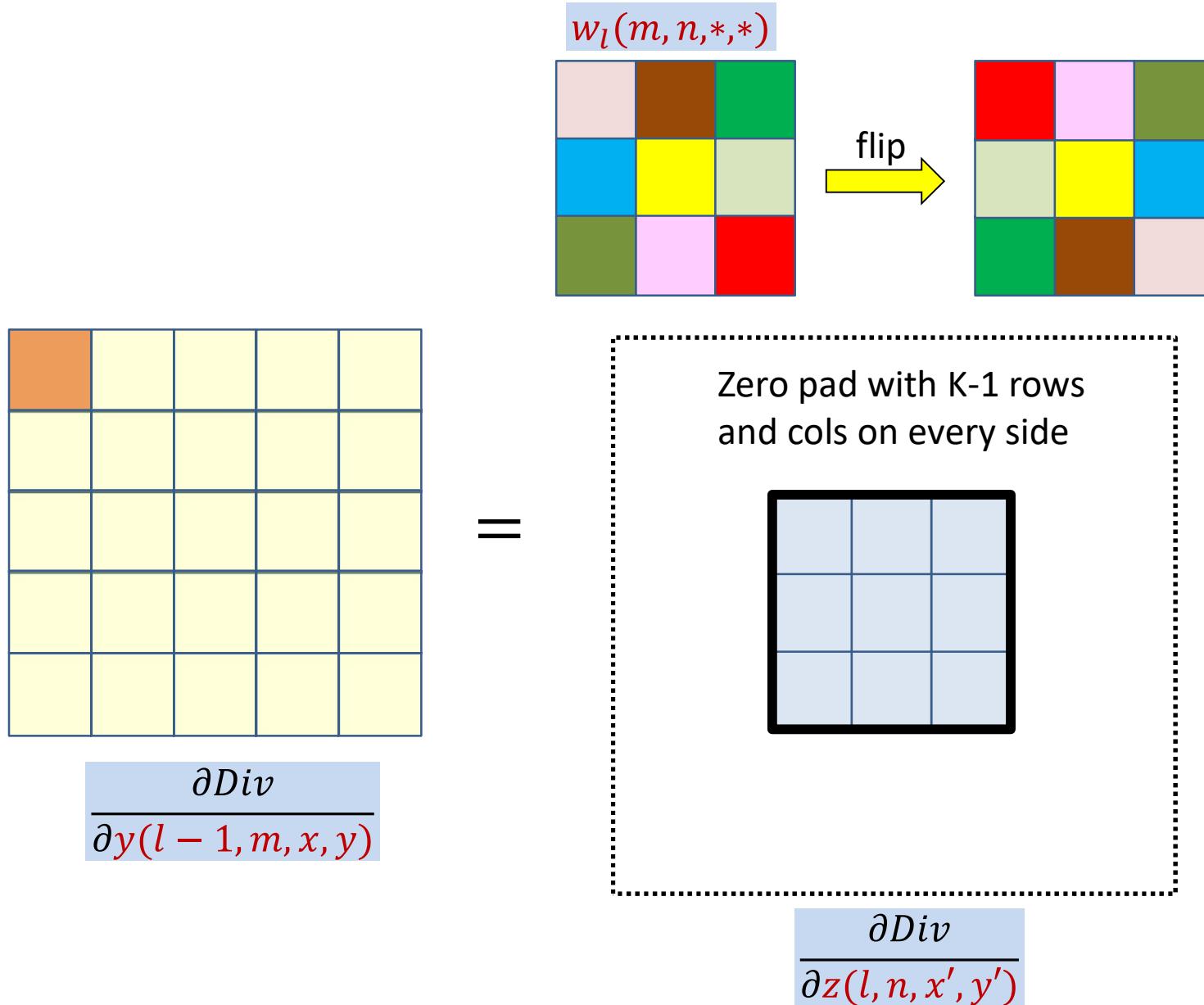
Contribution of the entire n th affine map $z(l, n, *, *)$

Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map

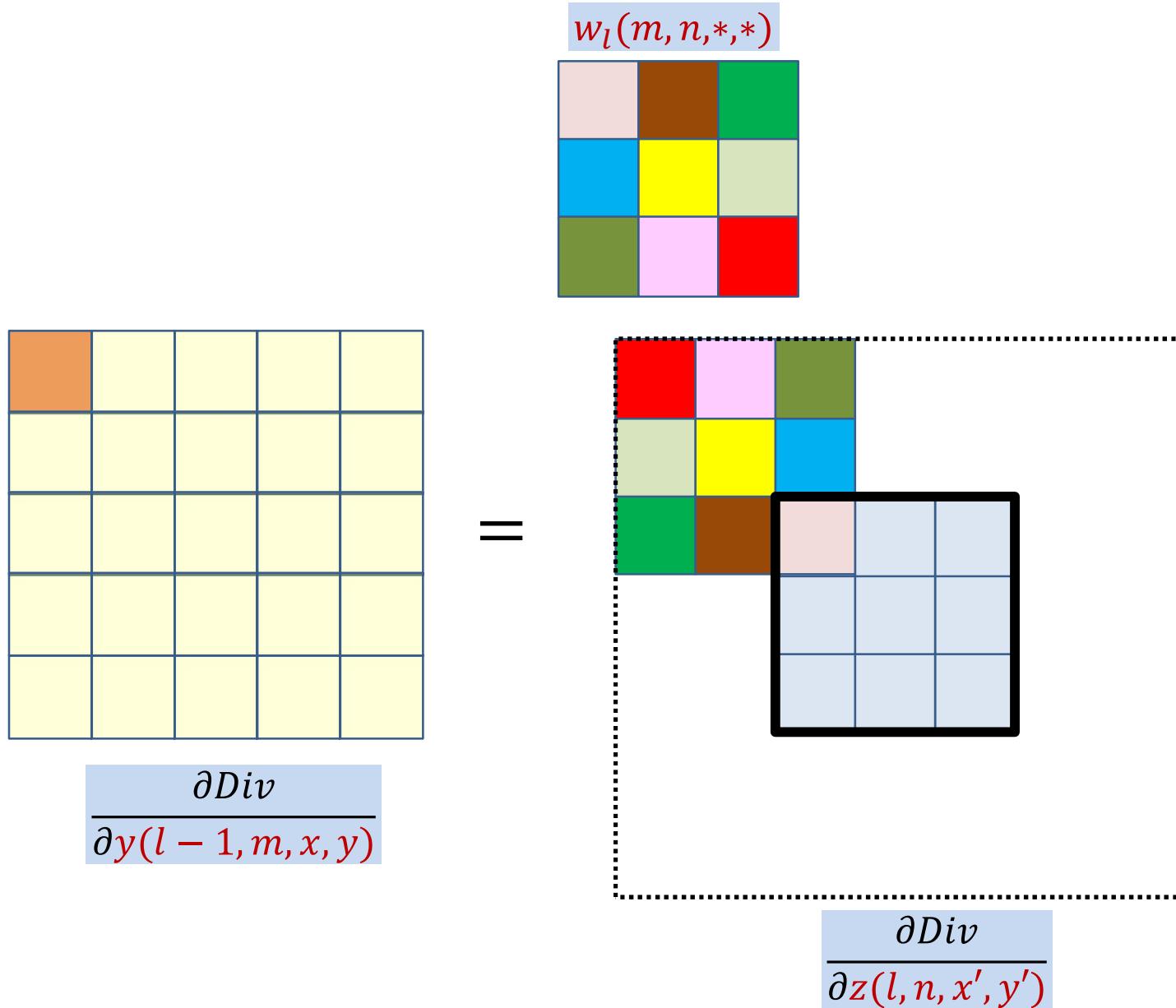
$$\frac{\partial \text{Div}}{\partial y(l - 1, m, x, y)} = w_l(m, n, *, *) \cdot \frac{\partial \text{Div}}{\partial z(l, n, x', y')}$$

The diagram illustrates the computation of a derivative from a single input map. On the left, a large 5x5 grid represents the input map $Y(l - 1, m)$. A single orange cell in the top-left corner is highlighted. An equals sign follows the input grid. To the right of the equals sign is a 3x3 kernel labeled $w_l(m, n, *, *)$, which is itself a 3x3 grid of colored cells. The result of the multiplication is shown as a 3x3 grid with a black border, representing the output map $Z(l, n)$.

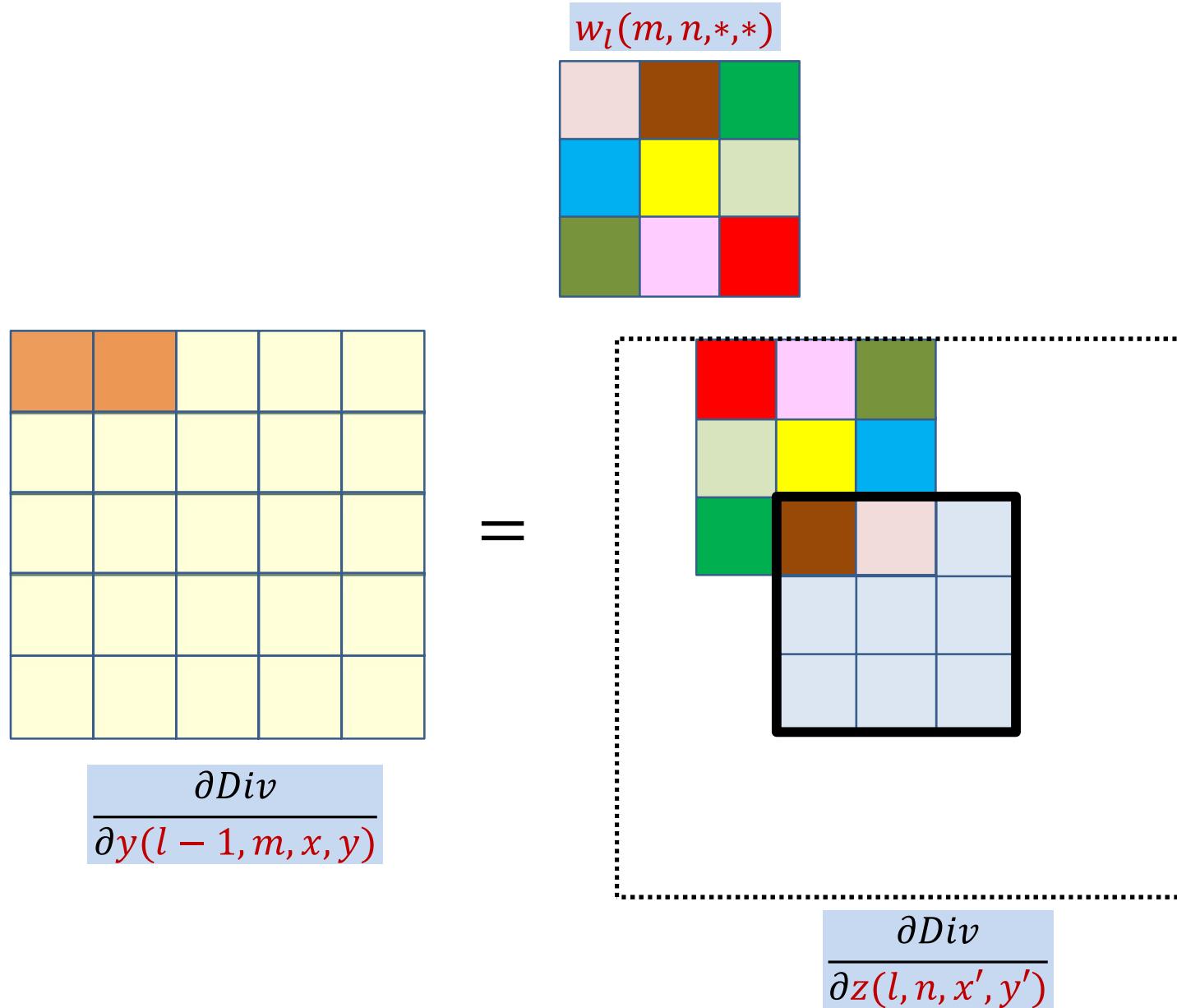
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



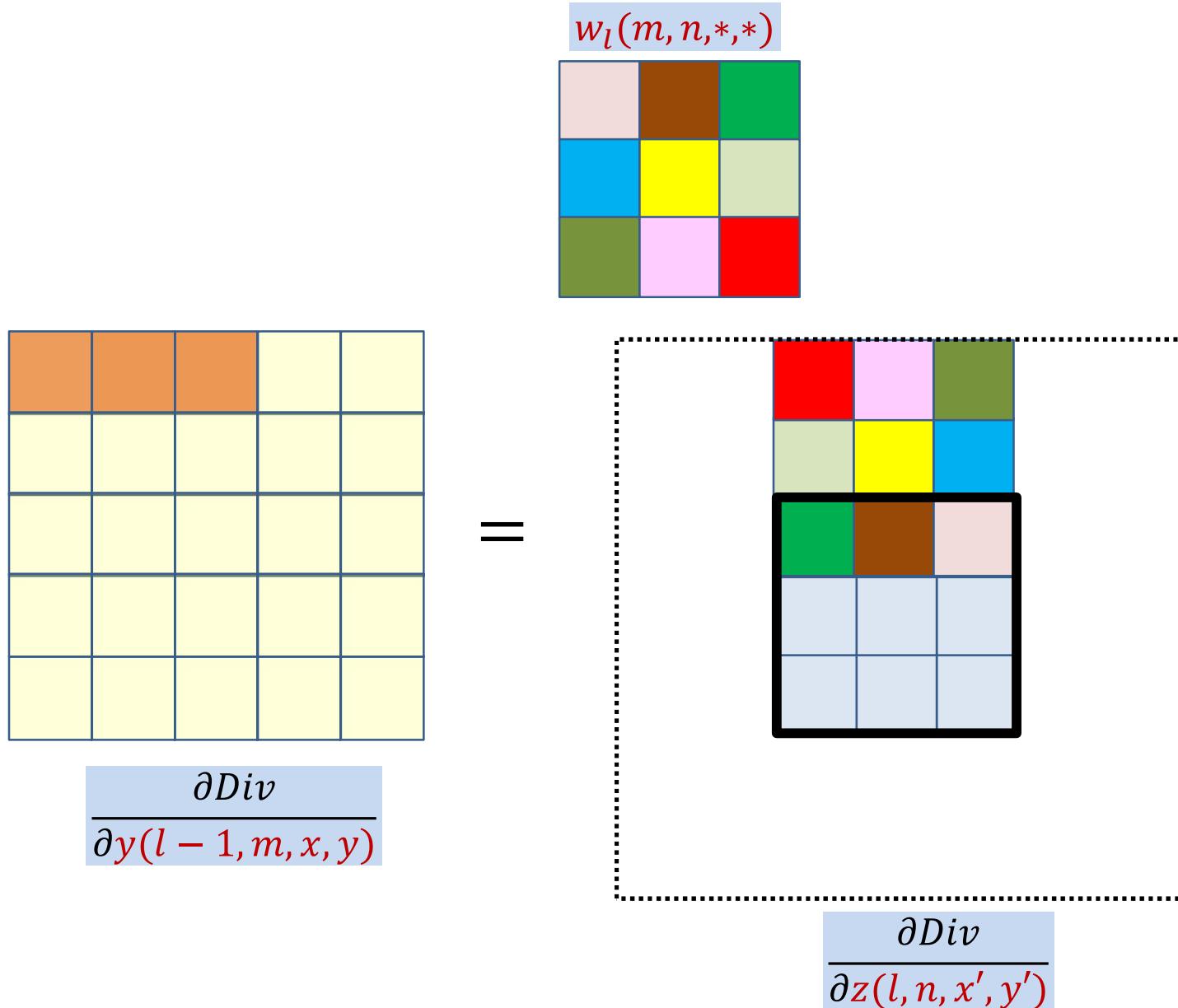
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



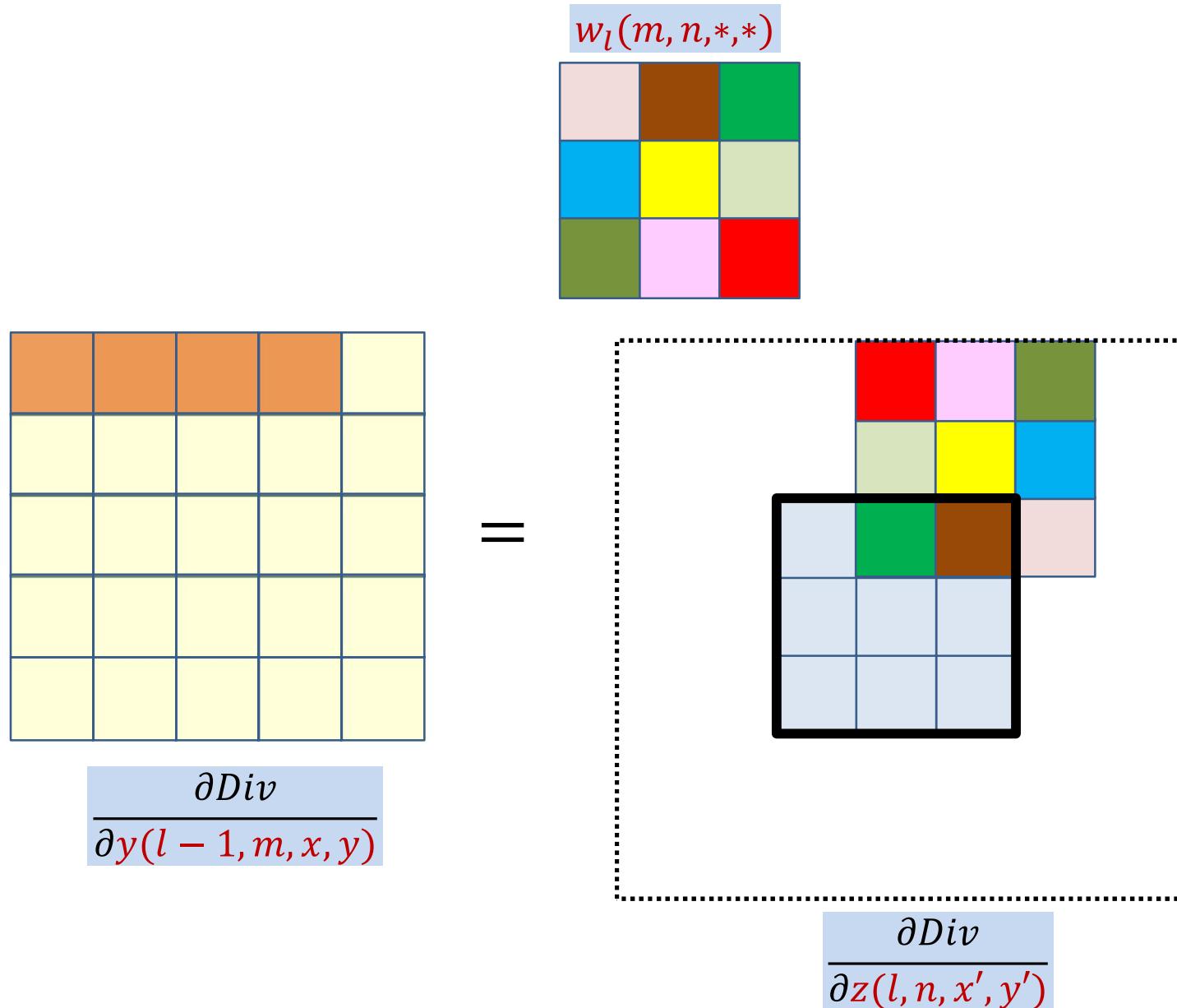
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



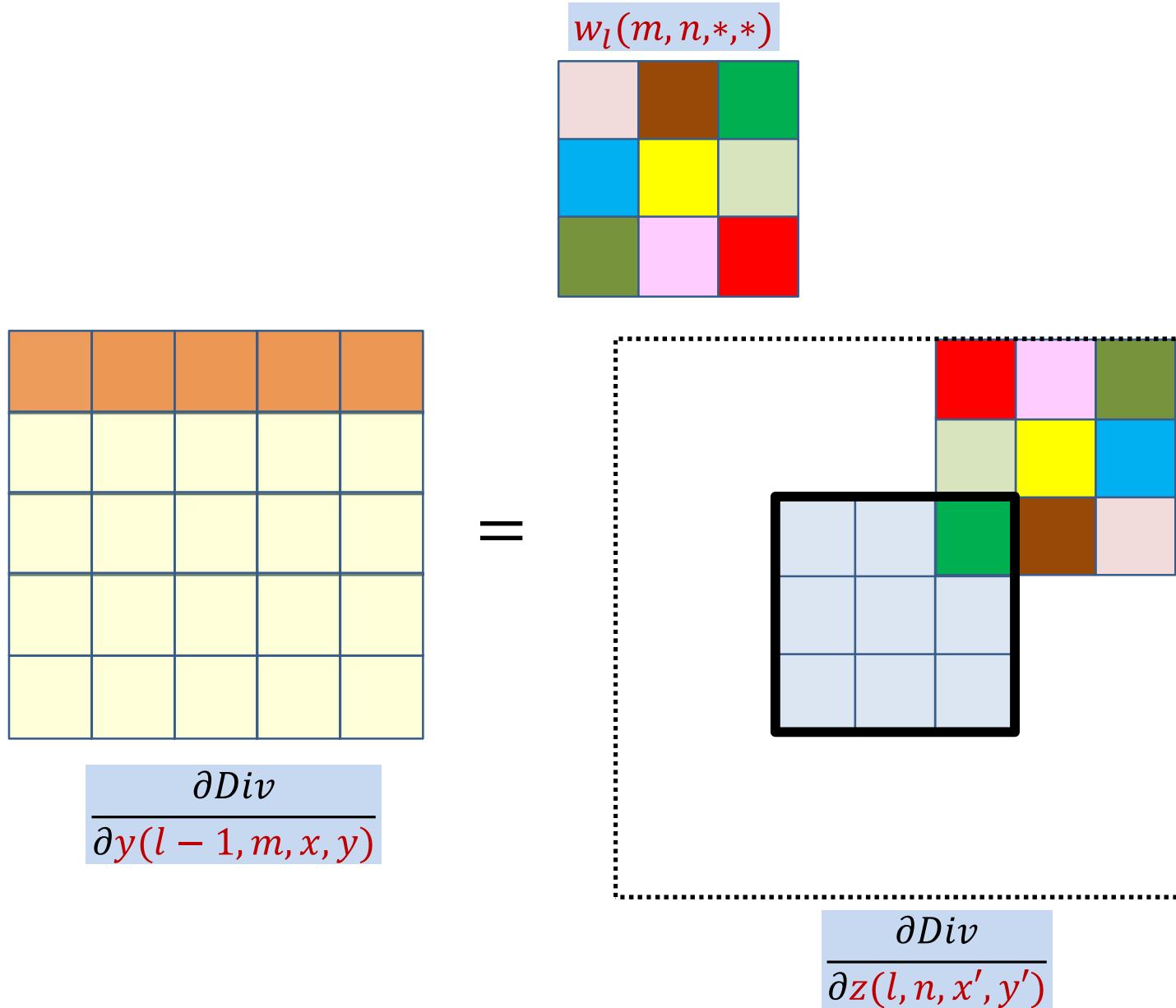
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



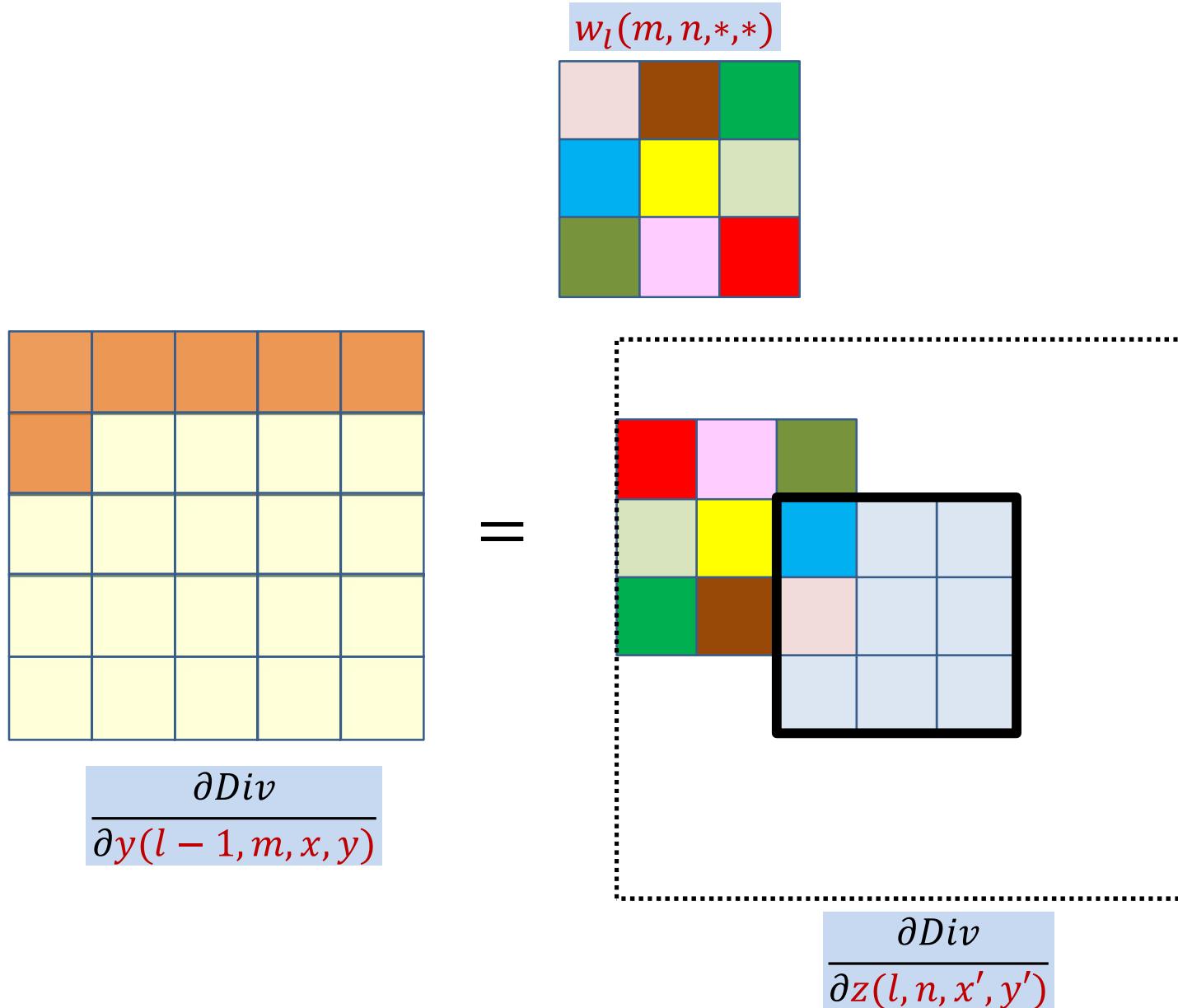
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



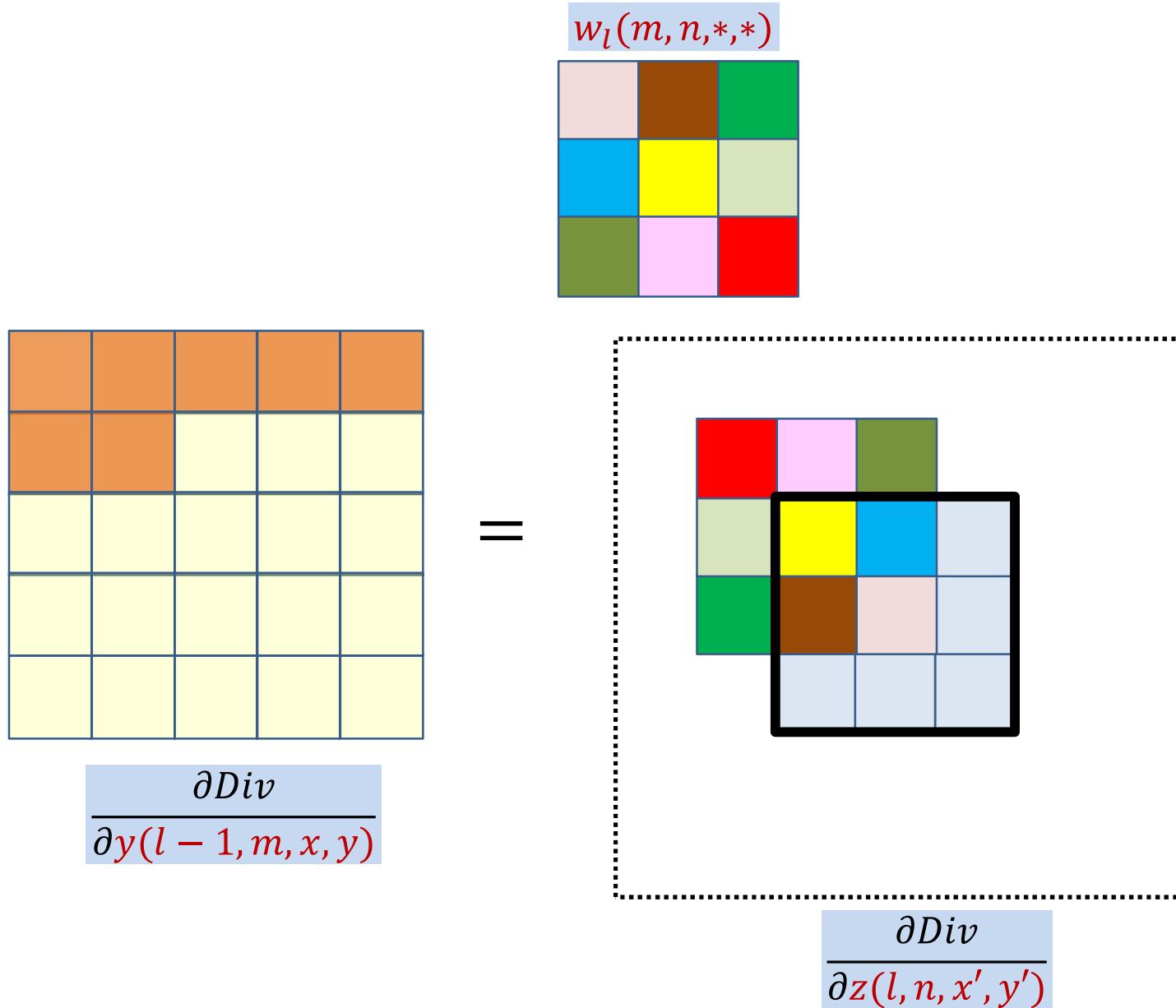
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



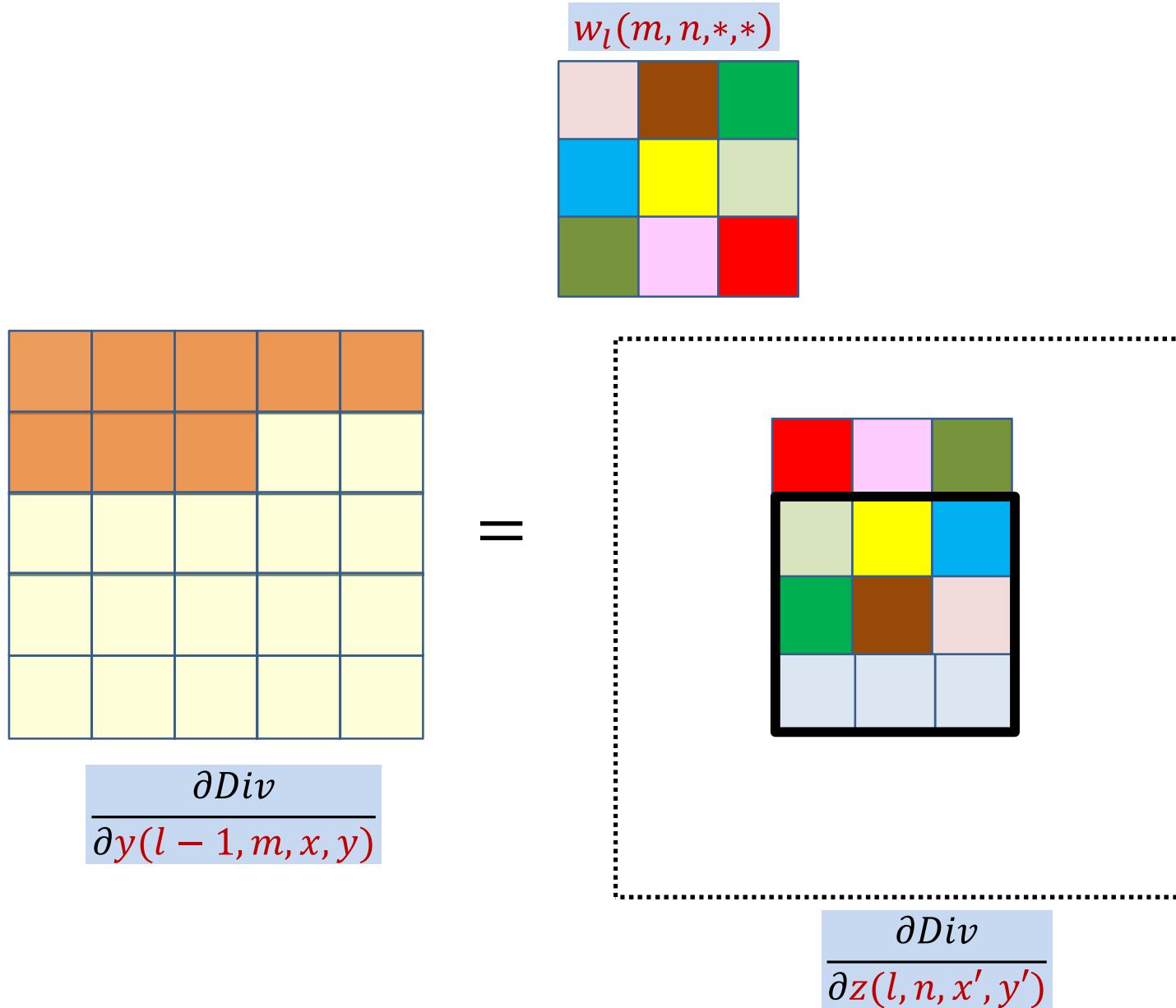
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



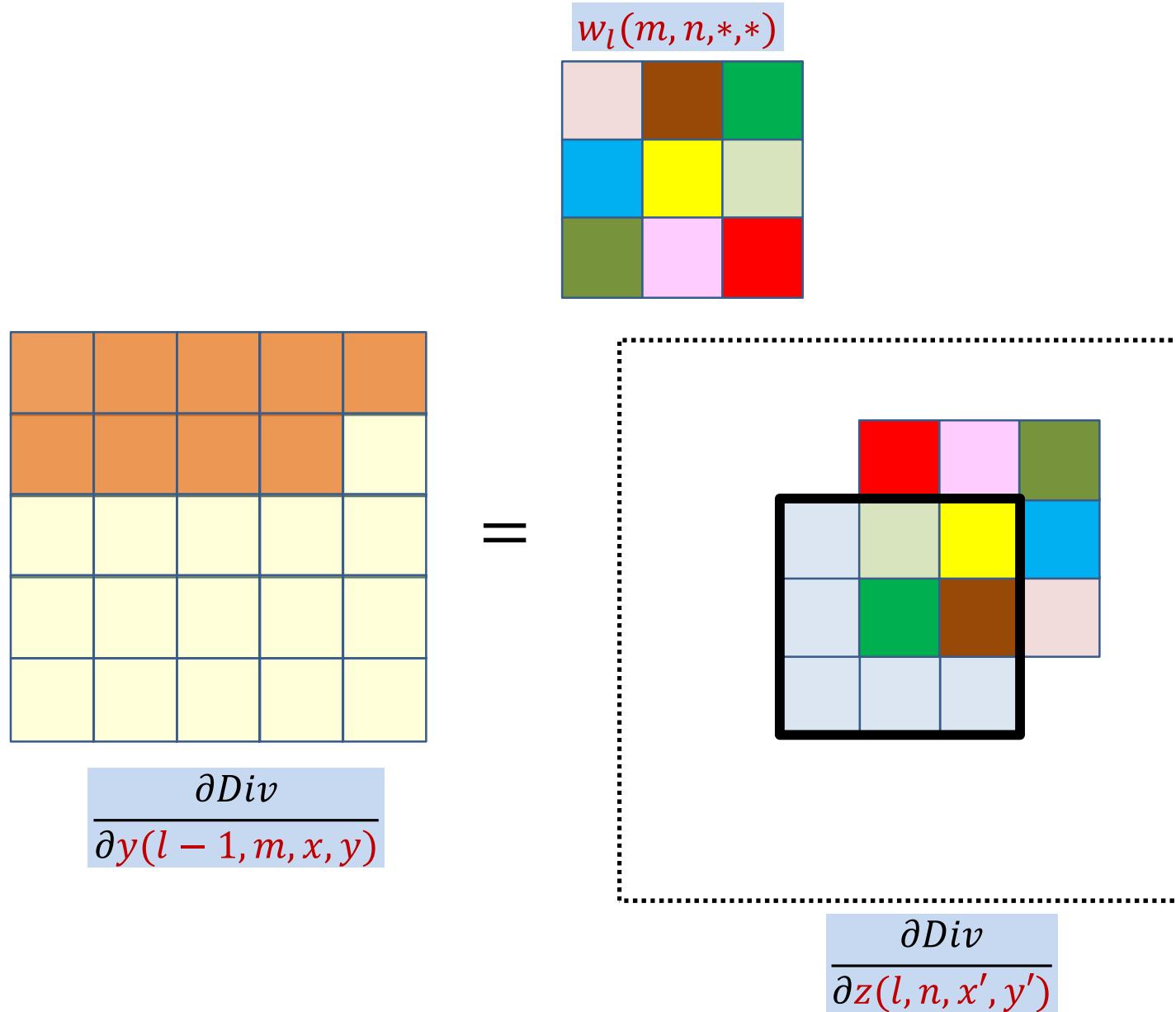
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



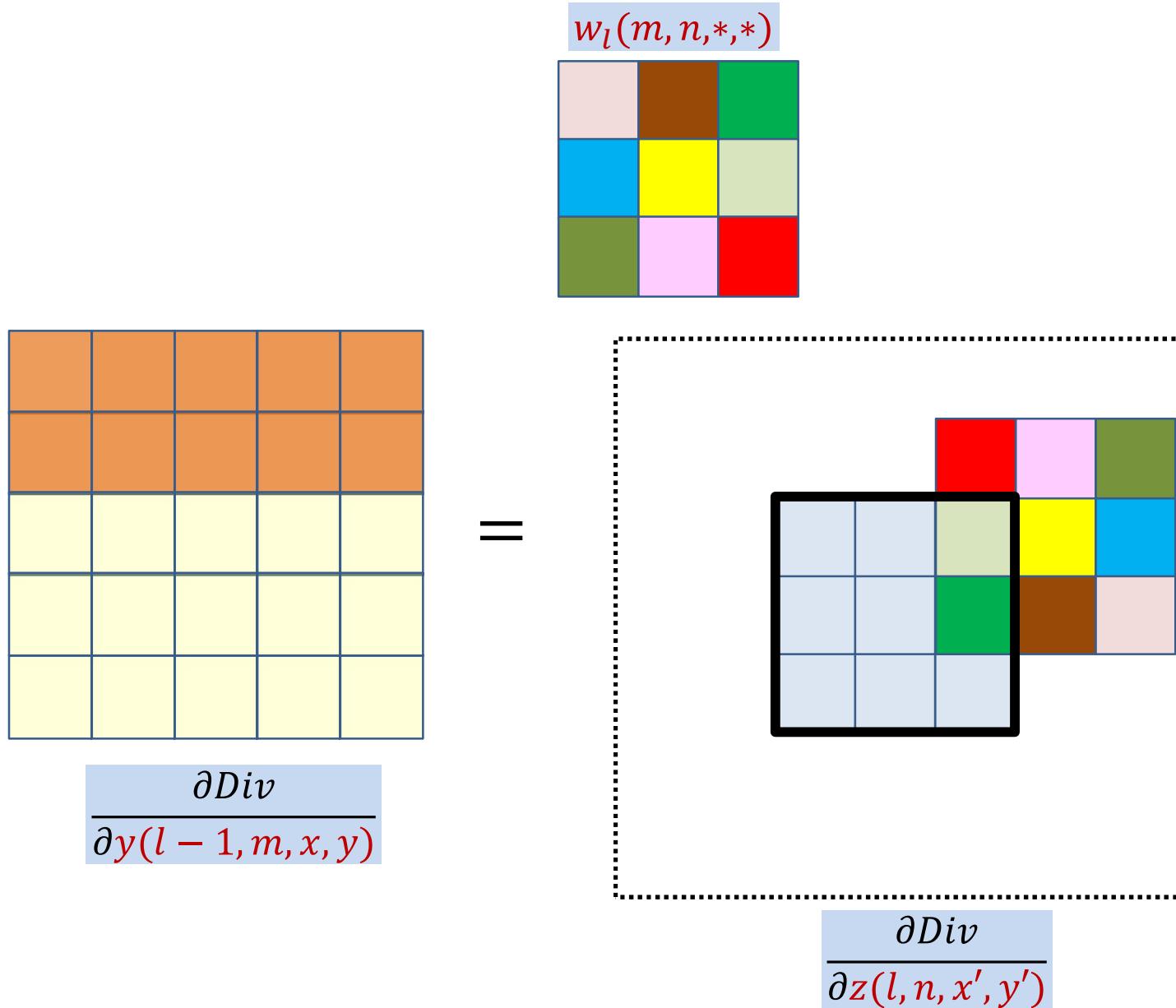
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



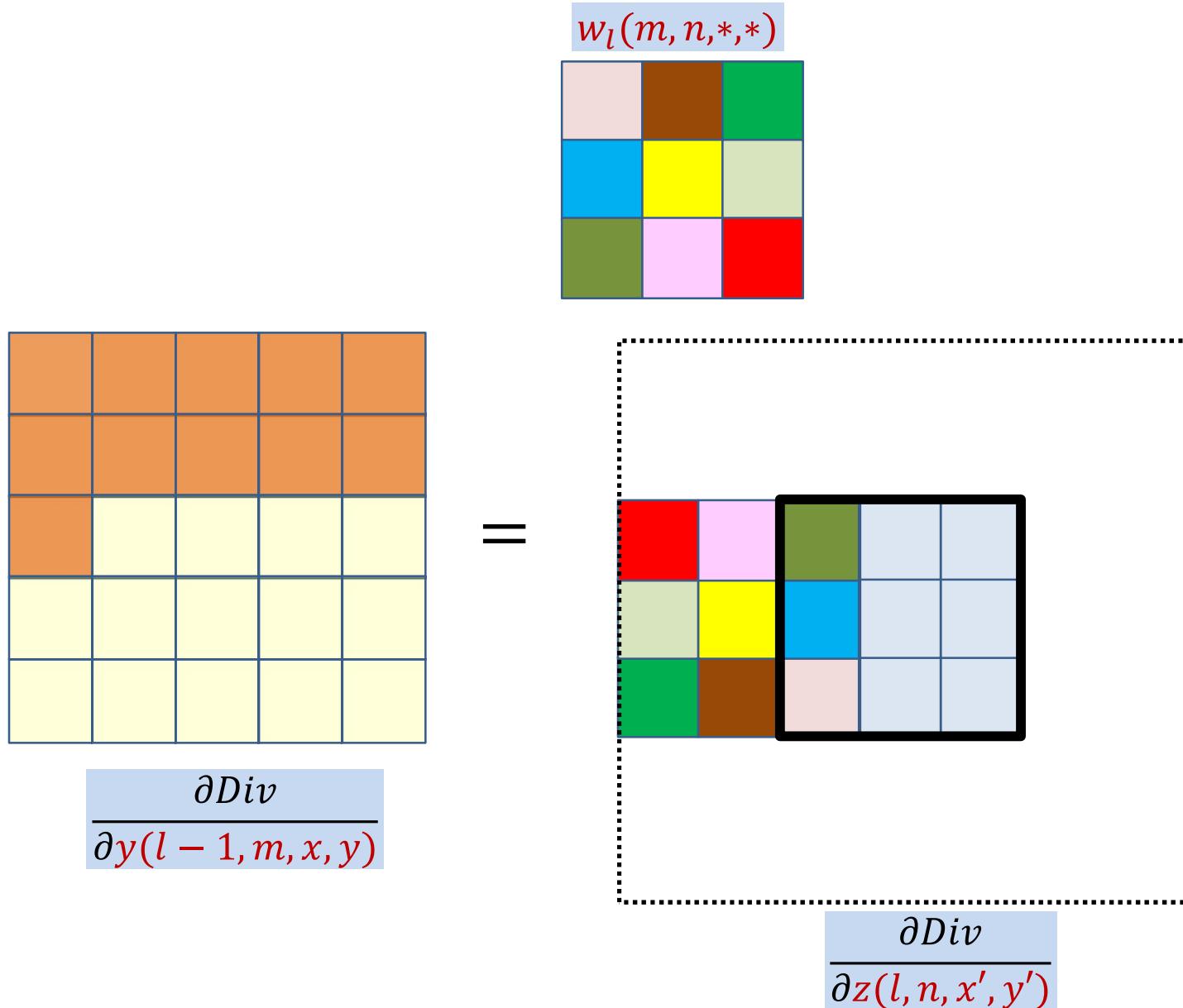
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



