Artificial Neural Networks
人工神经网络

权小军教授
中山大学计算机学院

quanxj3@mail.sysu.edu.cn

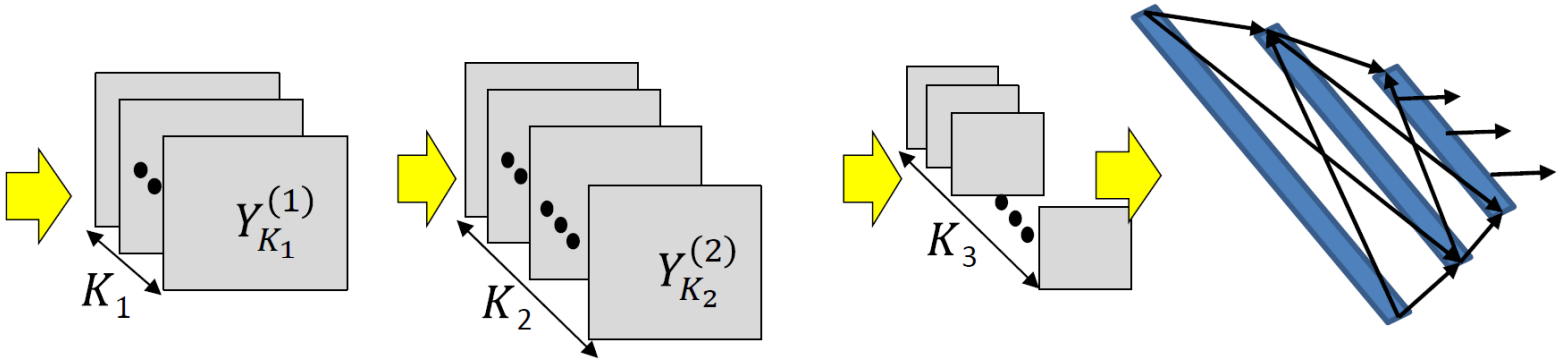
2023 年 4 月 27 日

Lecture 7 - Convolutional Neural Networks II

(*Part of the slides are adapted from CMU 11-785*)

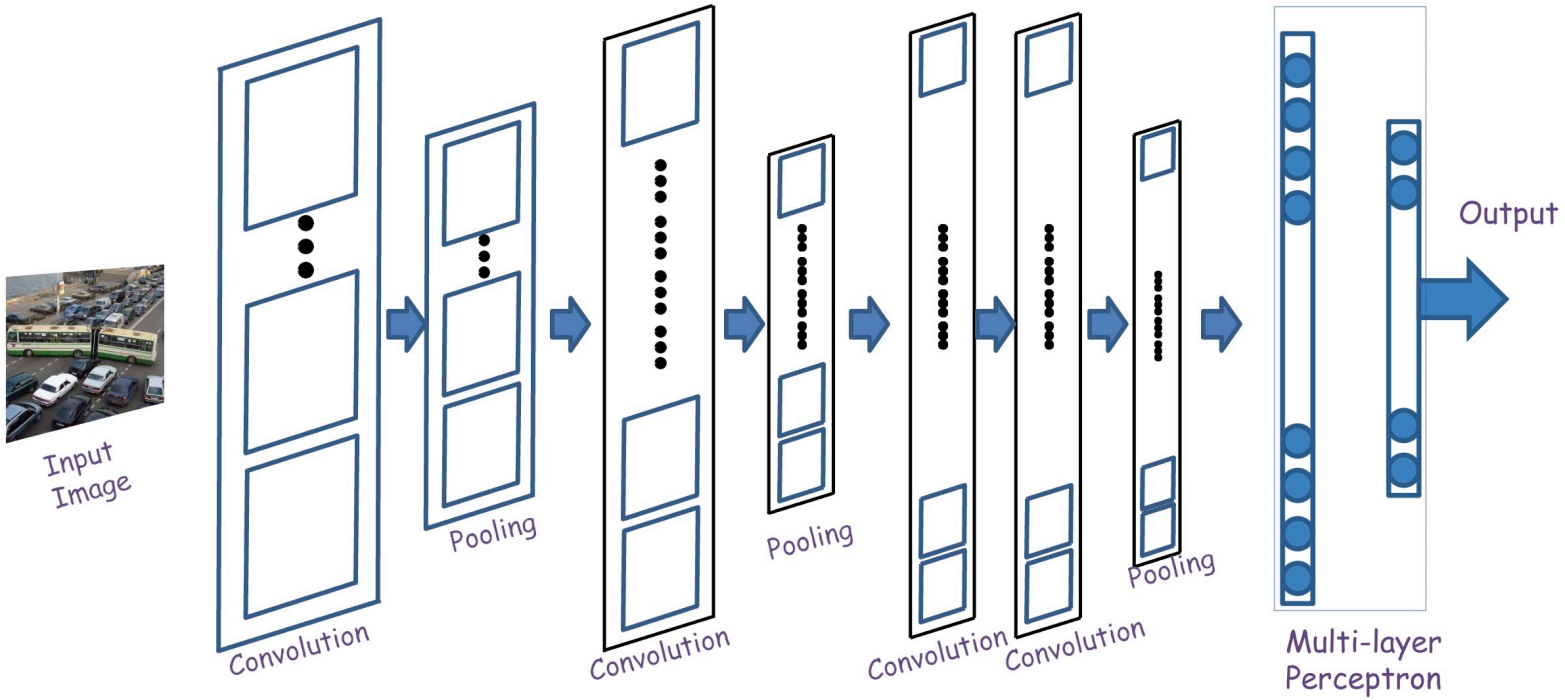
Lecture 7.1 Training

Training



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

Learning the network



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

Problem Setup

- Given a training set of input-output pairs $(X_1, d_1), (X_2, d_2), ..., (X_T, d_T)$
- The loss on the i-th instance is $div(Y_i, d_i)$
- The total loss

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

- Minimize $Loss$ w.r.t W_m, b_m

Training CNNs through Gradient Descent

Total training loss:

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

Assuming the bias is also represented as a weight

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

The derivative

Total training loss:

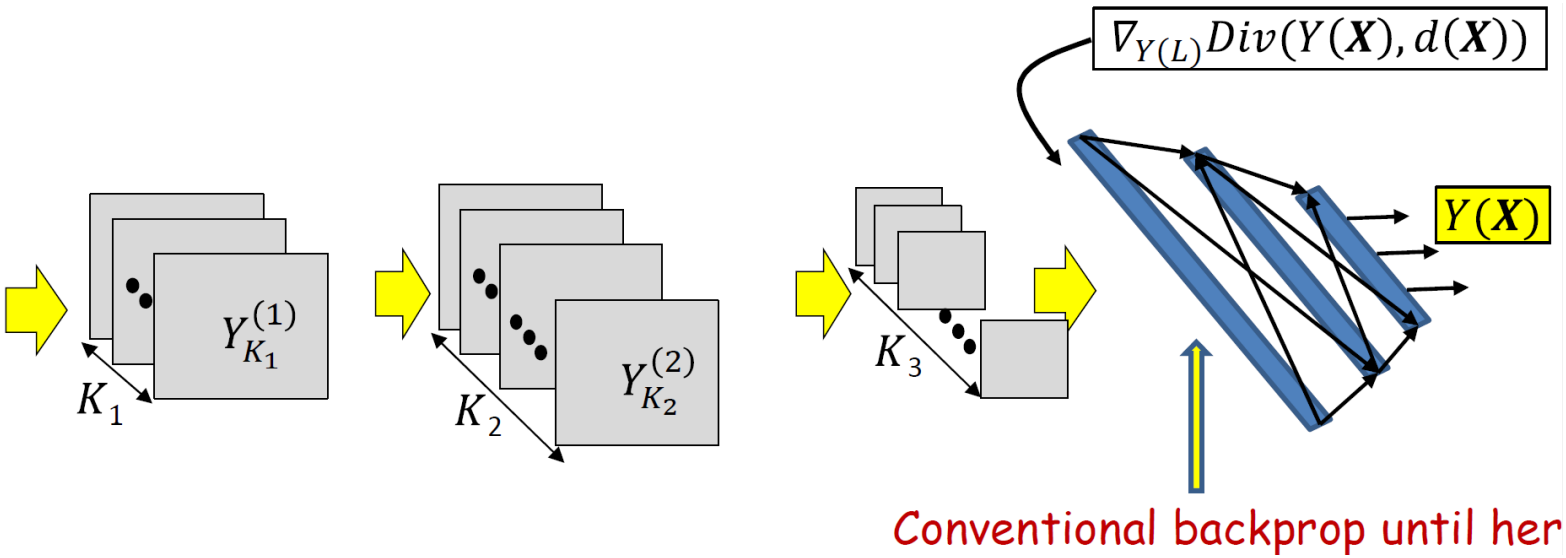
$$Loss = \frac{1}{T}\sum_{i} Div(Y_i, d_i)$$

- Computing the derivative

Total derivative:

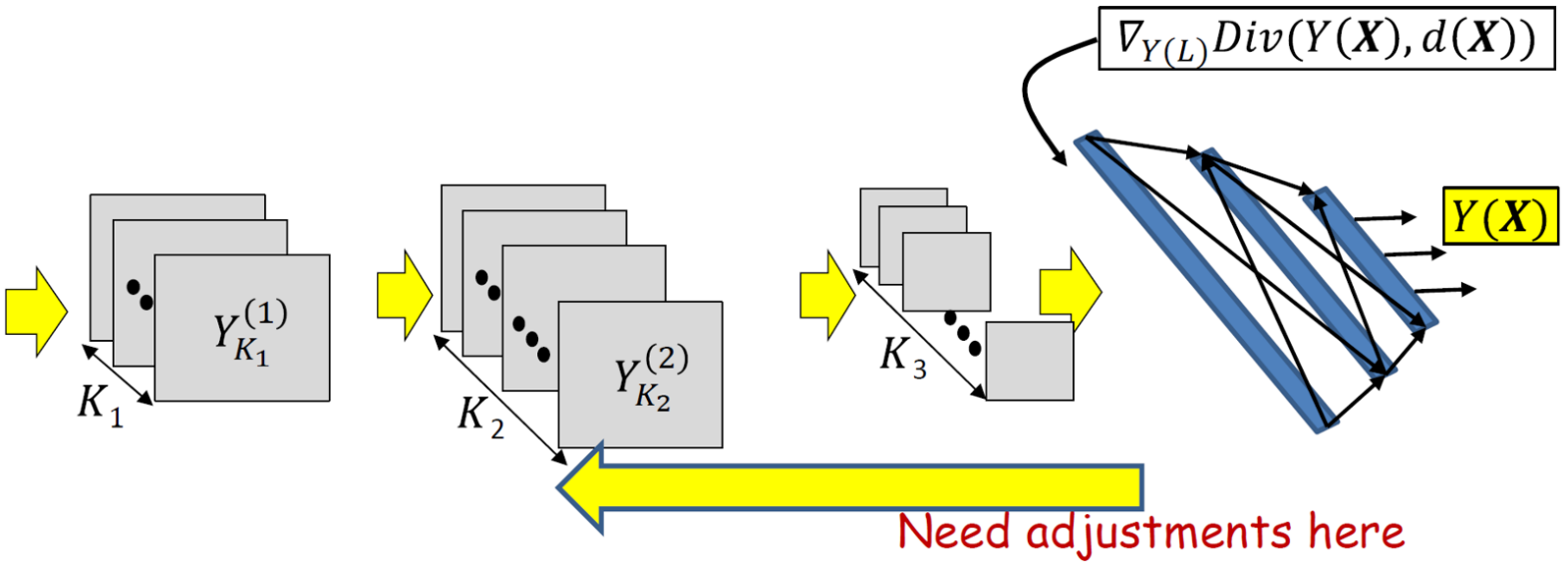
$$\frac{dLoss}{dw(l,m,j,x,y)} = \frac{1}{T}\sum_{i} \frac{dDiv(Y_i, d_i)}{dw(l,m,j,x,y)}$$

Backpropagation: Final flat layers



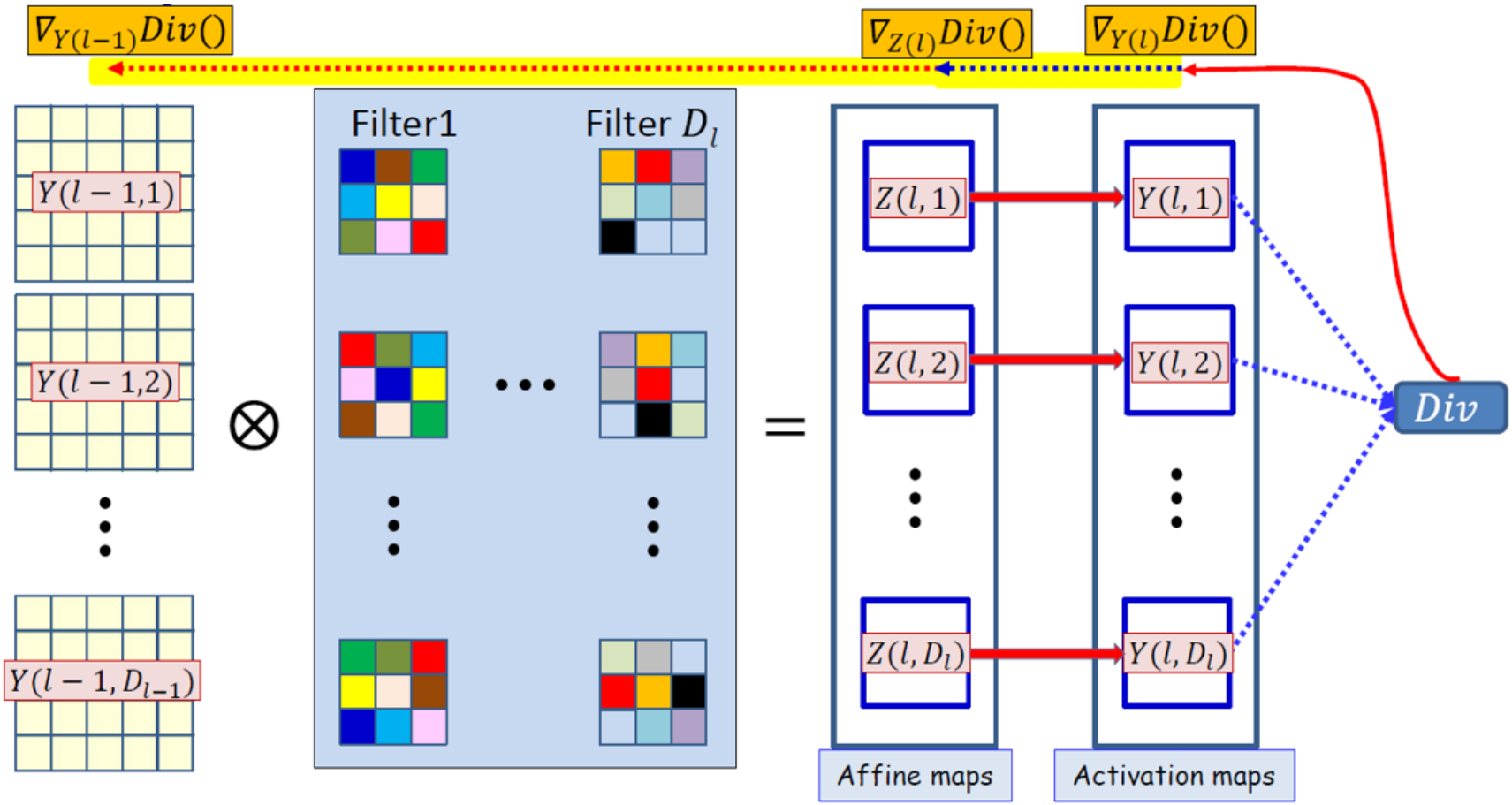
- Backpropagation continues in the usual manner until the computation of the derivative of the divergence w.r.t the inputs to the first "flat" layer
  - Recall: the first flat layer is only the "flattening" (压平) of the maps from the final convolutional layer

Backpropagation: Convolutional and Pooling layers



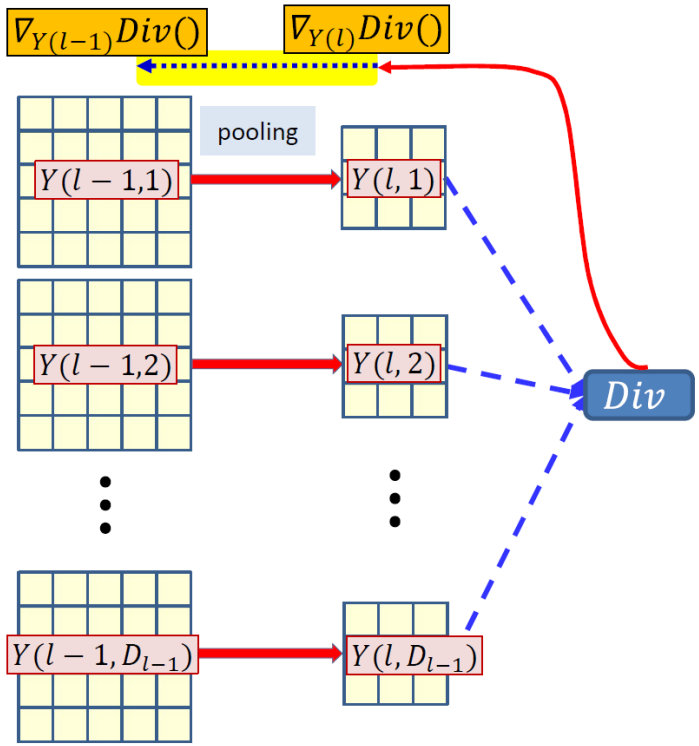
- Backpropagation from the flat MLP requires special consideration of
  - The shared computation in the convolutional layers
  - The pooling layers

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 layer



Pooling 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 maps $Z(l)$ 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. $Y(l)$

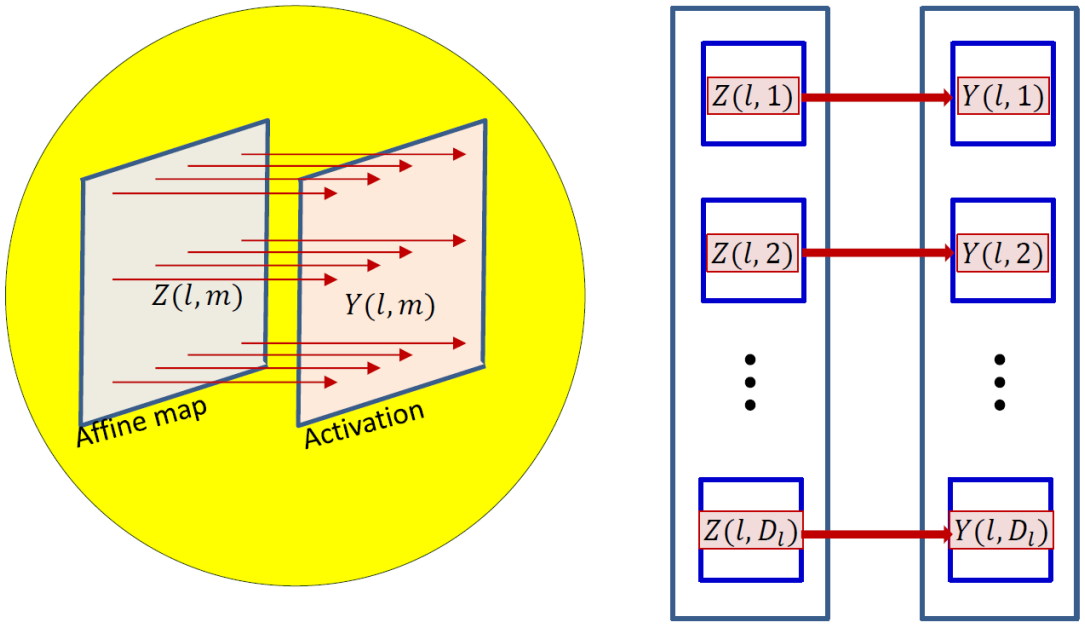
Lecture 7.2 Derivative of Convolution Layer

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

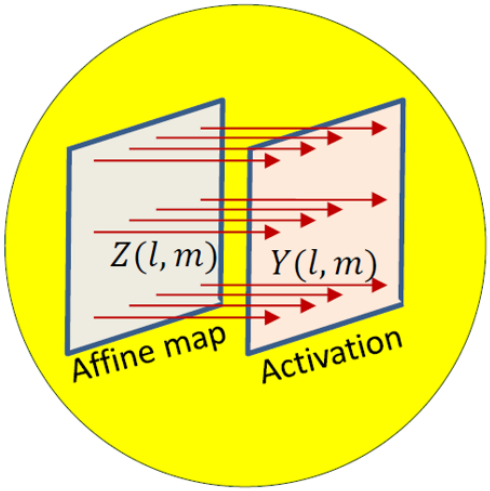


- Forward computation: The activation maps are obtained by point-wise application of activation function to affine maps