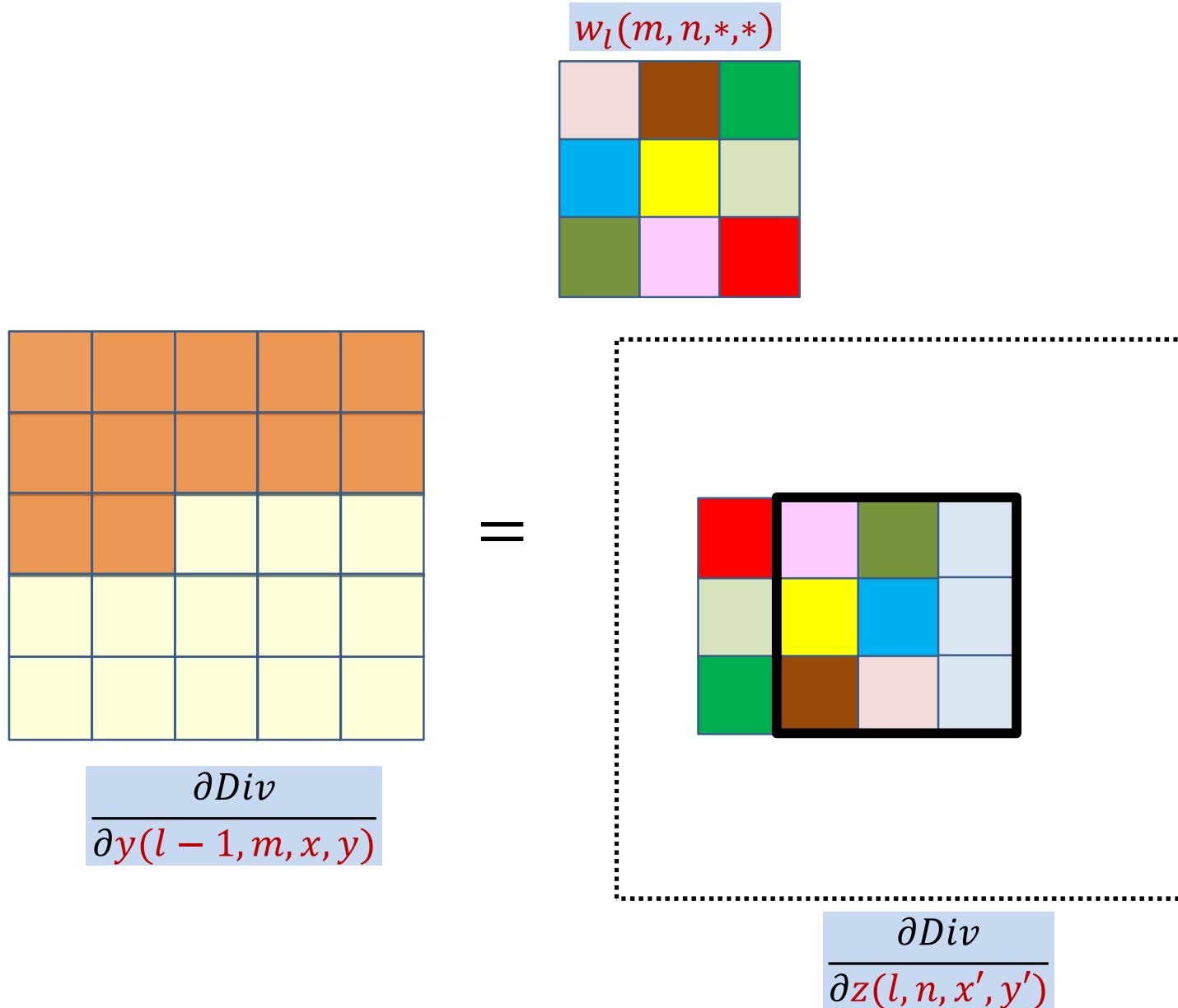
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



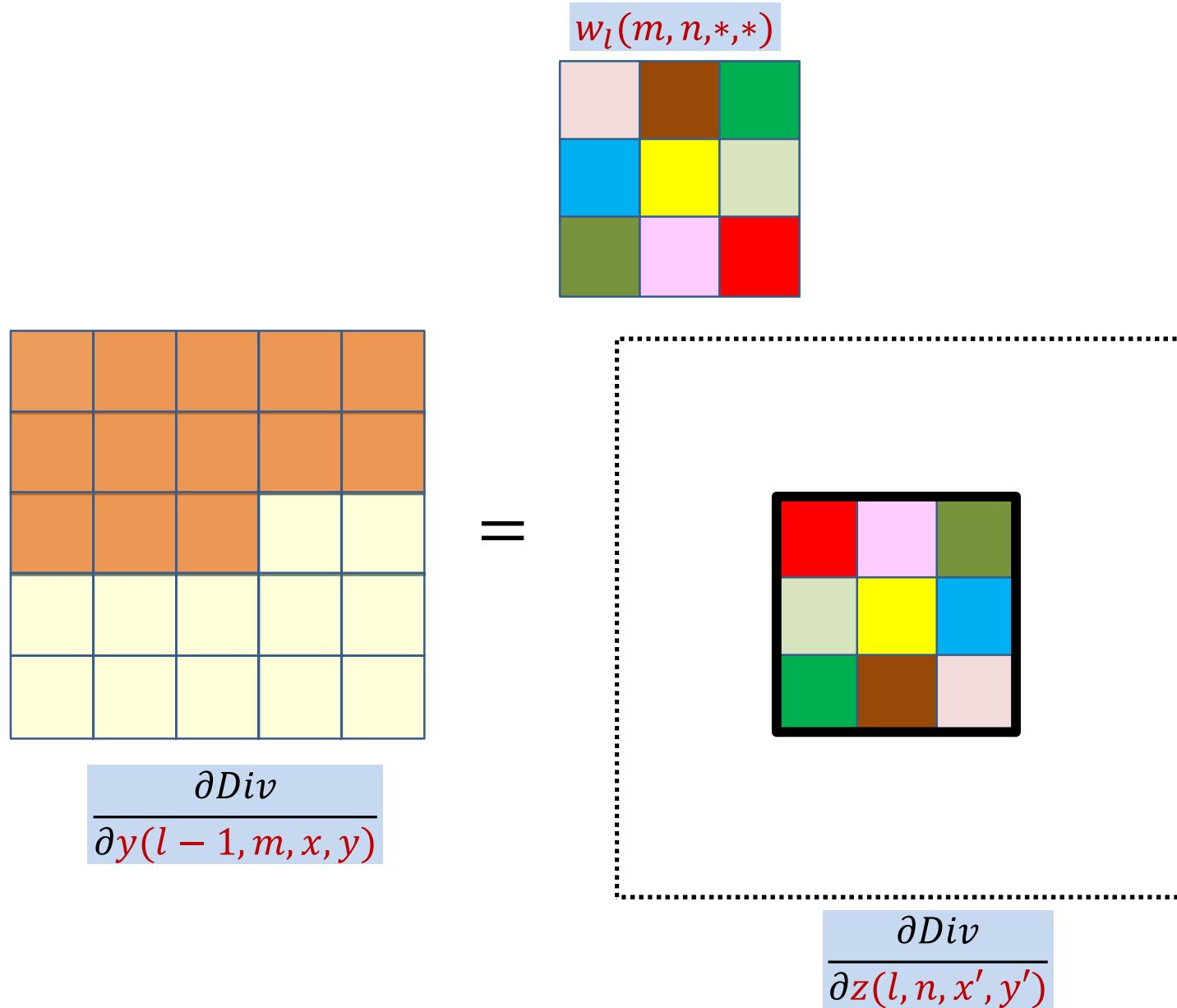
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



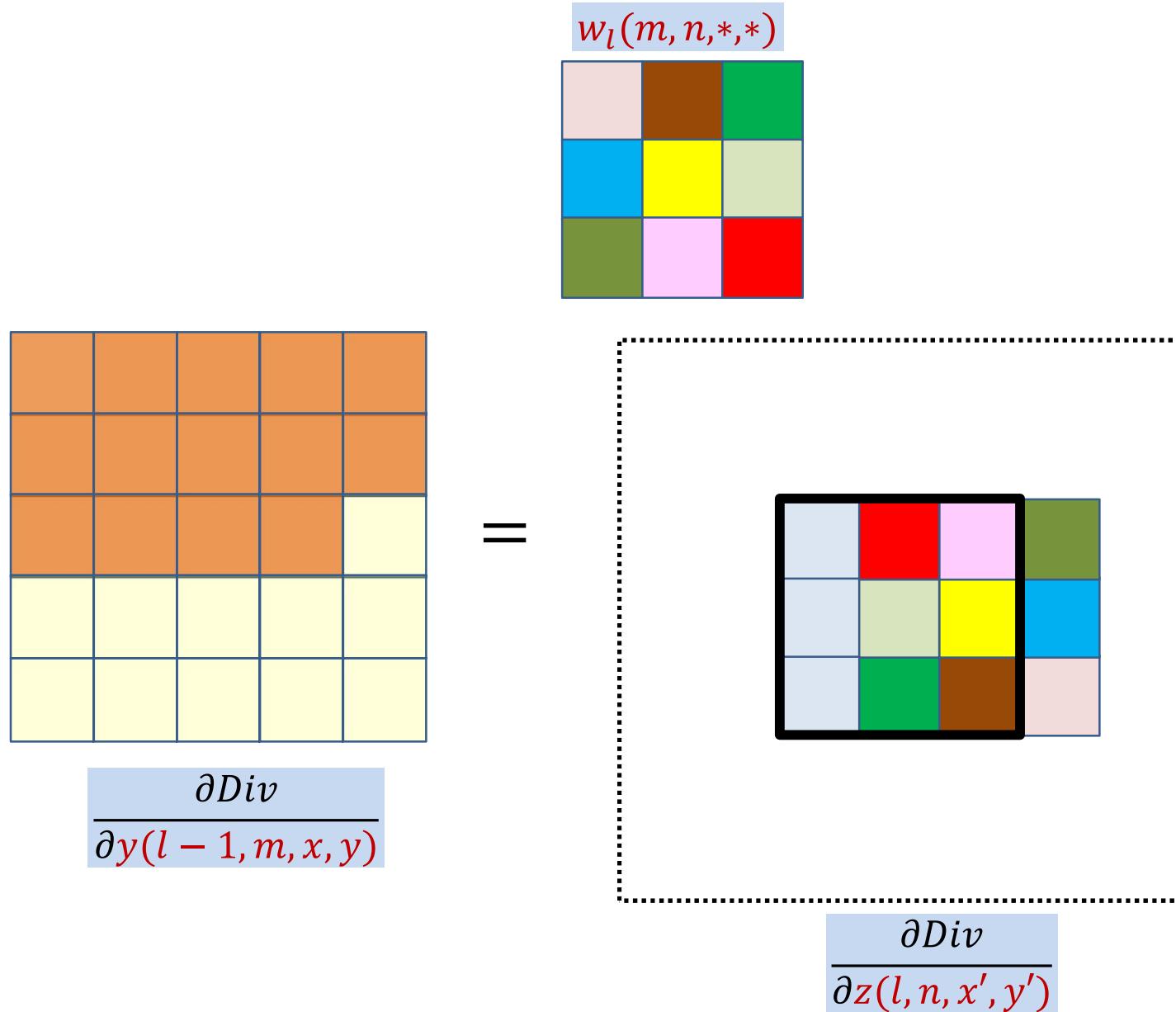
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



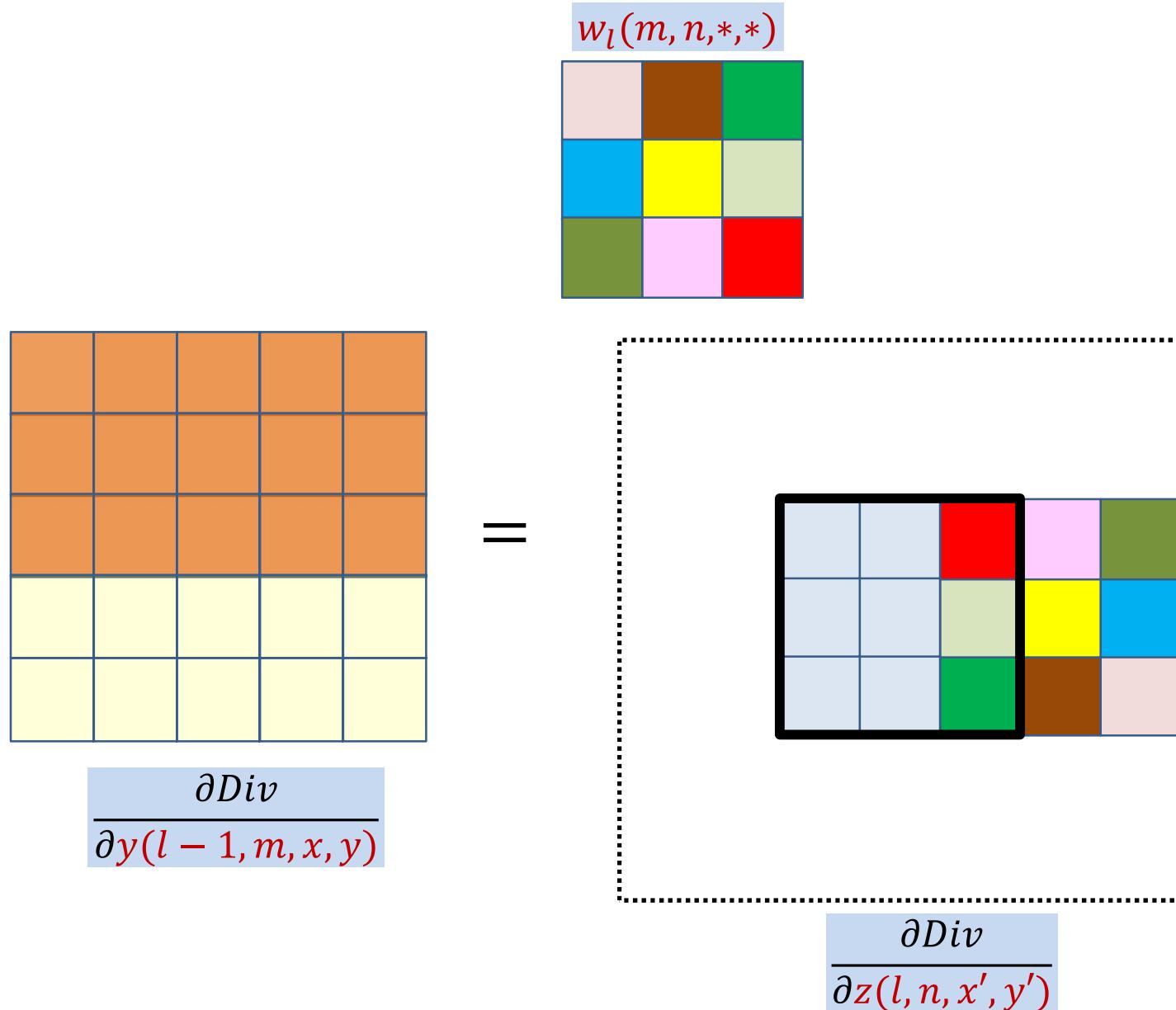
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map

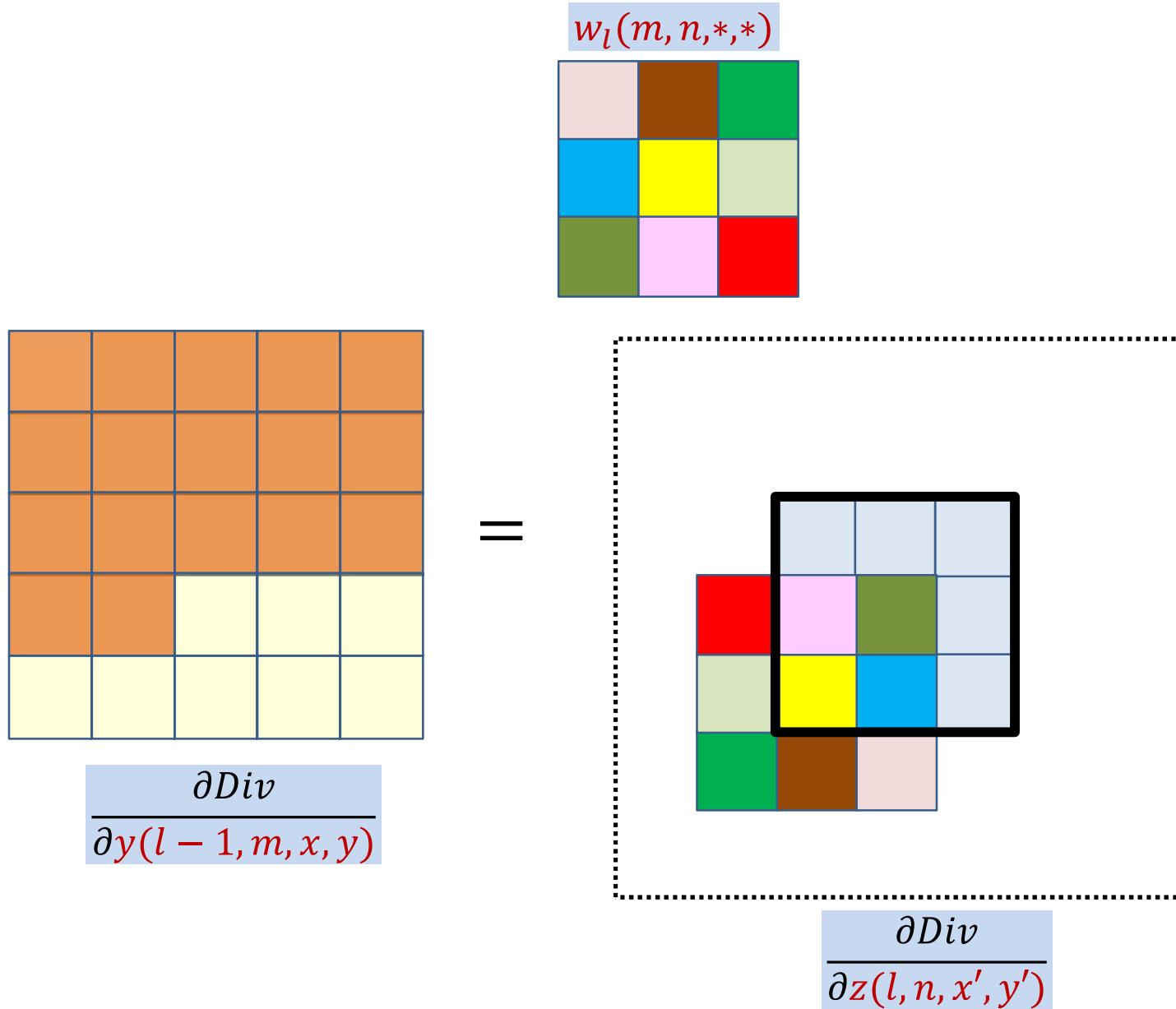


Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map

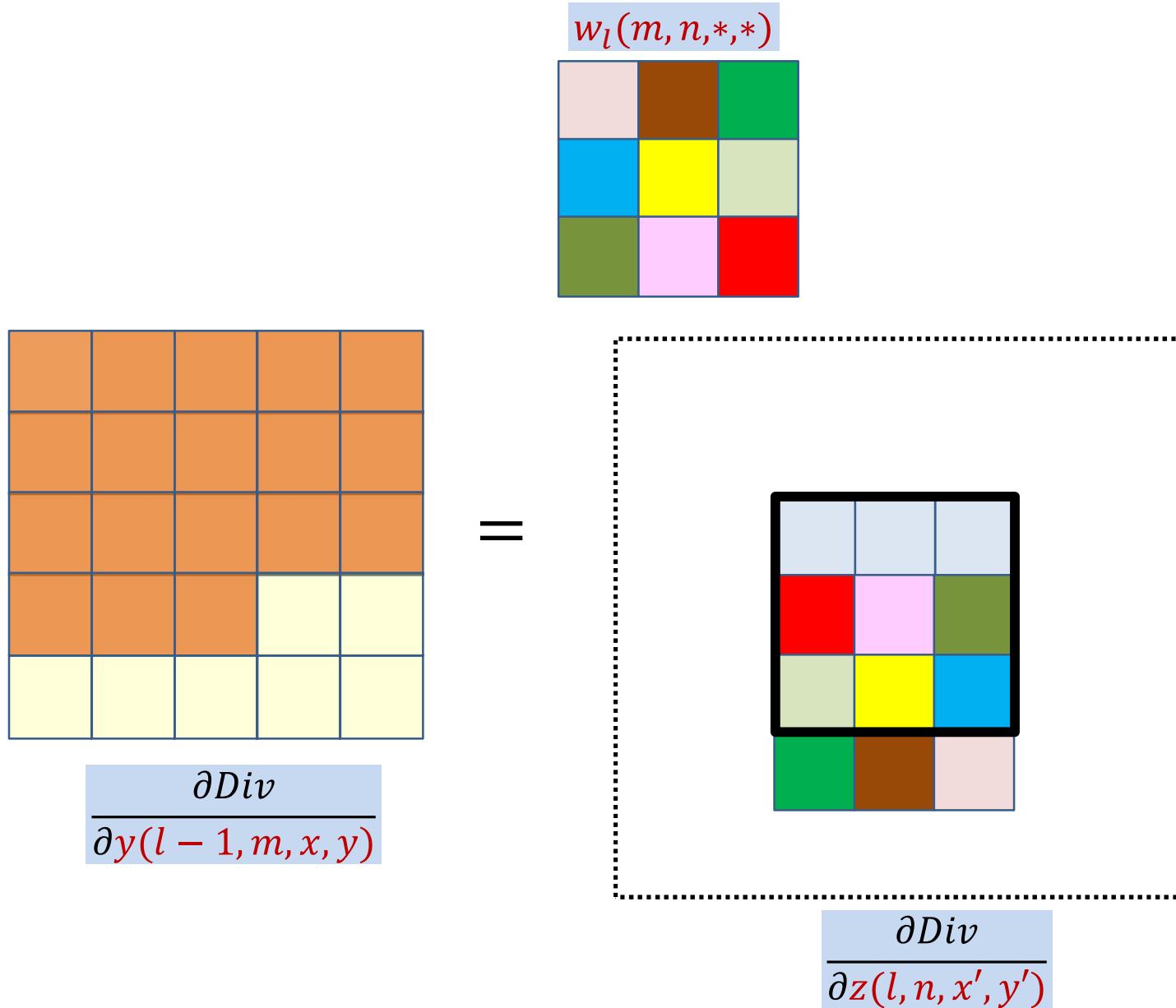
$$\frac{\partial \text{Div}}{\partial y(l - 1, m, x, y)} = w_l(m, n, *, *) \cdot \frac{\partial \text{Div}}{\partial z(l, n, x', y')}$$

The diagram illustrates the computation of the derivative of the divergence operator with respect to a specific element of a grid. On the left, a 5x5 grid of orange squares represents the input field y . A 3x3 subgrid in the bottom-left corner is highlighted in yellow, green, and red, representing the receptive field of the element $y(l-1, m)$. This subgrid is multiplied by a 3x3 weight matrix $w_l(m, n, *, *)$, shown above as a 3x3 grid with colored cells (pink, brown, green; blue, yellow, light green; olive, pink, red). The result is a 3x3 grid where each cell's color is the sum of the products of the corresponding row and column from the weight matrix and the highlighted subgrid. This result is then divided by the derivative of the divergence operator with respect to the element $z(l, n, x', y')$.

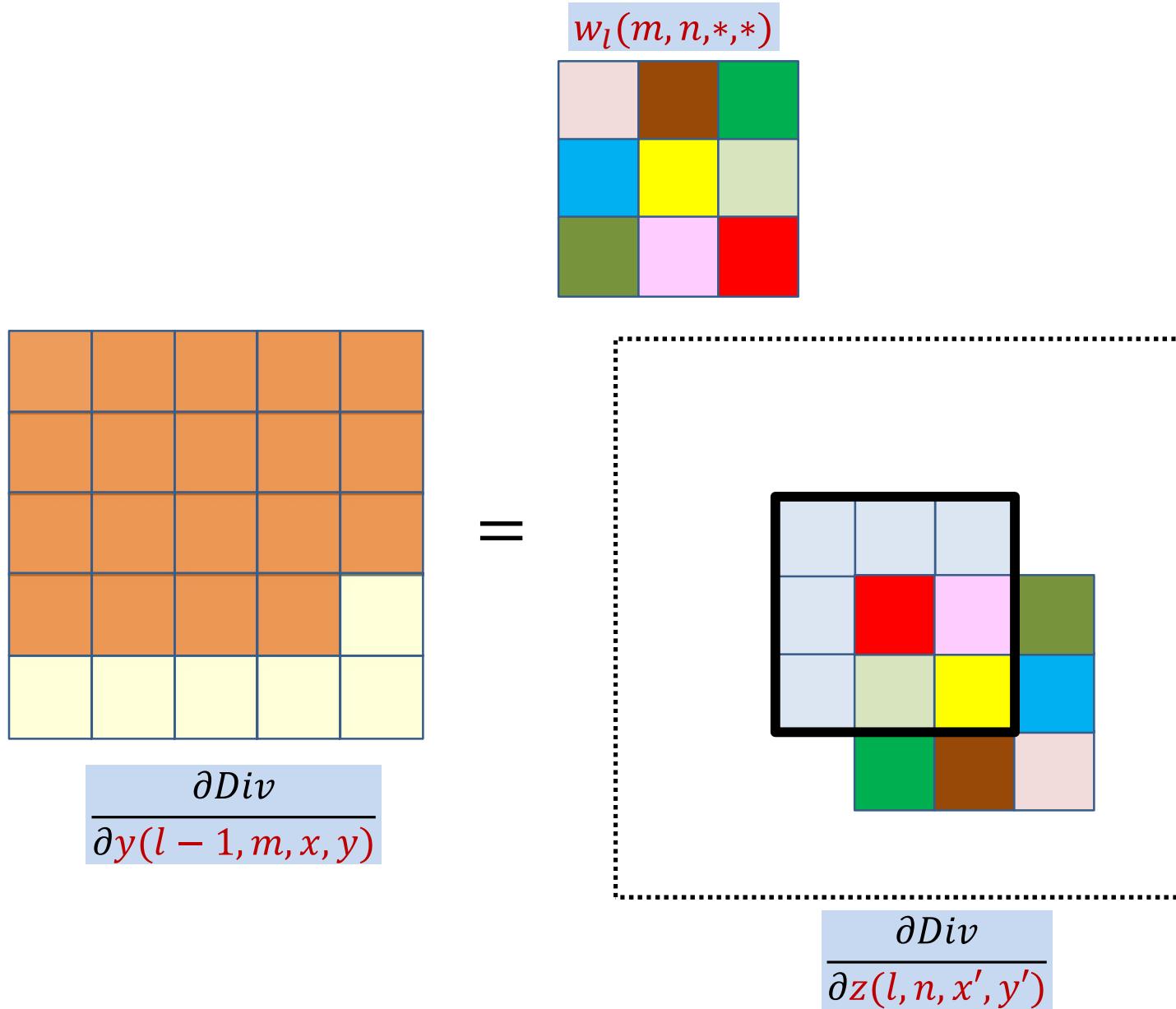
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



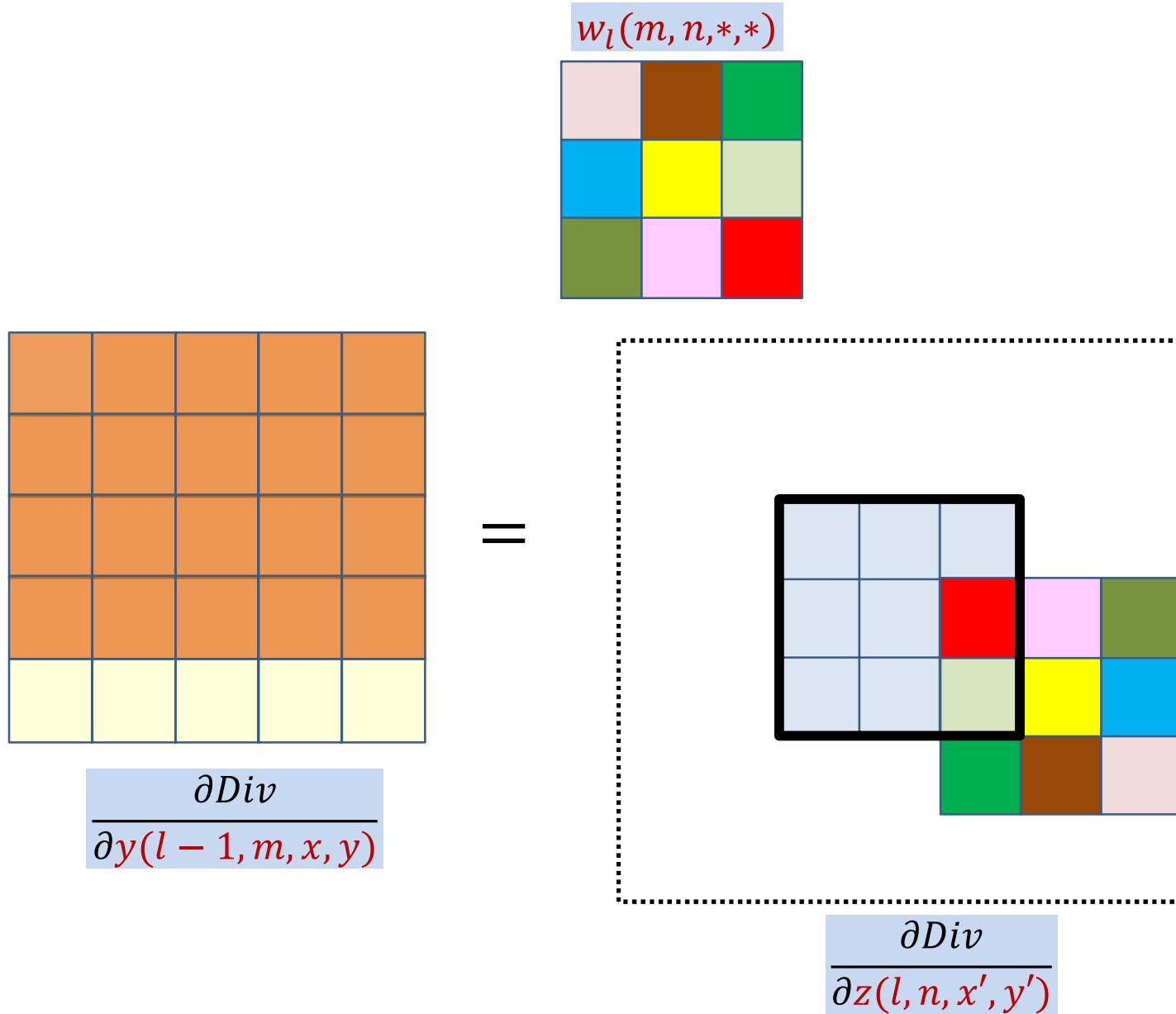
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map

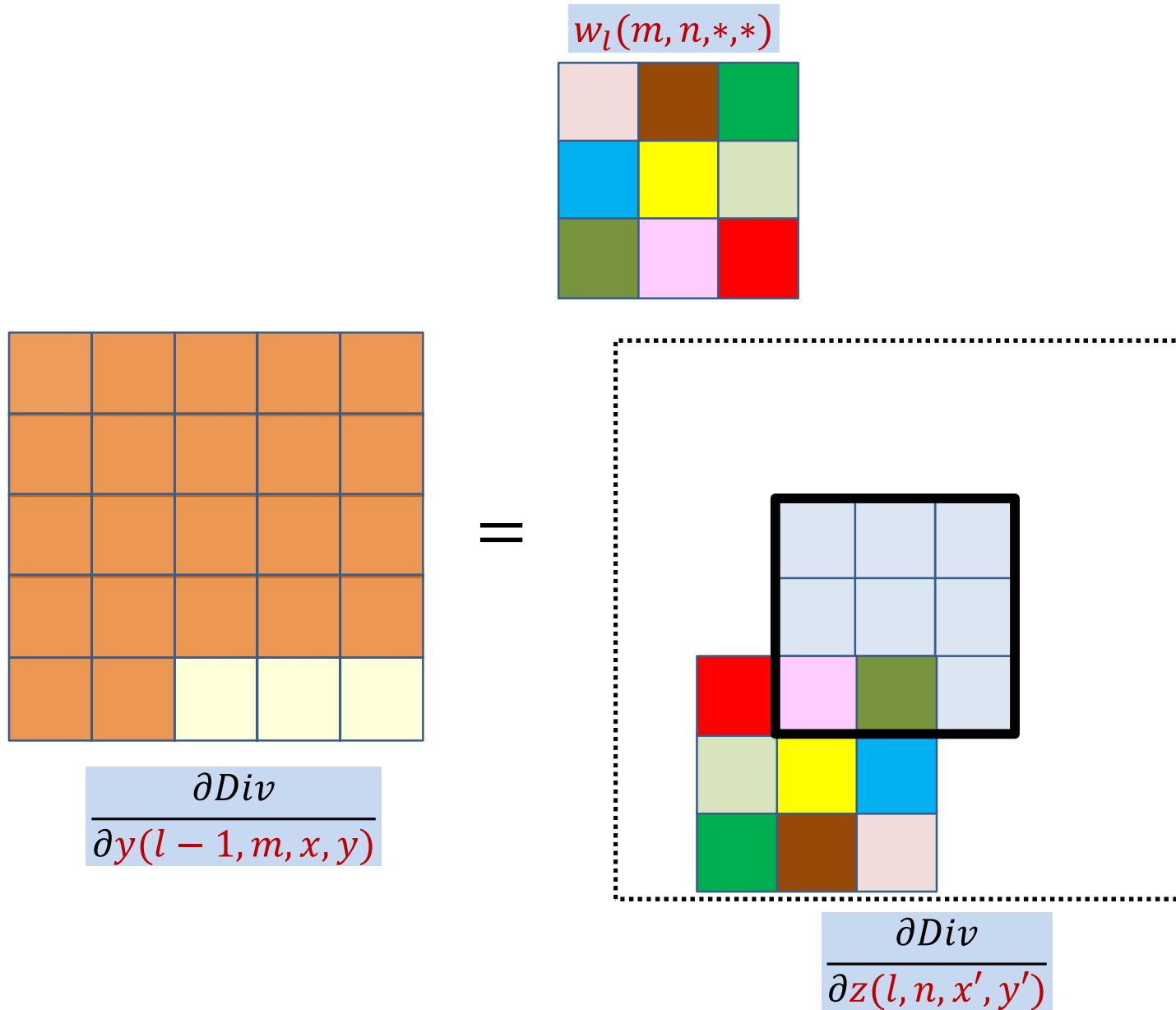
$$\frac{\partial \text{Div}}{\partial y(l - 1, m, x, y)} = w_l(m, n, *, *) \cdot \frac{\partial \text{Div}}{\partial z(l, n, x', y')}$$

The diagram illustrates the computation of the derivative of the divergence operator with respect to a specific element of a matrix. It consists of three main components:

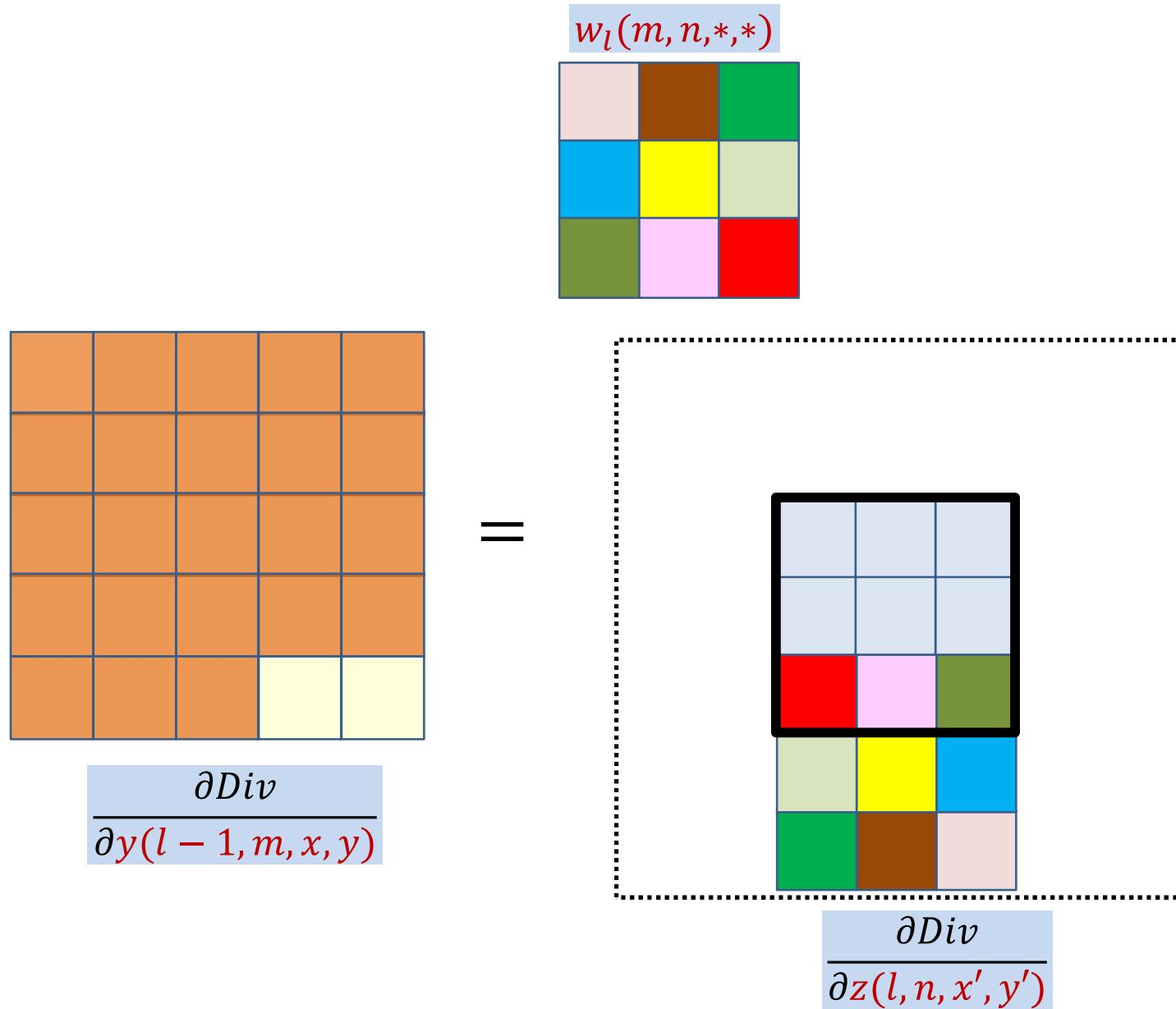
- Input Matrix:** A 5x5 grid of orange squares, representing the input matrix $Y(l - 1, m)$.
- Weight Matrix:** A 3x3 grid of colored squares labeled $w_l(m, n, *, *)$, representing the weight matrix.
- Output Matrix:** A 5x5 grid of colored squares, representing the result of the derivative operation.

The output matrix is formed by multiplying the input matrix by the weight matrix. The result is a 5x5 grid where each element is the sum of products of corresponding elements from the input and weight matrices. The weight matrix is highlighted with a black border, indicating it is the active element being differentiated.

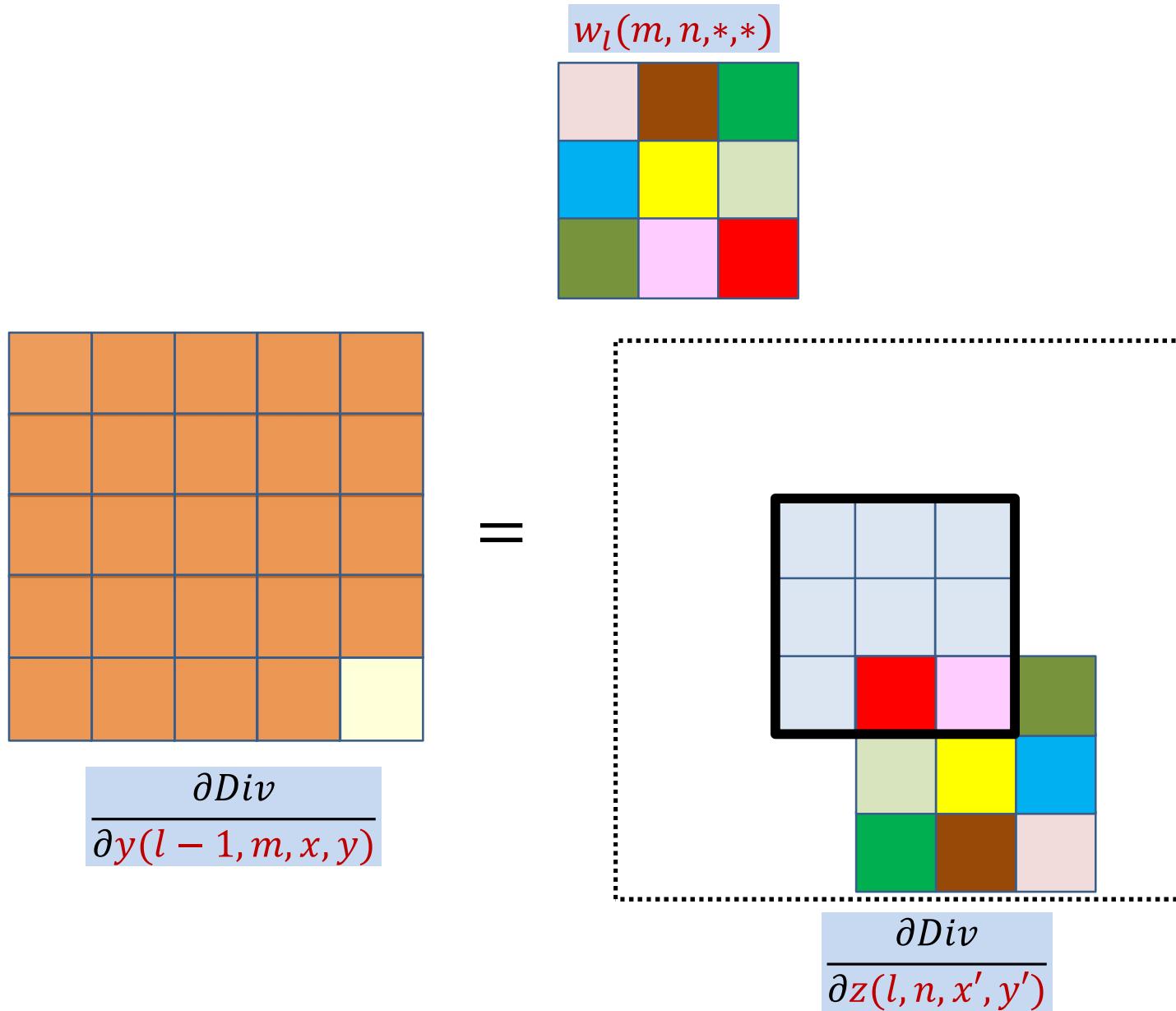
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



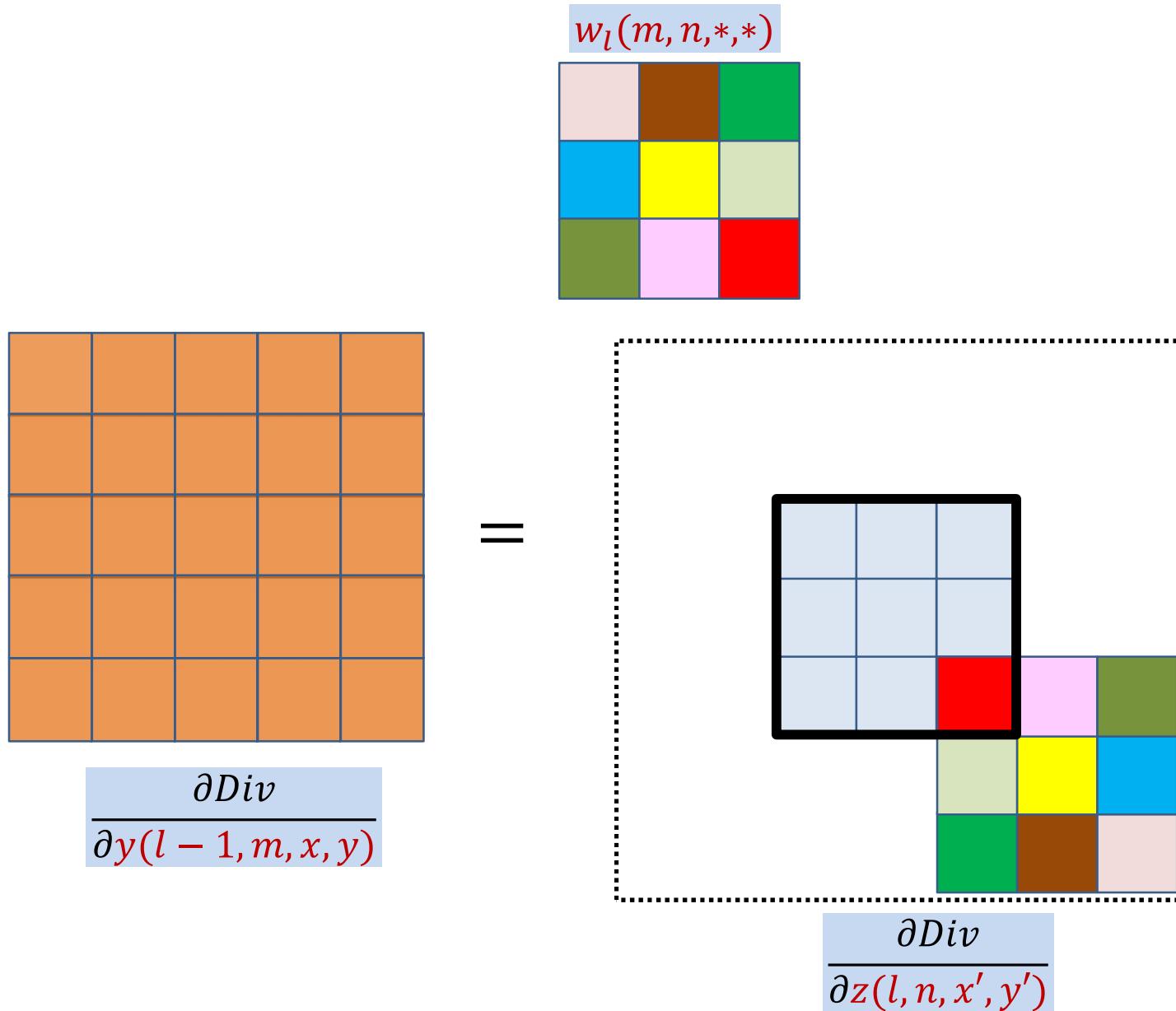
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



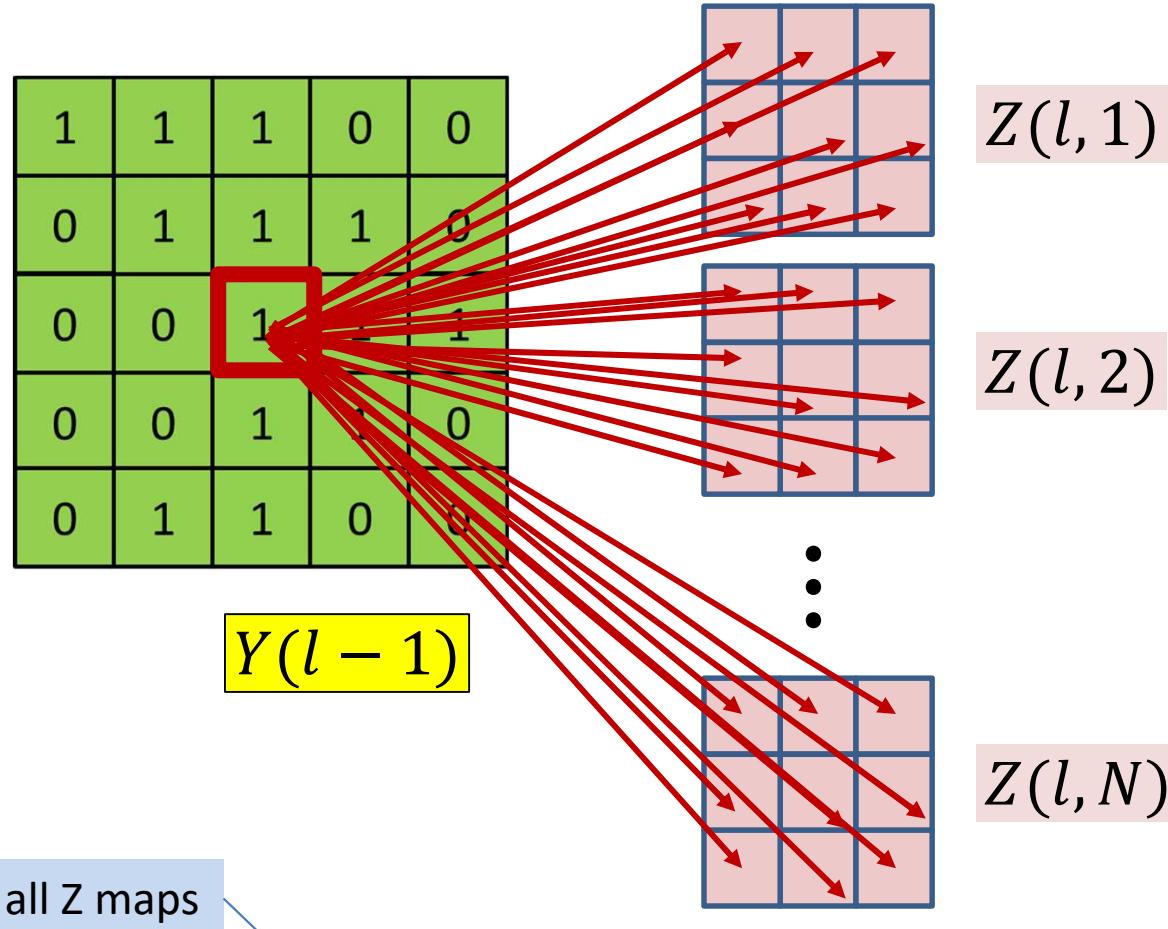
Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map



Derivative at $Y(l - 1, m)$ from a single $Z(l, n)$ map

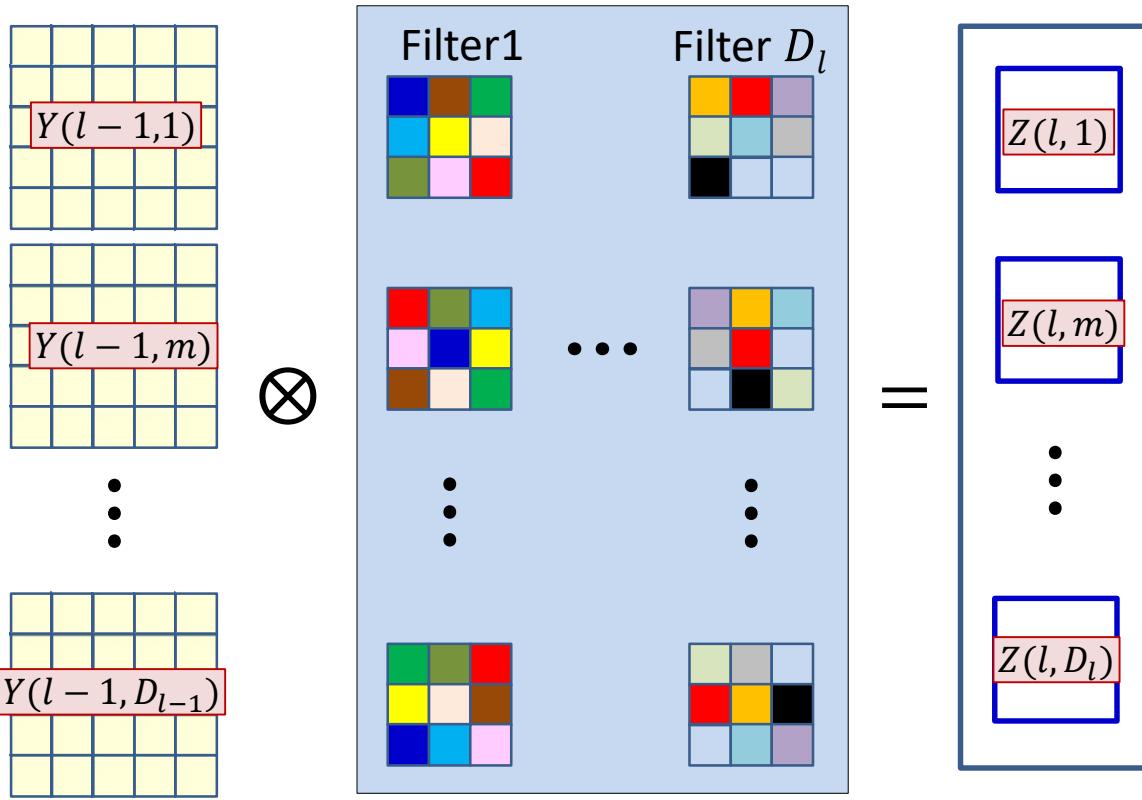


BP: Convolutional layer



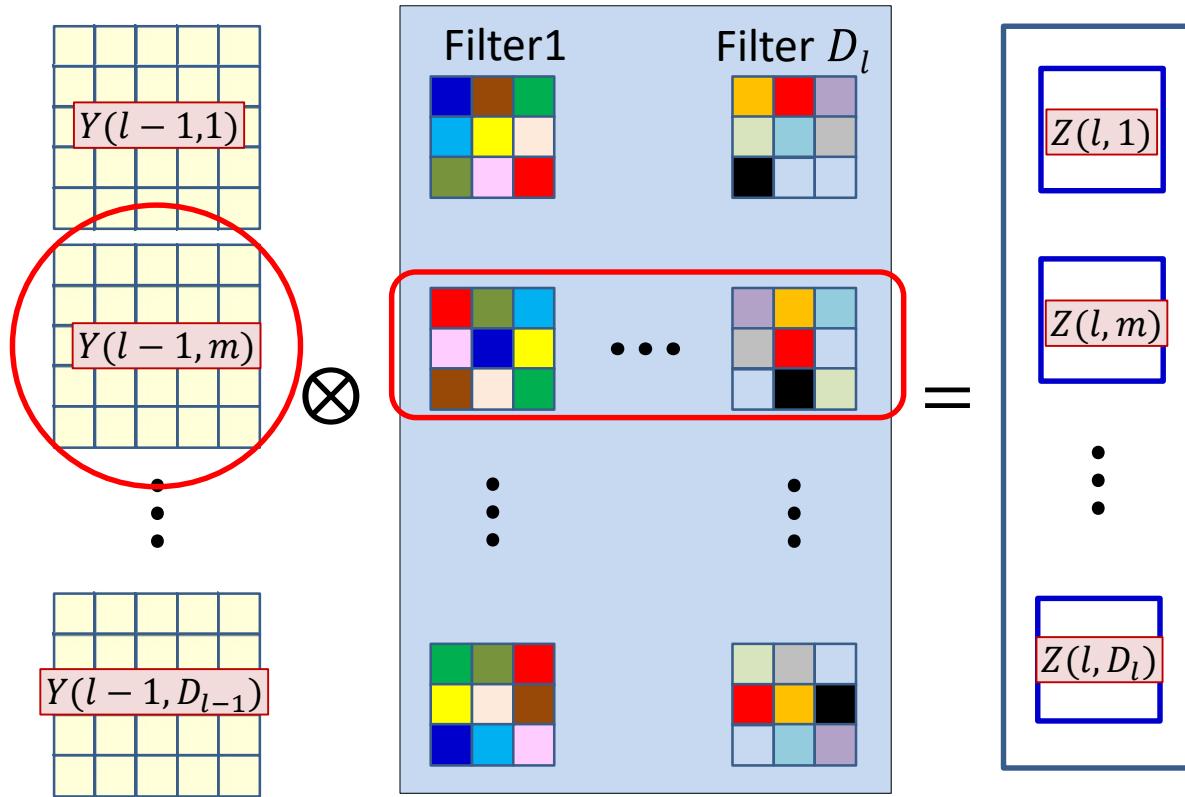
$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

The actual convolutions



- The D_l affine maps are produced by convolving with D_l filters

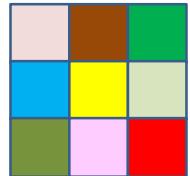
The actual convolutions



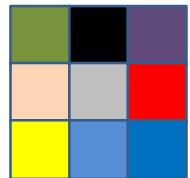
- The D_l affine maps are produced by convolving with D_l filters
- The m^{th} Y map always convolves the m^{th} plane of the filters
- The derivative for the m^{th} Y map will invoke the m^{th} plane of *all* the filters

$$w_l(m, n, x, y)$$

$$n = 1$$

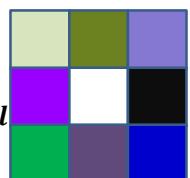


$$n = 2$$



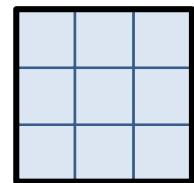
⋮

$$n = D_l$$

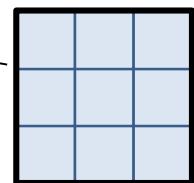


In reality, the derivative at each (x,y) location is obtained from *all* z maps

$$\frac{\partial \text{Div}}{\partial z(l, n, x', y')}$$

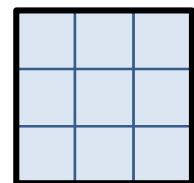


=



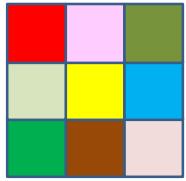
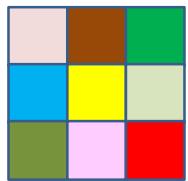
⋮

$$\frac{\partial \text{Div}}{\partial y(l-1, m, x, y)}$$

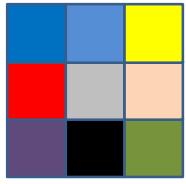
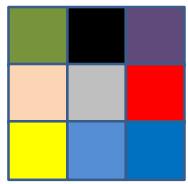


$$w_l(m, n, x, y)$$

$n = 1$



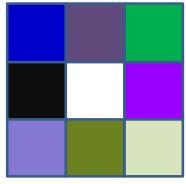
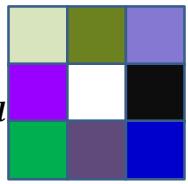
$n = 2$



⋮

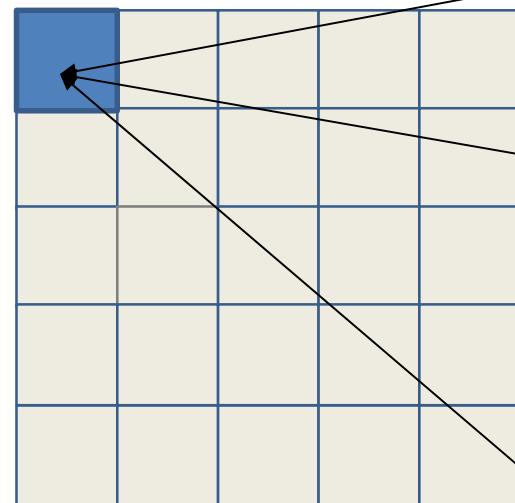
⋮

$n = D_l$

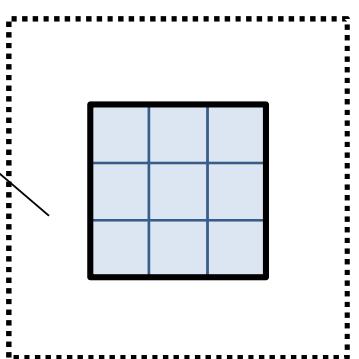
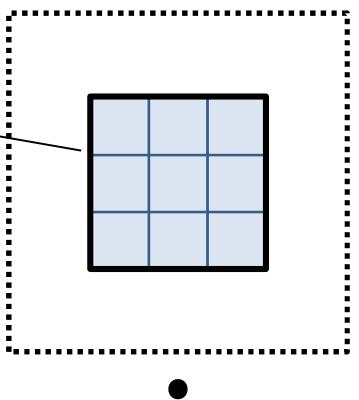
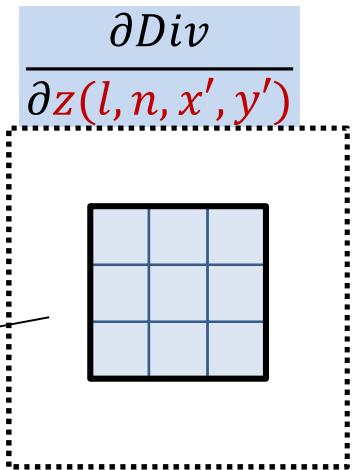


$$w_l(m, n, K + 1 - x, K + 1 - y)$$

In reality, the derivative at each (x, y) location is obtained from *all* z maps

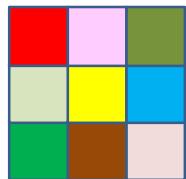
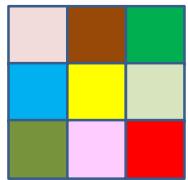


=

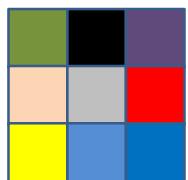


$w_l(m, n, x, y)$

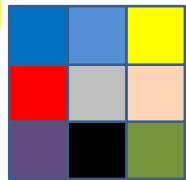
$n = 1$



$n = 2$



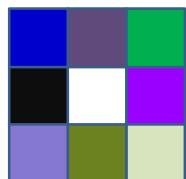
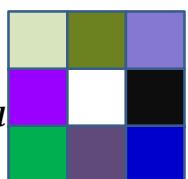
flip



⋮

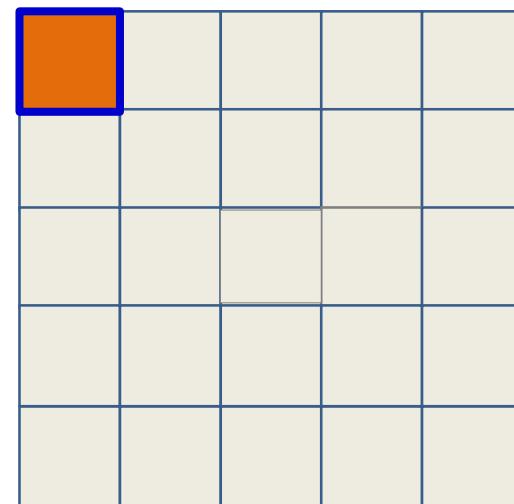
⋮

$n = D_l$

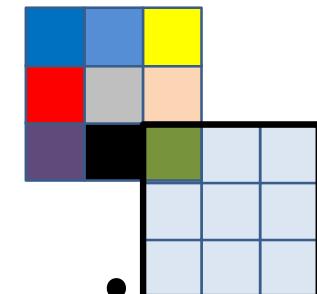
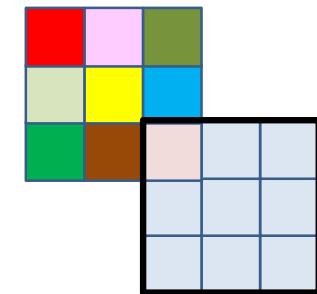


$w_l(m, n, K + 1 - x, K + 1 - y)$

$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

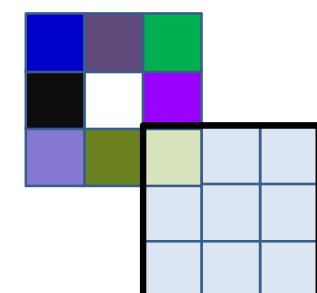


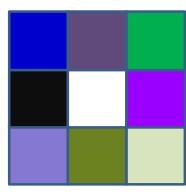
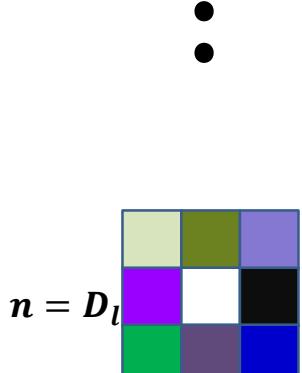
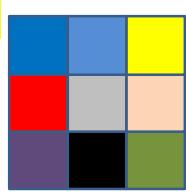
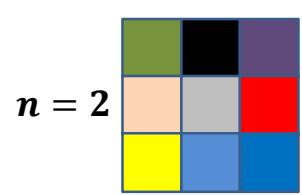
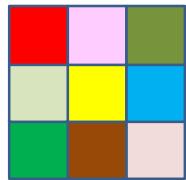
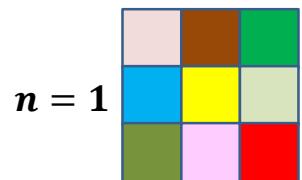
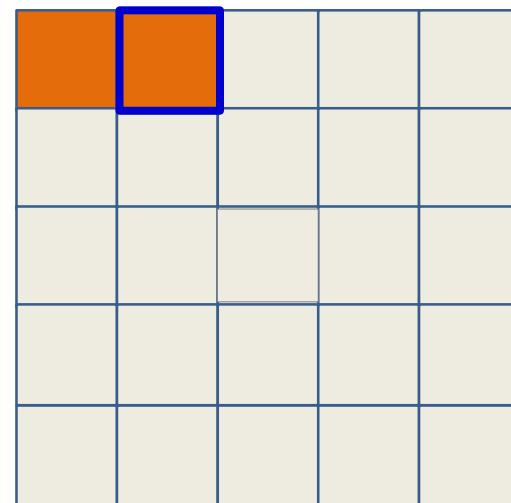
=



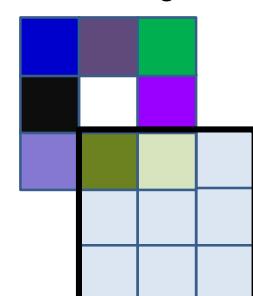
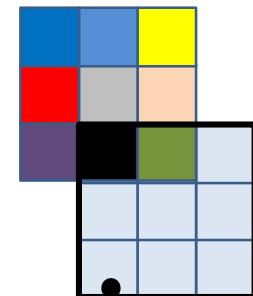
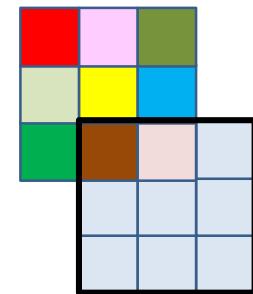
⋮

⋮



$w_l(m, n, x, y)$  $w_l(m, n, K + 1 - x, K + 1 - y)$ 

=

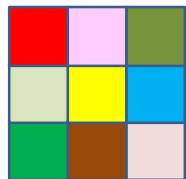
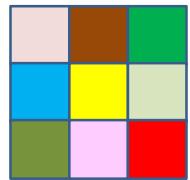


$$\frac{\partial \text{Div}}{\partial y(l-1, m, x, y)}$$

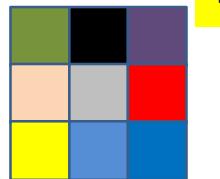
$$\frac{d\text{Div}}{dY(l-1, m, x, y)} = \sum_n \sum_{x',y'} \frac{d\text{Div}}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

$w_l(m, n, x, y)$

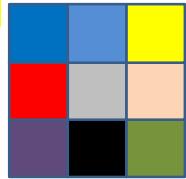
$n = 1$



$n = 2$



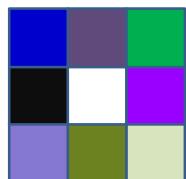
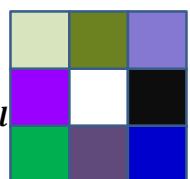
flip



⋮

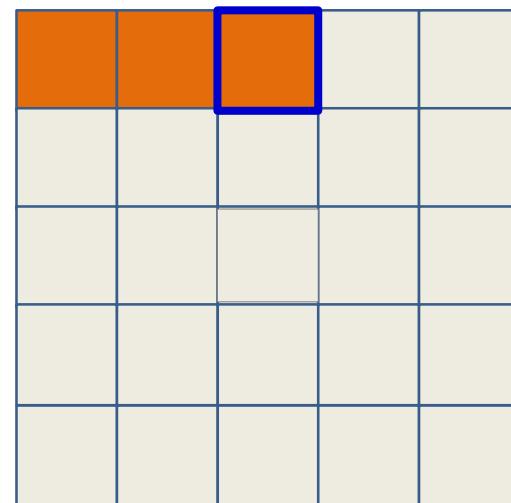
⋮

$n = D_l$

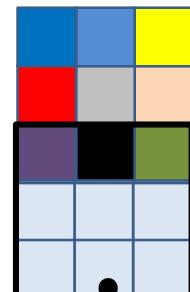
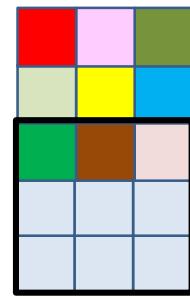


$w_l(m, n, K + 1 - x, K + 1 - y)$

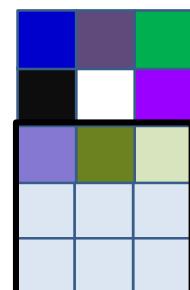
$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$



=

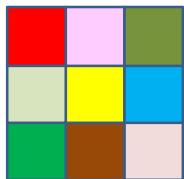
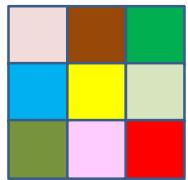


⋮

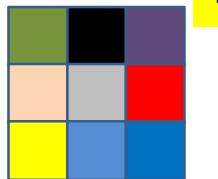


$w_l(m, n, x, y)$

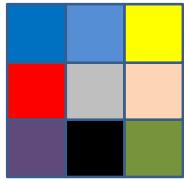
$n = 1$



$n = 2$



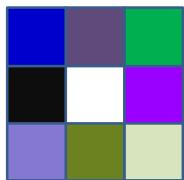
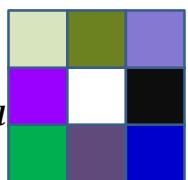
flip



⋮

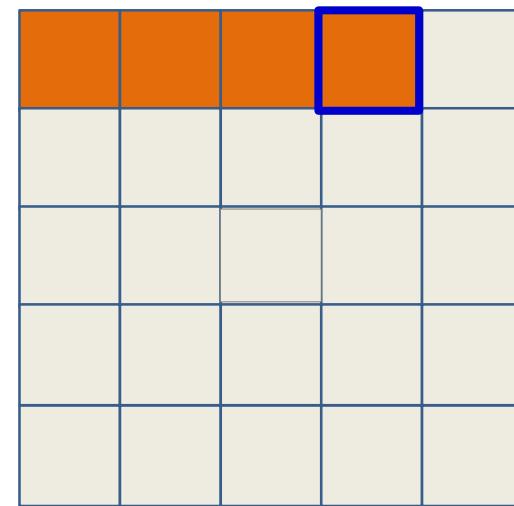
⋮

$n = D_l$

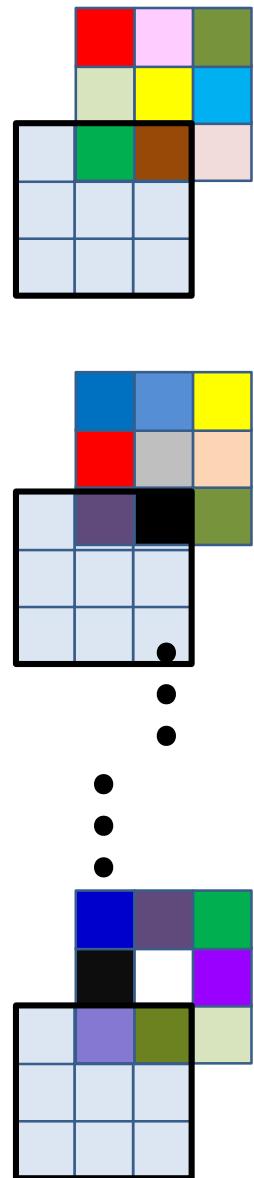


$w_l(m, n, K + 1 - x, K + 1 - y)$

$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

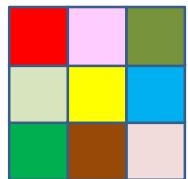
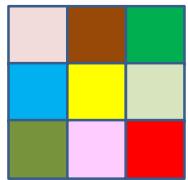


=

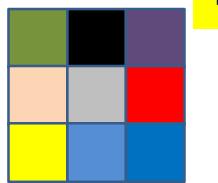


$w_l(m, n, x, y)$

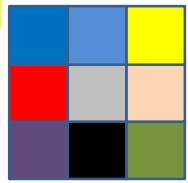
$n = 1$



$n = 2$



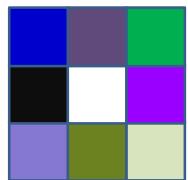
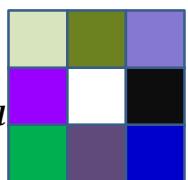
flip



⋮

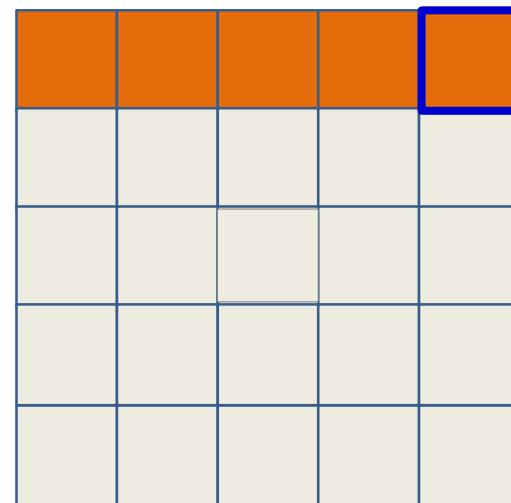
⋮

$n = D_l$

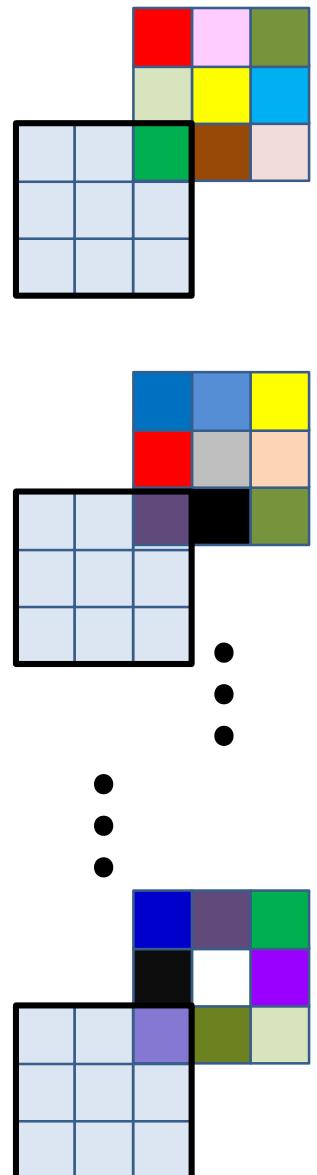


$w_l(m, n, K + 1 - x, K + 1 - y)$

$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

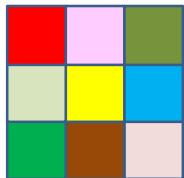
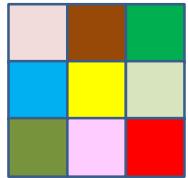


=

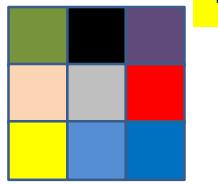


$w_l(m, n, x, y)$

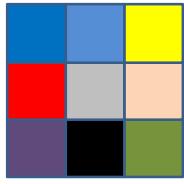
$n = 1$



$n = 2$



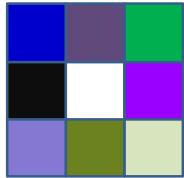
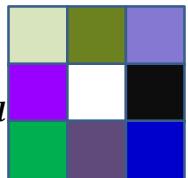
flip



⋮

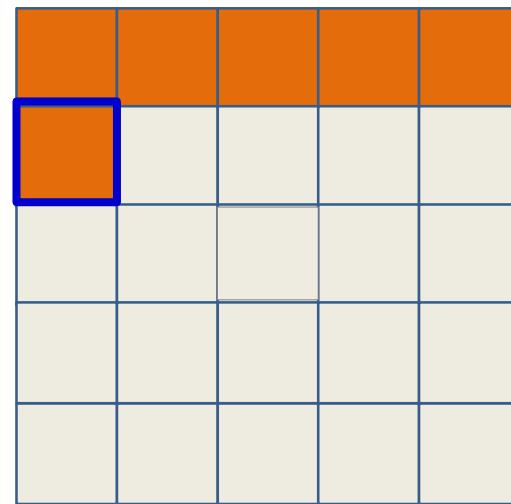
⋮

$n = D_l$

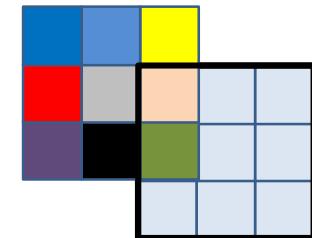
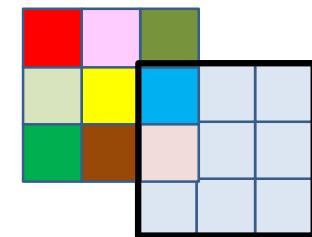


$w_l(m, n, K + 1 - x, K + 1 - y)$

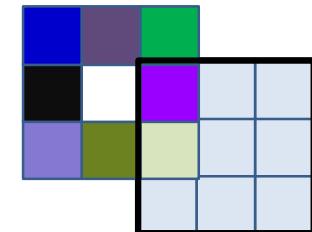
$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

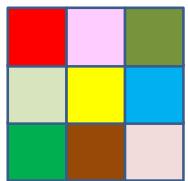
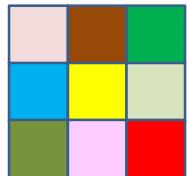
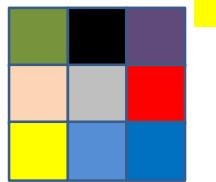


=

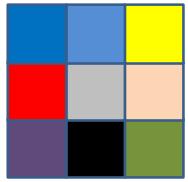


⋮
⋮
⋮
⋮



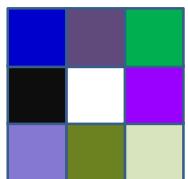
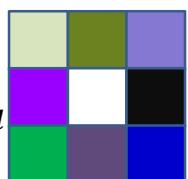
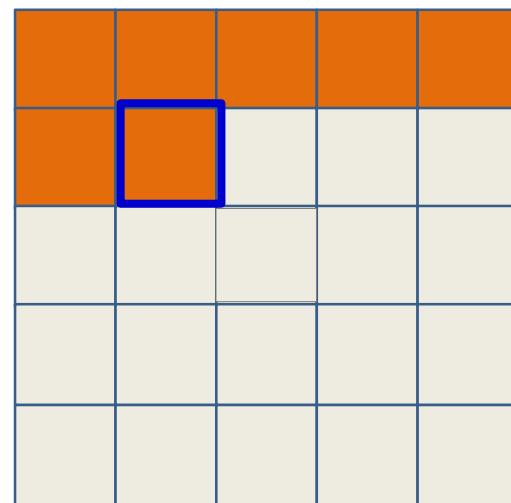
$w_l(m, n, x, y)$ $n = 1$  $n = 2$ 

flip

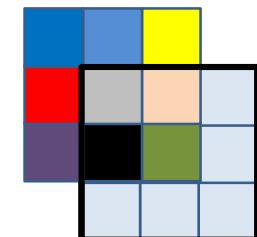
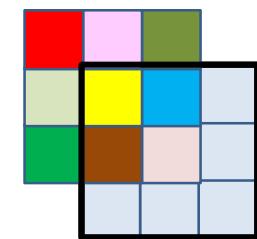


⋮

⋮

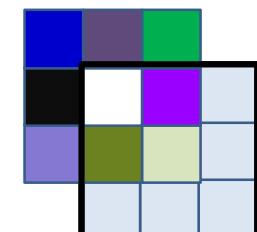
 $n = D_l$  $w_l(m, n, K + 1 - x, K + 1 - y)$ 

=



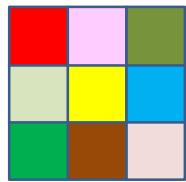
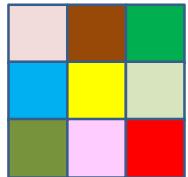
⋮

$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

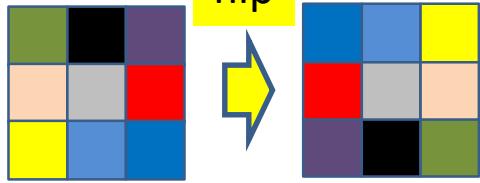


$w_l(m, n, x, y)$

$n = 1$



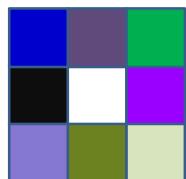
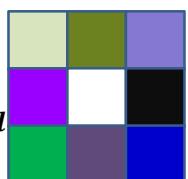
$n = 2$



⋮

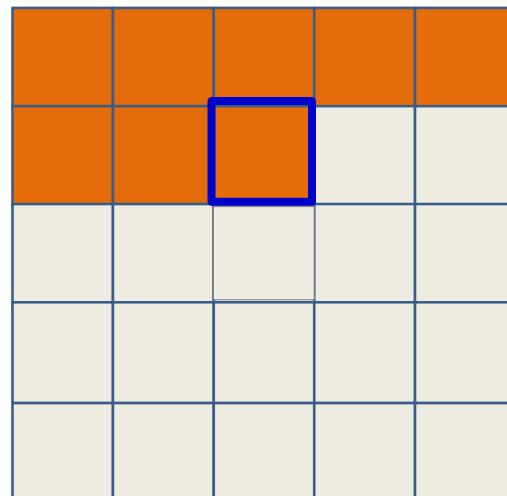
⋮

$n = D_l$

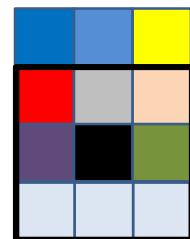
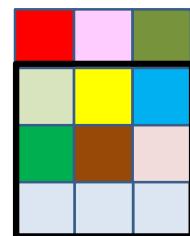


$w_l(m, n, K + 1 - x, K + 1 - y)$

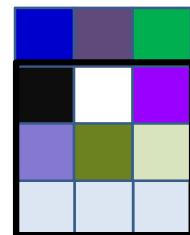
$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

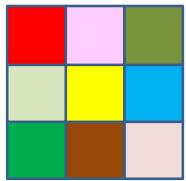
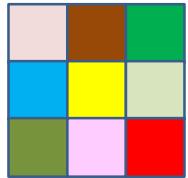
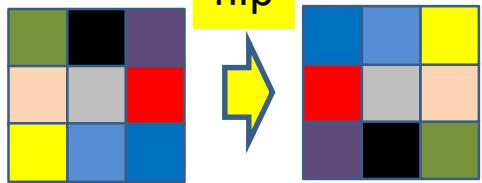
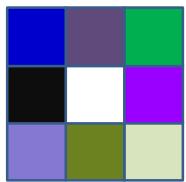
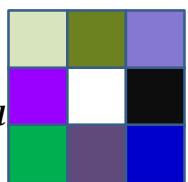


=

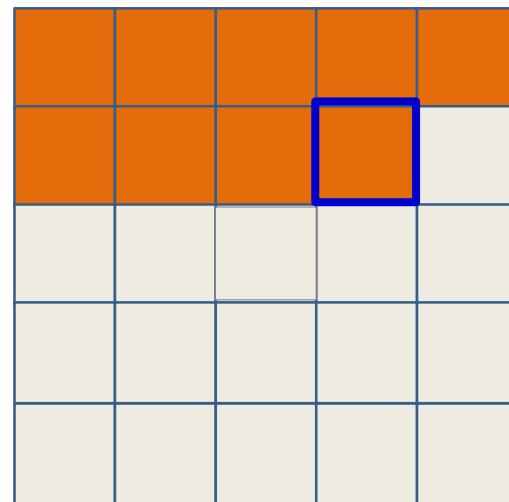
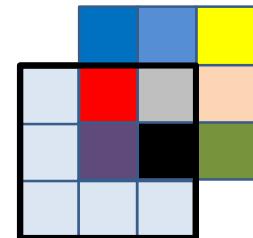
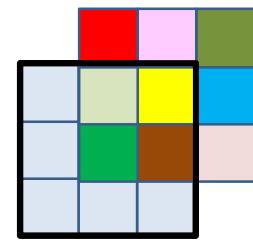
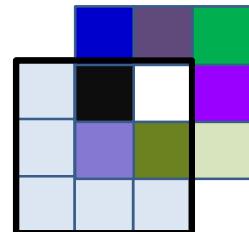


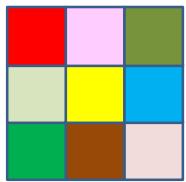
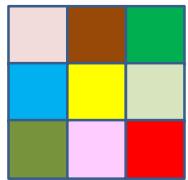
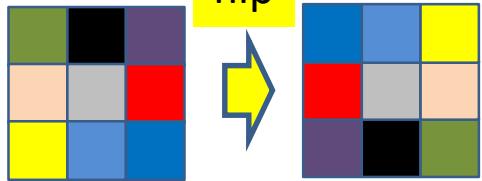
⋮



$w_l(m, n, x, y)$ $n = 1$  $n = 2$  \vdots \vdots $n = D_l$  $w_l(m, n, K + 1 - x, K + 1 - y)$

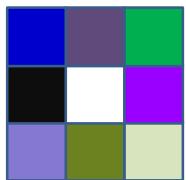
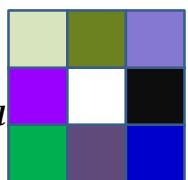
$$\frac{dDiv}{dy(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

 $=$  \vdots 

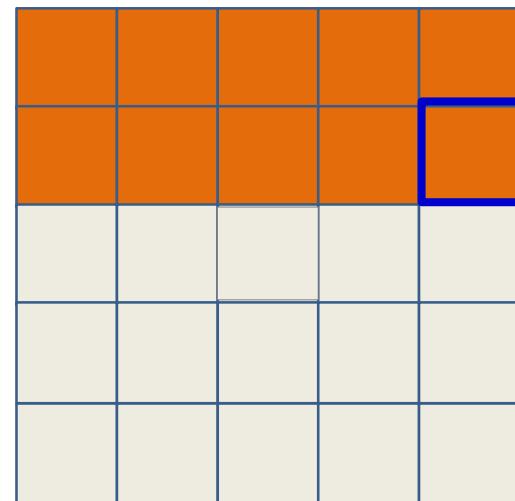
$w_l(m, n, x, y)$ $n = 1$  $n = 2$ 

⋮

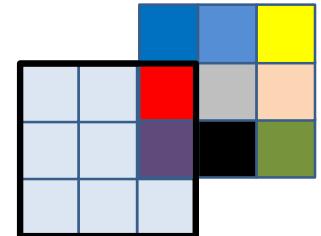
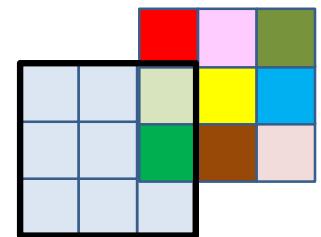
⋮

 $n = D_l$  $w_l(m, n, K + 1 - x, K + 1 - y)$

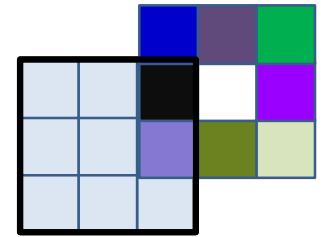
$$\frac{dDiv}{dy(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

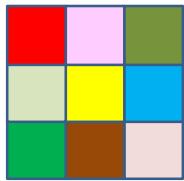
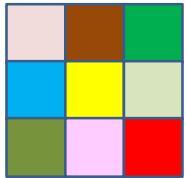
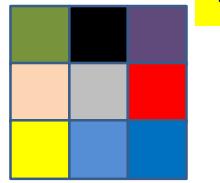


=

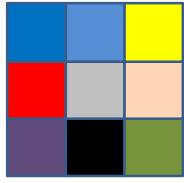
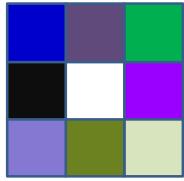
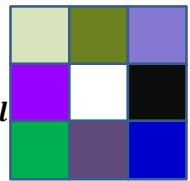


⋮
⋮
⋮

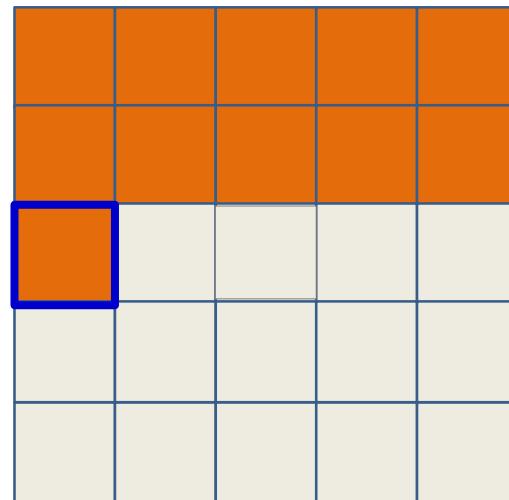


$w_l(m, n, x, y)$ $n = 1$  $n = 2$ 

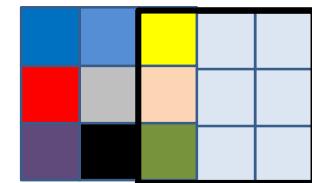
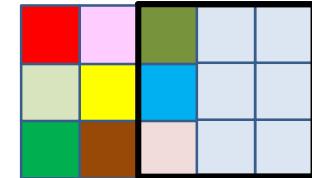
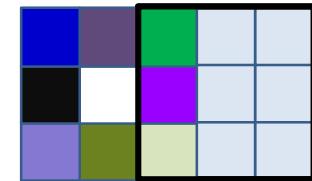
flip

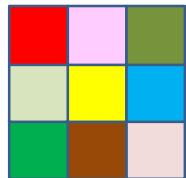
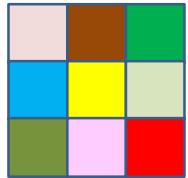
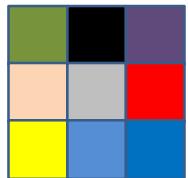
 \vdots \vdots $n = D_l$  $w_l(m, n, K + 1 - x, K + 1 - y)$

$$\frac{dDiv}{dy(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

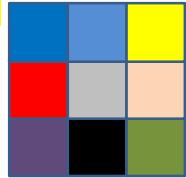
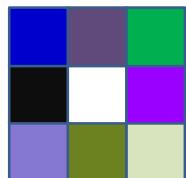
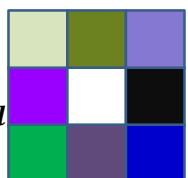
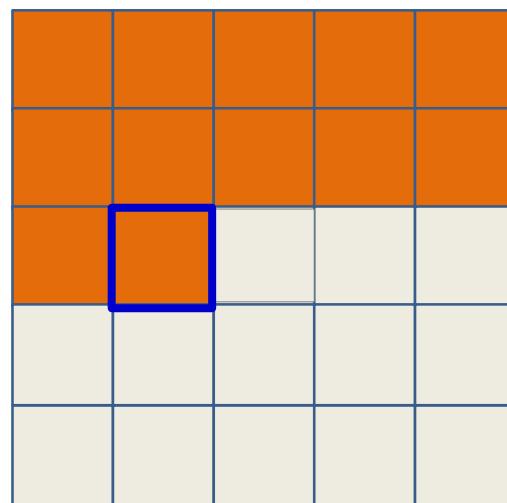


=

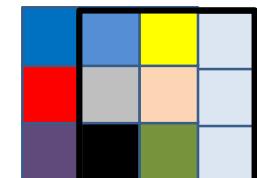
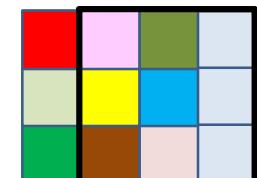
 \vdots 

$w_l(m, n, x, y)$ $n = 1$  $n = 2$ 

flip

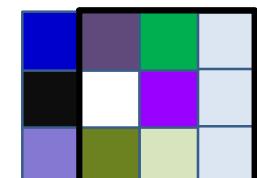
 \vdots \vdots $n = D_l$  $w_l(m, n, K + 1 - x, K + 1 - y)$ 

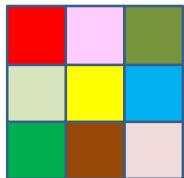
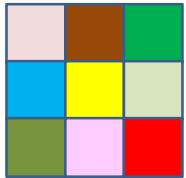
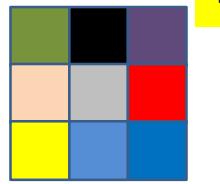
=

 \vdots
 \vdots
 \vdots

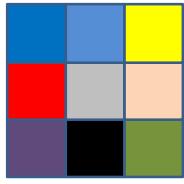
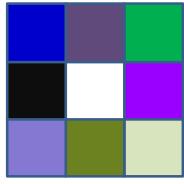
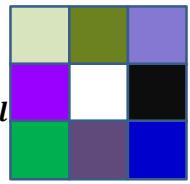
$$\frac{\partial \text{Div}}{\partial y(l-1, m, x, y)}$$

$$d\text{Div} = \sum_n \sum_{x', y'} \frac{d\text{Div}}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

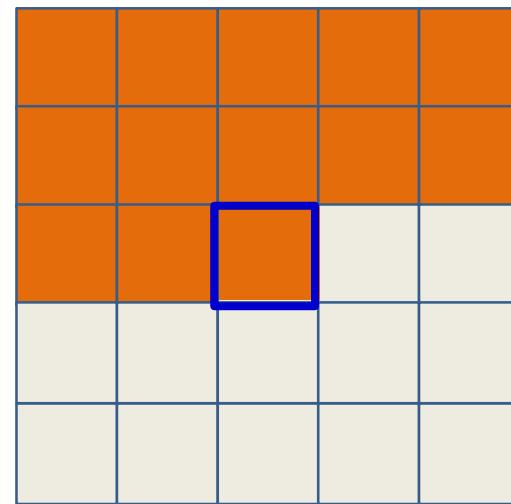


$w_l(m, n, x, y)$ $n = 1$  $n = 2$ 

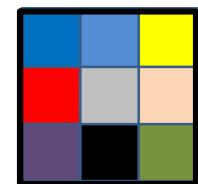
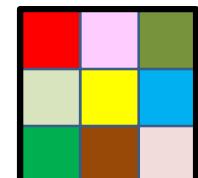
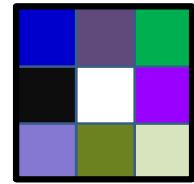
flip