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

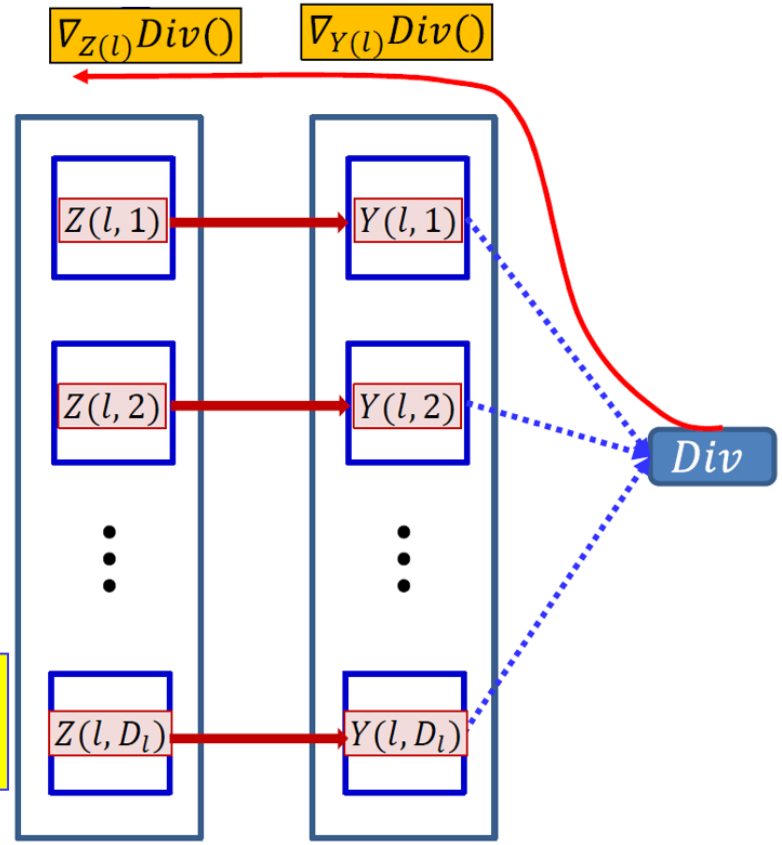
–The affine map entries $z(l, m, x, y)$ have already been computed via convolutions over the previous layer

Backpropagating through the activation



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

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



Backpropagation: Convolutional and Pooling layers

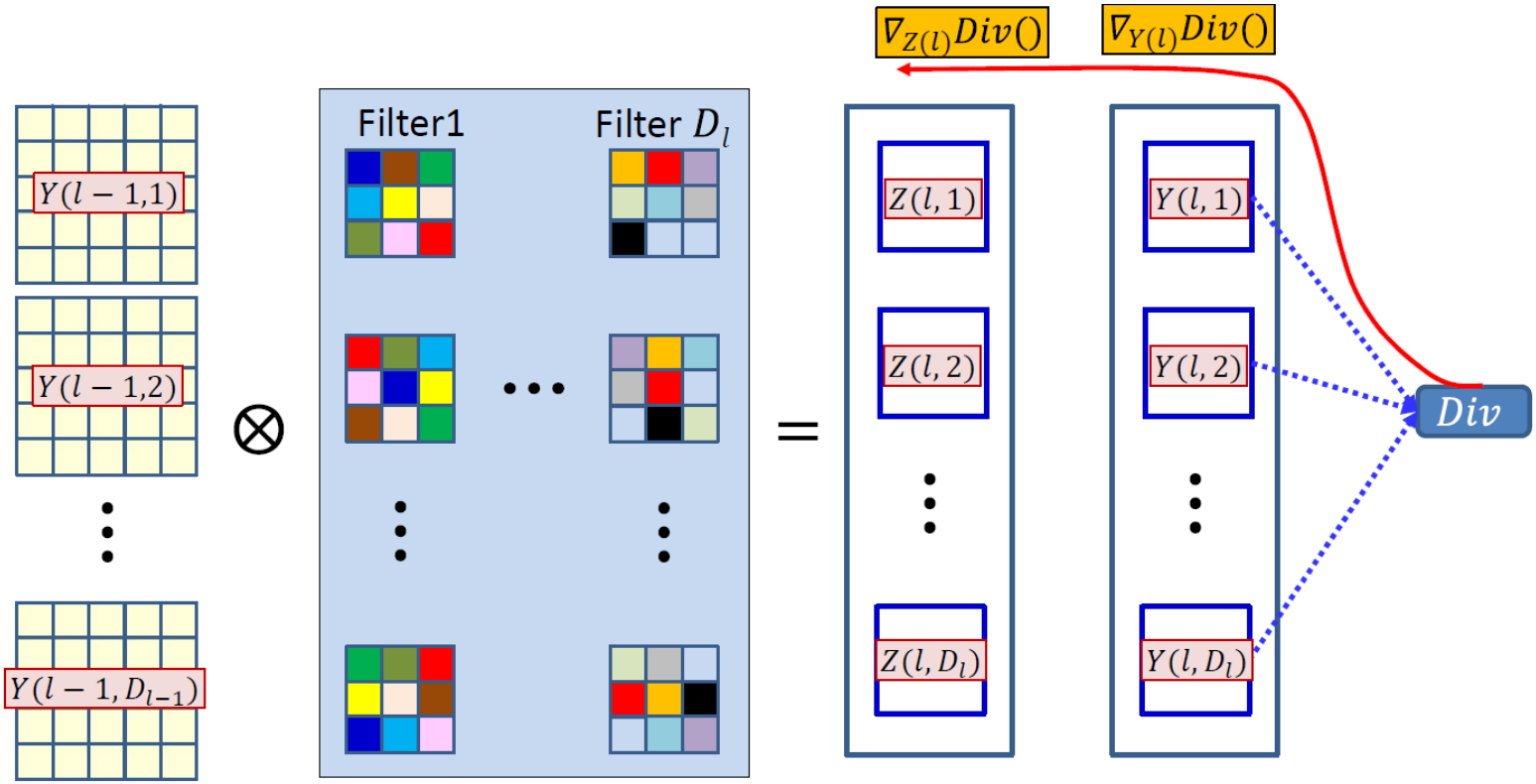
- **Assumption:** We already have the derivatives w.r.t. the elements of the maps output by the final convolutional (or pooling) layer
  - Obtained as a result of backpropagating through the flat MLP

- **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. $Y(l)$

Backpropagating through affine map

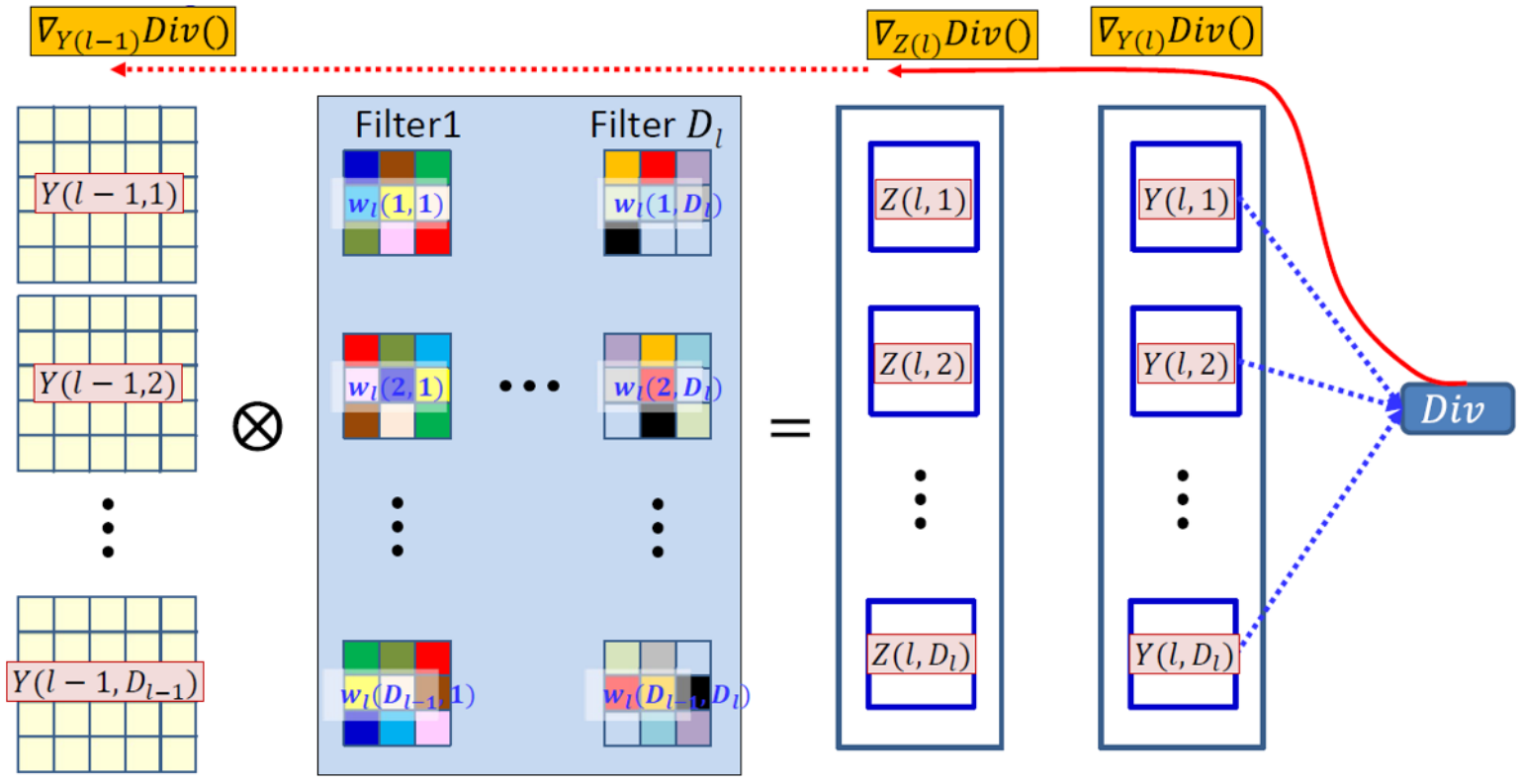
- Forward affine computation:
  - Compute affine maps $z(l, n, x, y)$ from previous layer maps $y(l-1, m, x, y)$ and filters $w_l(m, n, x, y)$
- Backpropagation: Given $\frac{dDiv}{dz(l,n,x,y)}$
  - Compute derivative w.r.t. $y(l-1, m, x, y)$
  - Compute derivative w.r.t. $w_l(m, n, x, y)$