 \vdots \vdots $n = D_l$  $w_l(m, n, K + 1 - x, K + 1 - y)$

$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

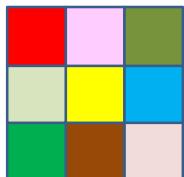
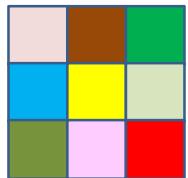


=

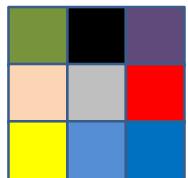
 \vdots 

$w_l(m, n, x, y)$

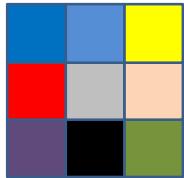
$n = 1$



$n = 2$



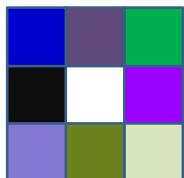
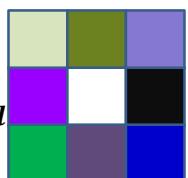
flip



⋮

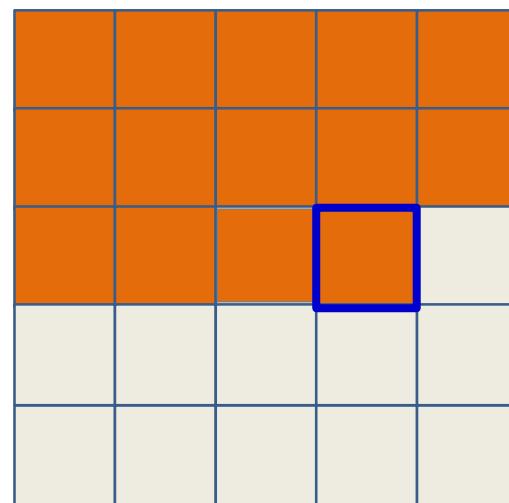
⋮

$n = D_l$

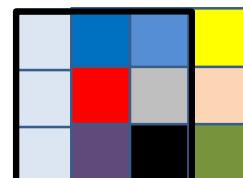
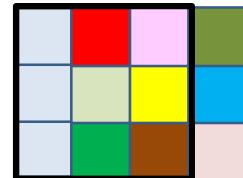


$w_l(m, n, K + 1 - x, K + 1 - y)$

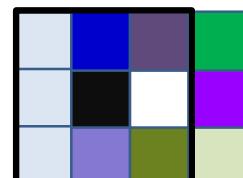
$$\frac{dDiv}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$



=

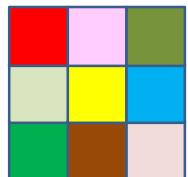
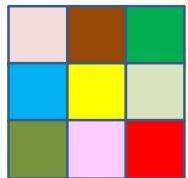


⋮

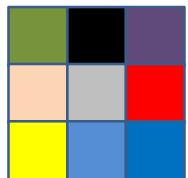


$w_l(m, n, x, y)$

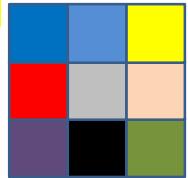
$n = 1$



$n = 2$



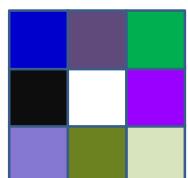
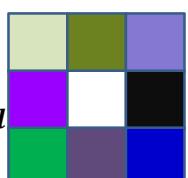
flip



⋮

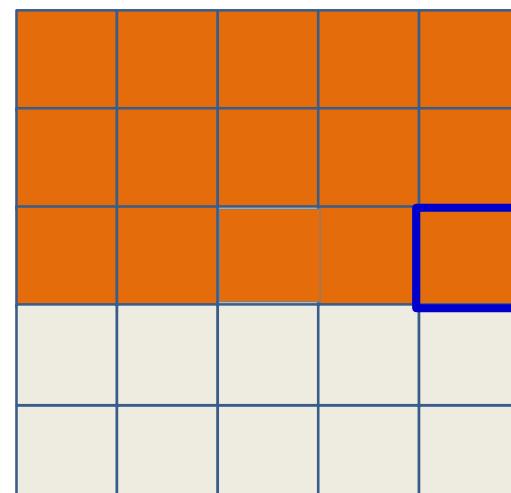
⋮

$n = D_l$

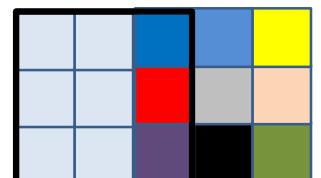
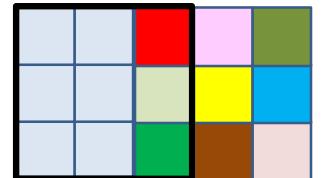


$w_l(m, n, K + 1 - x, K + 1 - y)$

$$dDiv = \sum_n \sum_{x',y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

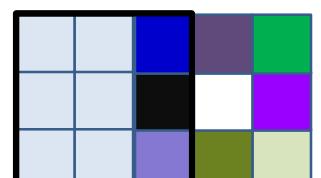


=

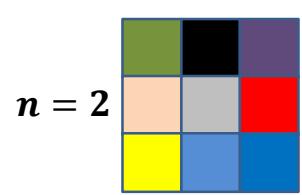
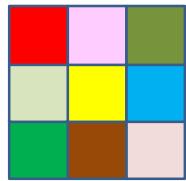
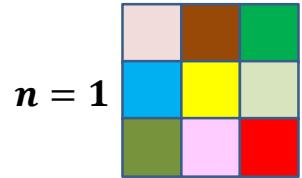


⋮

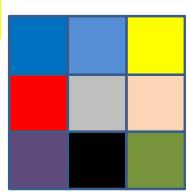
⋮



$w_l(m, n, x, y)$

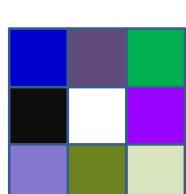
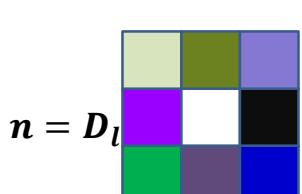


flip

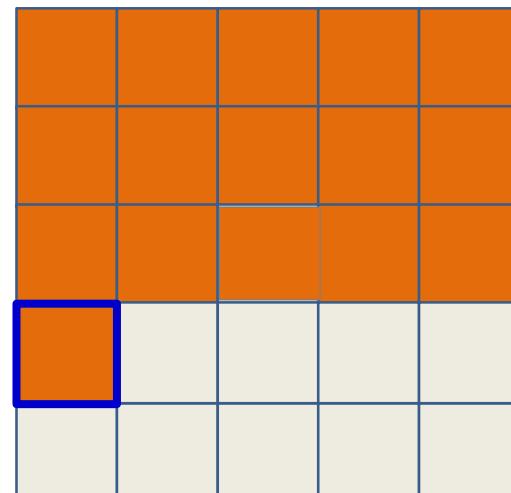


⋮

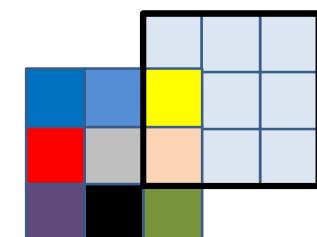
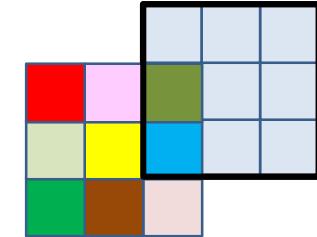
⋮



$w_l(m, n, K + 1 - x, K + 1 - y)$



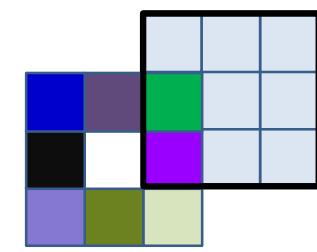
=

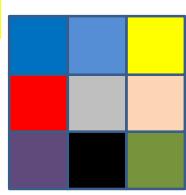
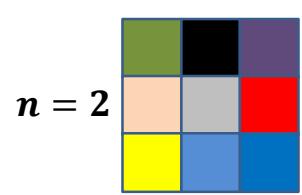
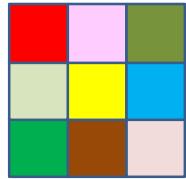
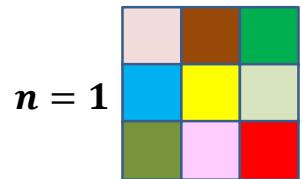


⋮
⋮
⋮

$$\frac{\partial \text{Div}}{\partial y(l-1, m, x, y)}$$

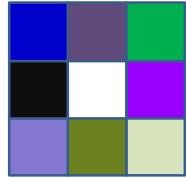
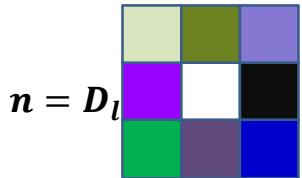
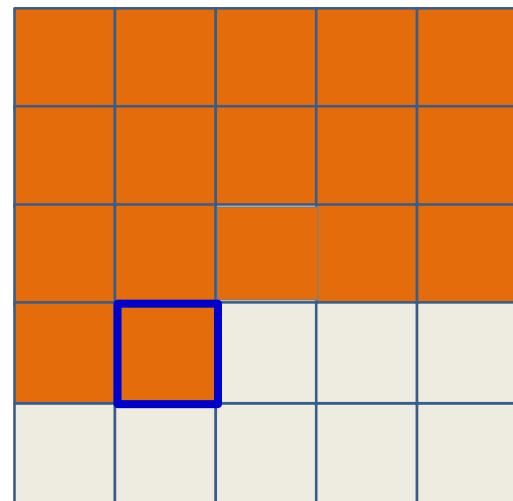
$$d\text{Div} = \sum_n \sum_{x', y'} \frac{d\text{Div}}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$



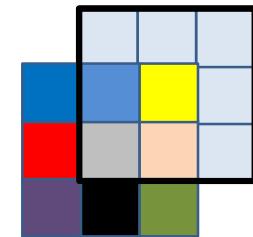
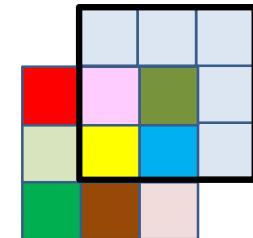
$w_l(m, n, x, y)$ 

⋮

⋮

 $w_l(m, n, K + 1 - x, K + 1 - y)$ 

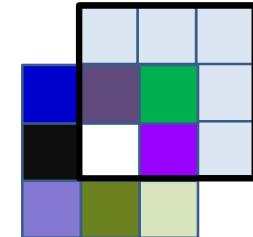
=



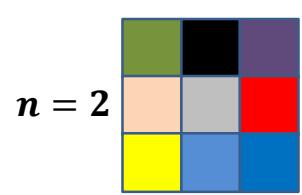
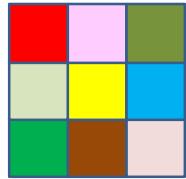
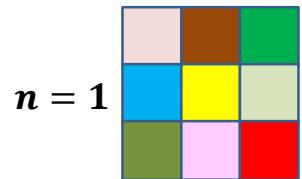
⋮
⋮
⋮

$$\frac{\partial \text{Div}}{\partial y(l-1, m, x, y)}$$

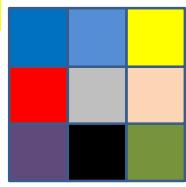
$$d\text{Div} = \sum_n \sum_{x', y'} \frac{d\text{Div}}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$



$w_l(m, n, x, y)$

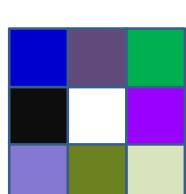
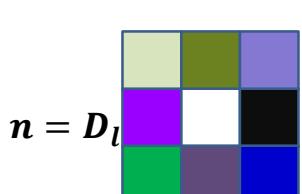


flip

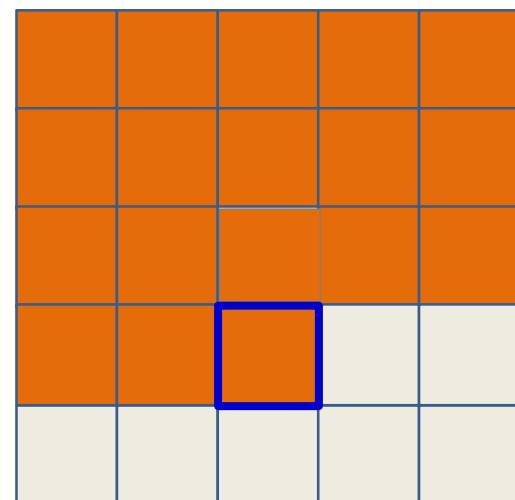


⋮

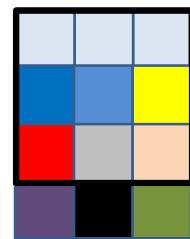
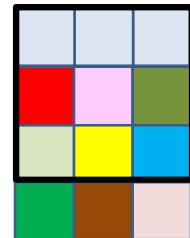
⋮



$w_l(m, n, K + 1 - x, K + 1 - y)$

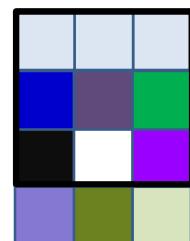


=

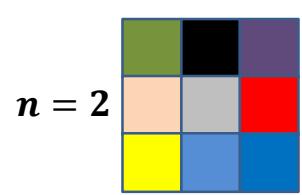
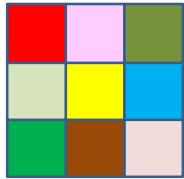
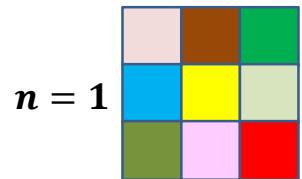


⋮

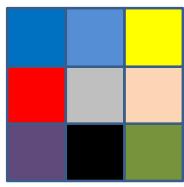
$$dY(l-1, m, x, y) = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$



$w_l(m, n, x, y)$

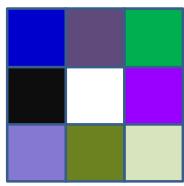
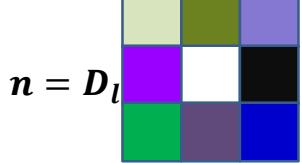


flip



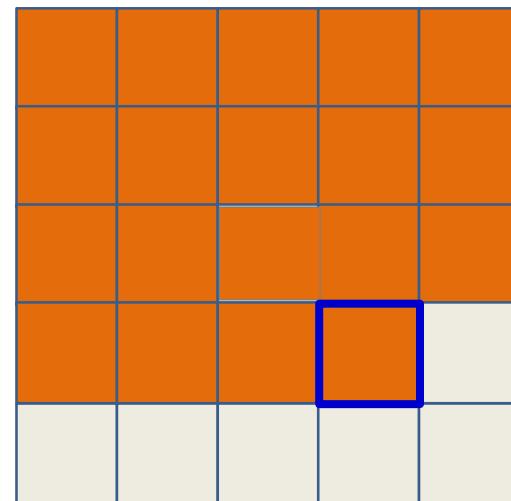
⋮

⋮

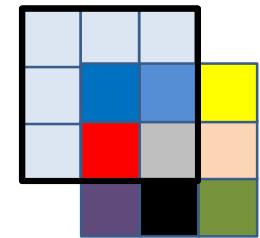
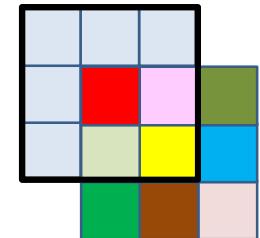


$w_l(m, n, K + 1 - x, K + 1 - y)$

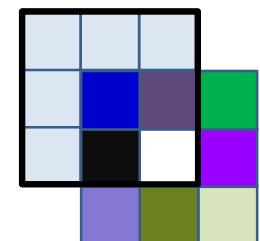
$$\frac{dDiv}{dy(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$



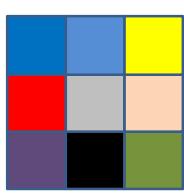
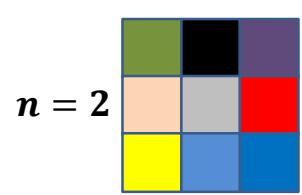
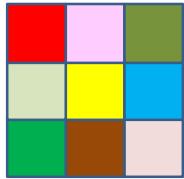
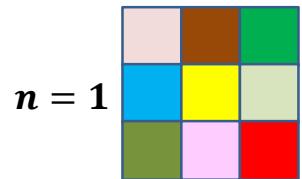
=



⋮

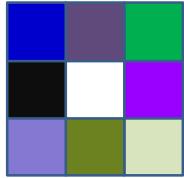
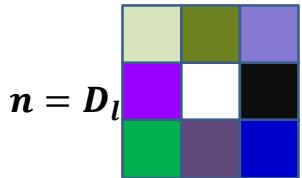


$w_l(m, n, x, y)$



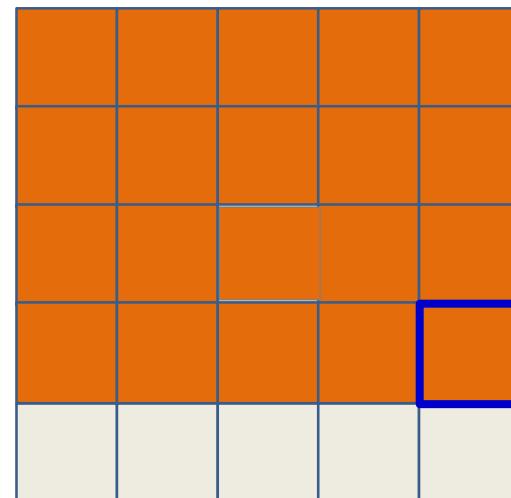
⋮

⋮

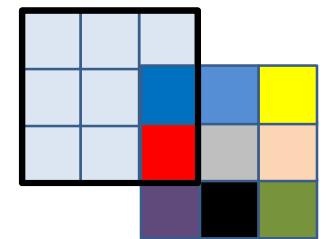
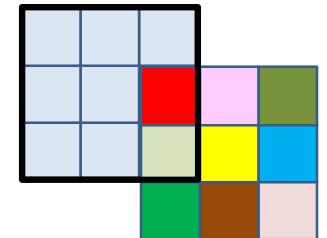


$w_l(m, n, K + 1 - x, K + 1 - y)$

$$\frac{dDiv}{dy(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

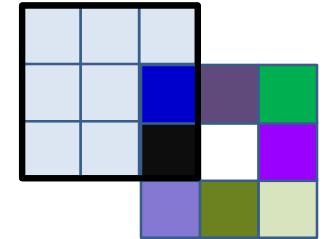


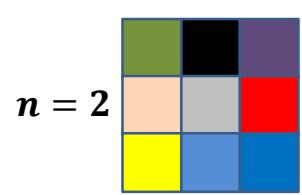
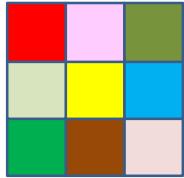
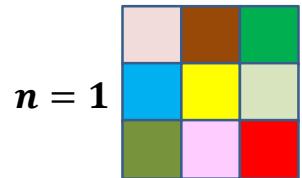
=



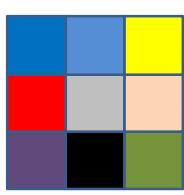
⋮

⋮



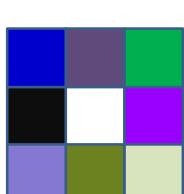
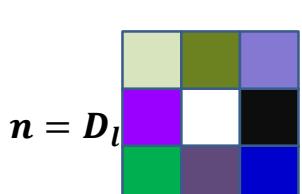
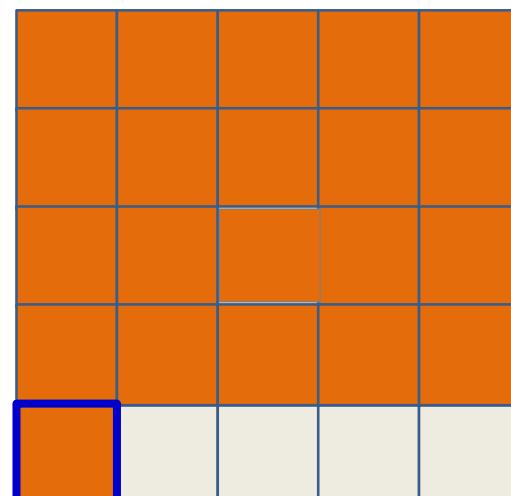
$w_l(m, n, x, y)$ 

flip



⋮

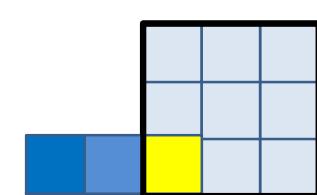
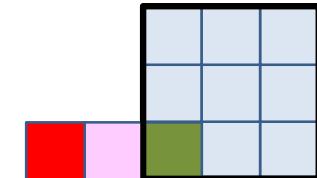
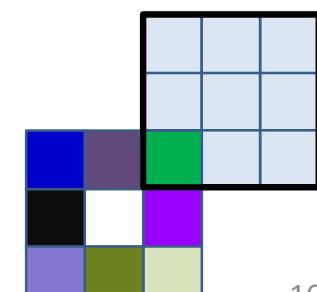
⋮

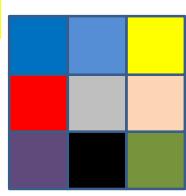
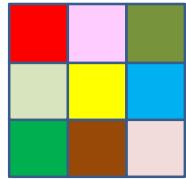
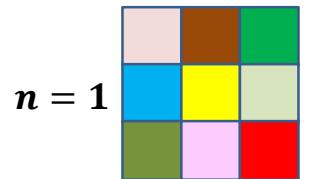
 $w_l(m, n, K + 1 - x, K + 1 - y)$ 

$$\frac{\partial \text{Div}}{\partial y(l-1, m, x, y)}$$

$$\frac{d\text{Div}}{dY(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{d\text{Div}}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

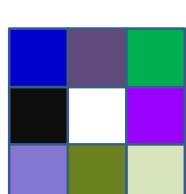
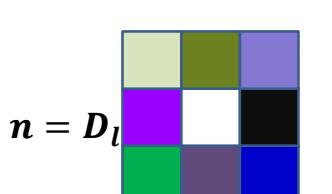
=

⋮
⋮
⋮

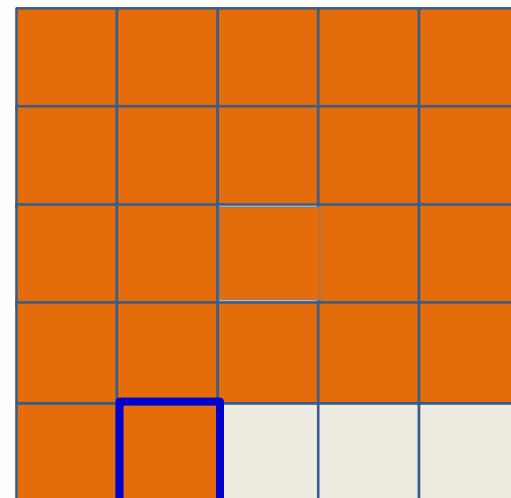
$w_l(m, n, x, y)$ 

⋮

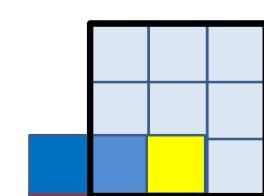
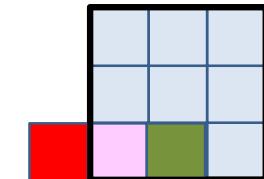
⋮



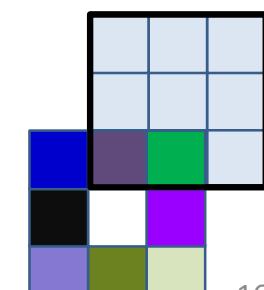
$w_l(m, n, K + 1 - x, K + 1 - y)$



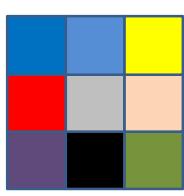
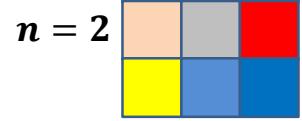
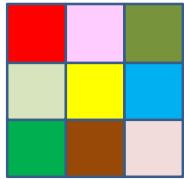
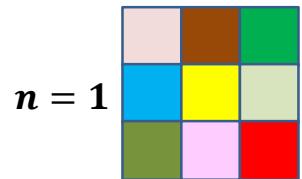
=



⋮
⋮
⋮

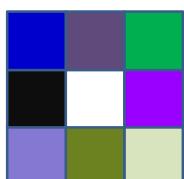
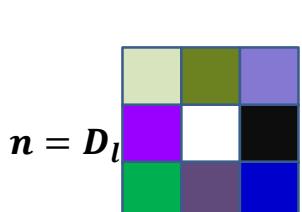


$$\frac{dDiv}{dy(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

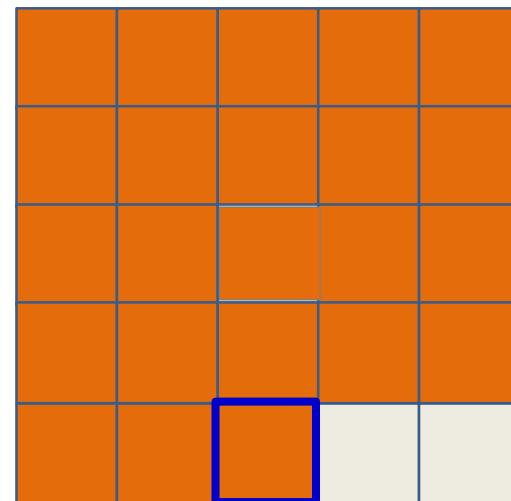
$w_l(m, n, x, y)$ 

⋮

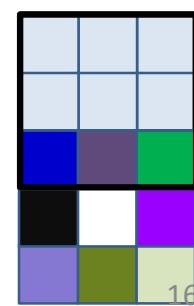
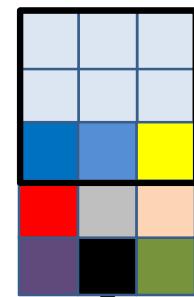
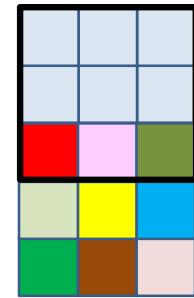
⋮



$w_l(m, n, K + 1 - x, K + 1 - y)$



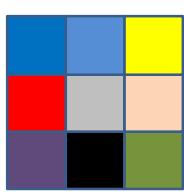
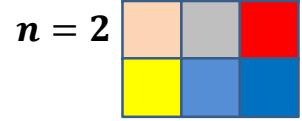
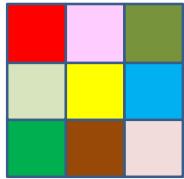
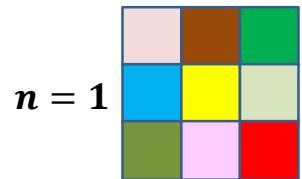
=



$$\frac{\partial \text{Div}}{\partial y(l-1, m, x, y)}$$

$$d\text{Div} = \sum_n \sum_{x',y'} \frac{d\text{Div}}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

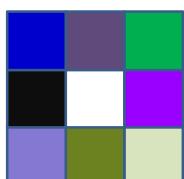
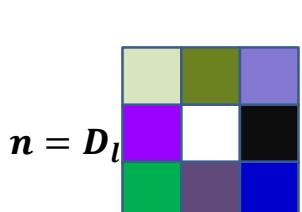
$w_l(m, n, x, y)$



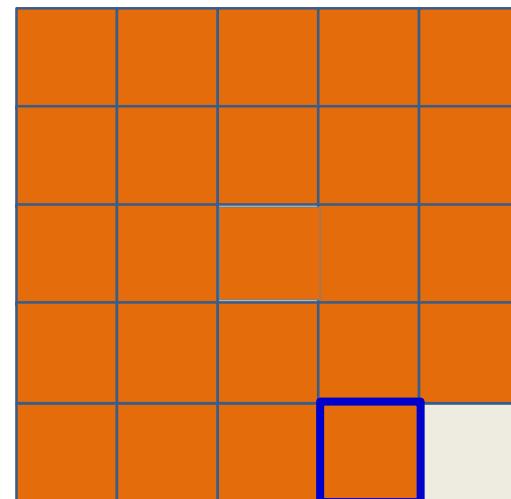
flip

⋮

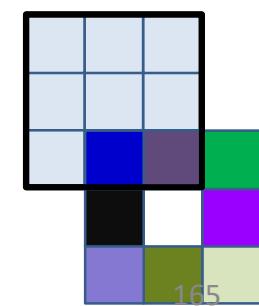
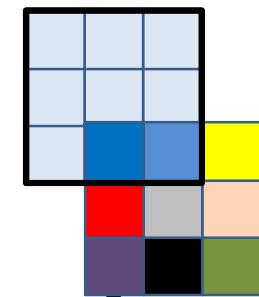
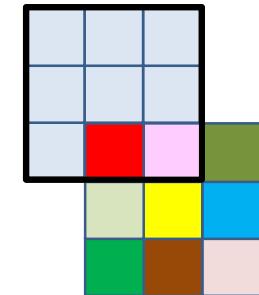
⋮



$w_l(m, n, K + 1 - x, K + 1 - y)$



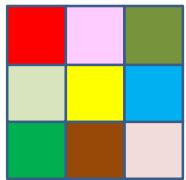
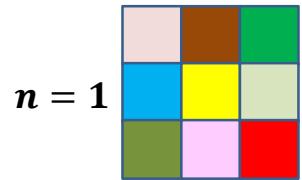
=



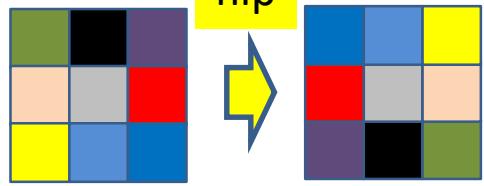
$$\frac{\partial \text{Div}}{\partial y(l-1, m, x, y)}$$

$$d\text{Div} = \sum_n \sum_{x', y'} \frac{d\text{Div}}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

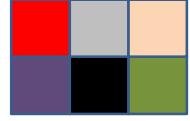
$w_l(m, n, x, y)$



$n = 2$



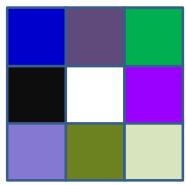
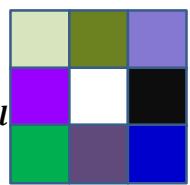
flip



⋮

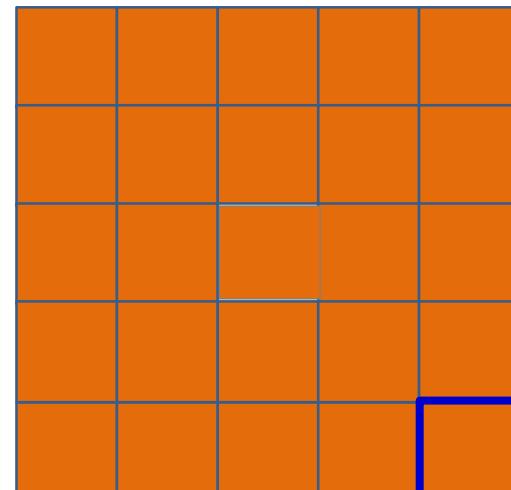
⋮

$n = D_l$

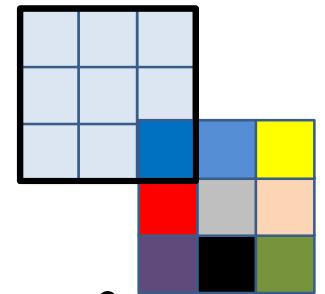
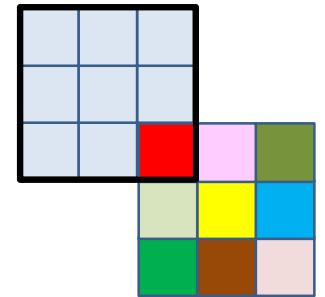


$w_l(m, n, K + 1 - x, K + 1 - y)$

$$\frac{dDiv}{dy(l-1, m, x, y)} = \sum_n \sum_{x', y'} \frac{dDiv}{dz(l, n, x', y')} w_l(m, n, x - x', y - y')$$

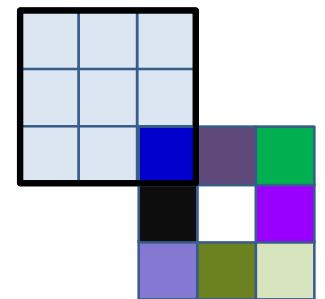


=

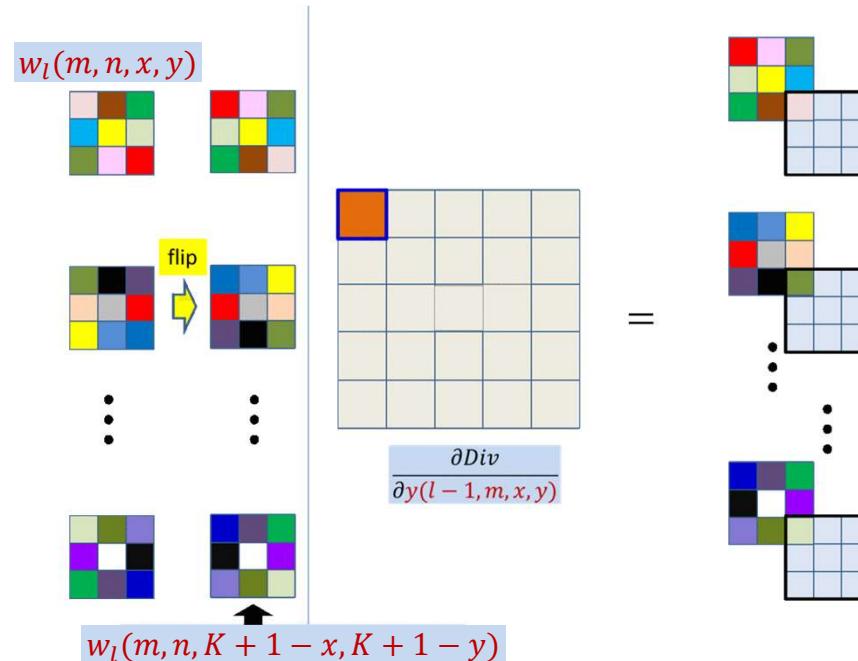


⋮

⋮

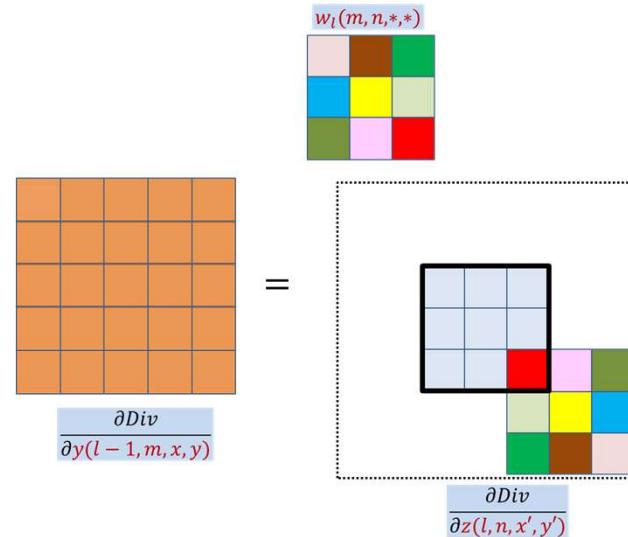


Computing the derivative for $Y(l - 1, m)$



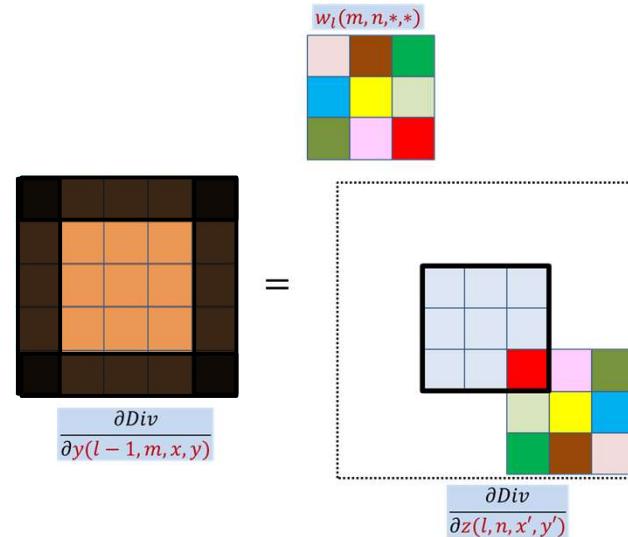
- This is just a convolution of the zero-padded maps by the transposed and flipped filter
 - After zero padding it first with $K - 1$ zeros on every side

The size of the Y-derivative map



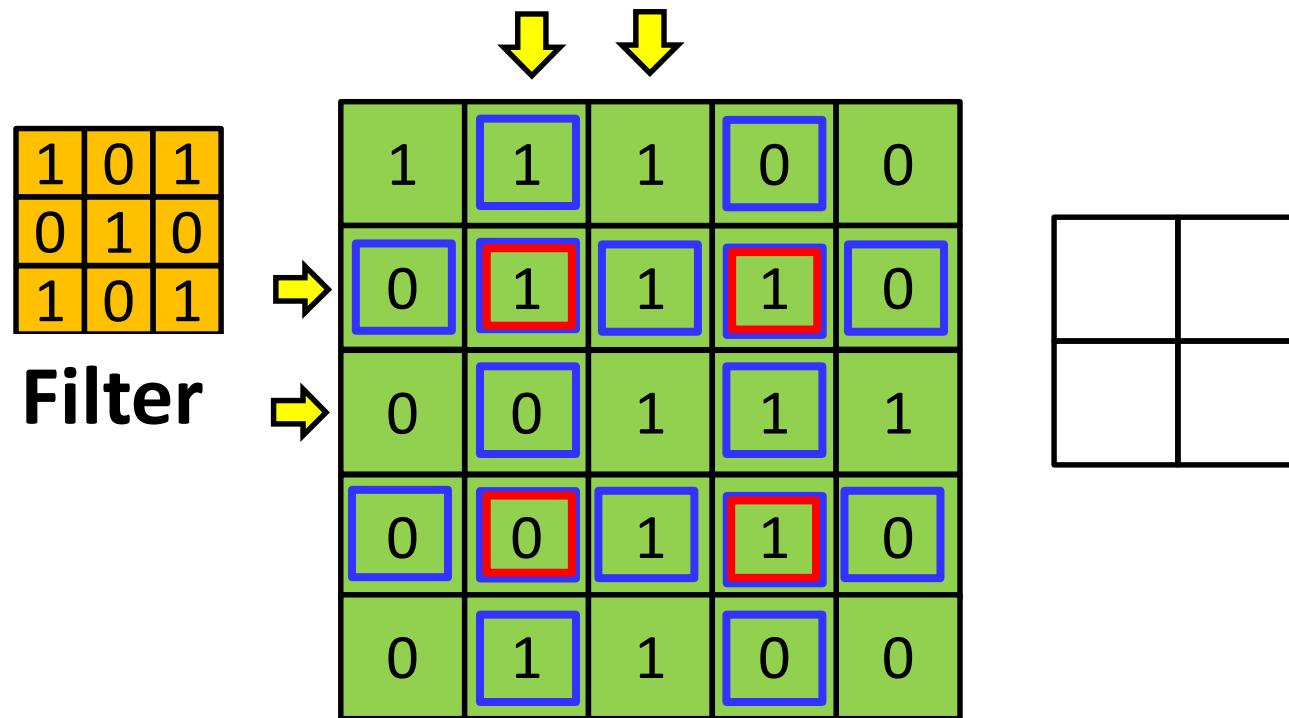
- We continue to compute elements for the derivative Y map as long as the (flipped) filter has at least one element in the (unpadded) derivative Zmap
 - I.e. so long as the Y derivative is non-zero
- The size of the Y derivative map will be $(H + K - 1) \times (W + K - 1)$
 - H and W are height and width of the Zmap
- This will be the size of the actual Y map that was originally convolved

The size of the Y-derivative map



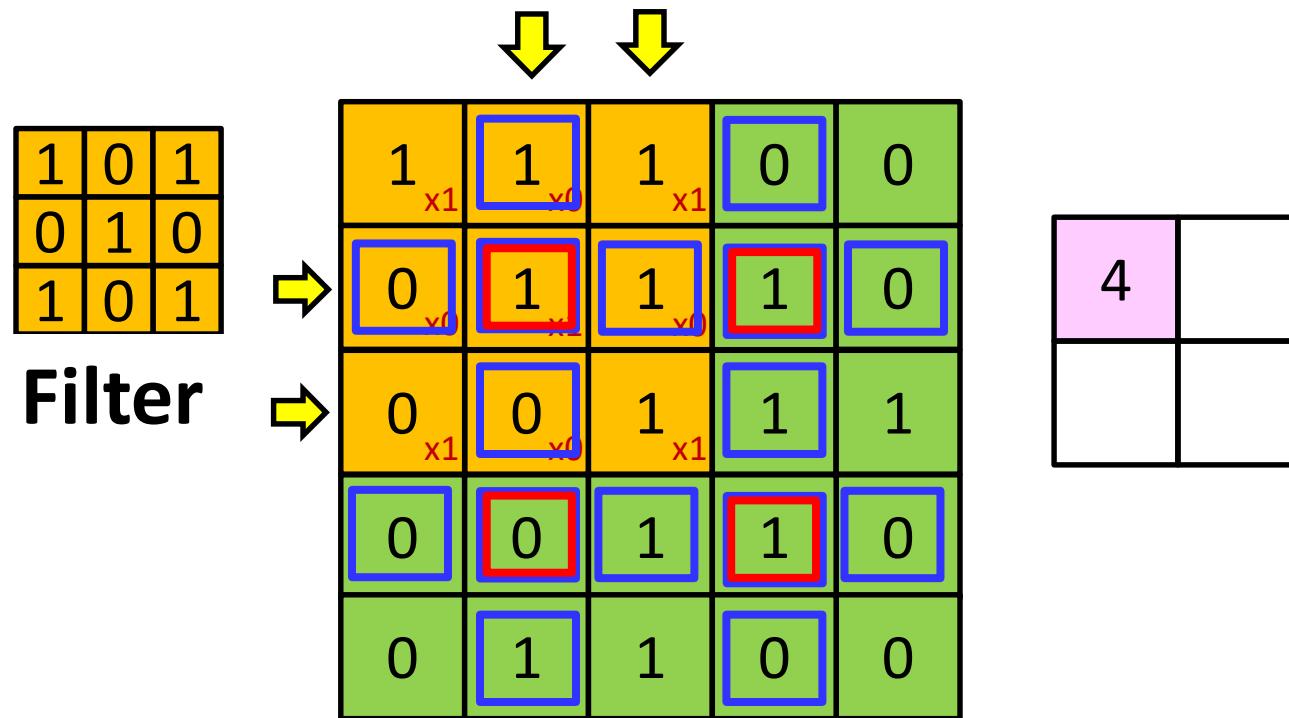
- If the Y map was zero-padded in the forward pass, the derivative map will be the size of the *zero-padded* map
 - The zero padding regions must be deleted before further backprop

When the stride is more than 1?



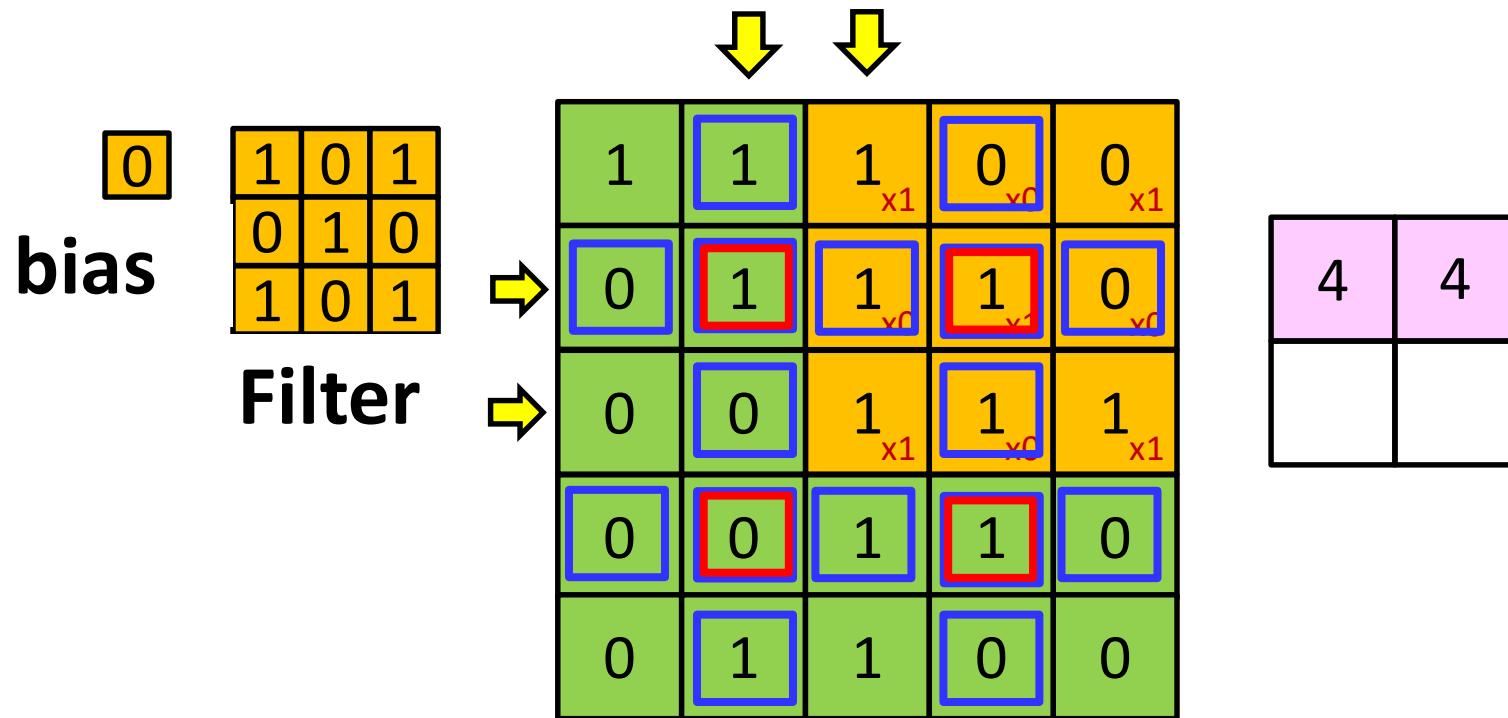
- When the stride is greater than 1, some positions of $Y(l - 1, m)$ contribute to more locations on the $Z(l, n)$ maps than others
 - With a stride of 2, the boxed-in-blue $Y(l - 1, m)$ locations contribute to half as many $Z(l, n)$ locations as the unboxed locations
 - The double-boxed (blue and red boxes) $Y(l - 1, m)$ locations contribute to only a quarter as many $Z(l, n)$ locations as the unboxed ones

When the stride is more than 1?



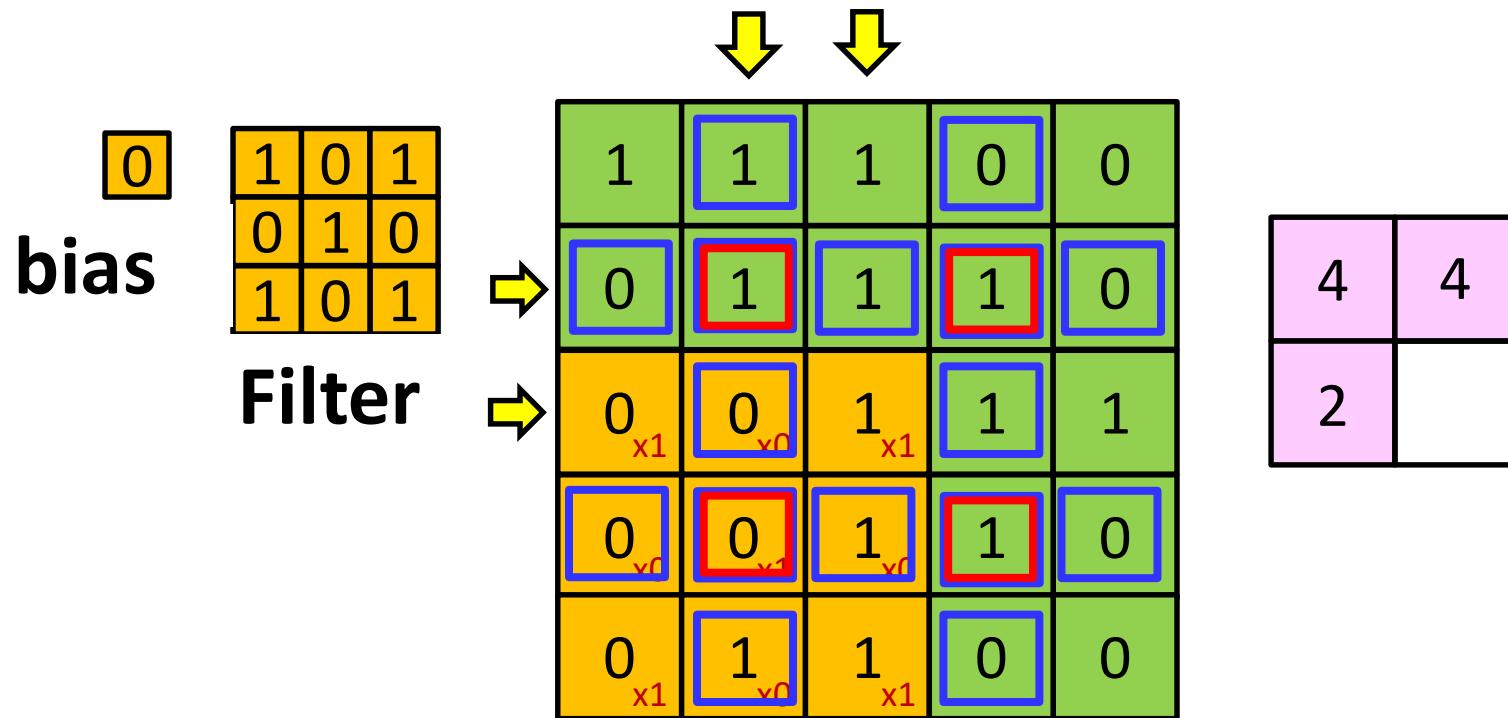
- When the stride is greater than 1, some positions of $Y(l - 1, m)$ contribute to more locations on the $Z(l, n)$ maps than others
 - With a stride of 2, the boxed-in-blue $Y(l - 1, m)$ locations contribute to half as many $Z(l, n)$ locations as the unboxed locations
 - The double-boxed (blue and red boxes) $Y(l - 1, m)$ locations contribute to only a quarter as many $Z(l, n)$ locations as the unboxed ones

When the stride is more than 1?



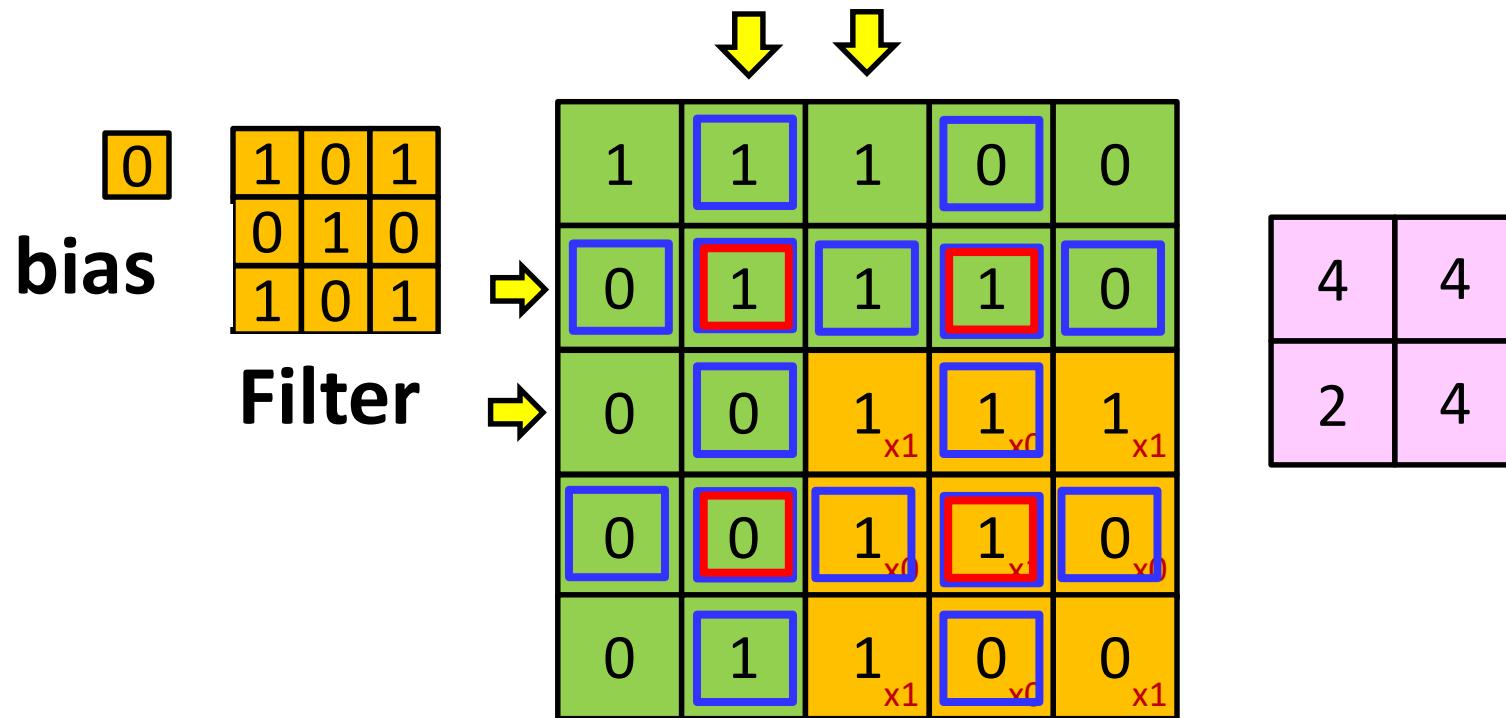
- When the stride is greater than 1, some positions of $Y(l - 1, m)$ contribute to more locations on the $Z(l, n)$ maps than others
 - With a stride of 2, the boxed-in-blue $Y(l - 1, m)$ locations contribute to half as many $Z(l, n)$ locations as the unboxed locations
 - The double-boxed (blue and red boxes) $Y(l - 1, m)$ locations contribute to only a quarter as many $Z(l, n)$ locations as the unboxed ones

When the stride is more than 1?



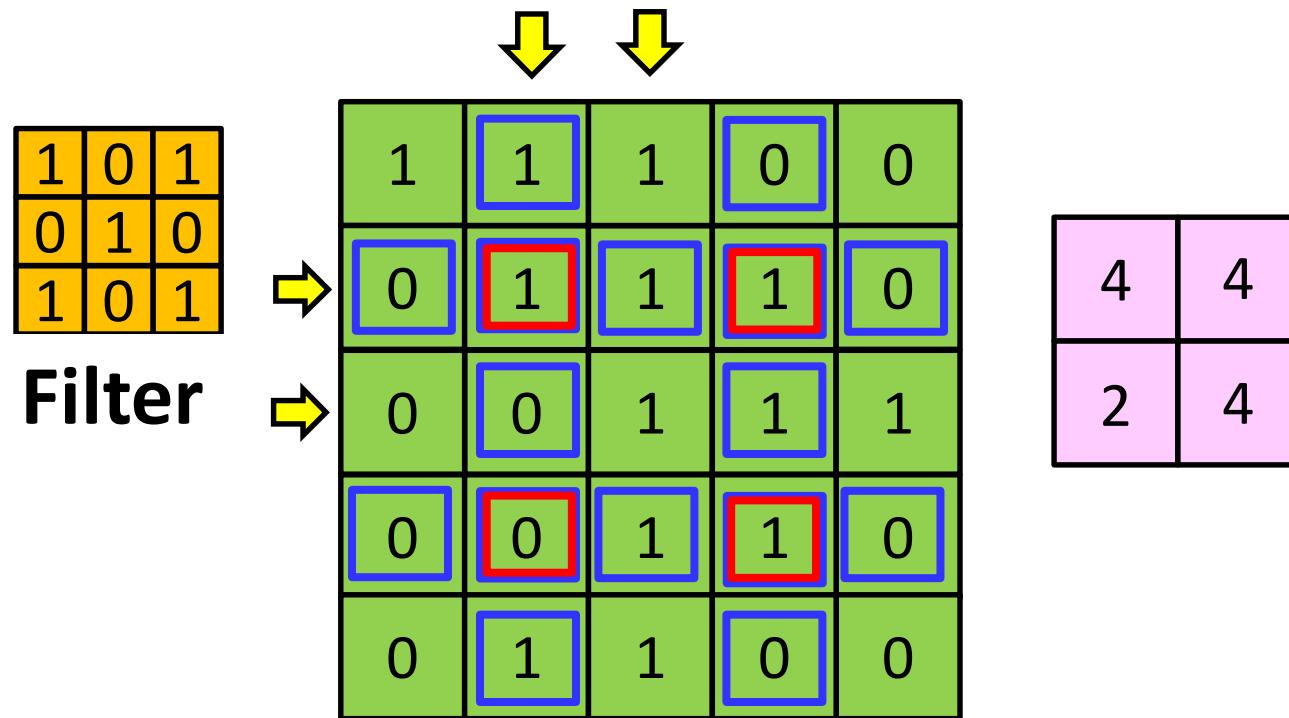
- When the stride is greater than 1, some positions of $Y(l - 1, m)$ contribute to more locations on the $Z(l, n)$ maps than others
 - With a stride of 2, the boxed-in-blue $Y(l - 1, m)$ locations contribute to half as many $Z(l, n)$ locations as the unboxed locations
 - The double-boxed (blue and red boxes) $Y(l - 1, m)$ locations contribute to only a quarter as many $Z(l, n)$ locations as the unboxed ones

When the stride is more than 1?



- When the stride is greater than 1, some positions of $Y(l - 1, m)$ contribute to more locations on the $Z(l, n)$ maps than others
 - With a stride of 2, the boxed-in-blue $Y(l - 1, m)$ locations contribute to half as many $Z(l, n)$ locations as the unboxed locations
 - The double-boxed (blue and red boxes) $Y(l - 1, m)$ locations contribute to only a quarter as many $Z(l, n)$ locations as the unboxed ones

When the stride is more than 1?



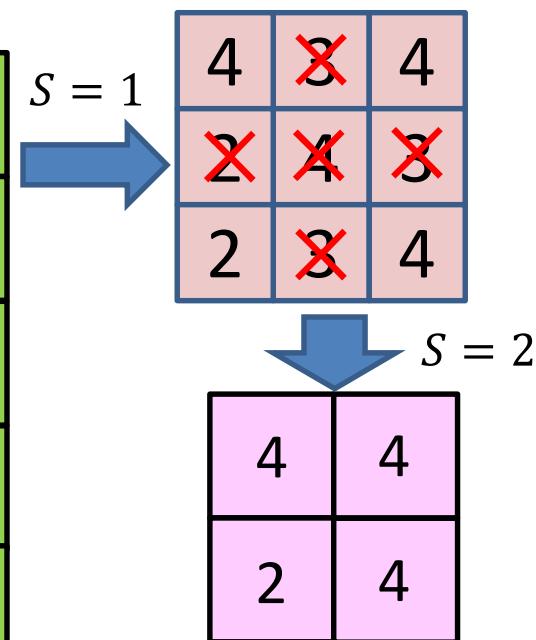
- We must make adjustments for when the stride is greater than 1.

Stride greater than 1

1	0	1
0	1	0
1	0	1

Filter

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



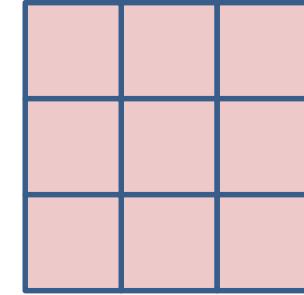
- **Observation:** Convolving with a stride S greater than 1 is the same as convolving with stride 1 and “dropping” $S - 1$ out of every S rows, and $S - 1$ of every S columns
 - **Downsampling by S**
 - E.g. for stride 2, it is the same as convolving with stride 1 and dropping every 2nd entry

Derivatives with Stride greater than 1

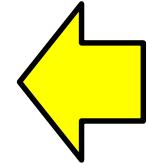
1	0	1
0	1	0
1	0	1

Filter

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



$\frac{dDiv}{dz(0,0)}$	$\frac{dDiv}{dz(1,0)}$
$\frac{dDiv}{dz(0,1)}$	$\frac{dDiv}{dz(1,1)}$



- **Derivatives:** Backprop gives us the derivatives of the divergence with respect to the elements of the *downsampled* (strided) Z map

Derivatives with Stride greater than 1

1	0	1
0	1	0
1	0	1

Filter

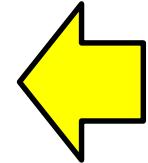
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

$dDiv$		$dDiv$
$dz(0,0)$		$dz(1,0)$
$dDiv$		$dDiv$
$dz(0,1)$		$dz(1,1)$

$\uparrow S = 2$

$dDiv$	$dDiv$
$dz(0,0)$	$dz(1,0)$

$dDiv$	$dDiv$
$dz(0,1)$	$dz(1,1)$



- **Derivatives:** Backprop gives us the derivatives of the divergence with respect to the elements of the *downsampled* (strided) Z map
- We can place these derivative values back into their original locations of the full-sized Z map

Derivatives with Stride greater than 1

1	0	1
0	1	0
1	0	1

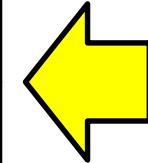
Filter

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

$dDiv$ $dz(0,0)$	0	$dDiv$ $dz(1,0)$
0	0	0
$dDiv$ $dz(0,1)$	0	$dDiv$ $dz(1,1)$

$\uparrow S = 2$

$dDiv$ $dz(0,0)$	$dDiv$ $dz(1,0)$
$dDiv$ $dz(0,1)$	$dDiv$ $dz(1,1)$



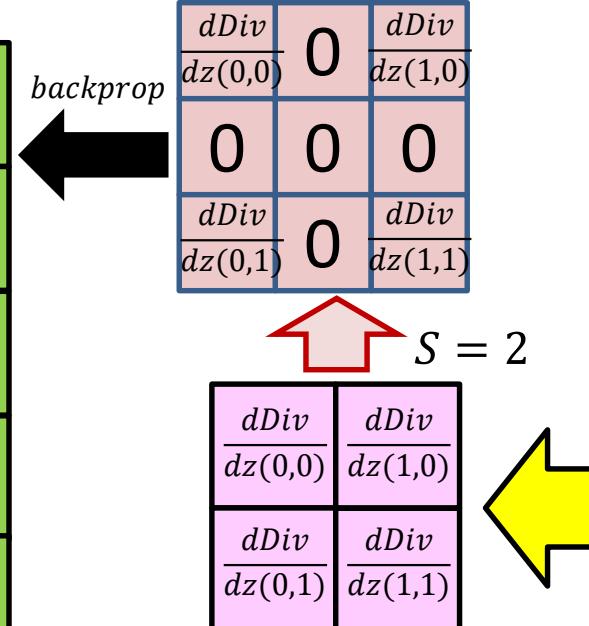
- **Derivatives:** Backprop gives us the derivatives of the divergence with respect to the elements of the *downsampled* (strided) Z map
- We can place these values back into their original locations of the full-sized Z map
- The remaining entries of the Z map do not affect the divergence
 - Since they get dropped out
- The derivative of the divergence w.r.t. these values is 0

Computing derivatives with Stride > 1

1	0	1
0	1	0
1	0	1

Filter

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



- **Upsampling derivative map:**
 - Upsample the downsampled derivatives
 - Insert zeros into the “empty” slots
 - This gives us the derivatives w.r.t. all the entries of a full-sized (stride 1) Z map
- We can compute the derivatives for Y, using the full map

Poll 3

- @888

Poll 3

Select all statements that are true about how to compute the derivative of the divergence w.r.t l th layer activation maps by backpropagation

- To compute the derivative w.r.t. the m th activation map of the l th convolutional layer, we must select the m th “planes” of all the $(l+1)$ th layer filters
- The selected filter planes must be flipped left-right and up-down
- They must convolve the derivative (maps) for the $(l+1)$ th layer affine values
- The output of the convolution must be flipped back left-right and up-down
- If the forward convolution has a stride S , the derivative maps must be upsampled by S prior to convolution
- If the forward convolution has stride S , the backpropagation convolution must also have a stride S

Overall algorithm for computing derivatives w.r.t. $Y(l - 1)$

- Given the derivatives $\frac{dDiv}{dz(l,n,x,y)}$
- If stride $S > 1$, upsample derivative map

$$\hat{z}(l, n, Sx, Sy) = \frac{dDiv}{dz(l, n, x, y)}$$

$\hat{z}(l, n, x, y) = 0 \quad \forall x, y \neq \text{integer multiples of } S$

- For $S = 1$,

$$\hat{z}(l, n, x, y) = \frac{dDiv}{dz(l, n, x, y)}$$

- Compute derivatives using:

$$\frac{dDiv}{dY(l - 1, m, x, y)} = \sum_n \sum_{x', y'} \hat{z}(l, n, x', y') w_l(m, n, x - x', y - y')$$

Can be computed by convolution with flipped filter

Derivatives for a single layer l :

Vector notation

```
# The weight W(l,m) is a 3D D_{l-1}xK_lxK_l
# Assuming dz has already been obtained via backprop
if (stride > 1)    #upsample
    dz = upsample(dz,stride, W_{l-1}, H_{l-1}, K_l)

dzpad = zeros(D_lx(H_l+2(K_l-1))x(W_l+2(K_l-1))) # zeropad
for j = 1:D_l
    for i = 1:D_{l-1}    # Transpose and flip
        Wflip(i,j,:,:,:) = flipLeftRight(flipUpDown(W(l,i,j,:,:,:)))
        dzpad(j,K_l:H_l-1,K_l:K_l+W_l-1) = dz(l,j,:,:,:) #center map
end

for j = 1:D_{l-1}
    for x = 1:W_{l-1}
        for y = 1:H_{l-1}
            segment = dzpad(:, x:x+K_l-1, y:y+K_l-1) #3D tensor
            dy(l-1,j,x,y) = Wflip.segment #tensor inner prod.
```

Upsampling

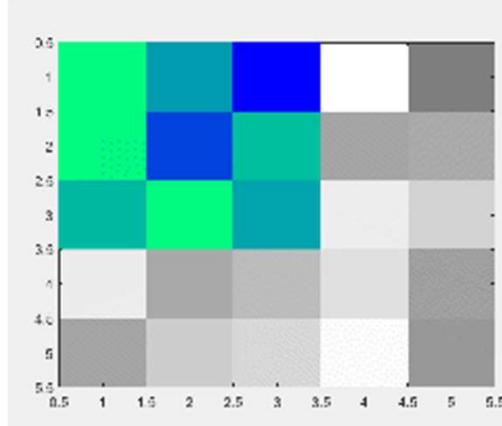
```
# Upsample dz to the size it would be if stride was 1
function upsample(dz, S, W, H, K)
    if (S > 1)    #Insert S-1 zeros between samples
        Hup = H - K + 1
        Wup = W - K + 1
        dzup = zeros(Wup, Hup)
        for x = 1:S:H
            xdownsamp = (x-1)/S+1    #Downsampled index
            for y = 1:S:W
                ydownsamp = (y-1)/S+1
                dzup(x, y) = dz(xdownsamp, ydownsamp)
        else
            dzup = dz
    return dzup
```

Backpropagating through affine map

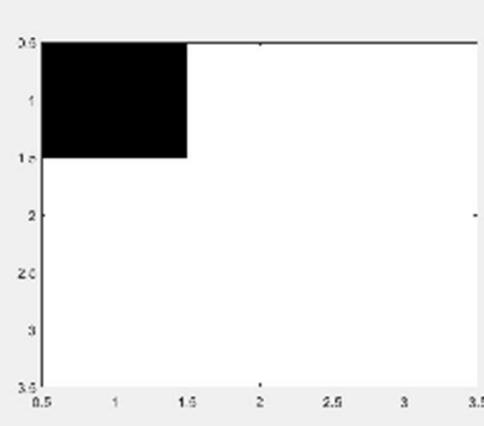
- Forward affine computation:
 - Compute affine maps $z(l, n, x, y)$ from previous layer maps $y(l - 1, m, x, y)$ and filters $w_l(m, n, x, y)$
- Backpropagation: Given $\frac{dDiv}{dz(l,n,x,y)}$
 - ✓ Compute derivative w.r.t. $y(l - 1, m, x, y)$
 - Compute derivative w.r.t. $w_l(m, n, x, y)$

The derivatives for the weights

$Y(l - 1, m) \otimes w_l(m, n)$



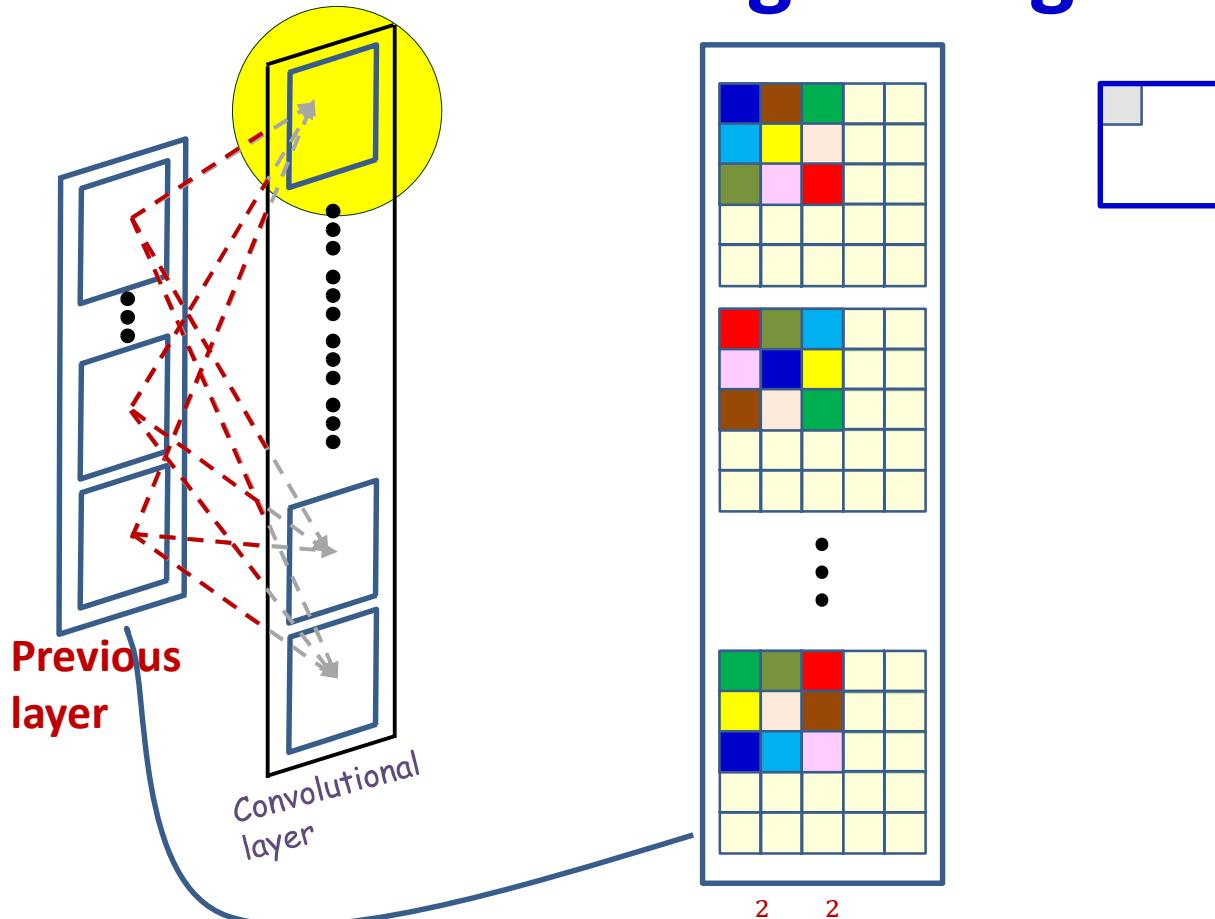
$Z(l, n)$



$$z(l, n, x, y) = \sum_m \sum_{x', y'} w_l(m, n, x', y') y(l - 1, m, x + x', y + y') + b_l(n)$$

- Each **weight** $w_l(m, n, x', y')$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components:
 $w_l(m, n, i, j)$ (e.g. $w_l(m, n, 1, 2)$)

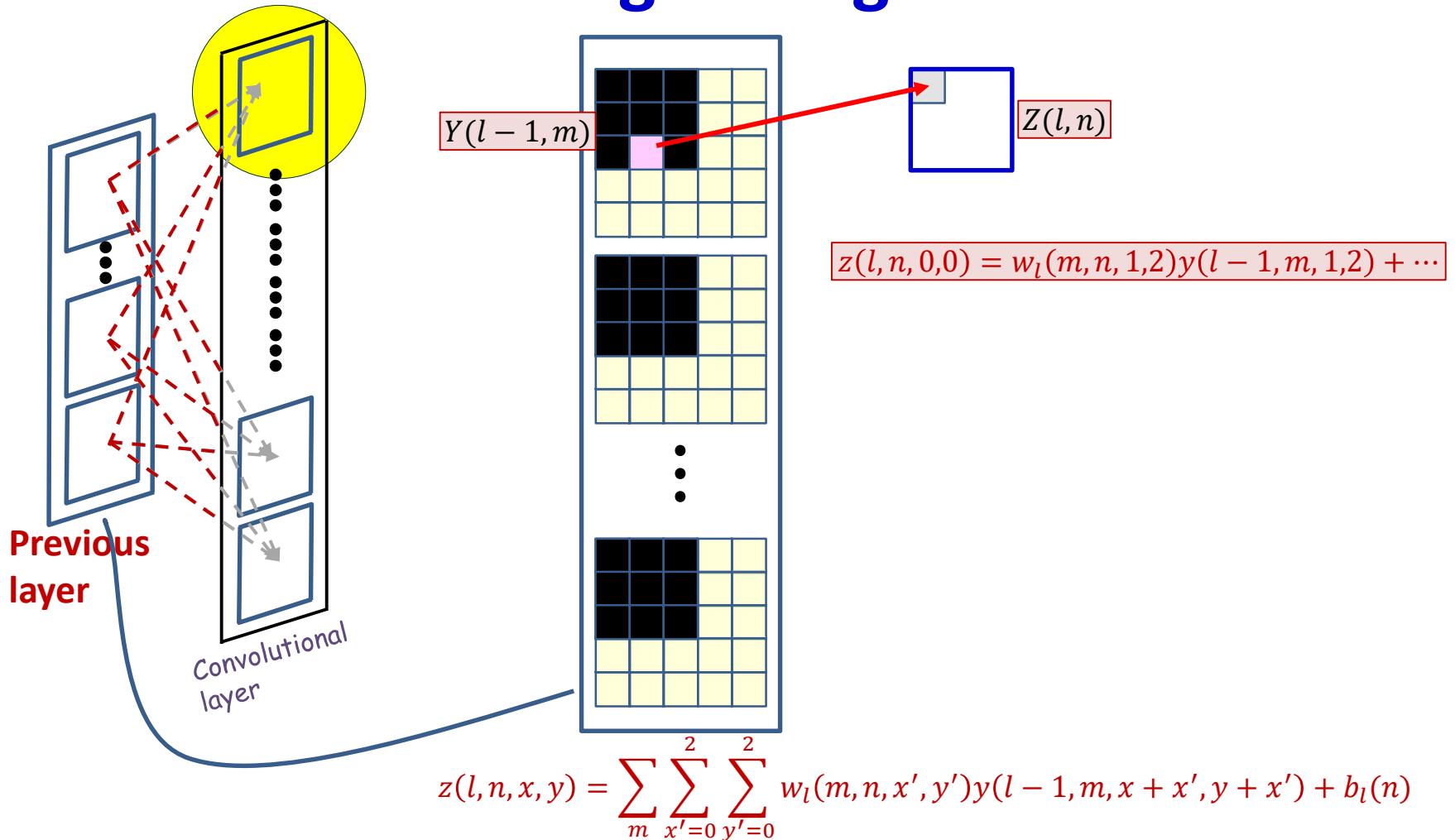
Convolution: the contribution of a single weight



$$z(l, n, x, y) = \sum_m \sum_{x'=0}^2 \sum_{y'=0}^2 w_l(m, n, x', y') y(l-1, m, x+x', y+y') + b_l(n)$$

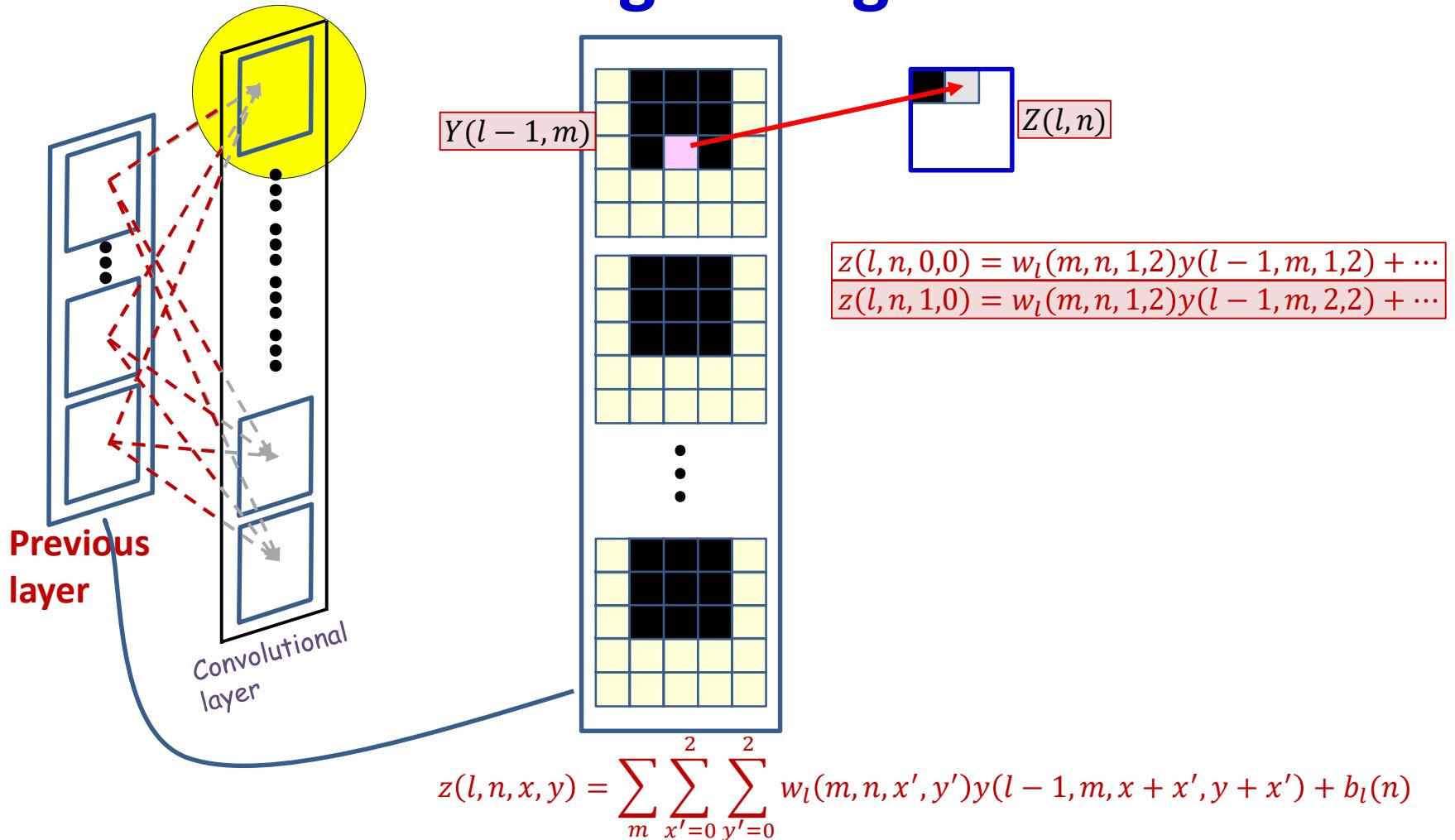
- Each affine output is computed from multiple input maps simultaneously
- Each **weight** $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$

Convolution: the contribution of a single weight



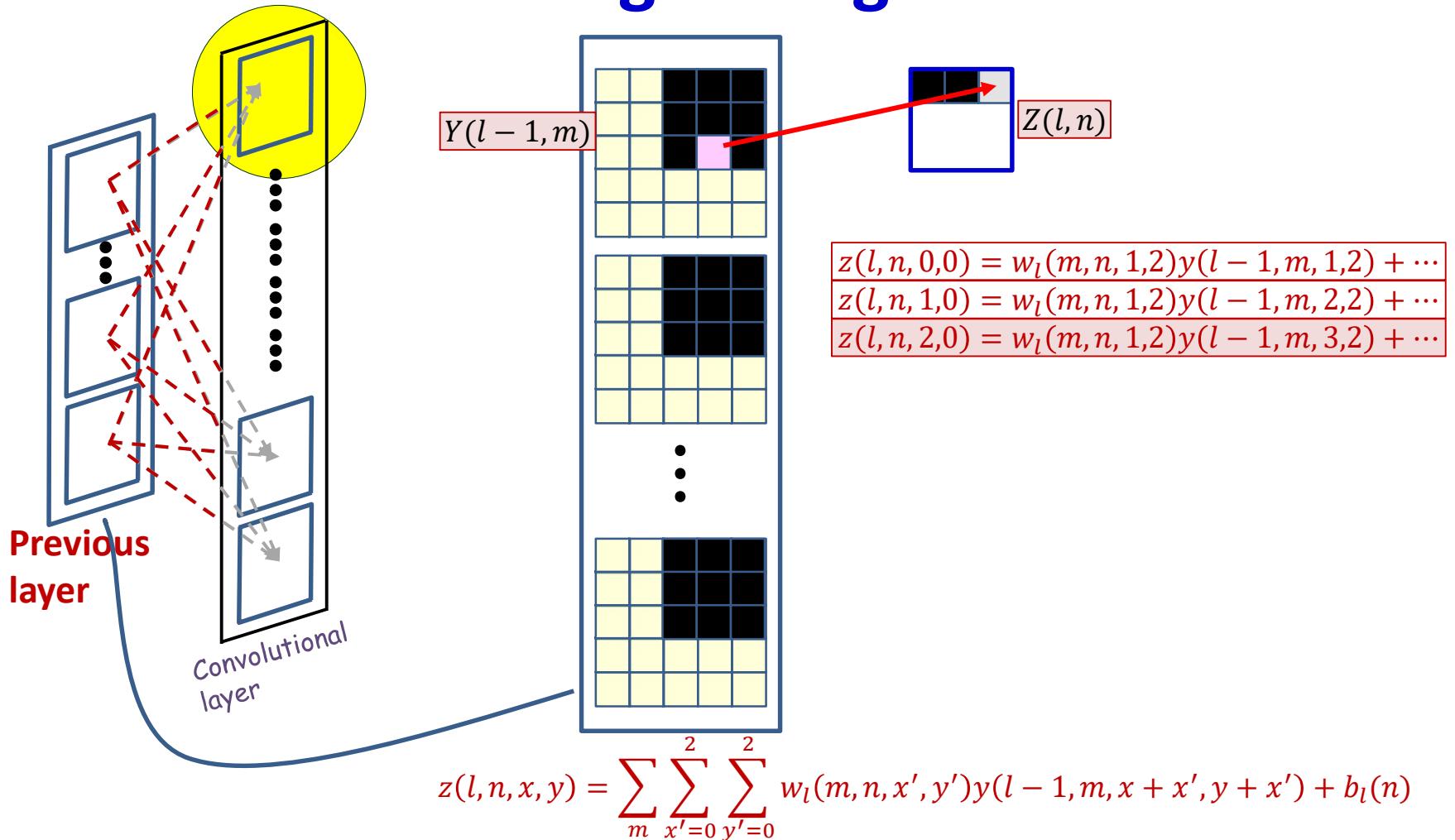
- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)$

Convolution: the contribution of a single weight



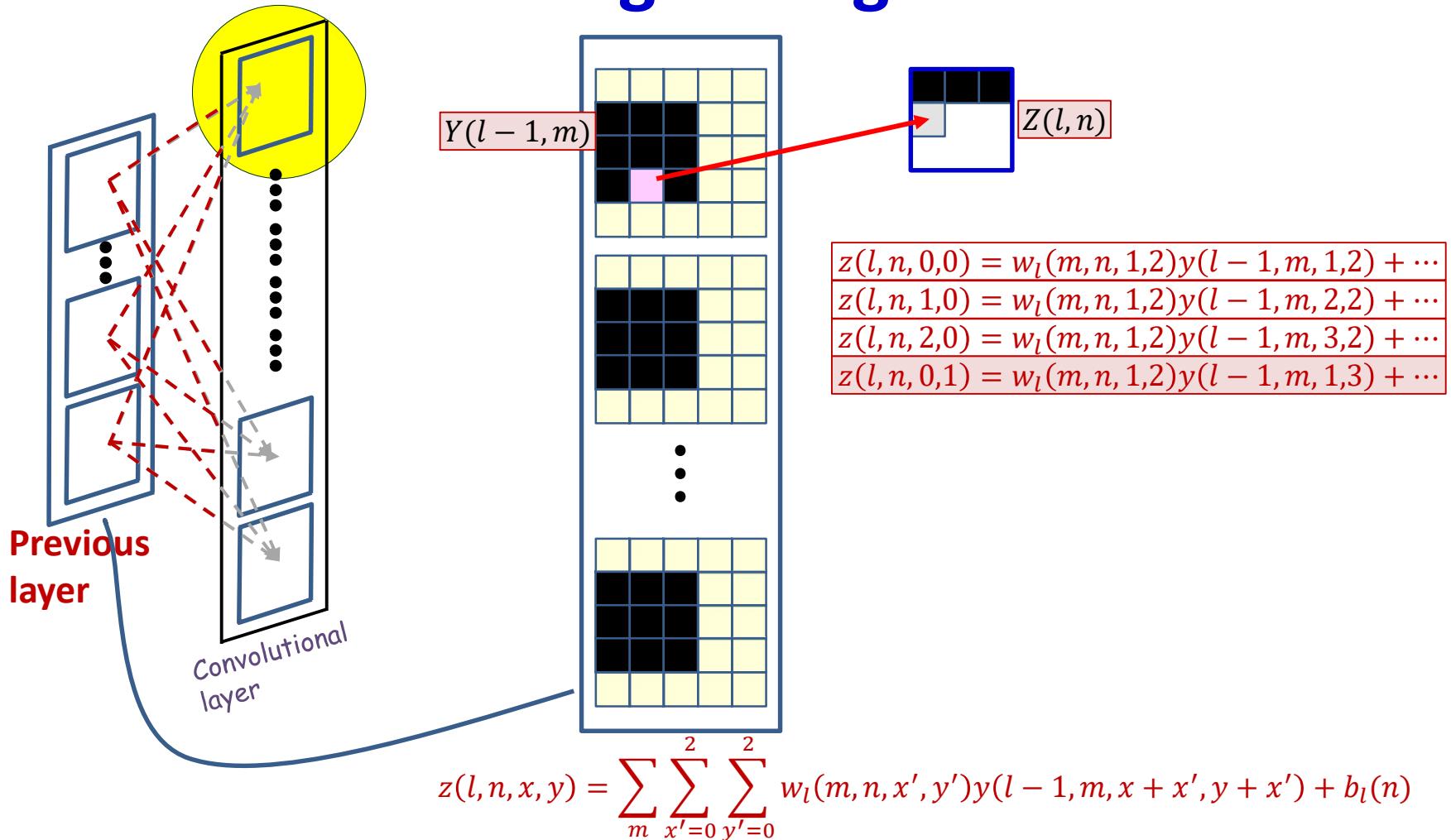
- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)$

Convolution: the contribution of a single weight



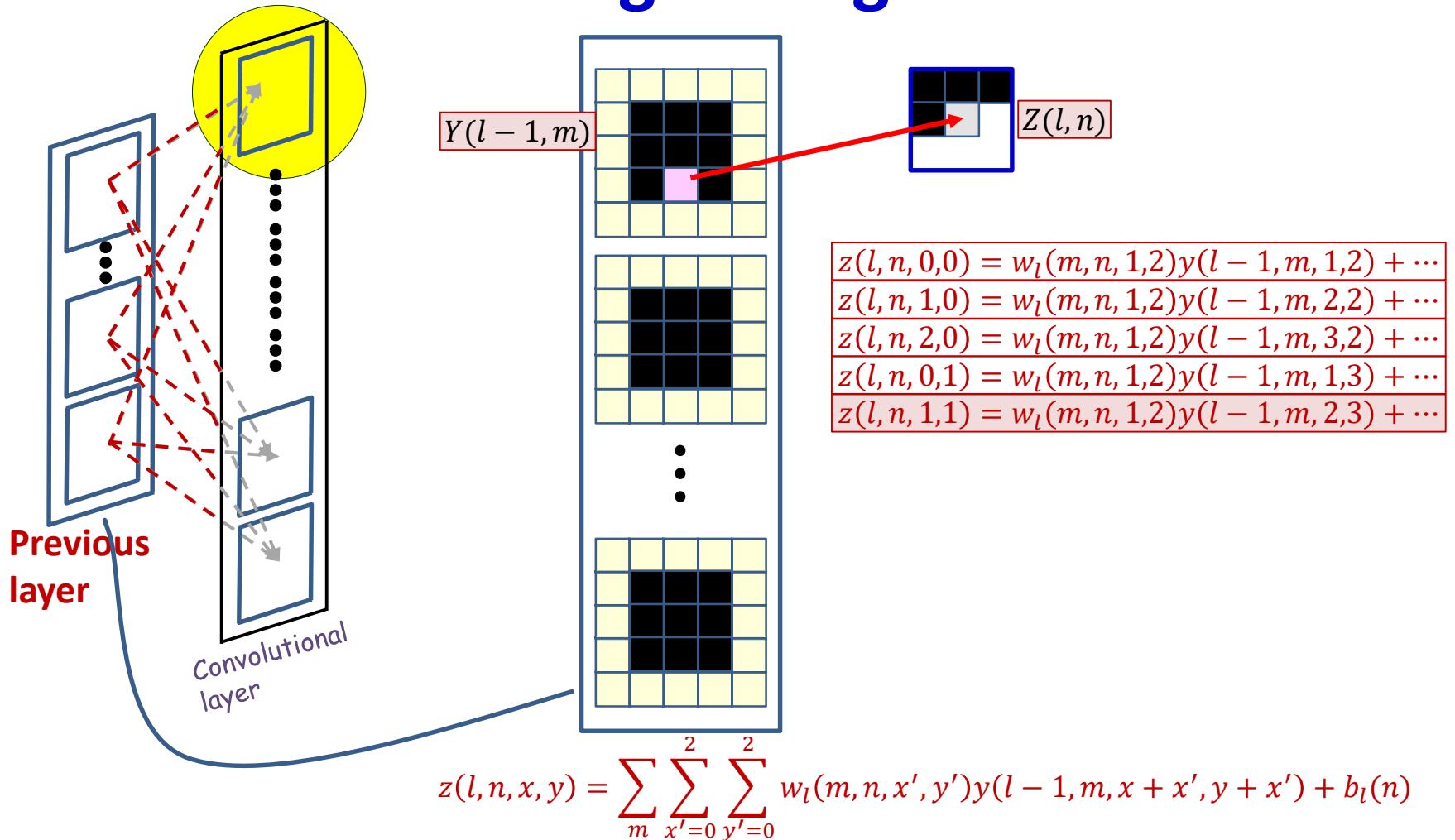
- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)_{\text{91}}$