Backpropagating through the affine map



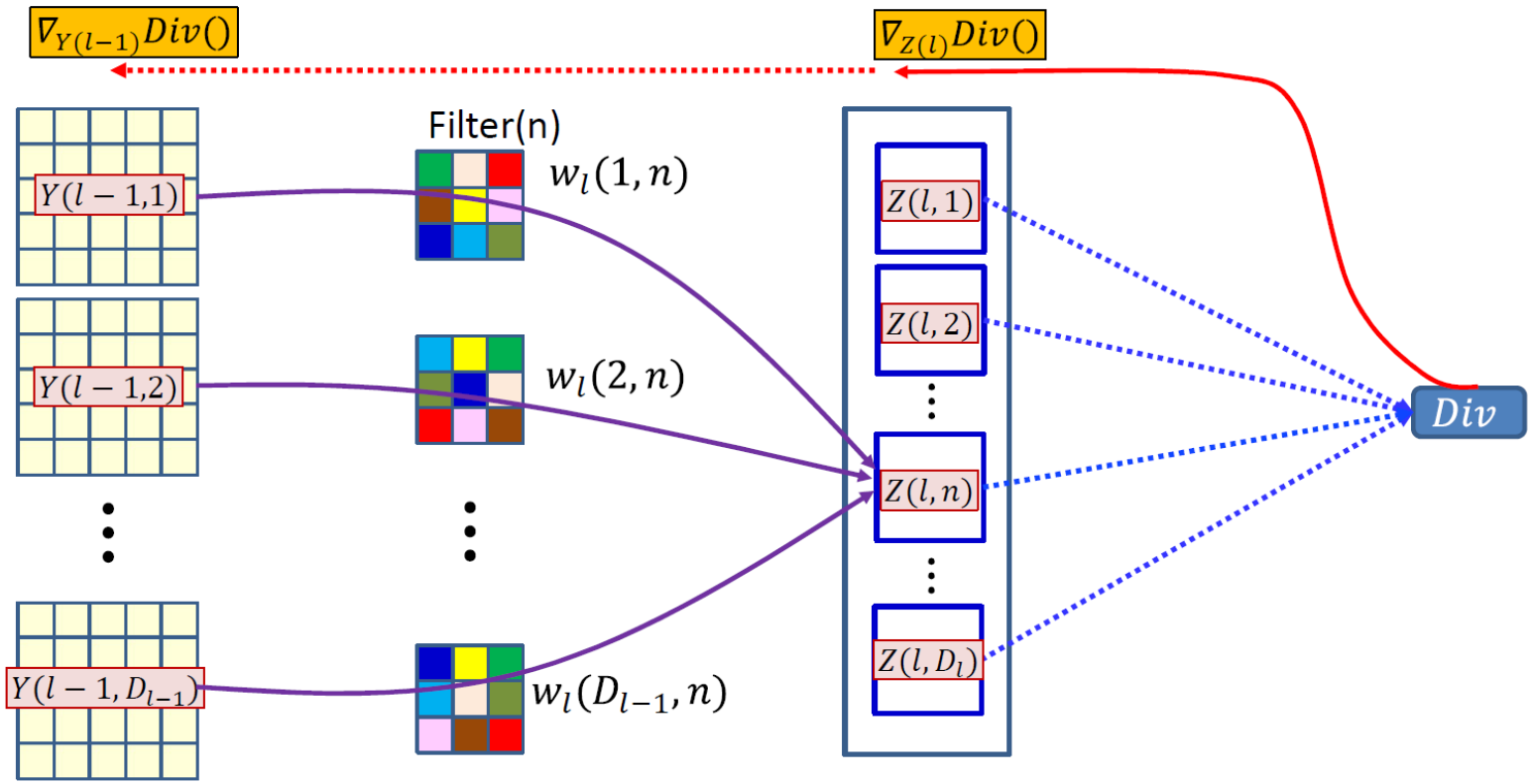
- We already have the derivative w.r.t $Z(l,*)$
  - Having backpropagated it past $Y(l,*)$

Backpropagating through the affine map



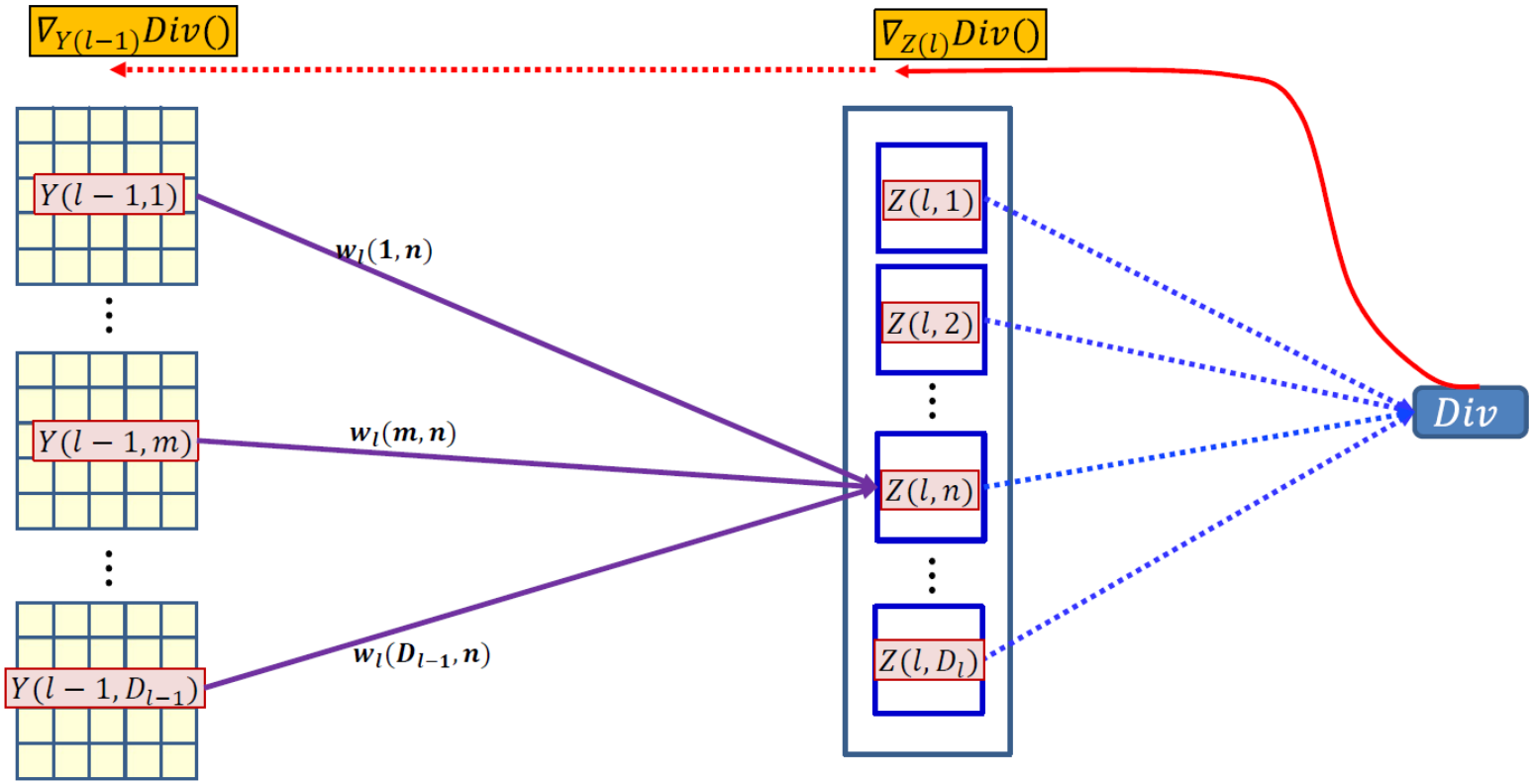
- We already have the derivative w.r.t $Z(l,*)$
  - Having backpropagated it past $Y(l,*)$
- We must compute the derivative w.r.t $Y(l-1,*)$

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



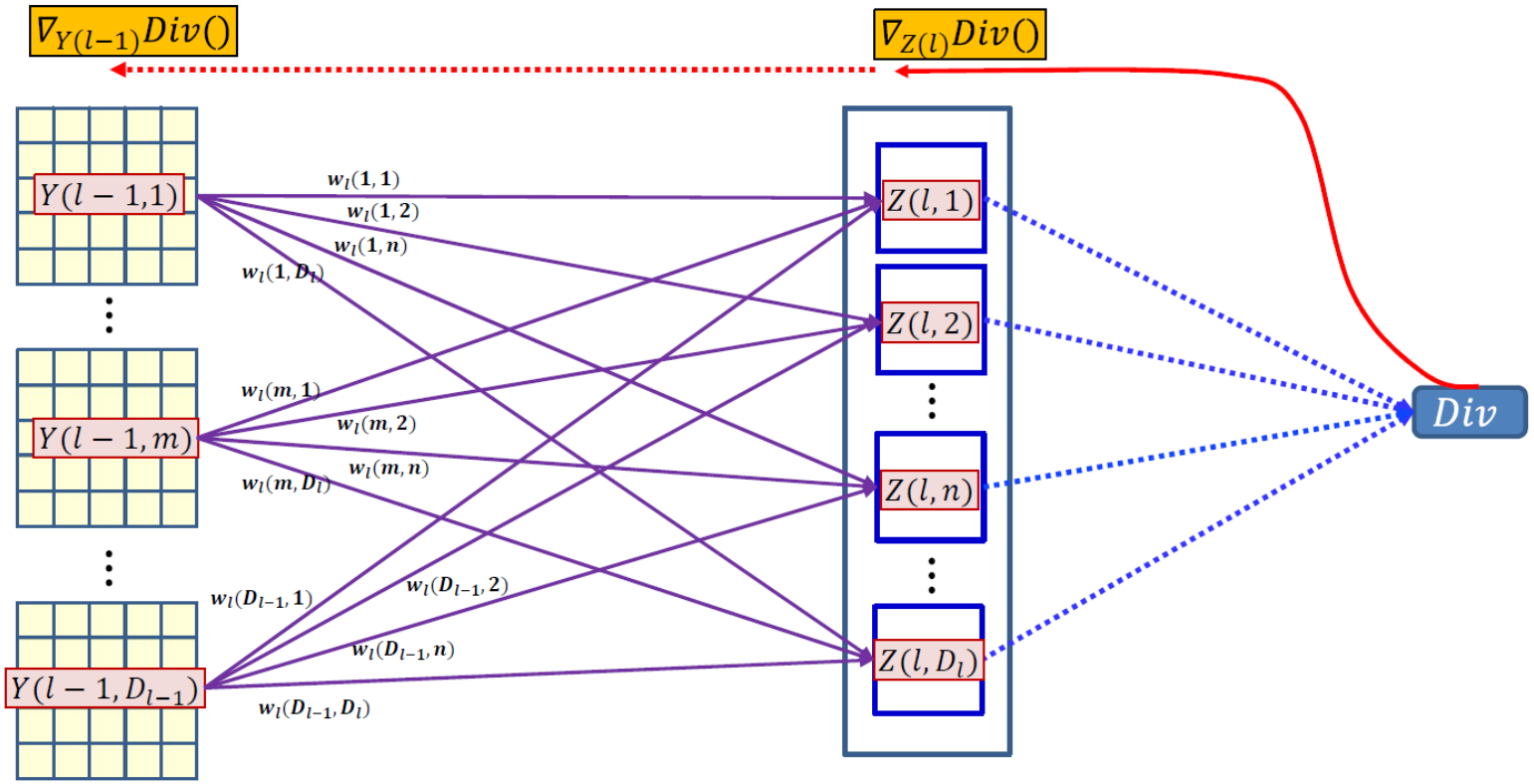
- Each $Y(l-1,m)$ map influences $Z(l,n)$ through the mth "plane" of the nth filter $w_l(m,n)$