Convolution: the contribution of a single weight



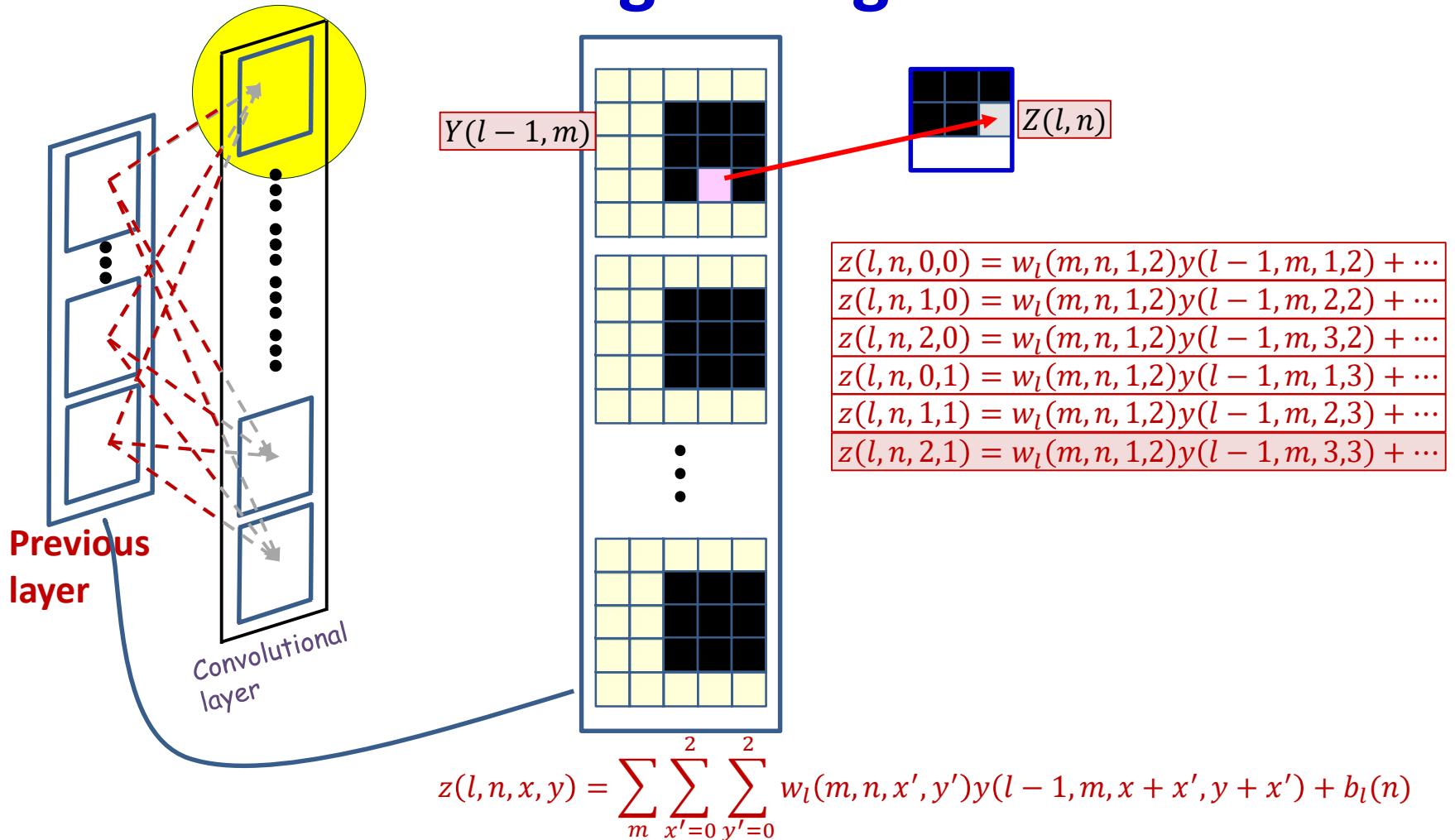
- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)$ ₉₂

Convolution: the contribution of a single weight



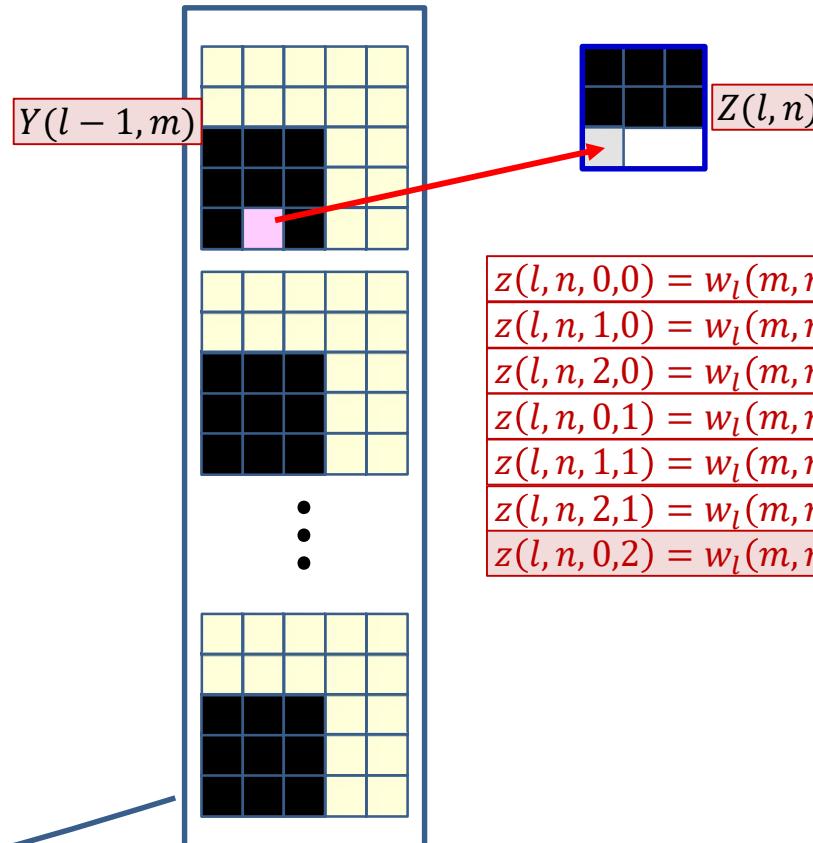
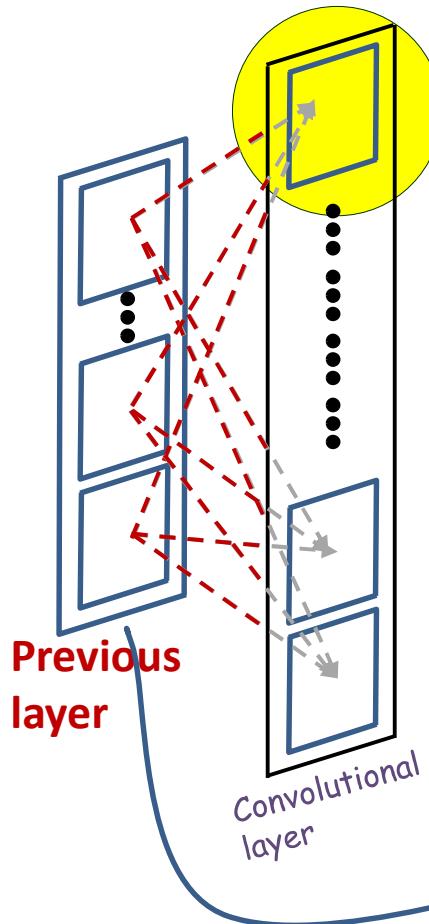
- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)$

Convolution: the contribution of a single weight



- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1,2)$

Convolution: the contribution of a single weight

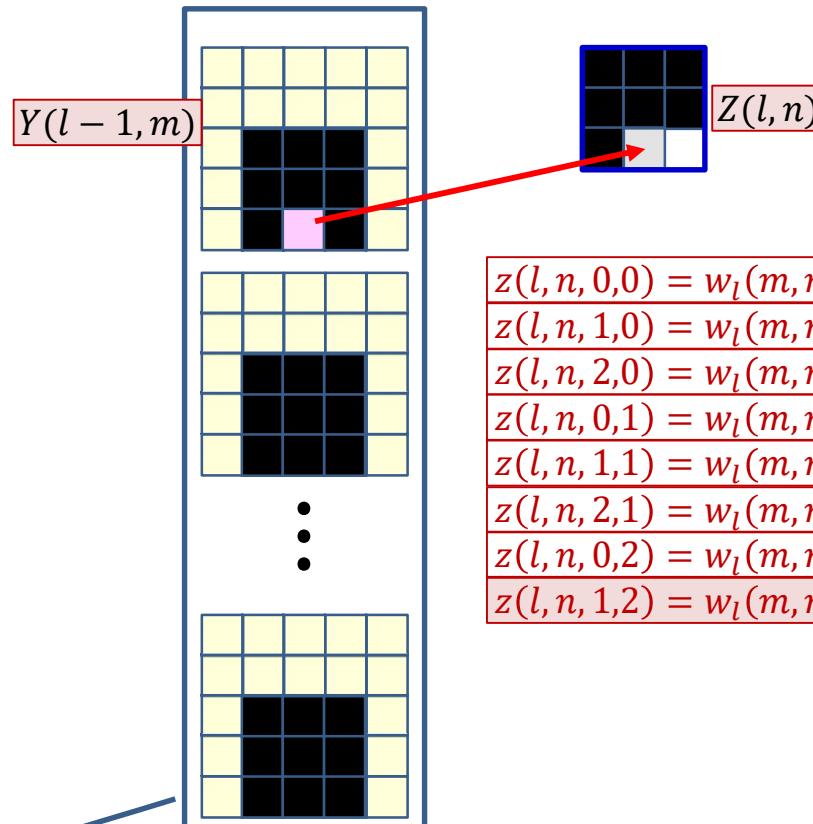
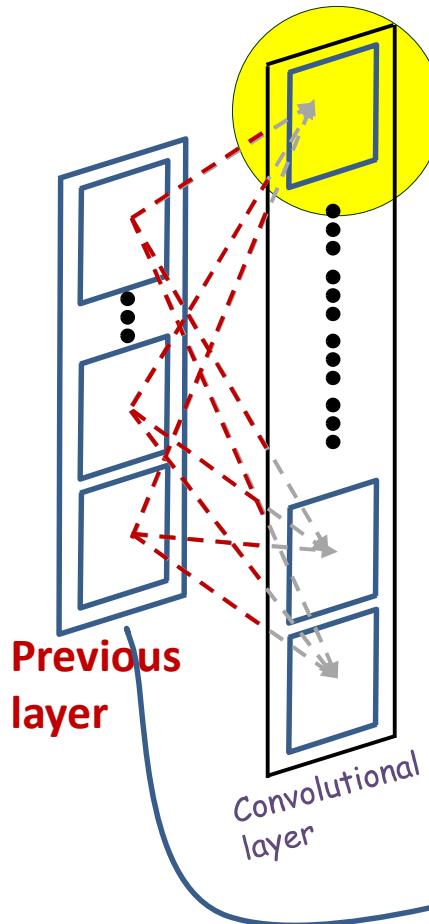


$$\begin{aligned}
 z(l, n, 0, 0) &= w_l(m, n, 1, 2)y(l - 1, m, 1, 2) + \dots \\
 z(l, n, 1, 0) &= w_l(m, n, 1, 2)y(l - 1, m, 2, 2) + \dots \\
 z(l, n, 2, 0) &= w_l(m, n, 1, 2)y(l - 1, m, 3, 2) + \dots \\
 z(l, n, 0, 1) &= w_l(m, n, 1, 2)y(l - 1, m, 1, 3) + \dots \\
 z(l, n, 1, 1) &= w_l(m, n, 1, 2)y(l - 1, m, 2, 3) + \dots \\
 z(l, n, 2, 1) &= w_l(m, n, 1, 2)y(l - 1, m, 3, 3) + \dots \\
 z(l, n, 0, 2) &= w_l(m, n, 1, 2)y(l - 1, m, 1, 4) + \dots
 \end{aligned}$$

$$z(l, n, x, y) = \sum_m \sum_{x'=0}^2 \sum_{y'=0}^2 w_l(m, n, x', y')y(l - 1, m, x + x', y + y') + b_l(n)$$

- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)$

Convolution: the contribution of a single weight

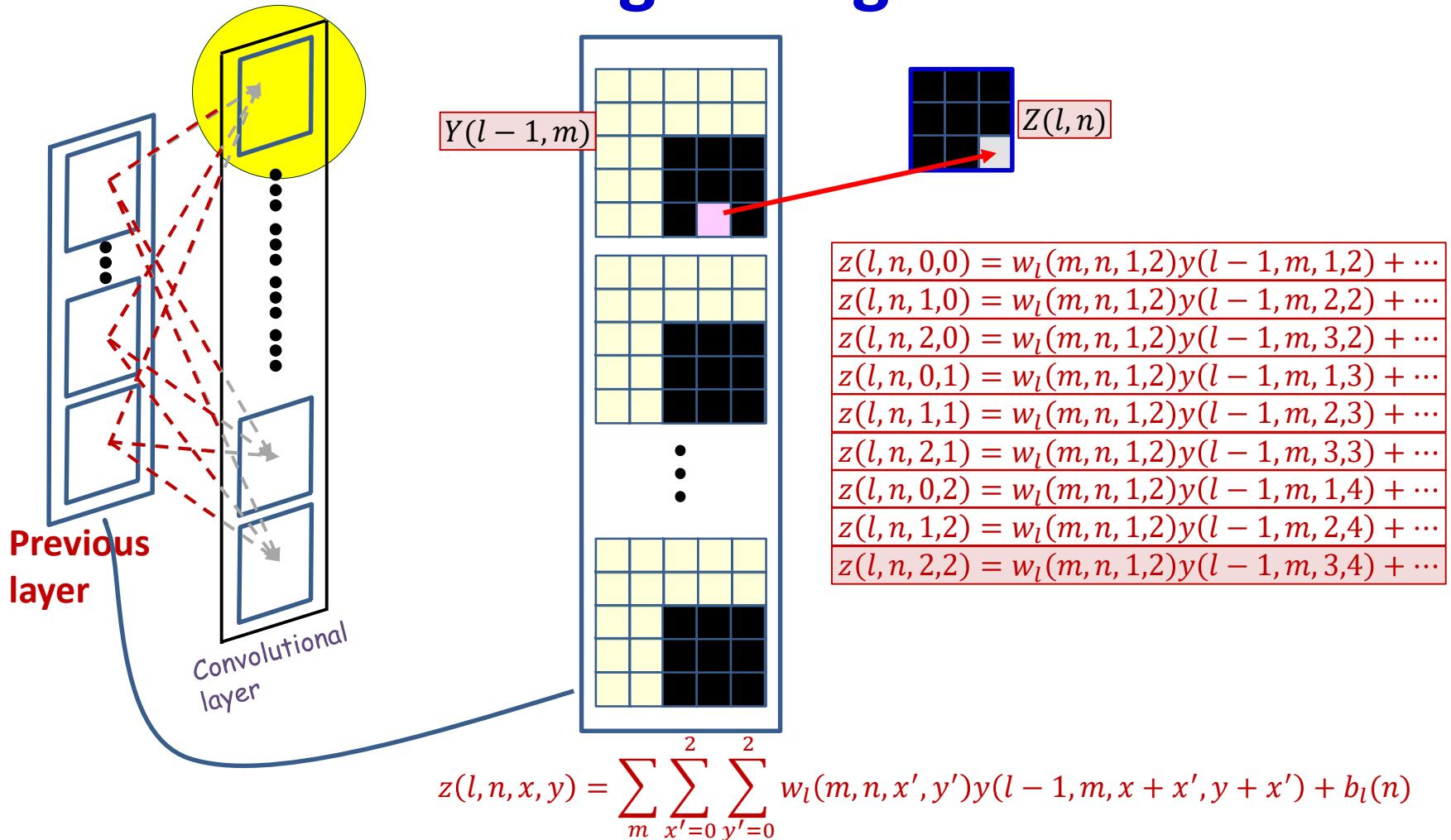


$$\begin{aligned}
 z(l, n, 0, 0) &= w_l(m, n, 1, 2)y(l - 1, m, 1, 2) + \dots \\
 z(l, n, 1, 0) &= w_l(m, n, 1, 2)y(l - 1, m, 2, 2) + \dots \\
 z(l, n, 2, 0) &= w_l(m, n, 1, 2)y(l - 1, m, 3, 2) + \dots \\
 z(l, n, 0, 1) &= w_l(m, n, 1, 2)y(l - 1, m, 1, 3) + \dots \\
 z(l, n, 1, 1) &= w_l(m, n, 1, 2)y(l - 1, m, 2, 3) + \dots \\
 z(l, n, 2, 1) &= w_l(m, n, 1, 2)y(l - 1, m, 3, 3) + \dots \\
 z(l, n, 0, 2) &= w_l(m, n, 1, 2)y(l - 1, m, 1, 4) + \dots \\
 z(l, n, 1, 2) &= w_l(m, n, 1, 2)y(l - 1, m, 2, 4) + \dots
 \end{aligned}$$

$$z(l, n, x, y) = \sum_m \sum_{x'=0}^2 \sum_{y'=0}^2 w_l(m, n, x', y') y(l - 1, m, x + x', y + y') + b_l(n)$$

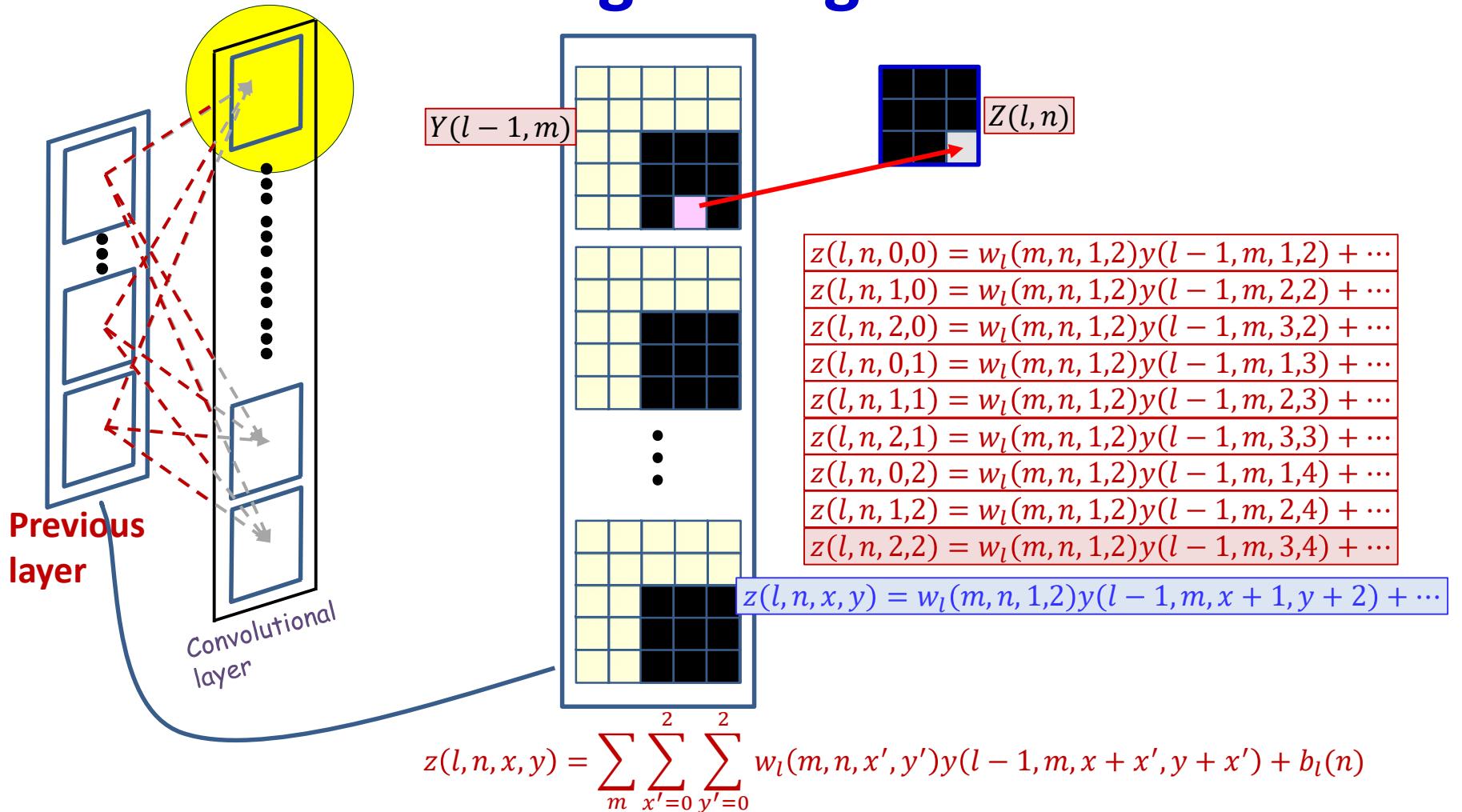
- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)$

Convolution: the contribution of a single weight



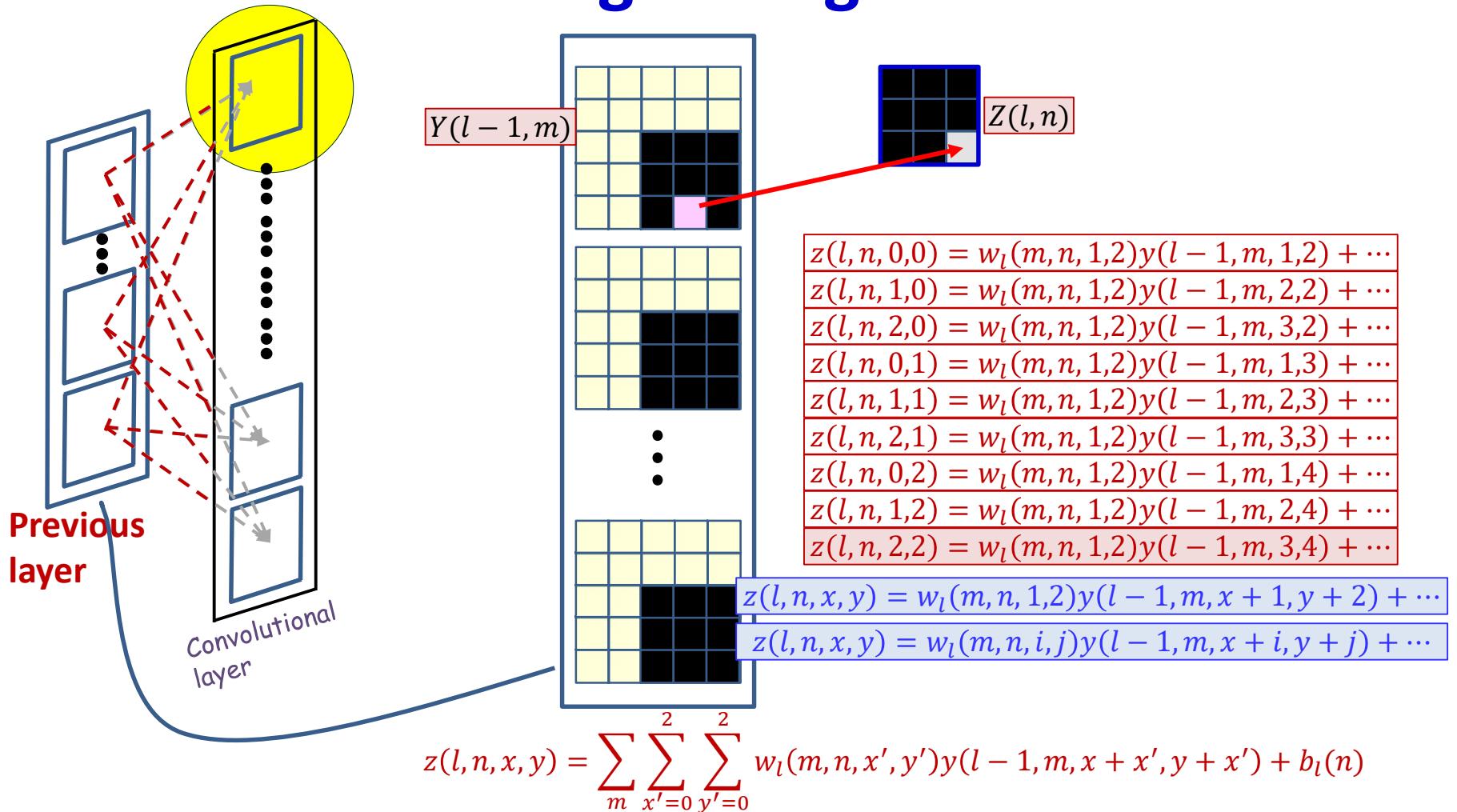
- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)$

Convolution: the contribution of a single weight



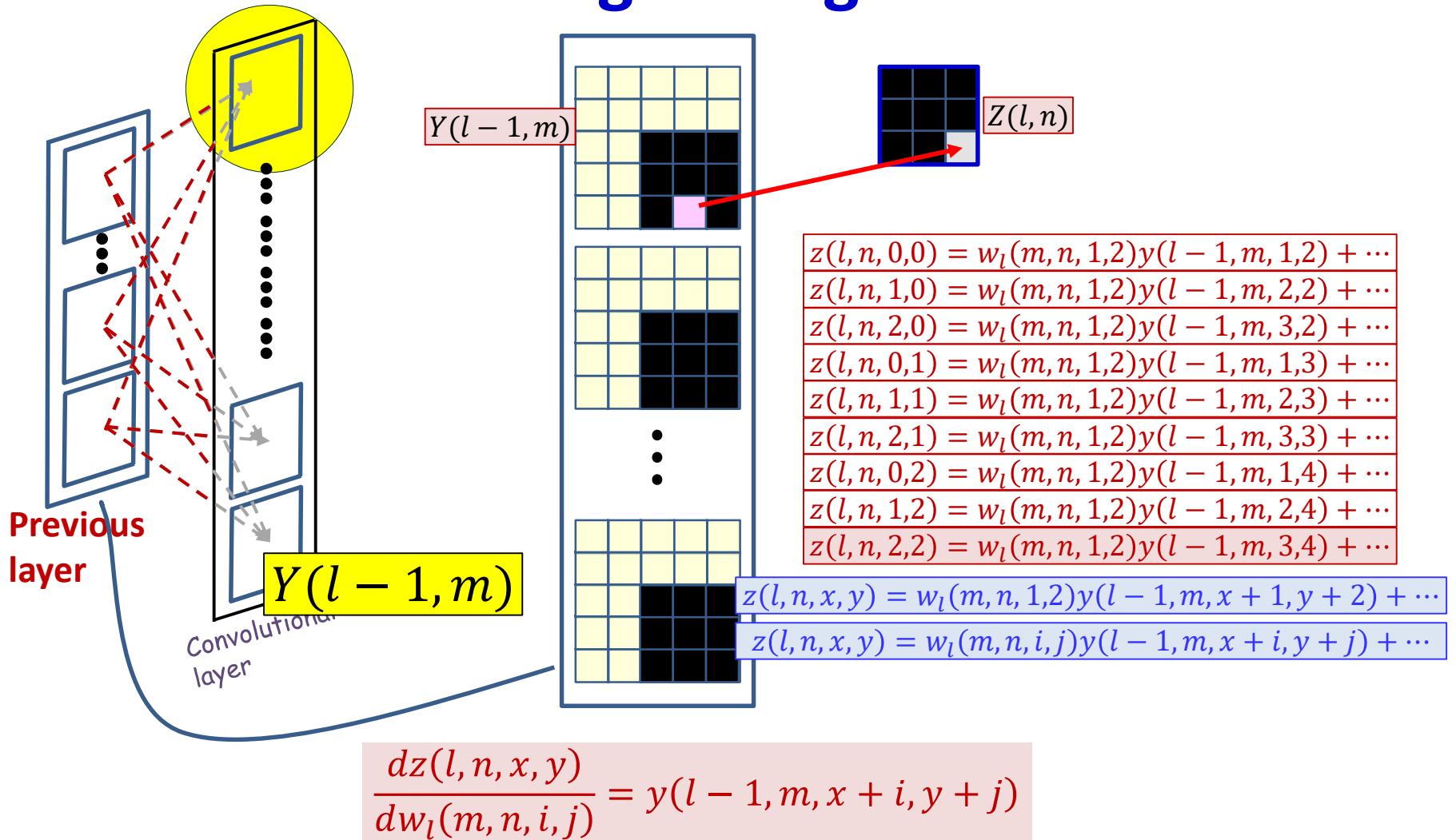
- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)$ ₉₈

Convolution: the contribution of a single weight

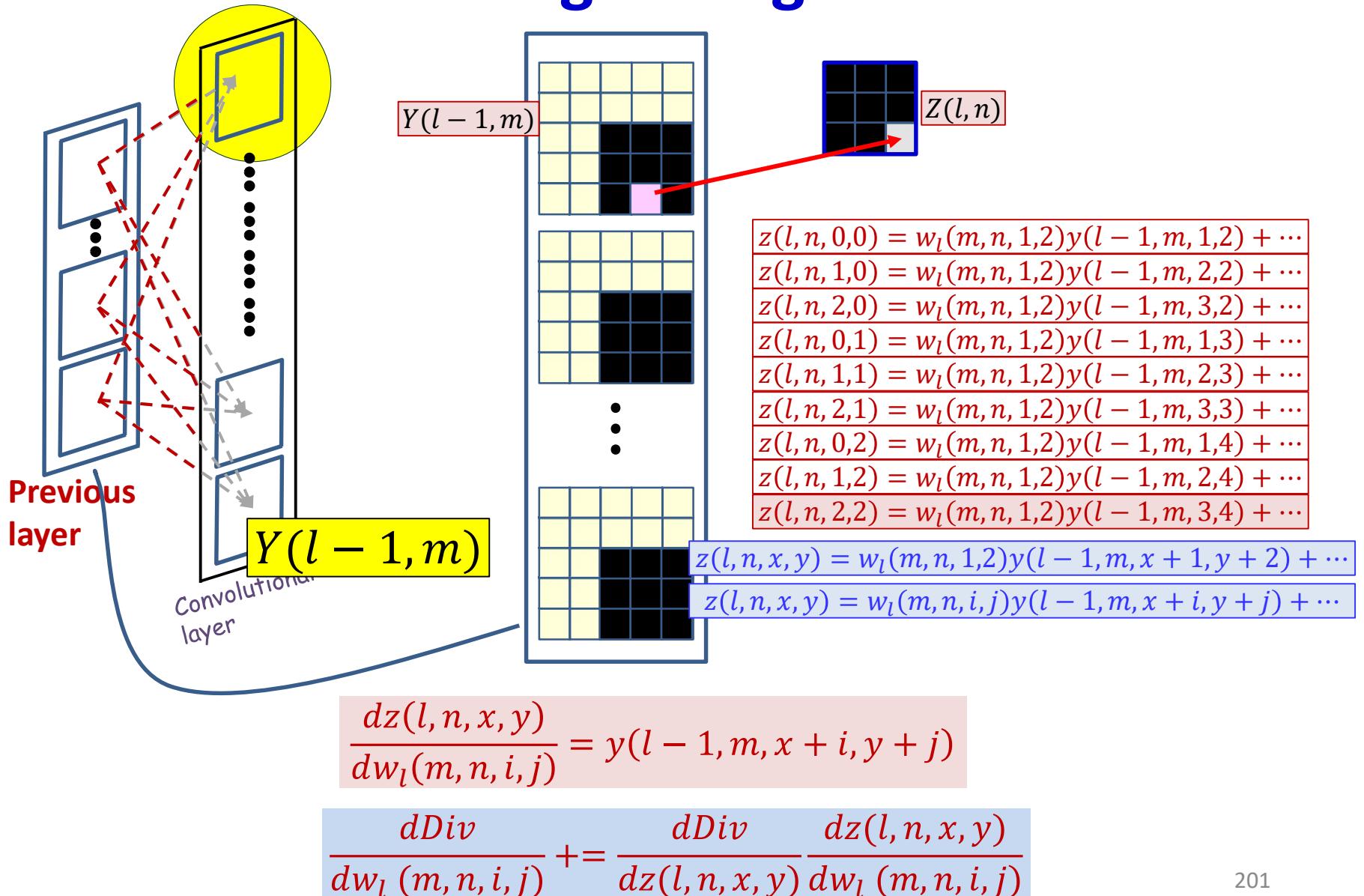


- Each weight $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - Consider the contribution of one filter components: e.g. $w_l(m, n, 1, 2)_{99}$

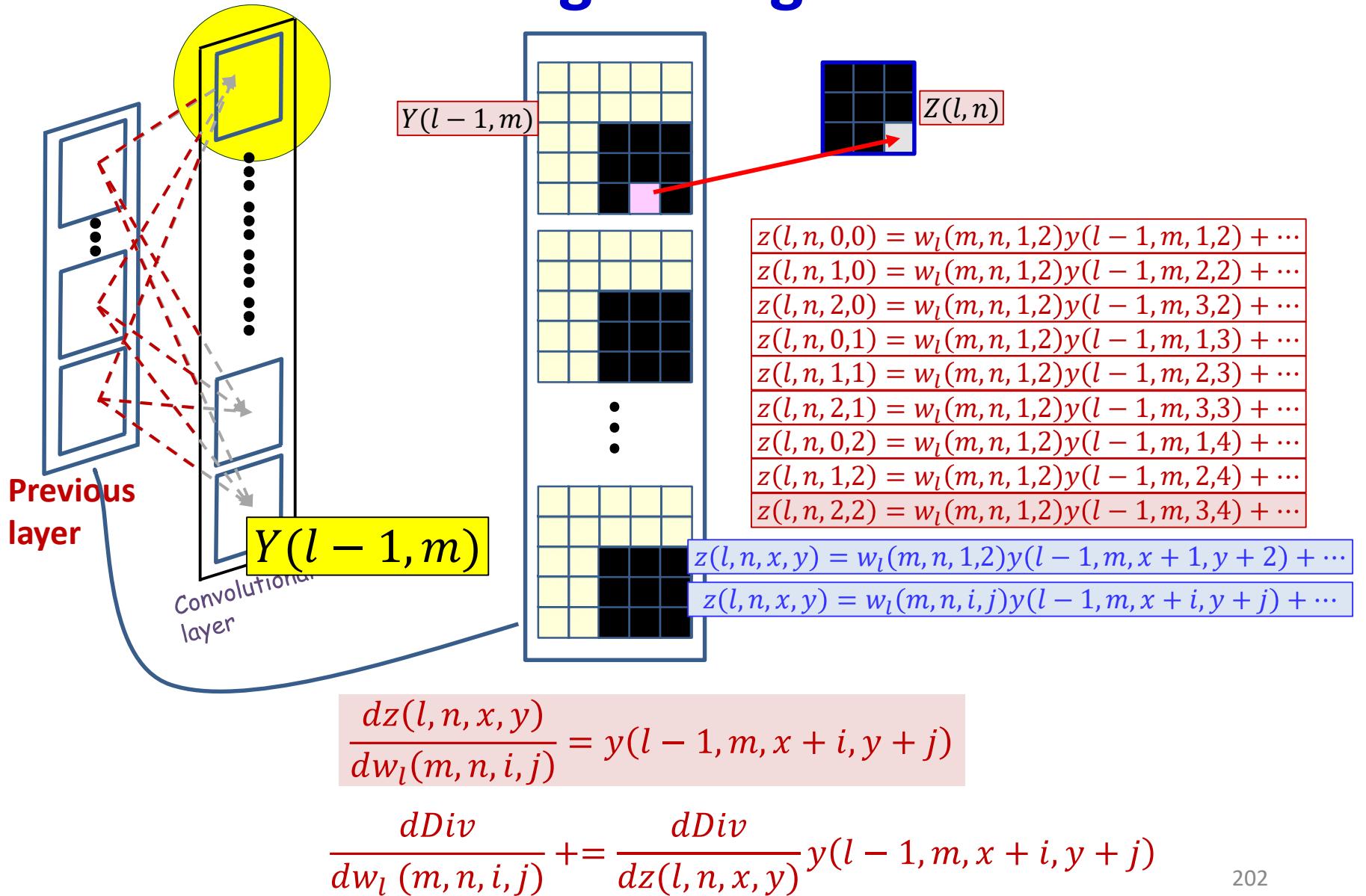
Convolution: the contribution of a single weight



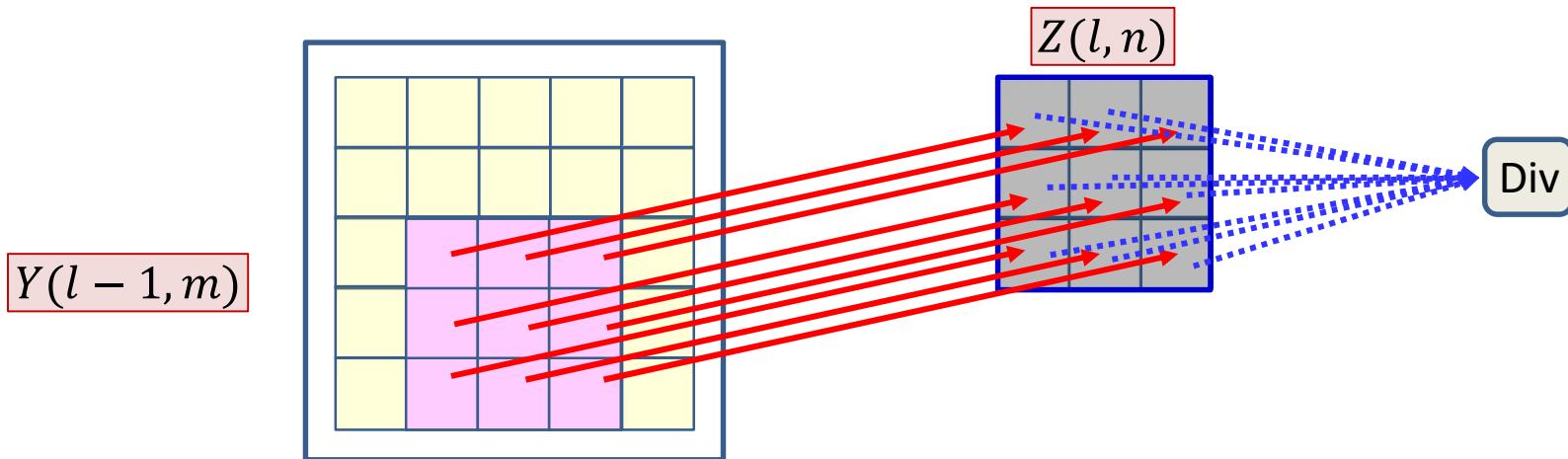
Convolution: the contribution of a single weight



Convolution: the contribution of a single weight



The derivative for a single weight



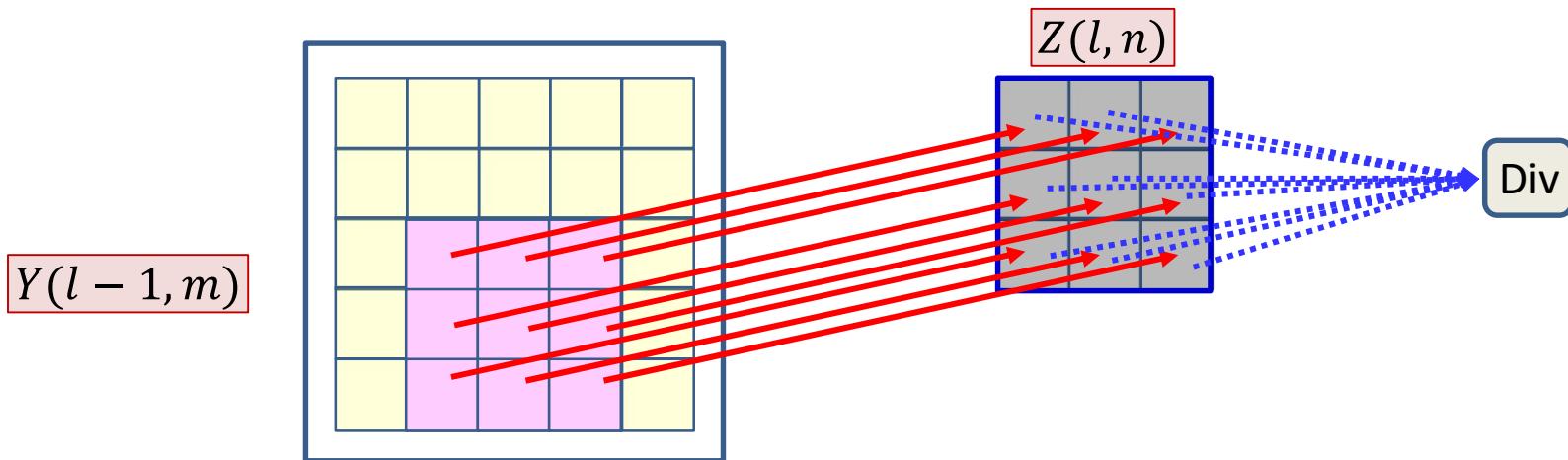
- Each filter component $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - The derivative of each $z(l, n, x, y)$ w.r.t. $w_l(m, n, i, j)$ is given by

$$\frac{dz(l, n, x, y)}{dw_l(m, n, i, j)} = y(l-1, m, x + i, y + j)$$

- The final divergence is influenced by *every* $z(l, n, x, y)$
- The derivative of the divergence w.r.t $w_l(m, n, i, j)$ must sum over all $z(l, n, x, y)$ terms it influences

$$\frac{d\text{Div}}{dw_l(m, n, i, j)} = \sum_{x,y} \frac{d\text{Div}}{dz(l, n, x, y)} \frac{dz(l, n, x, y)}{dw_l(m, n, i, j)}$$

The derivative for a single weight



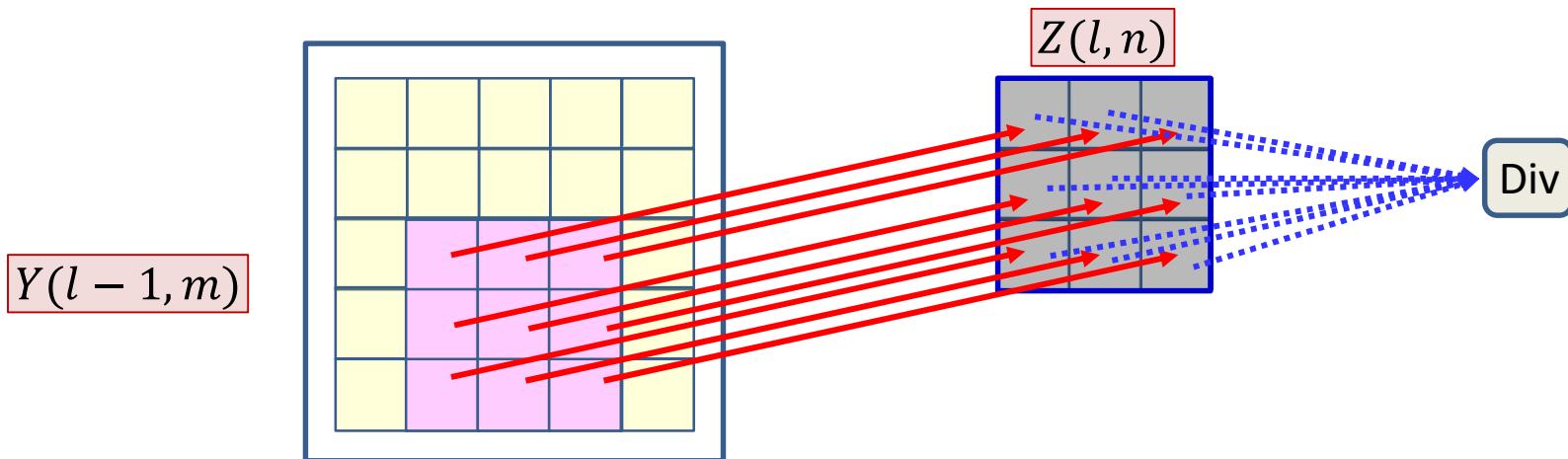
- Each filter component $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - The derivative of each $z(l, n, x, y)$ w.r.t. $w_l(m, n, i, j)$ is given by

$$\frac{dz(l, n, x, y)}{dw_l(m, n, i, j)} = y(l - 1, m, x + i, y + j)$$

- The final divergence is influenced by every $z(l, n, x, y)$
- The derivative w.r.t $w_l(m, n, i, j)$ must sum over all $z(l, n, x, y)$ terms it influences

$$\frac{dDiv}{dw_l(m, n, i, j)} = \sum_{x,y} \frac{dDiv}{dz(l, n, x, y)} \frac{dz(l, n, x, y)}{dw_l(m, n, i, j)}$$

The derivative for a single weight



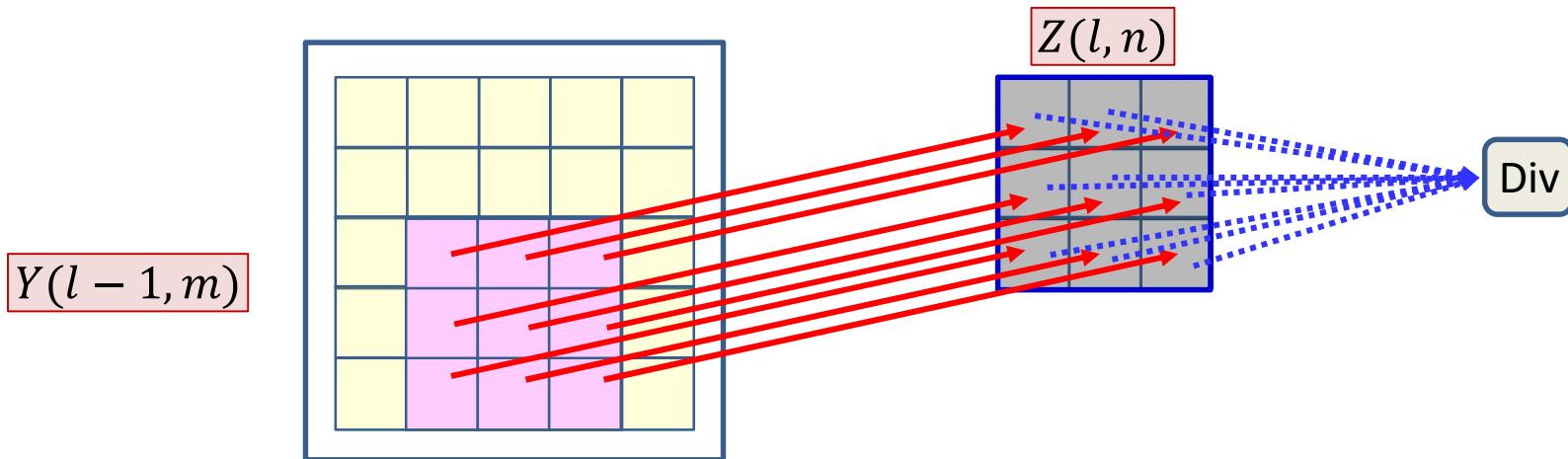
- Each filter component $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - The derivative of each $z(l, n, x, y)$ w.r.t. $w_l(m, n, i, j)$ is given by

$$\frac{dz(l, n, x, y)}{dw_l(m, n, i, j)} = y(l - 1, m, x + i, y + j)$$

- The final divergence is influenced by every $z(l, n, x, y)$
- The derivative w.r.t $w_l(m, n, i, j)$ must sum over all $z(l, n, x, y)$ terms it influences

$$\frac{dDiv}{dw_l(m, n, i, j)} = \sum_{x,y} \frac{dDiv}{dz(l, n, x, y)} \frac{dz(l, n, x, y)}{dw_l(m, n, i, j)}$$

The derivative for a single weight



- Each filter component $w_l(m, n, i, j)$ affects several $z(l, n, x, y)$
 - The derivative of each $z(l, n, x, y)$ w.r.t. $w_l(m, n, i, j)$ is given by

$$\frac{dz(l, n, x, y)}{dw_l(m, n, i, j)} = y(l - 1, m, x + i, y + j)$$

- The final divergence is influenced by *every* $z(l, n, x, y)$
- The derivative of the divergence w.r.t $w_l(m, n, i, j)$ must sum over all $z(l, n, x, y)$ terms it influences

$$\frac{dDiv}{dw_l(m, n, i, j)} = \sum_{x,y} \frac{dDiv}{dz(l, n, x, y)} y(l - 1, m, x + i, y + j)$$

But this too is a convolution

$$\frac{dDiv}{dw_l(m, n, i, j)} = \sum_{x,y} \frac{dDiv}{dz(l, n, x, y)} y(l - 1, m, x + i, y + j)$$

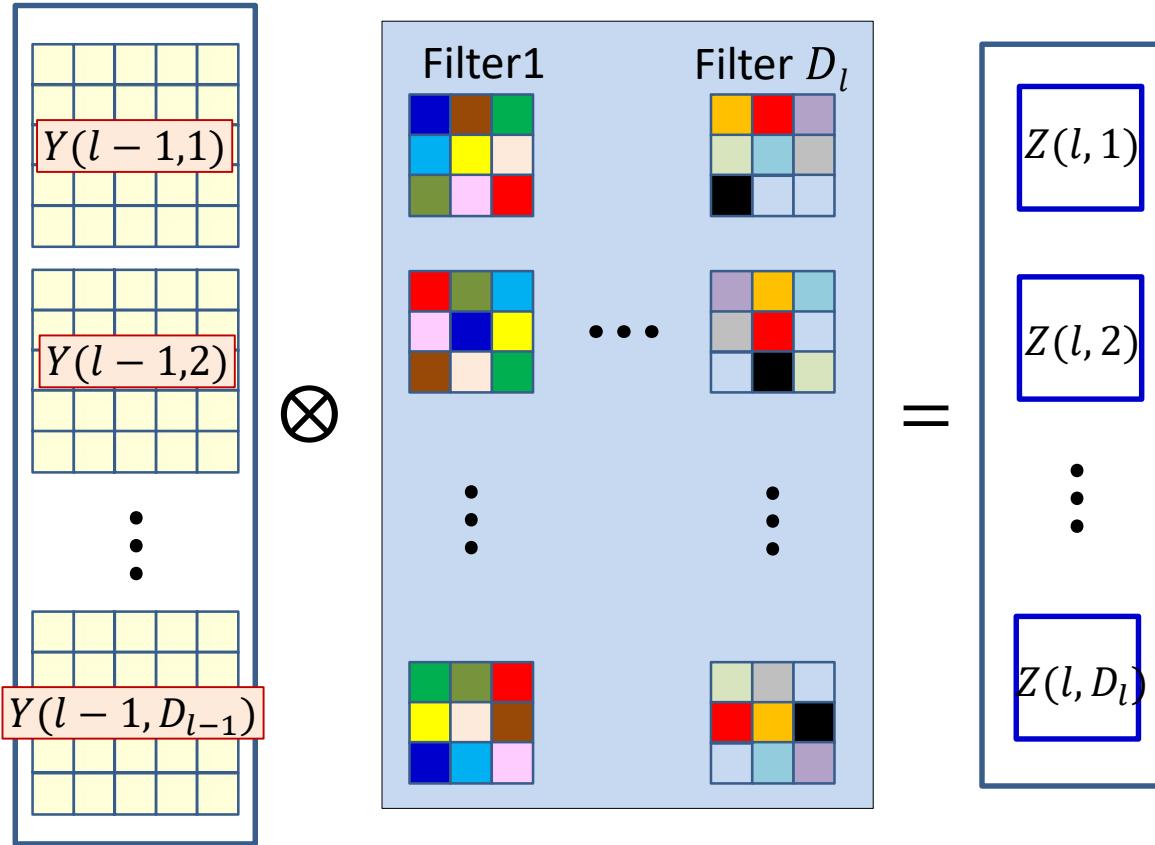
- The derivatives for all components of all filters can be computed directly from the above formula

- In fact it is just a convolution

$$\frac{dDiv}{dw_l(m, n, i, j)} = \frac{dDiv}{dz(l, n)} \otimes y(l - 1, m)$$

- How?