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



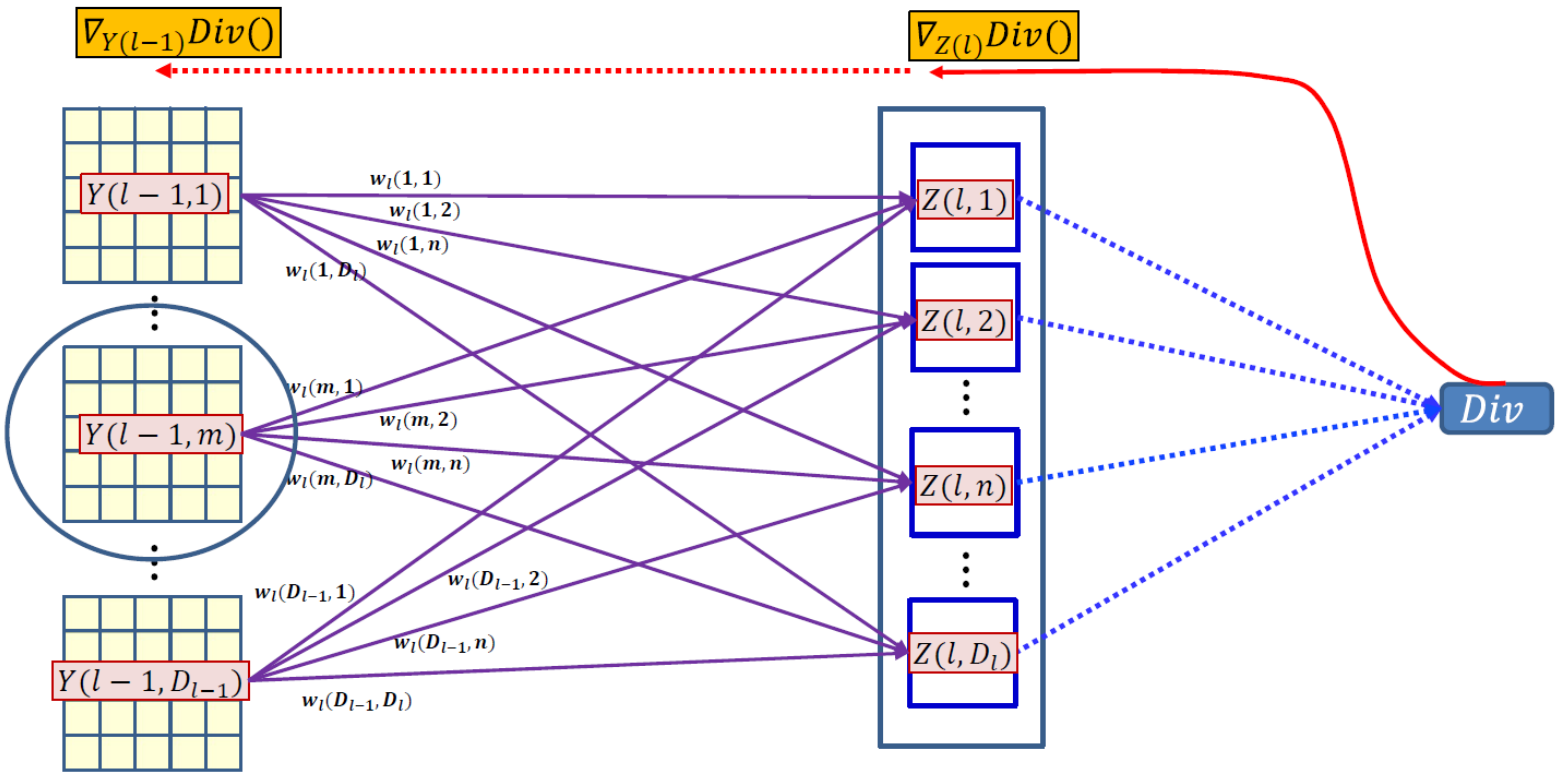
- Each $Y(l-1,m)$ map influences $Z(l,n)$ through the mth "plane" of the nth filter $w_l(m,n)$

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



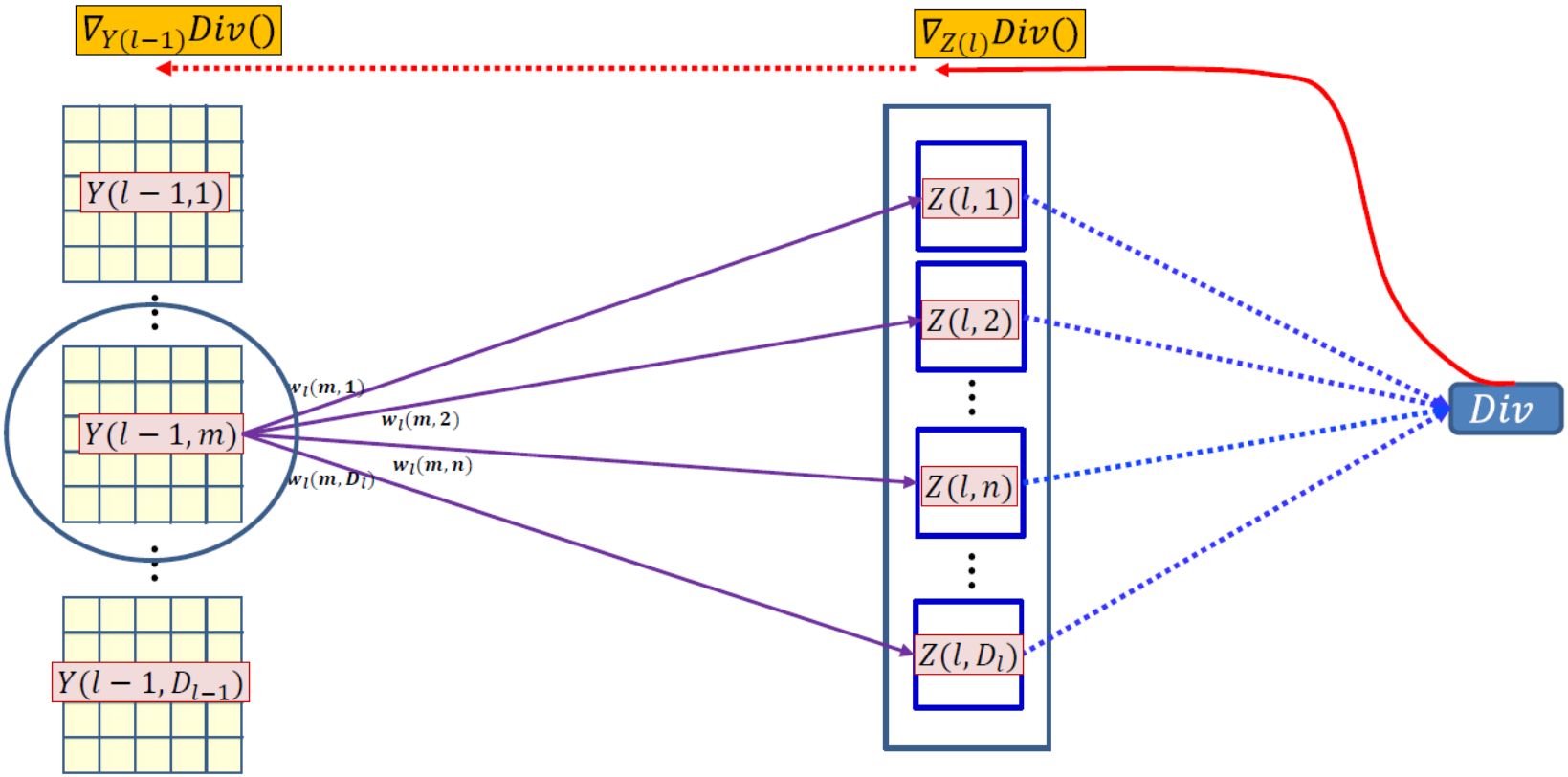
- Each $Y(l-1,m)$ map influences $Z(l,n)$ through the mth "plane" of the nth filter $w_l(m,n)$

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



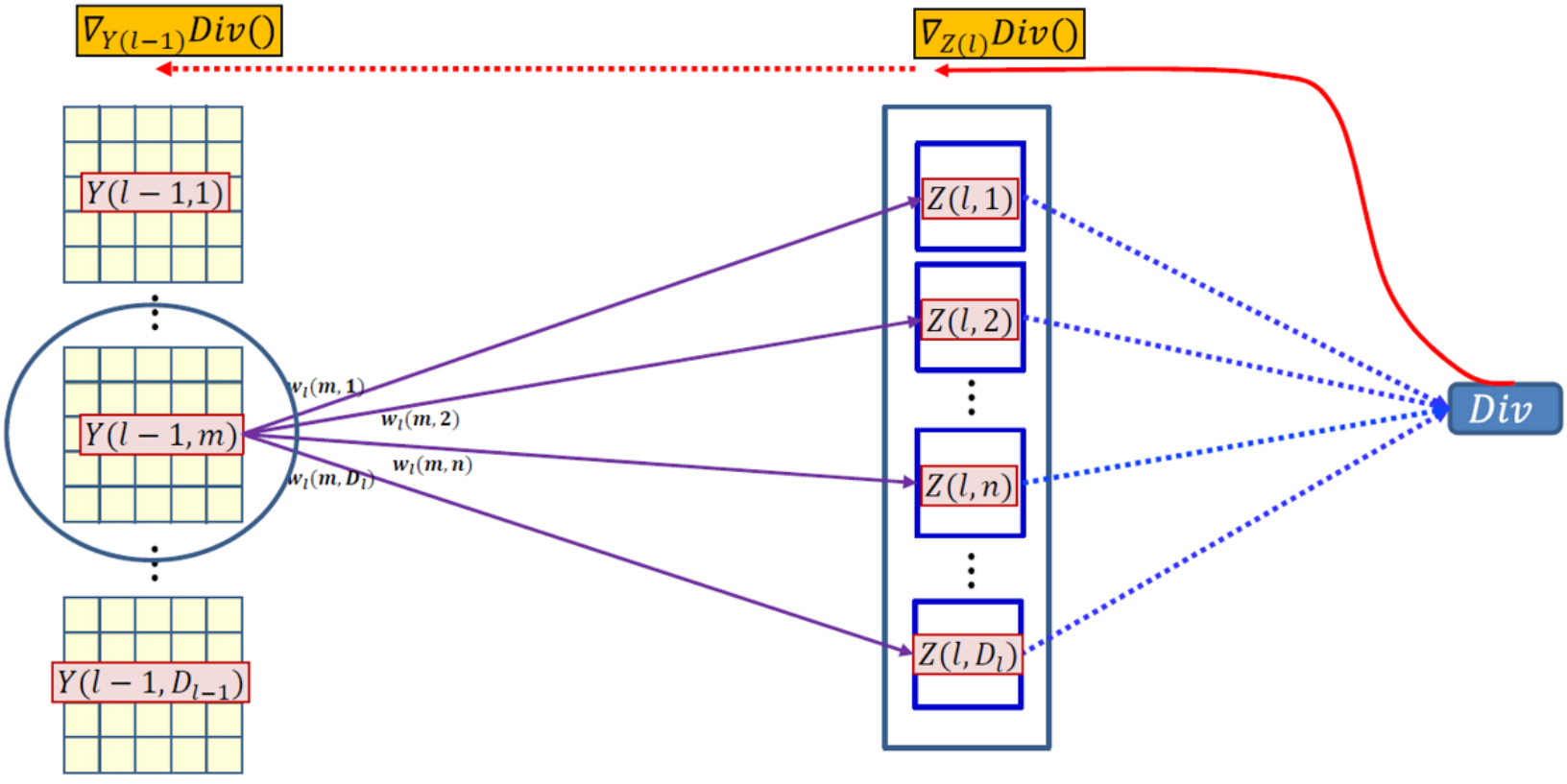
- Each $Y(l-1,m)$ map influences $Z(l,n)$ through the mth "plane" of the nth filter $w_l(m,n)$

Dependency diagram for a single map



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

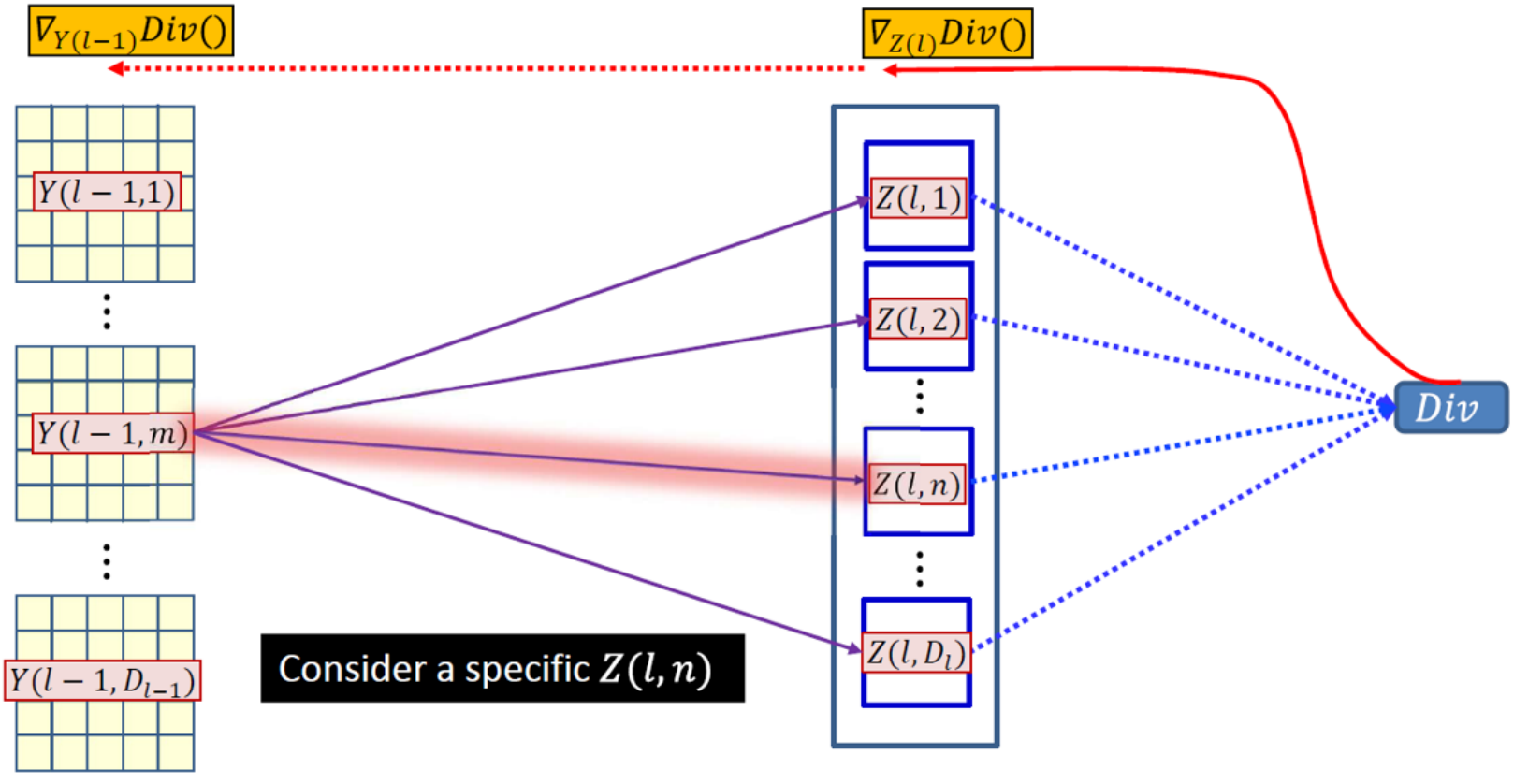
Dependency diagram for a single map



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

- 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

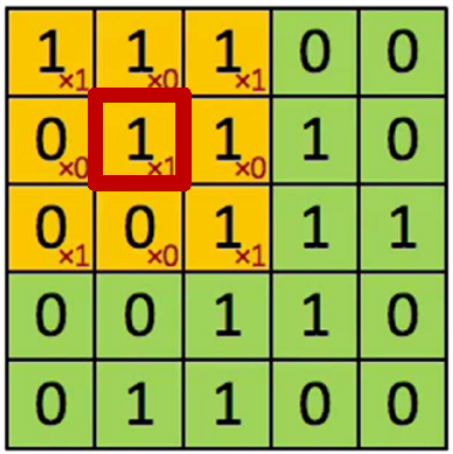
Dependency diagram for a single map



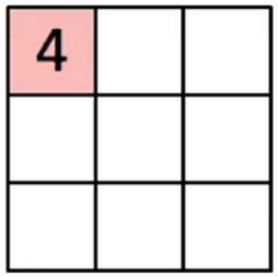
$$\nabla_{Y(l-1,m)}Div(.) = \sum_n \nabla_{Z(l,n)}Div(.)\,\nabla_{Y(l-1,m)}Z(l,n)$$