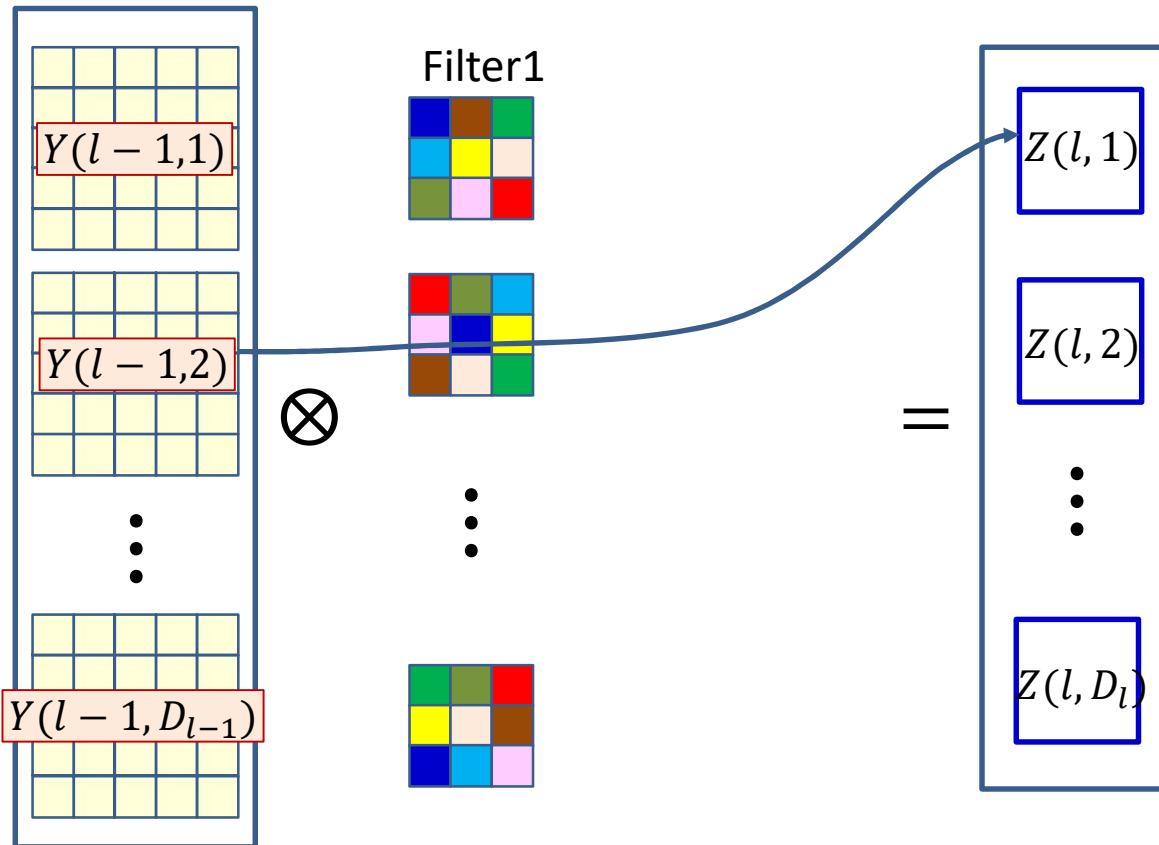
Recap: Convolution



$$z(l, n, x, y) = \sum_m \sum_{i=0}^2 \sum_{j=0}^2 w_l(m, n, i, j) y(l-1, m, x+i, y+j) + b_l(n)$$

- Forward computation: Each filter produces an affine map

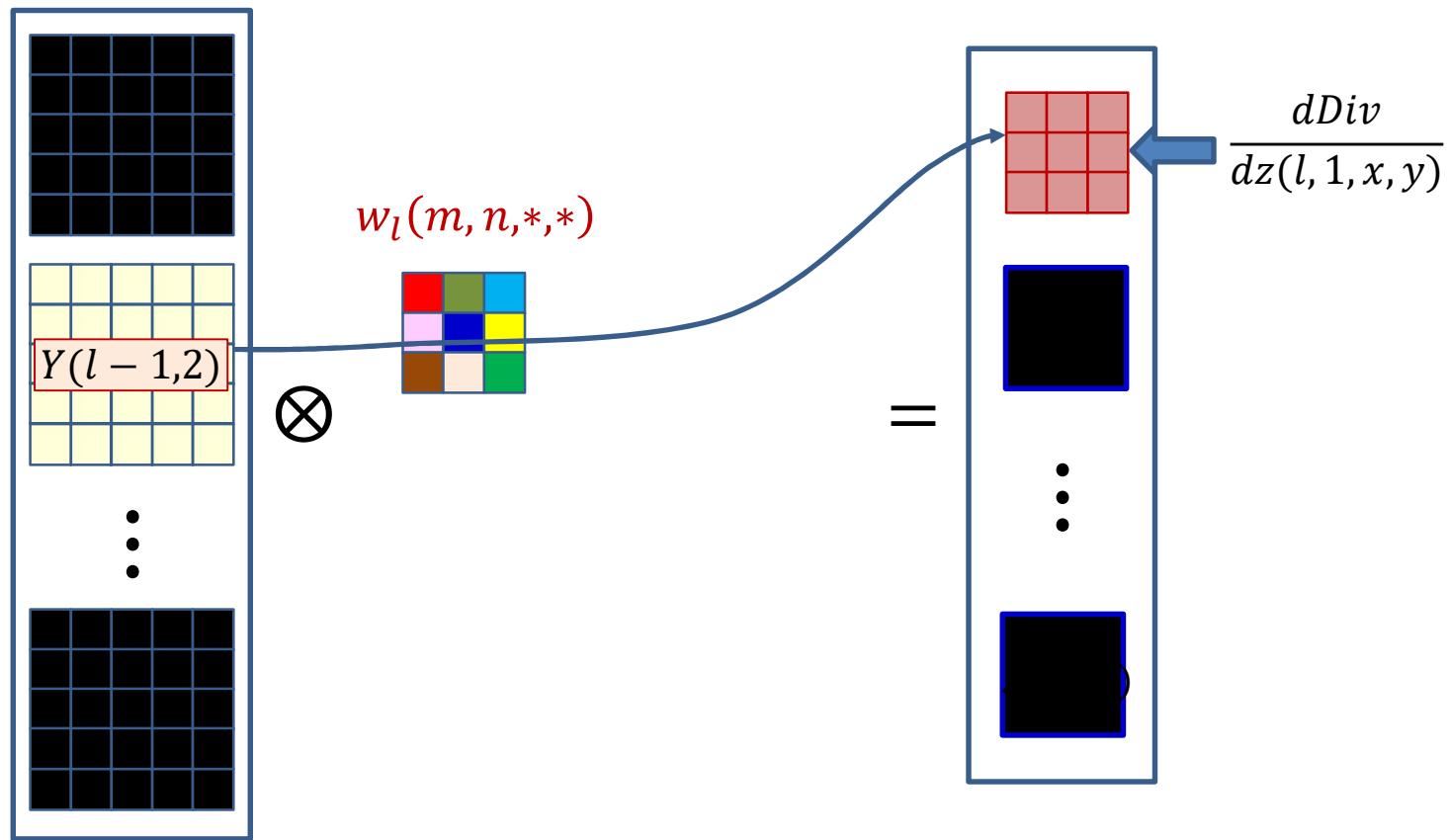
Recap: Convolution



$$z(l, n, x, y) = \sum_m \sum_{i=0}^2 \sum_{j=0}^2 w_l(m, n, i, j) y(l-1, m, x+i, y+j) + b_l(n)$$

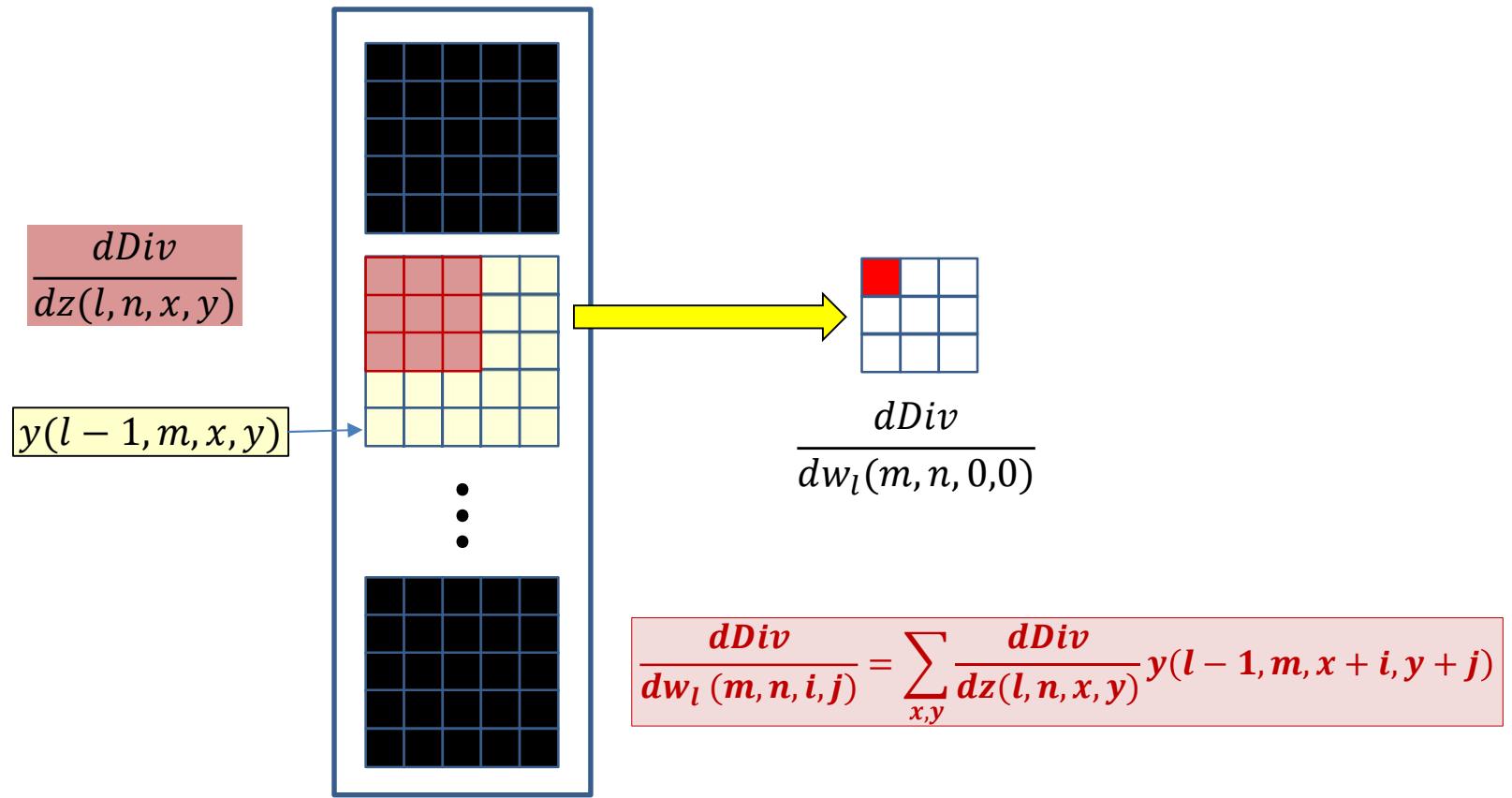
- $Y(l-1, m)$ influences $Z(l, n)$ through $w_l(m, n)$

The filter derivative



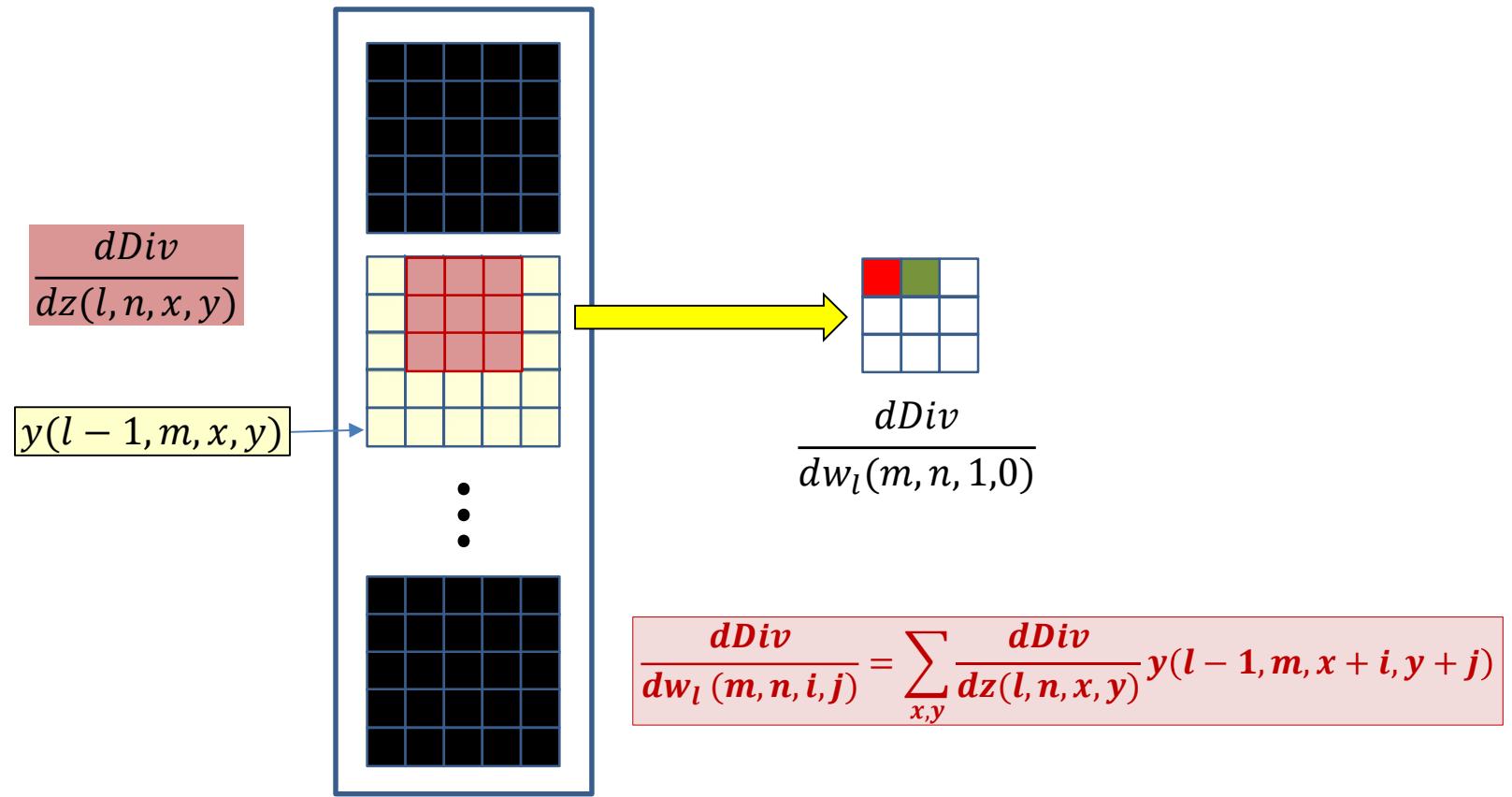
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)$ ¹¹⁰

The filter derivative



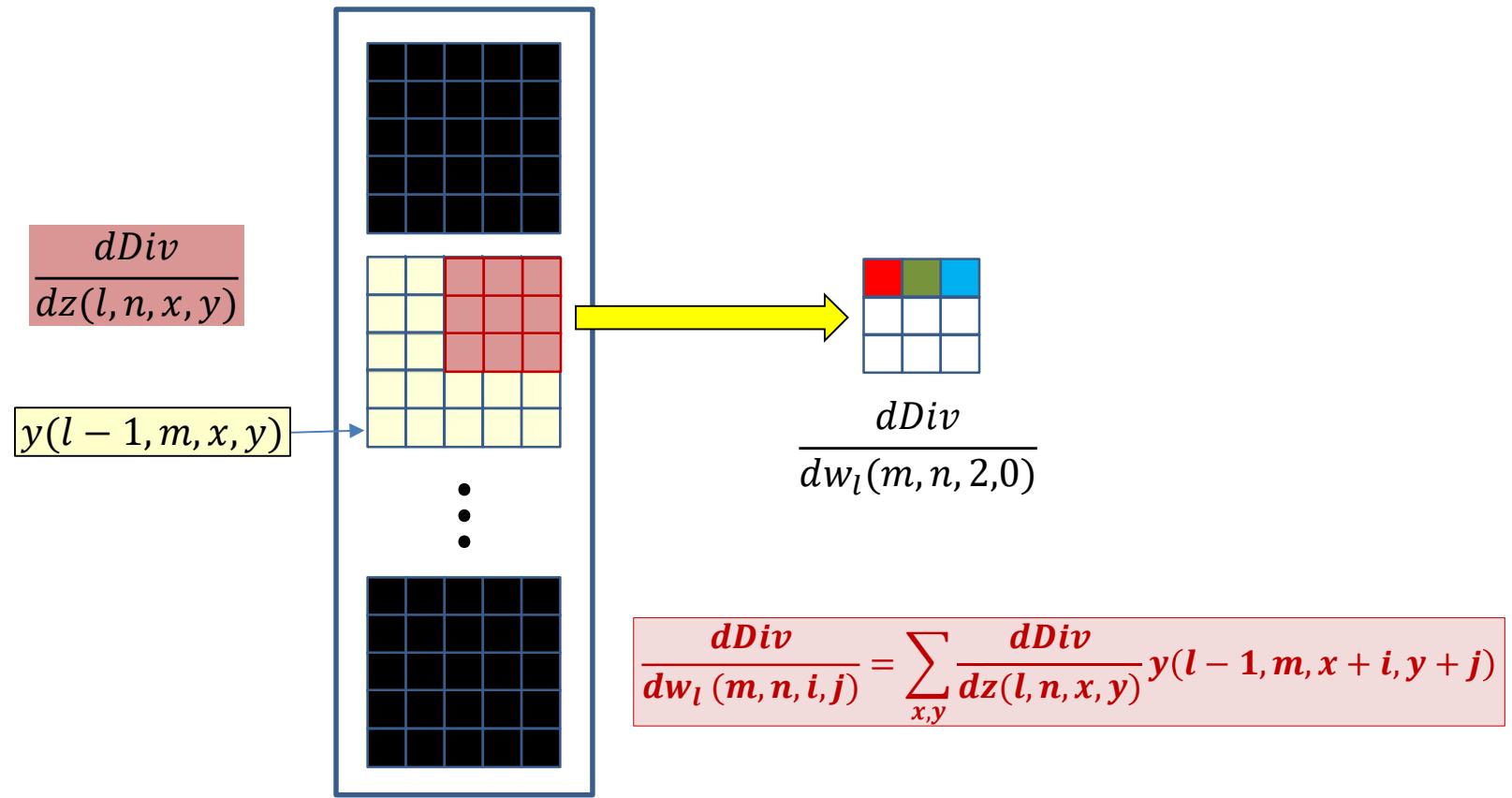
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)^{111}$

The filter derivative



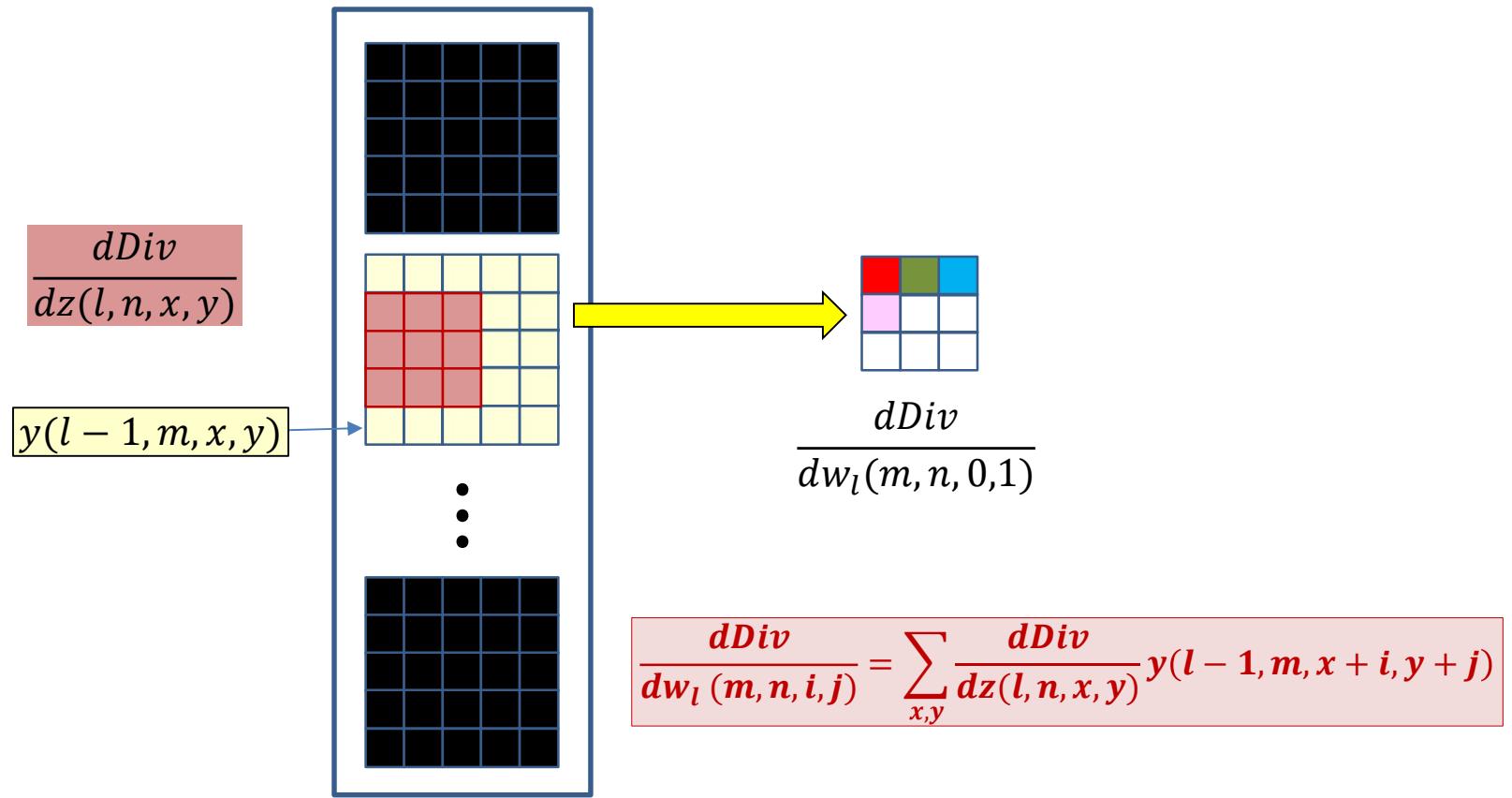
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)^{112}$

The filter derivative



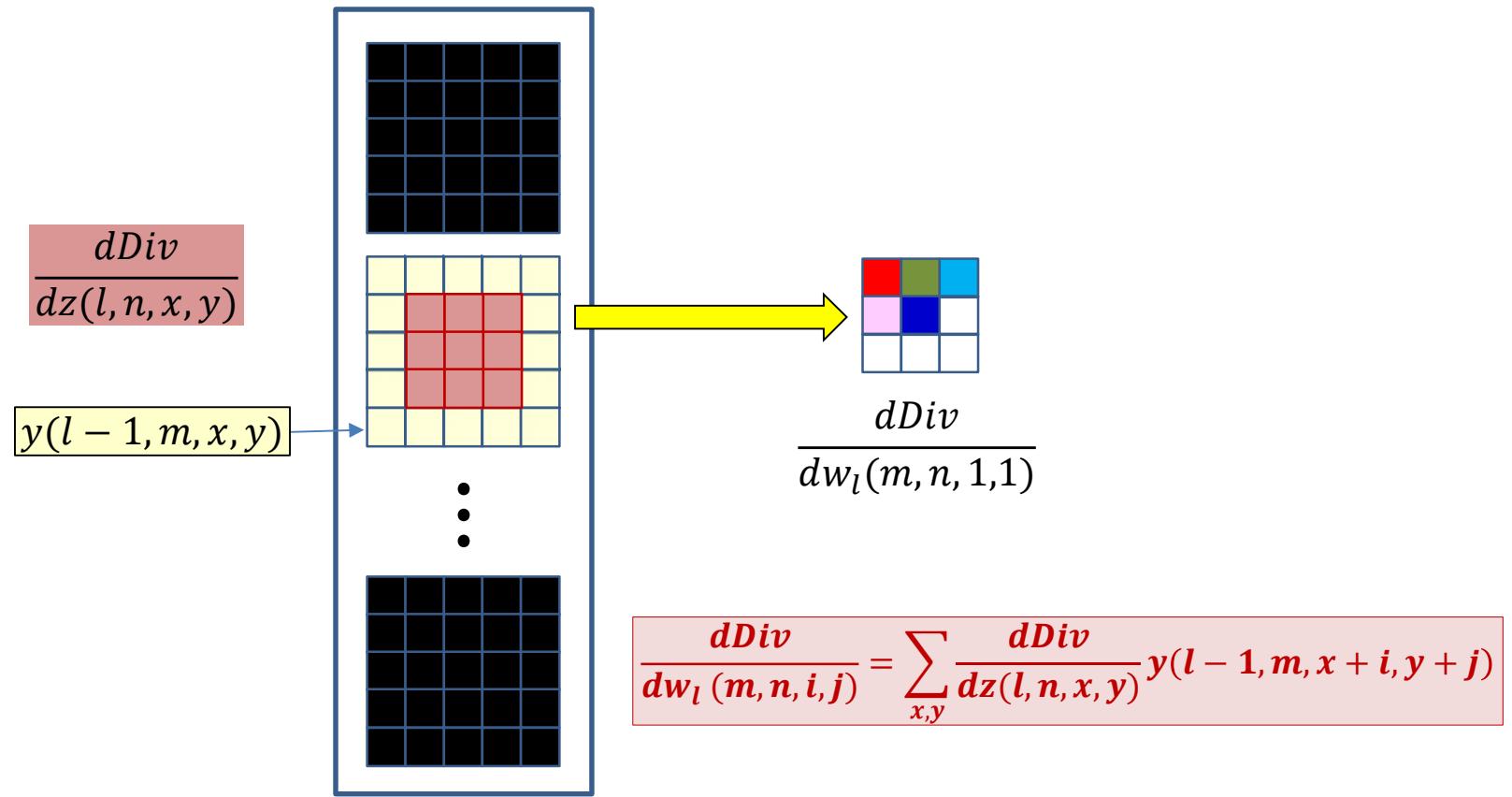
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)^{113}$

The filter derivative



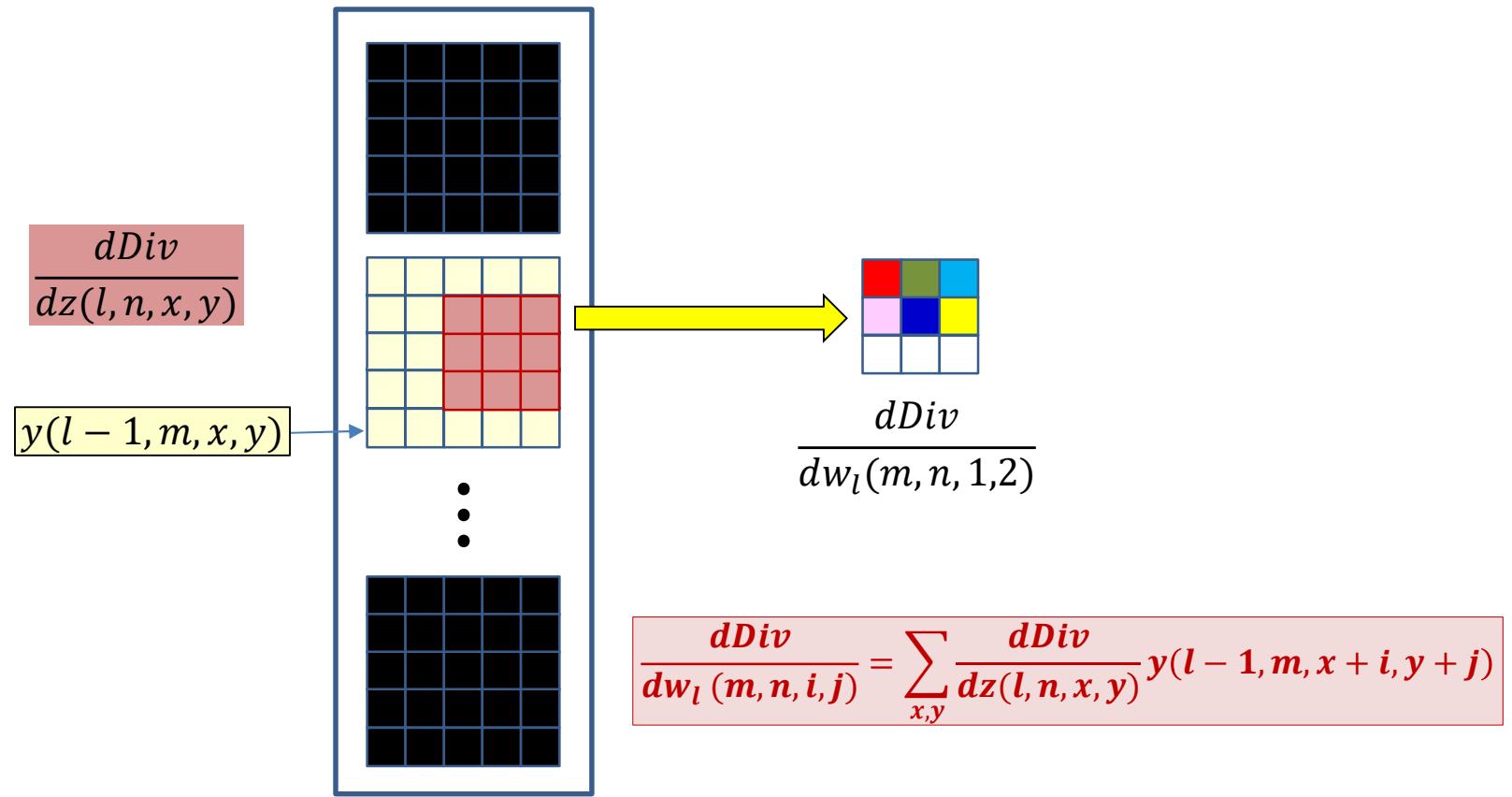
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)$ ²¹⁴

The filter derivative



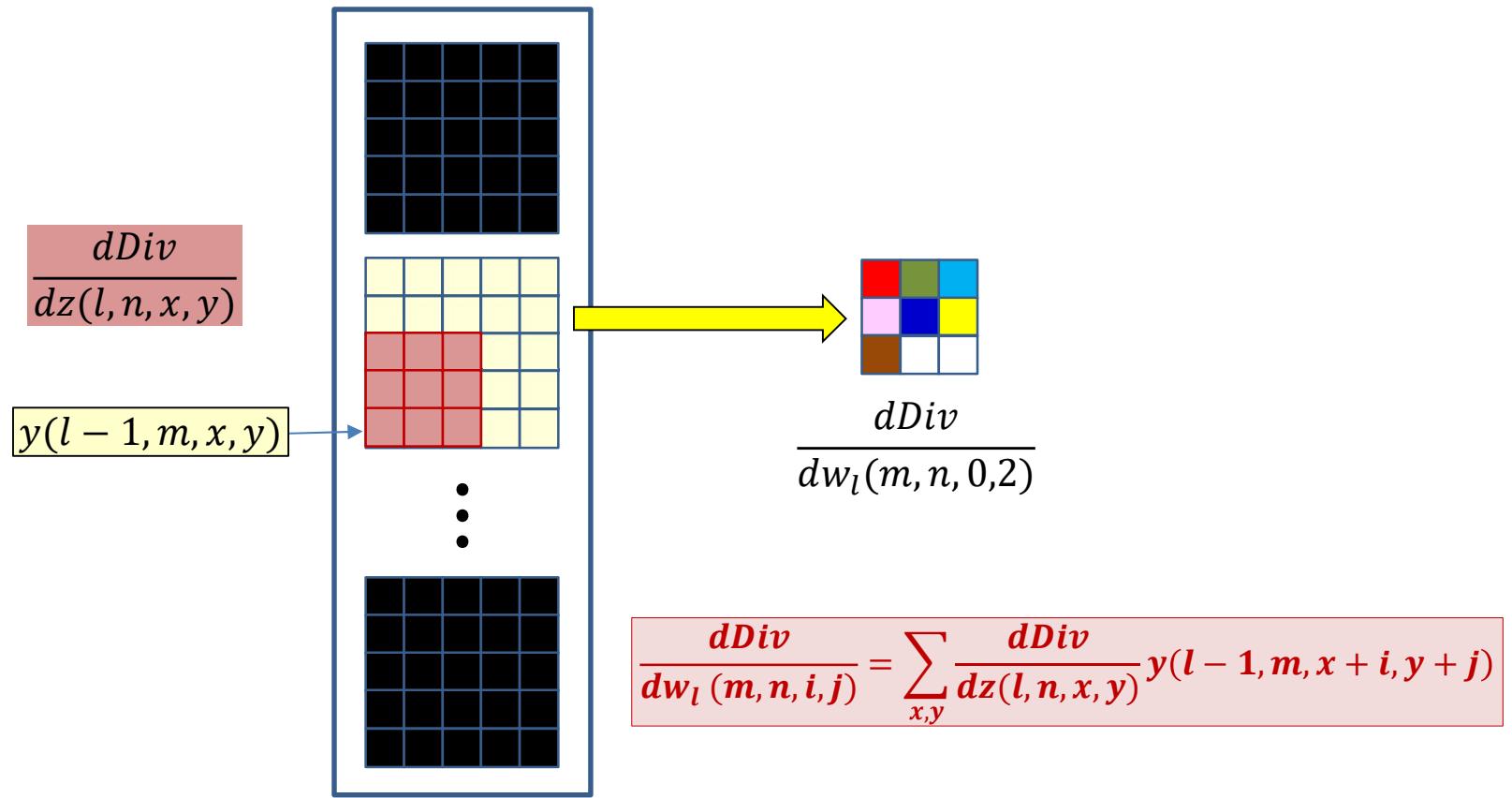
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)$ ²¹⁵

The filter derivative



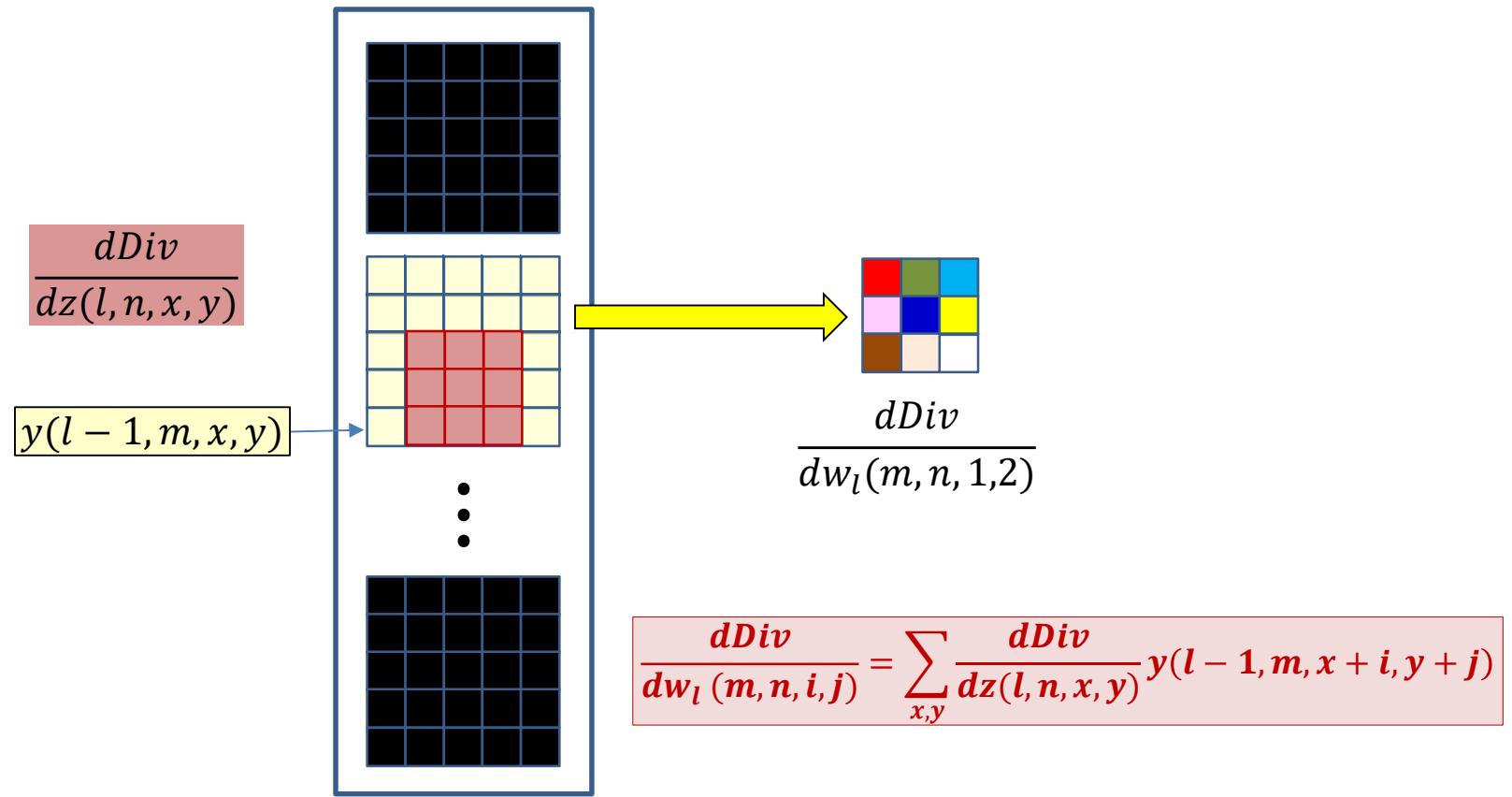
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)^{116}$

The filter derivative



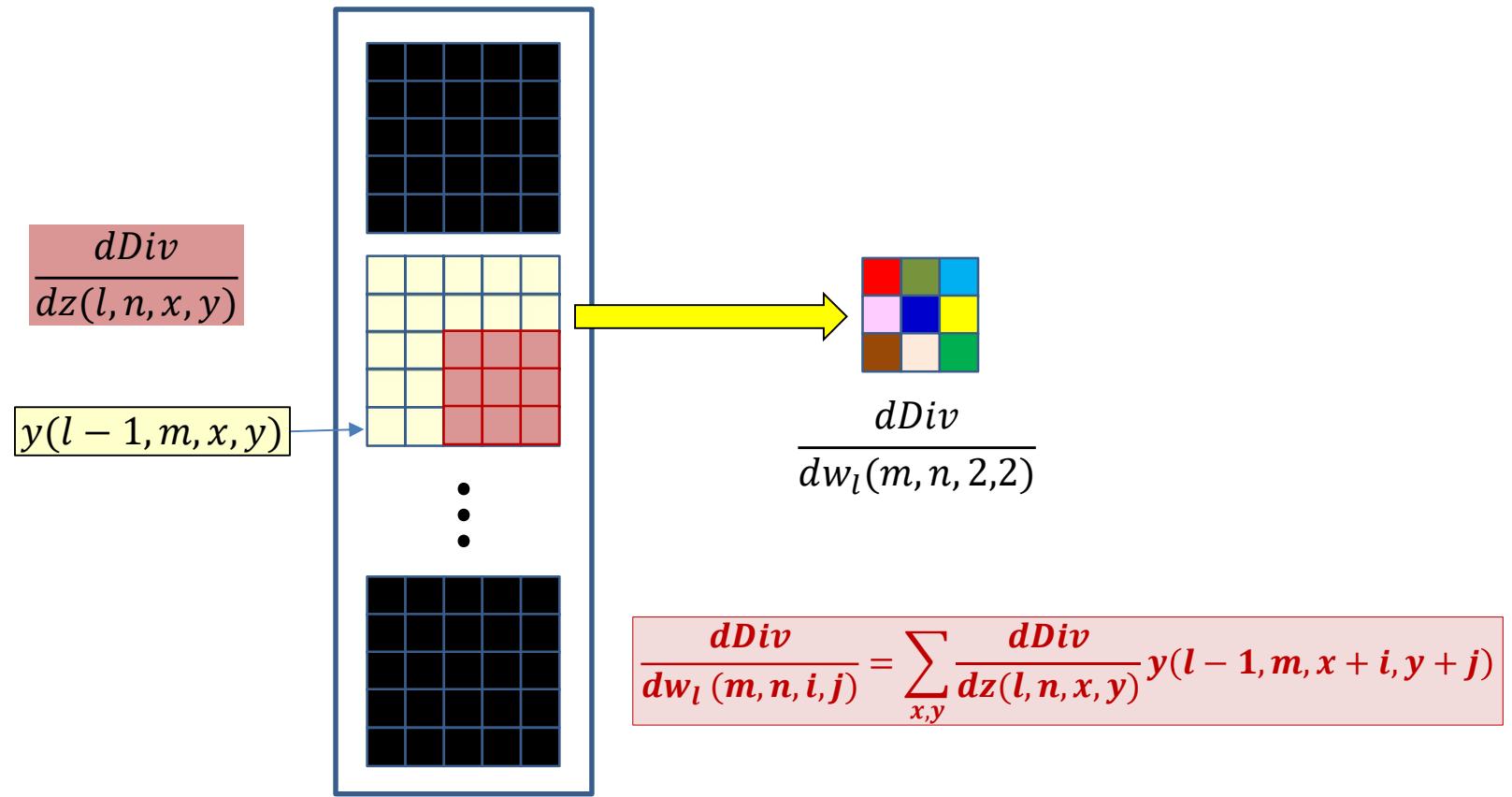
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)^{117}$

The filter derivative



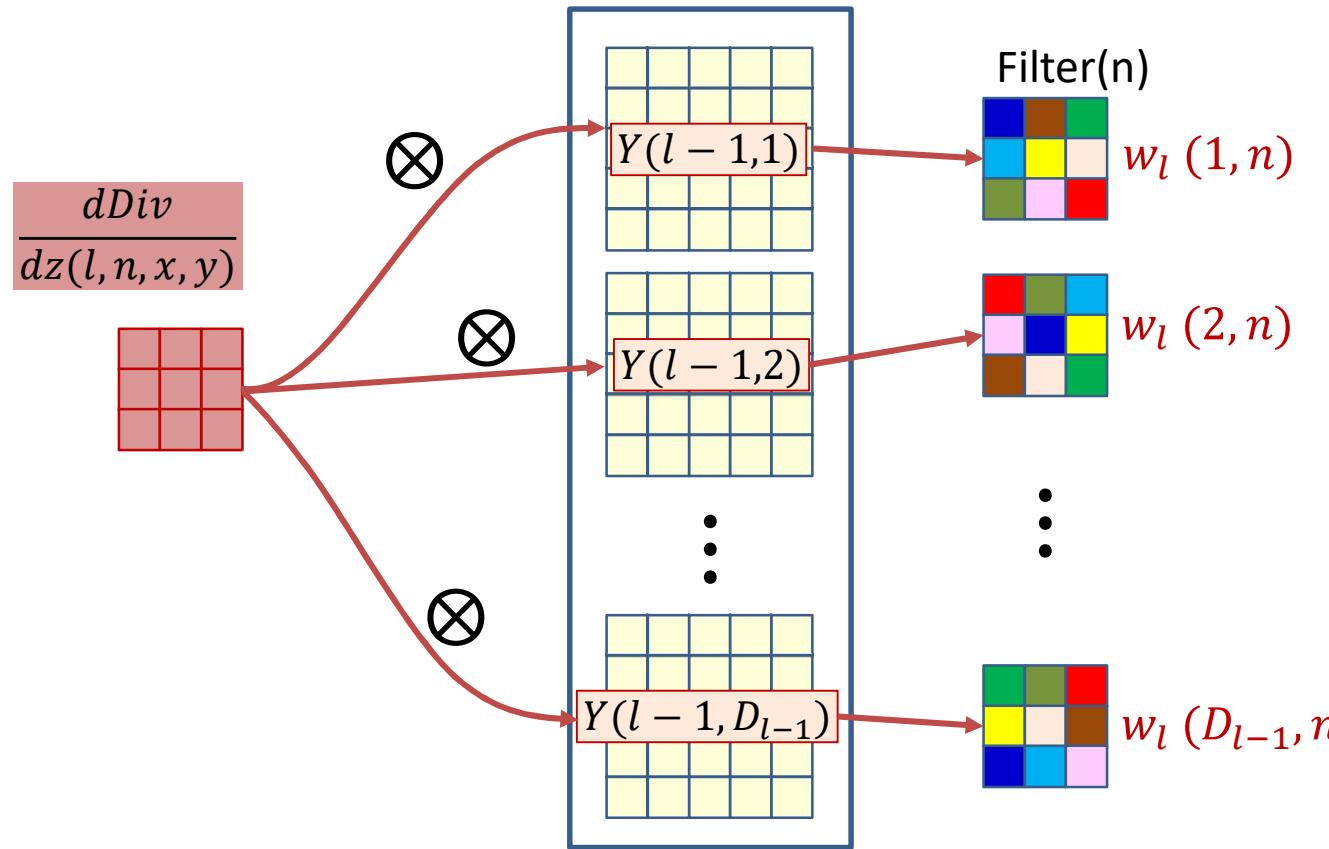
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)$ ²¹⁸

The filter derivative



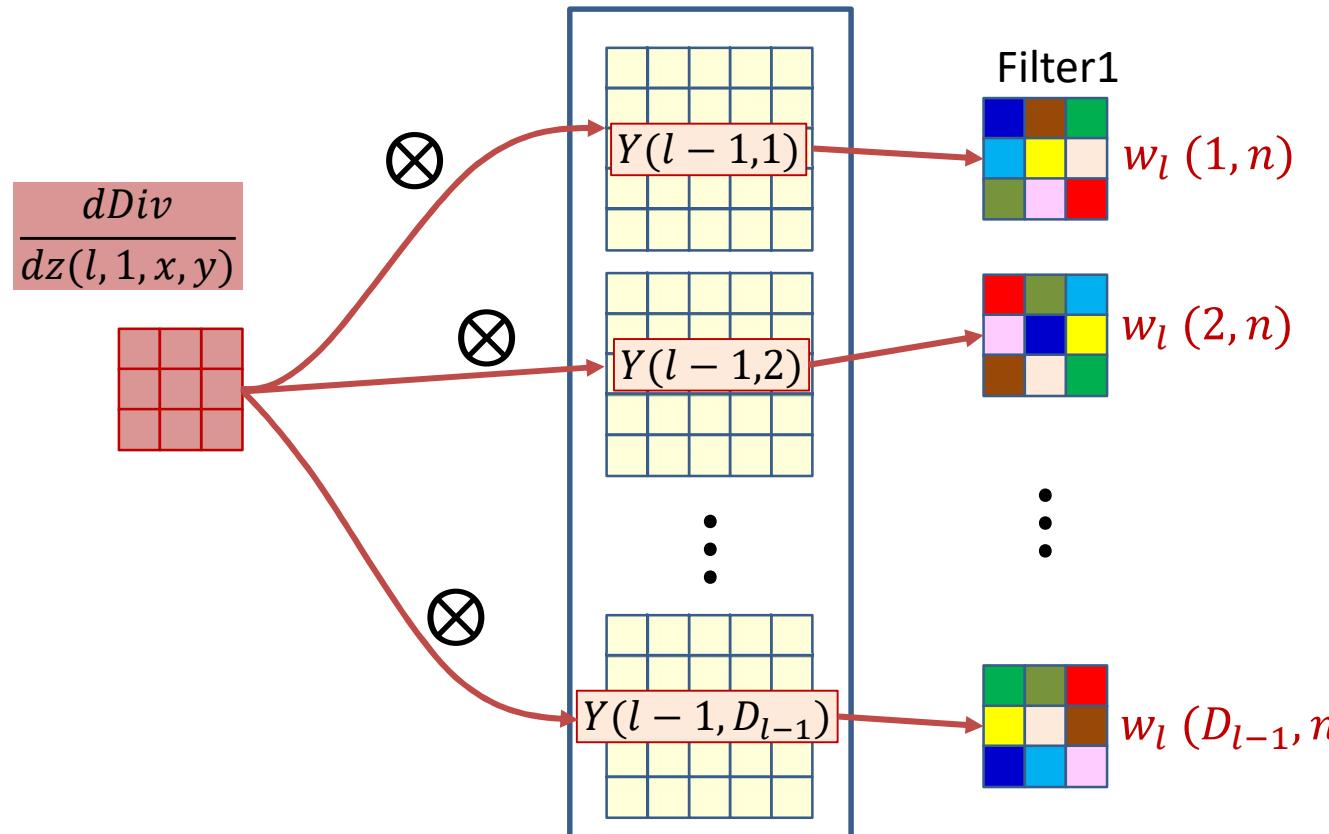
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
 - Must use them to compute the derivative for $w_l(m, n, *, *)^{119}$

The filter derivative



- The derivative of the n^{th} affine map $Z(l, n)$ convolves with every output map $Y(l - 1, m)$ of the $(l - 1)^{\text{th}}$ layer, to get the derivative for $w_l(m, n)$, the m^{th} “plane” of the n^{th} filter

The filter derivative



$$\frac{dDiv}{dw_l(m, n, i, j)} = \sum_{x,y} \frac{dDiv}{dz(l, n, x, y)} y(l - 1, m, x + i, y + j) = \frac{dDiv}{dz(l, n)} \otimes y(l - 1, m)$$

$\frac{dDiv}{dw_l(m, n, i, j)}$ must be upsampled if the stride was greater than 1 in the forward pass

If $Y(l - 1, m)$ was zero padded in the forward pass, it must be zero padded for backprop

Poll 4

- @889

Poll 4

Select all statements that are true about how to compute the derivative of the divergence w.r.t l^{th} layer filters using backpropagation

- **The derivative for the m^{th} plane of the n^{th} filter is computed by convolving the m^{th} input ($l-1^{th}$) layer map with the n^{th} output (l^{th}) layer affine derivative map**
- The output map must be flipped left-right/up-down before convolution
- **If the forward convolution has a stride S , the derivative maps must be upsampled by S prior to convolution**
- If the forward convolution has stride S , the backpropagation convolution must also have a stride S

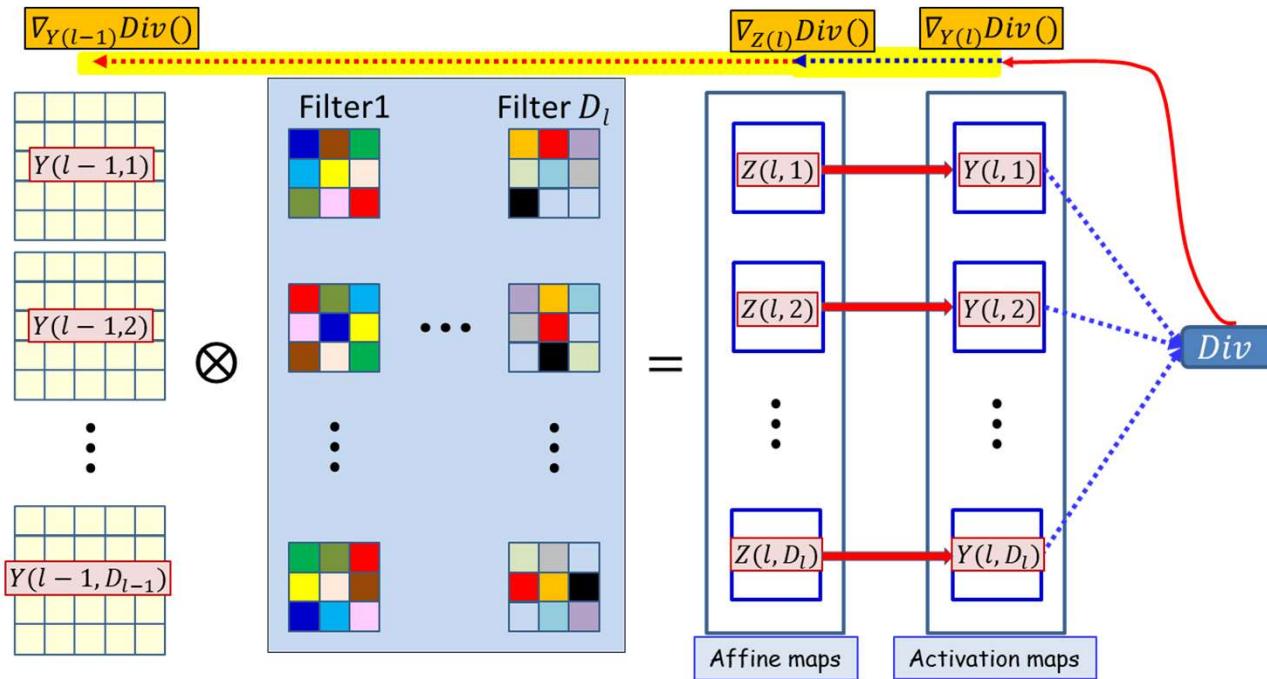
Derivatives for the filters at layer l :

Vector notation

```
# The weight  $W(l, j)$  is a 3D  $D_{l-1} \times K_l \times K_l$ 
# Assuming that derivative maps have been upsampled
#   if stride > 1
# Also assuming y map has been zero-padded if this was
#   also done in the forward pass
```

```
for n = 1:Dl
  for x = 1:Kl
    for y = 1:Kl
      for m = 1:Dl-1
        dw(l,m,n,x,y) = dz(l,n,:,:,:) . #dot product
                                         y(l-1,m,x:x+Kl-1,y:y+Kl-1)
```

Backpropagation: Convolutional layers



- **For convolutional layers:**



How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$



How to compute the derivative w.r.t. $Y(l - 1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$

CNN: Forward

```
Y(0,:,:,:, :) = Image
for l = 1:L  # layers operate on vector at (x,y)
    for x = 1:Wl-1-Kl+1
        for y = 1:Hl-1-Kl+1
            for j = 1:Dl
                z(l,j,x,y) = 0
                for i = 1:Dl-1
                    for x' = 1:Kl
                        for y' = 1:Kl
                            z(l,j,x,y) += w(l,j,i,x',y')
                            Y(l-1,i,x+x'-1,y+y'-1)
                Y(l,j,x,y) = activation(z(l,j,x,y))
Y = softmax( Y(L,:,:1,1)..Y(L,:,:W-K+1,H-K+1) )
```

Switching to 1-based
indexing with appropriate
adjustments

Backward layer l

```
dw(l) = zeros(DlxDl-1xKlxKl)
dY(l-1) = zeros(Dl-1xWl-1xHl-1)
for x = Wl-1-Kl+1:downto:1
    for y = Hl-1-Kl+1:downto:1
        for j = Dl:downto:1
            dz(l,j,x,y) = dY(l,j,x,y).f'(z(l,j,x,y))
            for i = Dl-1:downto:1
                for x' = Kl:downto:1
                    for y' = Kl:downto:1
                        dY(l-1,i,x+x'-1,y+y'-1) +=
                            w(l,j,i,x',y')dz(l,j,x,y)
                        dw(l,j,i,x',y') +=
                            dz(l,j,x,y)Y(l-1,i,x+x'-1,y+y'-1)
```

Complete Backward (no pooling)

```
dY(L) = dDiv/dY(L)
for l = L:downto:1  # Backward through layers
    dw(l) = zeros(DlxDl-1xKlxKl)
    dY(l-1) = zeros(Dl-1xWl-1xHl-1)
    for x = Wl-1-Kl+1:downto:1
        for y = Hl-1-Kl+1:downto:1
            for j = Dl:downto:1
                dz(l,j,x,y) = dY(l,j,x,y).f'(z(l,j,x,y))
                for i = Dl-1:downto:1
                    for x' = Kl:downto:1
                        for y' = Kl:downto:1
                            dY(l-1,i,x+x'-1,y+y'-1) +=
                                w(l,j,i,x',y')dz(l,j,x,y)
                            dw(l,j,i,x',y') +=
                                dz(l,j,x,y)y(l-1,i,x+x'-1,y+y'-1)
```

Complete Backward (no pooling)

```
dY(L) = dDiv/dY(L)
for l = L:downto:1    # Backward through layers
    dw(l) = zeros(DlxDl-1xKlxKl)
    dY(l-1) = zeros(Dl-1xWl-1xHl-1)
    for x = Wl-1-Kl+1:downto:1
        for y = Hl-1-Kl+1:downto:1
            for j = Dl:downto:1
                dz(l,j,x,y) = dY(l,j,x,y).f'(z(l,j,x,y))
                for i = Dl-1:downto:1
                    for x' = Kl:downto:1
                        for y' = Kl:downto:1
                            dY(l-1,i,x+x'-1,y+y'-1) +=
                                w(l,j,i,x',y')dz(l,j,x,y)
                            dw(l,j,i,x',y') +=
                                dz(l,j,x,y)y(l-1,i,x+x'-1,y+y'-1)
```

Multiple ways of recasting this as tensor/ vector operations.

Will not discuss here

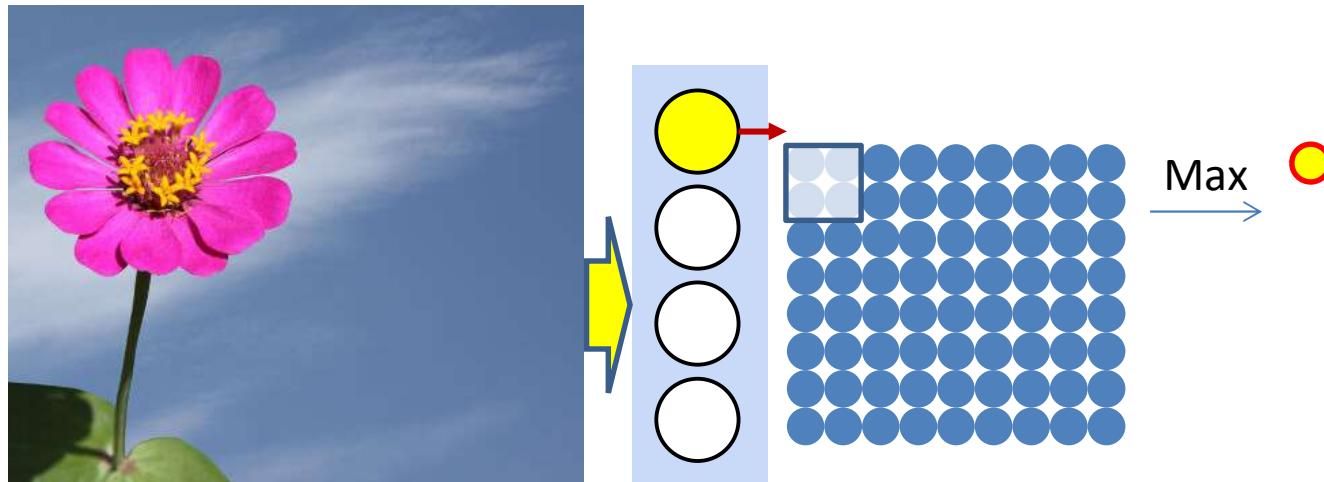
Complete Backward (with strides)

```
dY(L) = dDiv/dY(L)
for l = L:downto:1  # Backward through layers
    dw(l) = zeros(DlxDl-1xKlxKl)
    dY(l-1) = zeros(Dl-1xWl-1xHl-1)
    for x = Wl:downto:1
        m = (x-1) stride
        for y = Hl:downto:1
            n = (y-1) stride
            for j = Dl:downto:1
                dz(l,j,x,y) = dY(l,j,x,y).f'(z(l,j,x,y))
                for i = Dl-1:downto:1
                    for x' = Kl:downto:1
                        for y' = Kl:downto:1
                            dY(l-1,i,m+x',n+y') +=
                                w(l,j,i,x',y')dz(l,j,x,y)
                            dw(l,j,i,x',y') +=
                                dz(l,j,x,y)y(l-1,i,m+x',n+y')
```

Backpropagation: Convolutional and Pooling layers

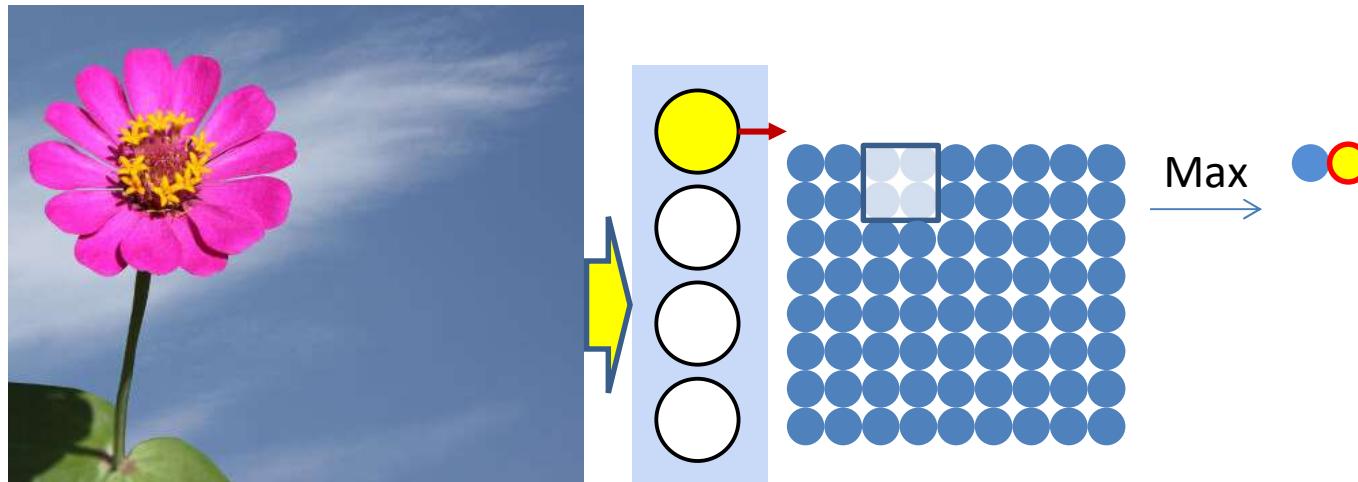
- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
 - Obtained as a result of backpropagating through the flat MLP
- **Required:**
 - **For convolutional layers:**
 - How to compute the derivatives w.r.t. the affine combination $Z(l)$ maps from the activation output maps $Y(l)$
 - How to compute the derivative w.r.t. $Y(l - 1)$ and $w(l)$ given derivatives w.r.t. $Z(l)$
 - **For pooling layers:**
 - How to compute the derivative w.r.t. $Y(l - 1)$ given derivatives w.r.t. $Y(l)$

Pooling and downsampling



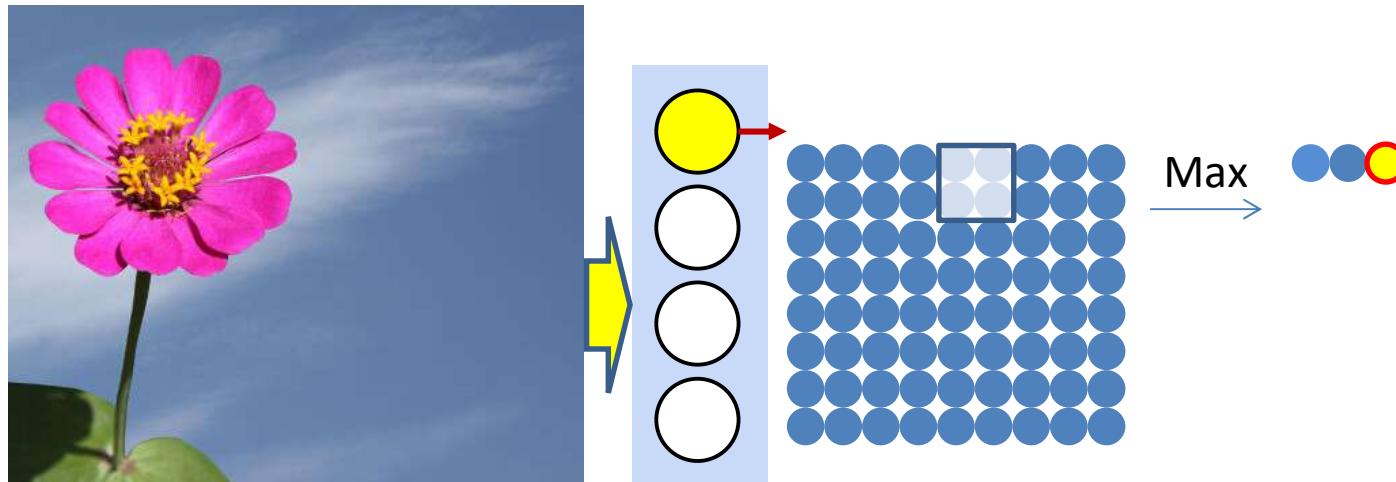
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



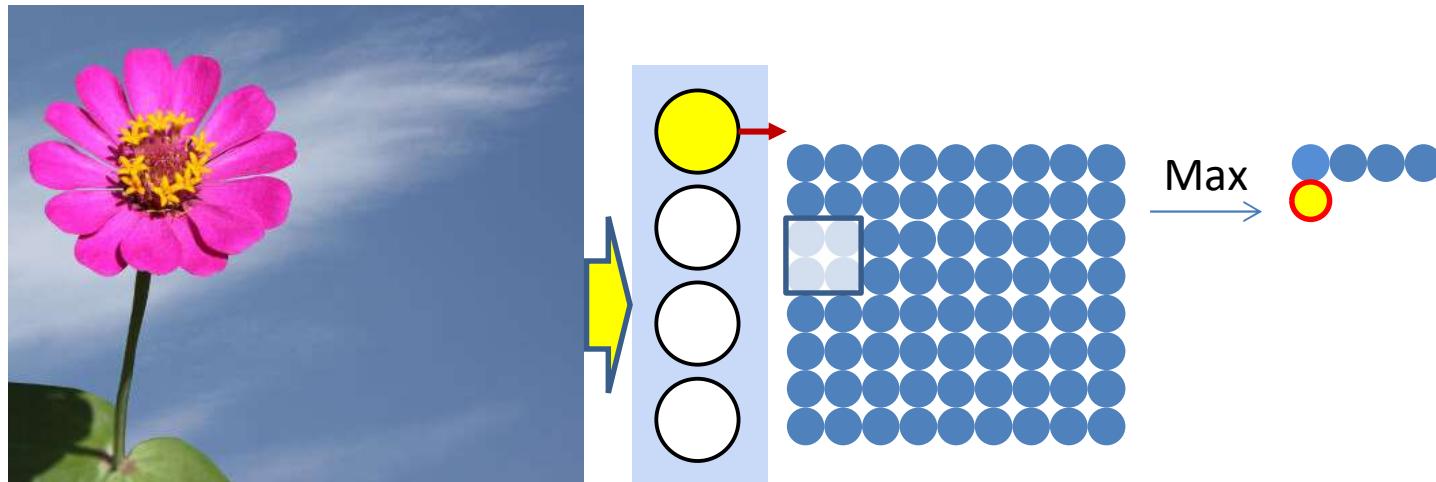
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



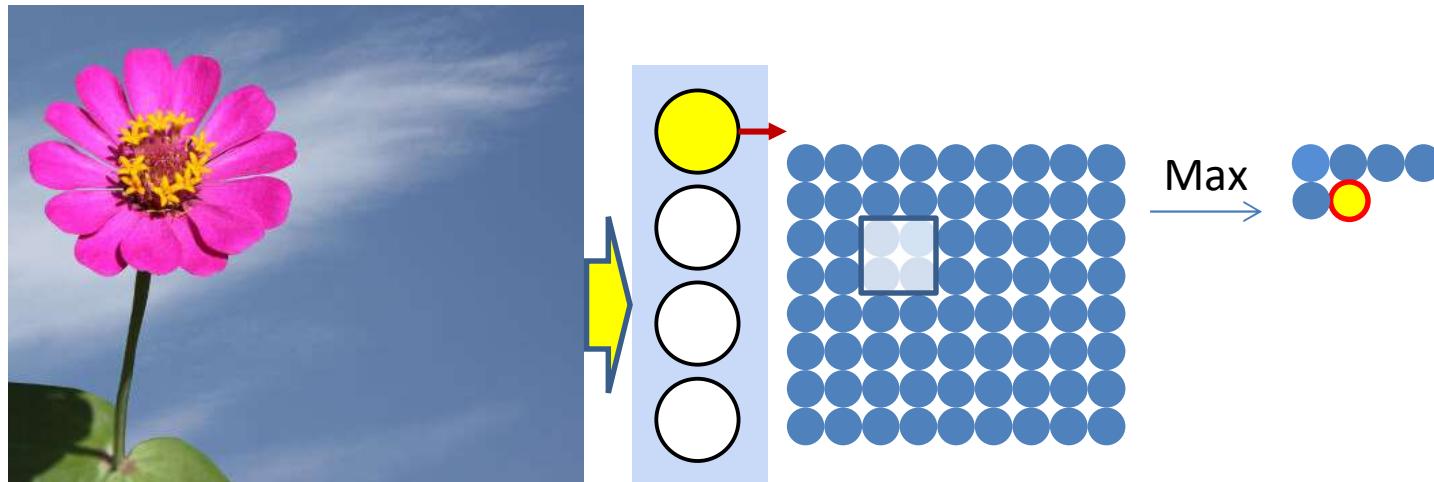
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



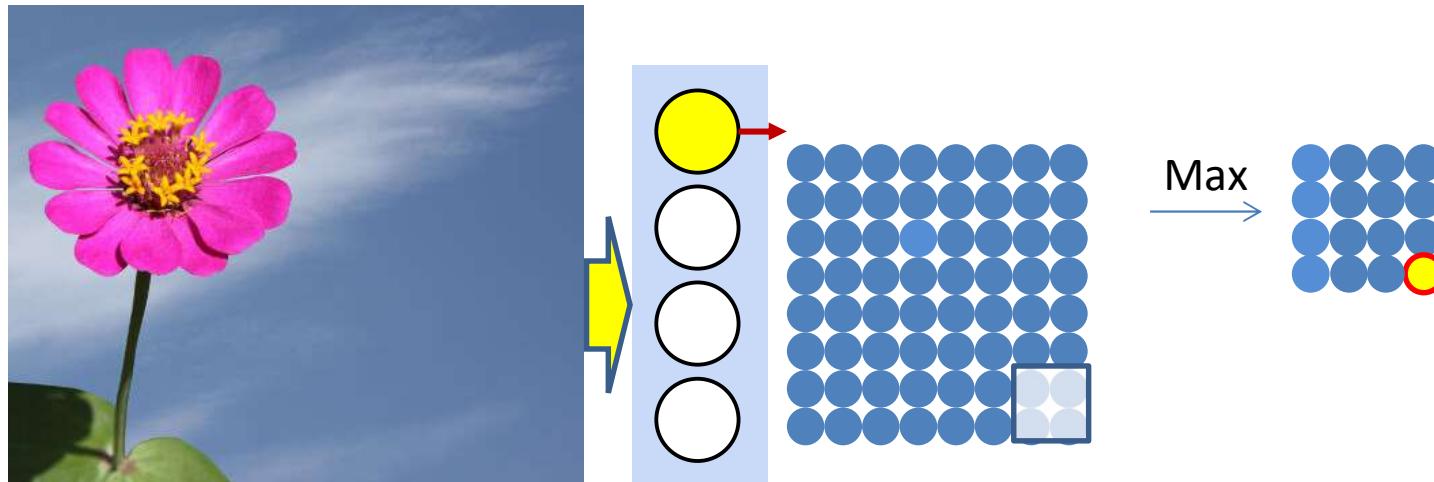
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



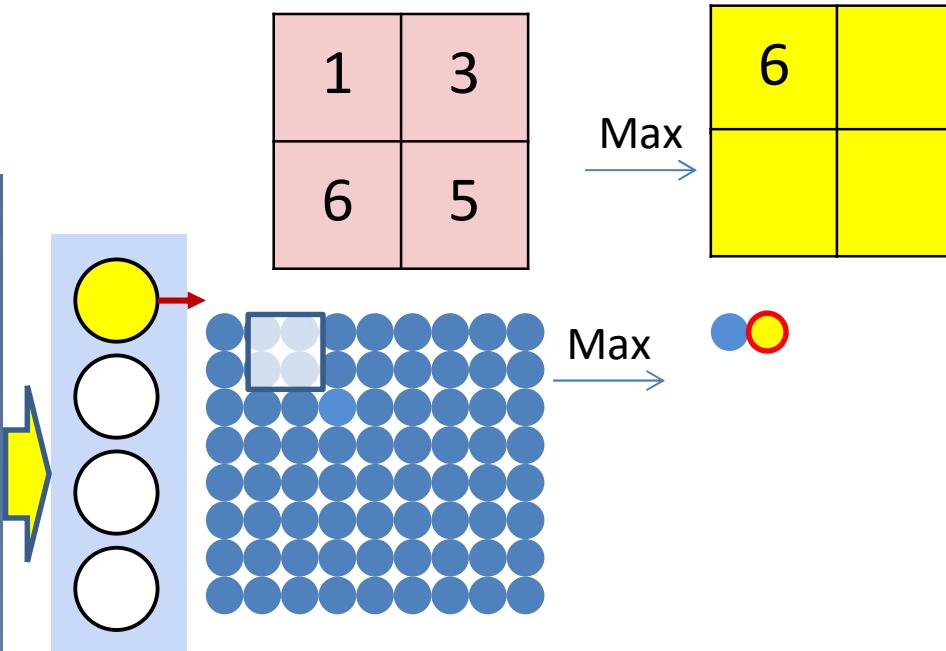
- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Pooling and downsampling



- Pooling is typically performed with strides > 1
 - Results in shrinking of the map
 - “Downsampling”

Max pooling

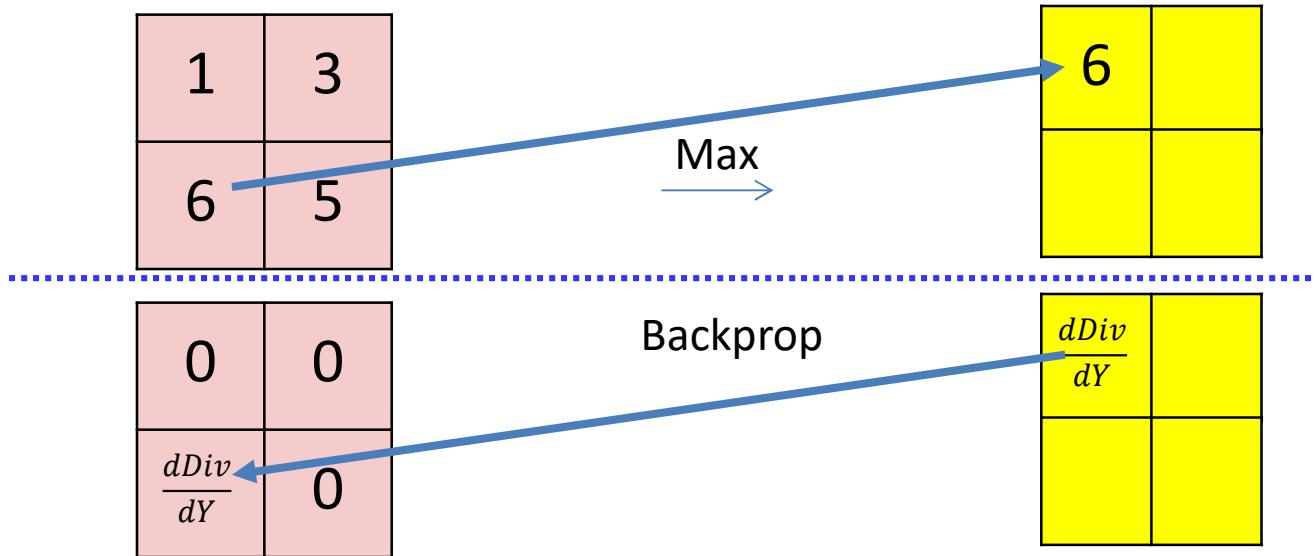


- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

$$P(l, m, i, j) = \operatorname{argmax}_{\substack{k \in \{(i-1)d+1, (i-1)d+K_{lpool}\}, \\ n \in \{(j-1)d+1, (j-1)d+K_{lpool}\}}} Y(l-1, m, k, n)$$

$$Y(l, m, i, j) = Y(l-1, m, P(l, m, i, j))$$

Derivative of Max pooling



$$\frac{dDiv}{dy(l-1, m, k, l)} = \begin{cases} \frac{dDiv}{dy(l, m, i, j)} & \text{if } (k, l) = P(l, m, i, j) \\ 0 & \text{otherwise} \end{cases}$$

- Max pooling selects the largest from a pool of elements

$$P(l, m, i, j) = \operatorname{argmax}_{\substack{k \in \{(i-1)d+1, (i-1)d+K_{lpool}\}, \\ n \in \{(j-1)d+1, (j-1)d+K_{lpool}\}}} y(l-1, m, k, n)$$

$$y(l, m, i, j) = y(l-1, m, P(l, m, i, j))$$

Max Pooling layer at layer l

- a) Performed separately for every map (j).
*) Not combining multiple maps within a single max operation.
- b) Keeping track of location of max

Max pooling

```
for j = 1:D1
    m = 1
    for x = 1:stride(l):Wl-1-Kl+1
        n = 1
        for y = 1:stride(l):Hl-1-Kl+1
            pidx(l,j,m,n) = maxidx(y(l-1,j,x:x+Kl-1,y:y+Kl-1))
            y(l,j,m,n) = y(l-1,j,pidx(l,j,m,n))
            n = n+1
        m = m+1
```



Derivative of max pooling layer at layer l

- a) Performed separately for every map (j).
*) Not combining multiple maps within a single max operation.
- b) Keeping track of location of max

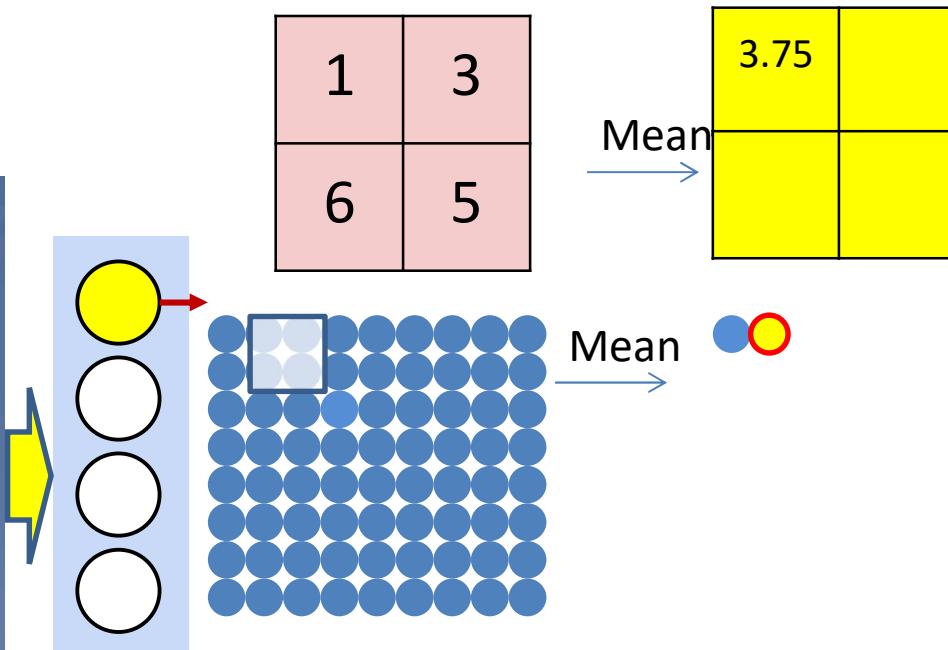


Max pooling

```
dy (:,:, :) = zeros (D1 x W1 x H1)
for j = 1:D1
    for x = 1:W1_downsampled
        for y = 1:H1_downsampled
            dy(l-1,j,pidx(l,j,x,y)) += dy(l,j,x,y)
```

“ $+=$ ” because this entry may be selected in multiple adjacent overlapping windows

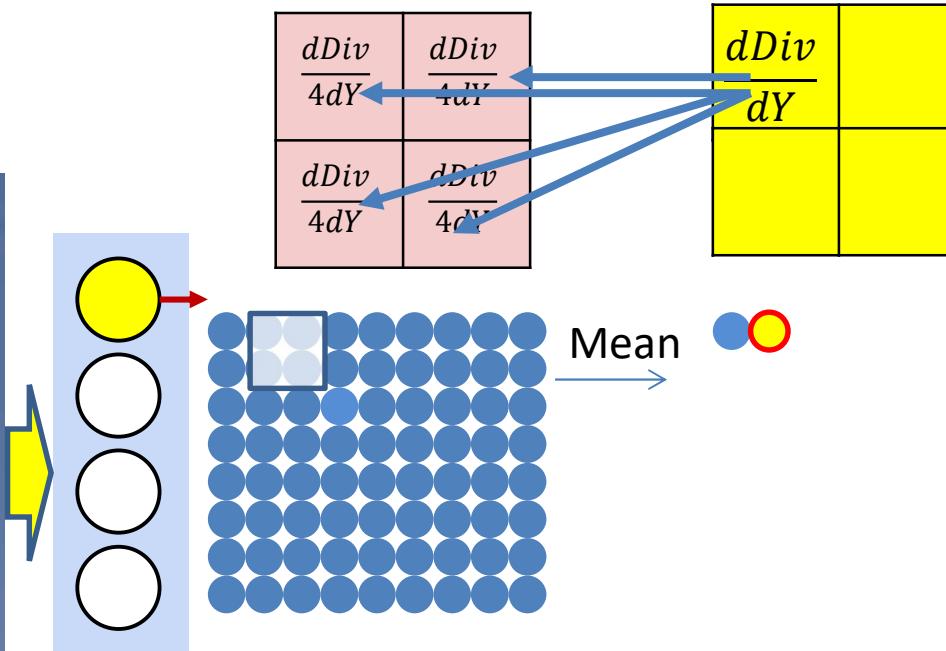
Mean pooling



- Mean pooling compute the mean of a pool of elements
- Pooling is performed by “scanning” the input

$$y(l, m, i, j) = \frac{1}{K_{lpool}^2} \sum_{\substack{k \in \{(i-1)d+1, (i-1)d+K_{lpool}\}, \\ n \in \{(j-1)d+1, (j-1)d+K_{lpool}\}}} y(l-1, m, k, n)$$

Derivative of mean pooling



- The derivative of mean pooling is distributed over the pool

$$k \in \{(i-1)d + 1, (i-1)d + K_{lpool}\}, n \in \{(j-1)d + 1, (j-1)d + K_{lpool}\} \quad dy(l-1, m, k, n) = \frac{1}{K_{lpool}^2} dy(l, m, k, n)$$

Mean Pooling layer at layer l

a) Performed separately for every map (j).

*) Not combining multiple maps within a single mean operation.

Mean pooling

```
for j = 1:D1 #Over the maps
    m = 1
    for x = 1:stride(l):Wl-1-K1+1 #K1 = pooling kernel size
        n = 1
        for y = 1:stride(l):Hl-1-K1+1
            y(l,j,m,n) = mean(y(l-1,j,x:x+K1-1,y:y+K1-1))
            n = n+1
        m = m+1
```

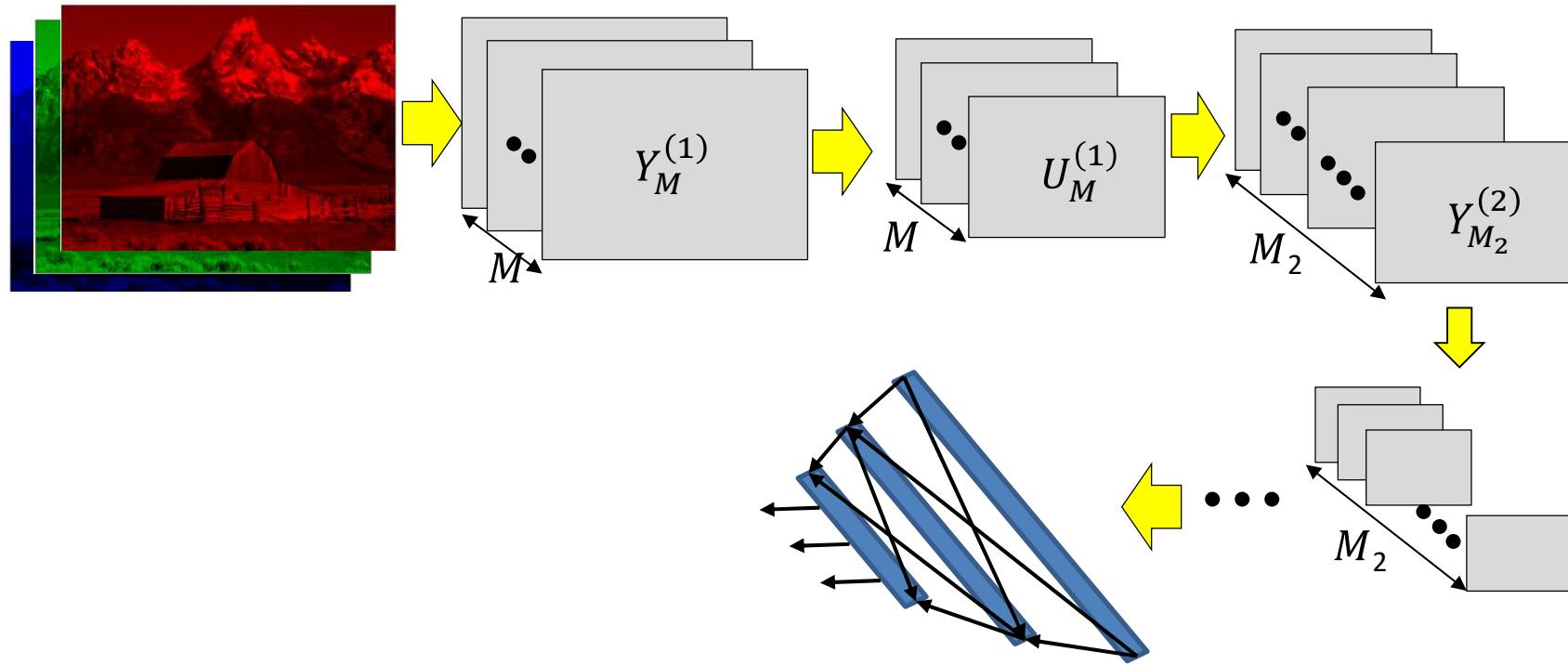
Derivative of mean pooling layer at layer l

Mean pooling

```
dy (:,:, :, :) = zeros (D1 x W1 x H1)
for j = 1:D1
    for x = 1:W1_downsampled
        n = (x-1)*stride
        for y = 1:H1_downsampled
            m = (y-1)*stride
            for i = 1:Klpool
                for j = 1:Klpool
                    dy (l-1, j, p, n+i, m+j) += (1/K2lpool) y (l, j, x, y)
```

“+=” because adjacent windows may overlap

Learning the network



- Have shown the derivative of divergence w.r.t every intermediate output, and every free parameter (filter weights)
- Can now be embedded in gradient descent framework to learn the network

Story so far

- The convolutional neural network is a supervised version of a computational model of mammalian vision
- It includes
 - Convolutional layers comprising learned filters that scan the outputs of the previous layer
 - Downsampling layers that operate over groups of outputs from the convolutional layer to reduce network size
- The parameters of the network can be learned through regular back propagation
 - Maxpooling layers must propagate derivatives only over the maximum element in each pool
 - Other pooling operators can use regular gradients or subgradients
 - Derivatives must sum over appropriate sets of elements to account for the fact that the network is, in fact, a shared parameter network