- 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



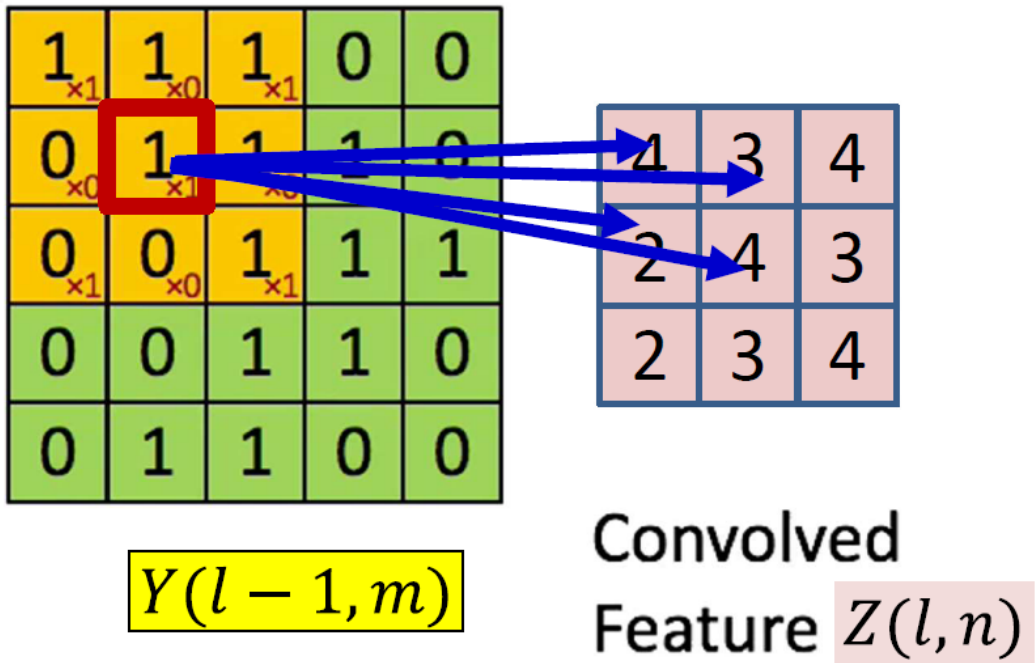
$Y(l-1,m)$



Convolved Feature $Z(l,n)$

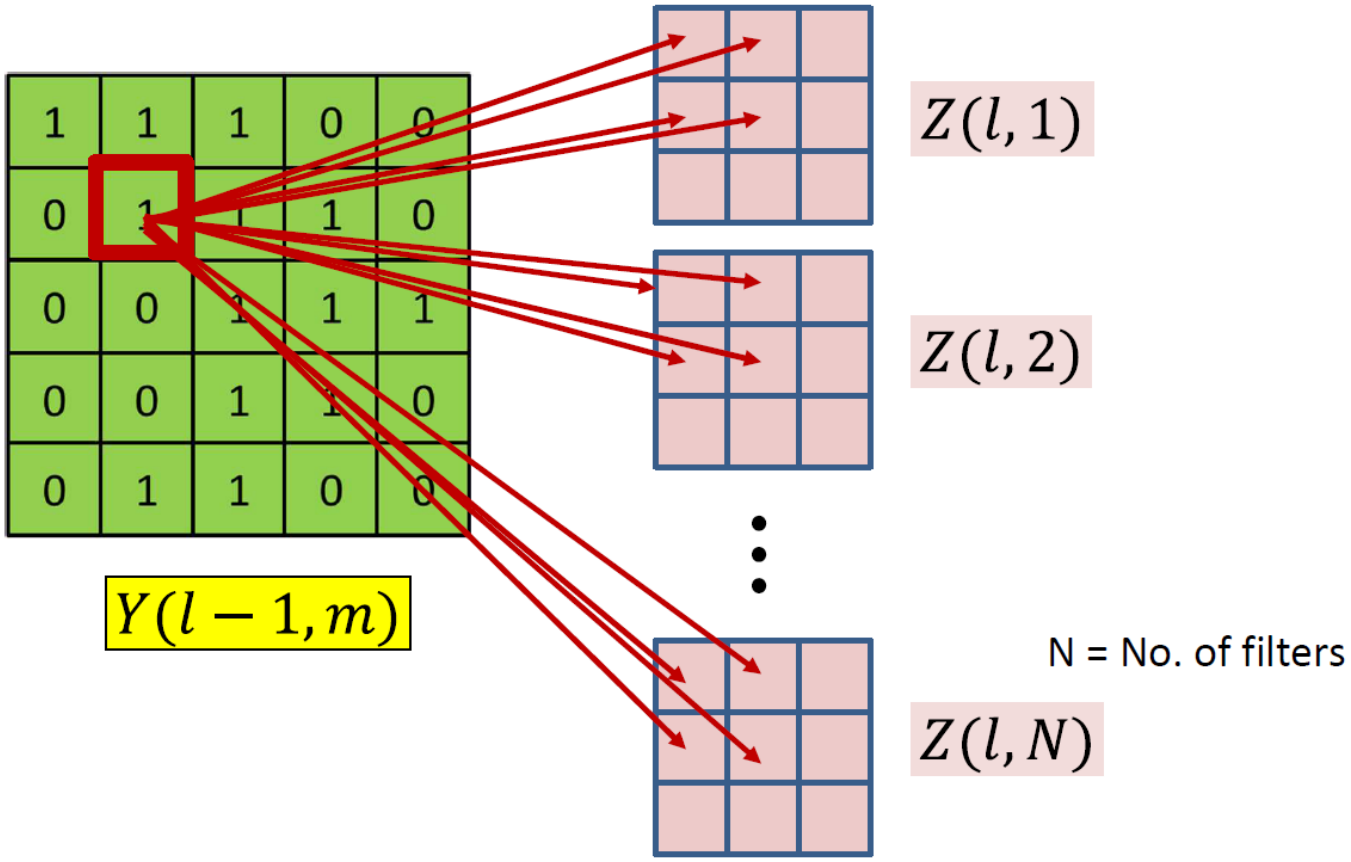
- Each $Y(l-1,m,x,y)$ affects several $z(l,n,x',y')$ terms

BP: Convolutional layer



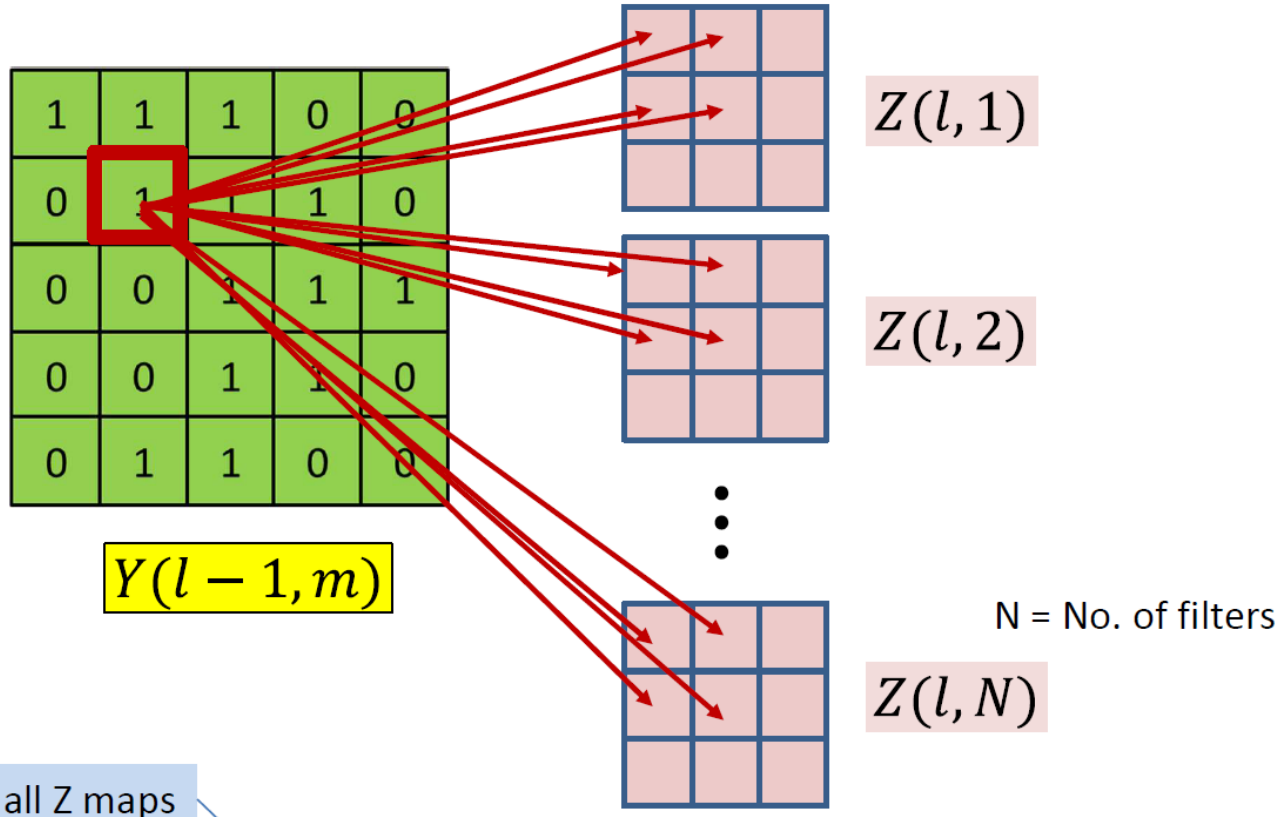
- Each $Y(l-1,m,x,y)$ affects several $z(l,n,x',y')$ terms

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

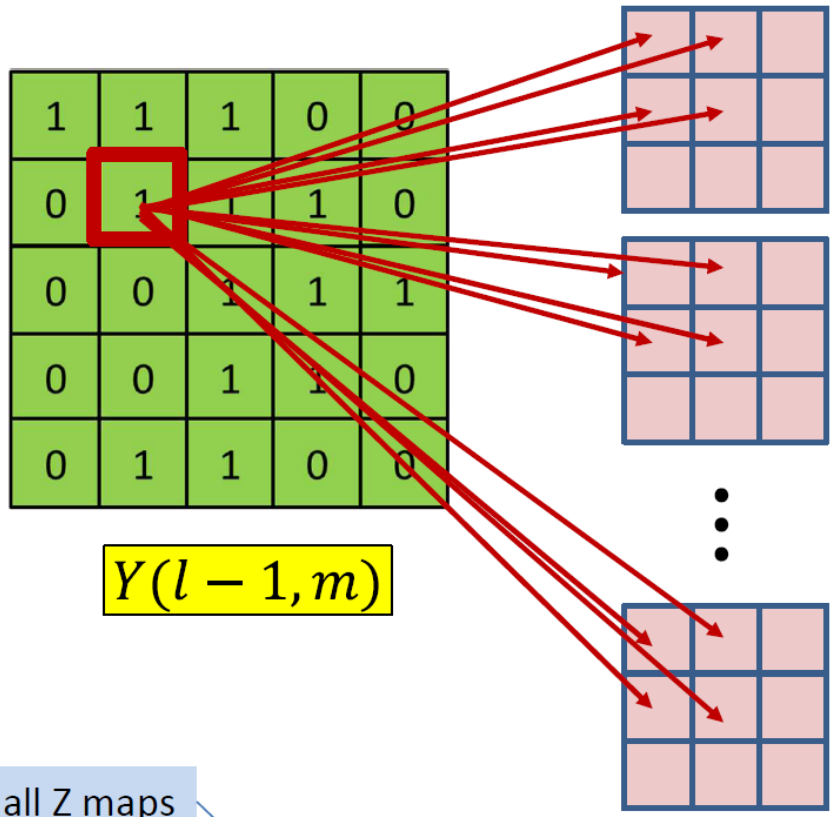
BP: Convolutional layer



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

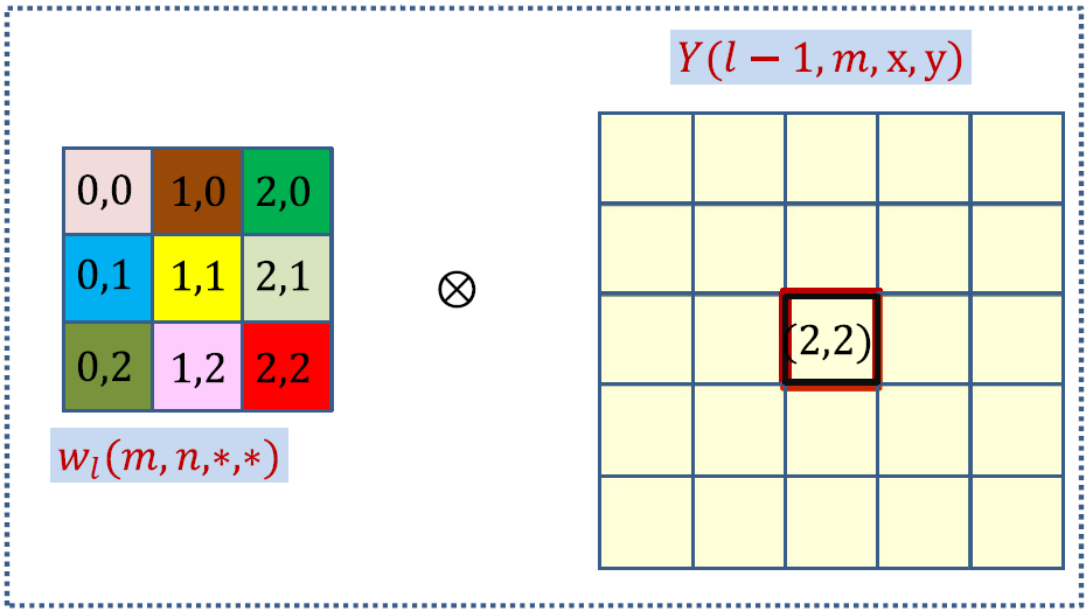


Summing over all Z maps

What is this?

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

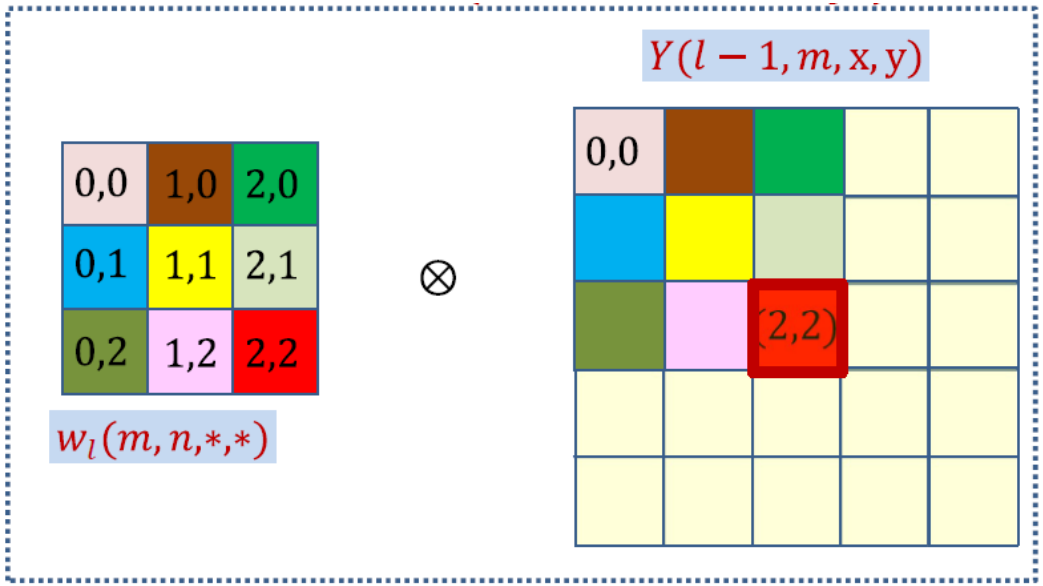
How a single $Y(l-1, m, x, y)$ influences $z(l, n, x', y')$



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 $z(l,n,x',y')$

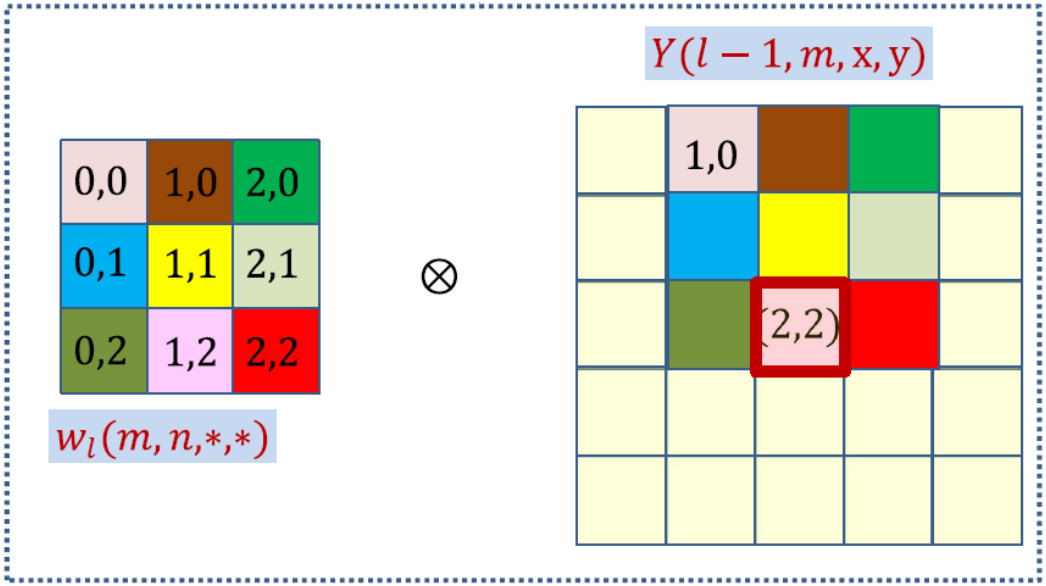


$$z(l,n,0,0) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,2,2)$$

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

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

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

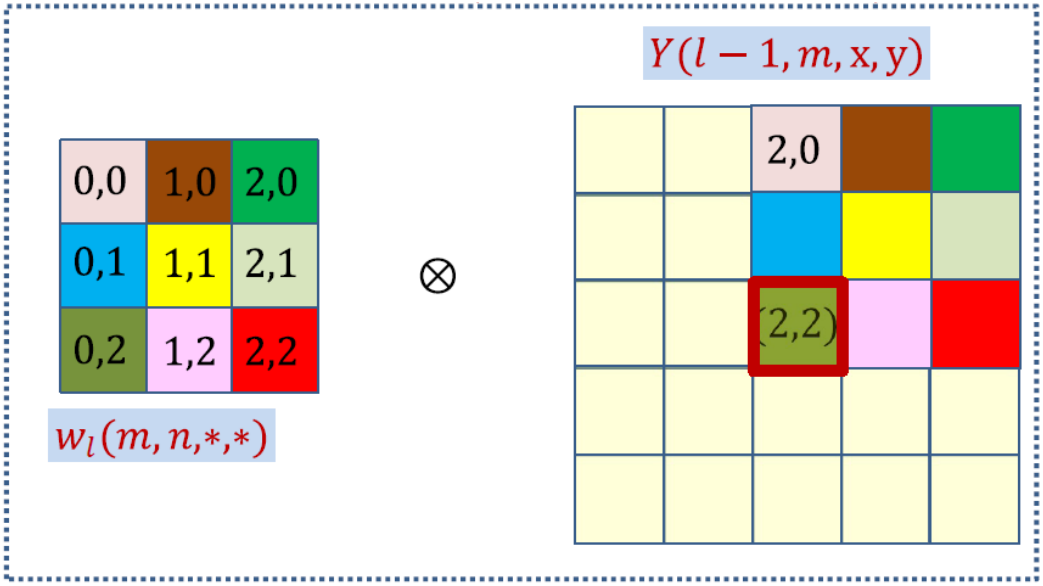


$$z(l,n,1,0) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,1,2)$$

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

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

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

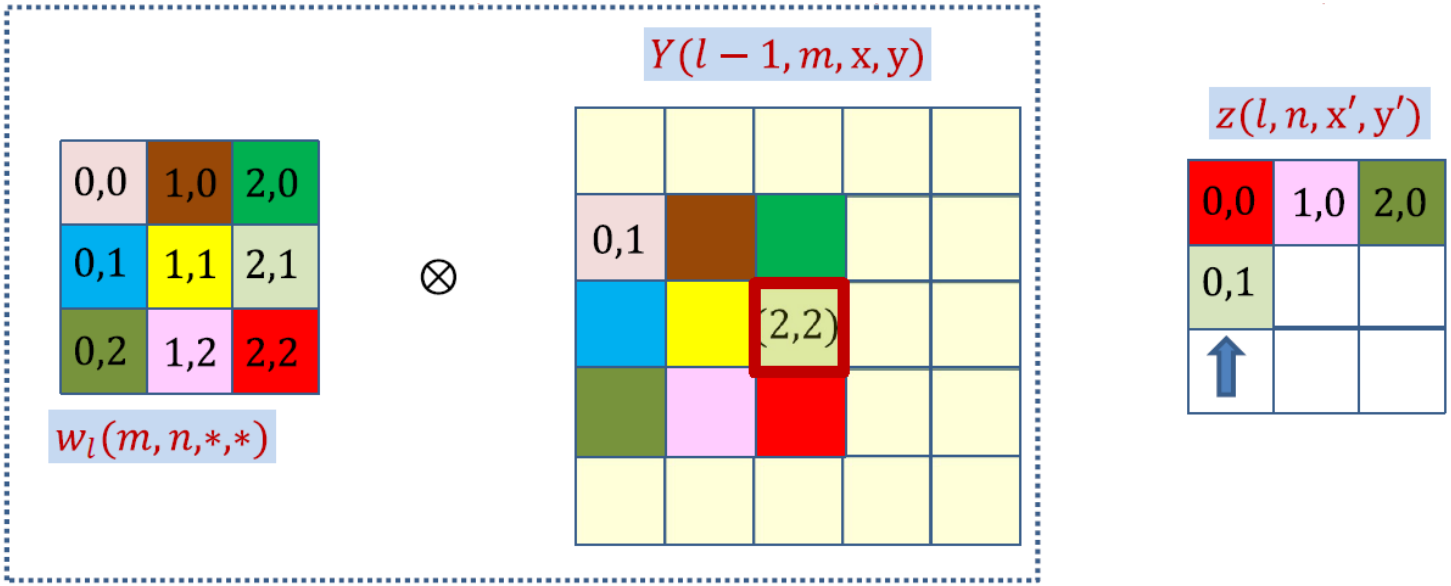


$$z(l,n,2,0) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,0,2)$$

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

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

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

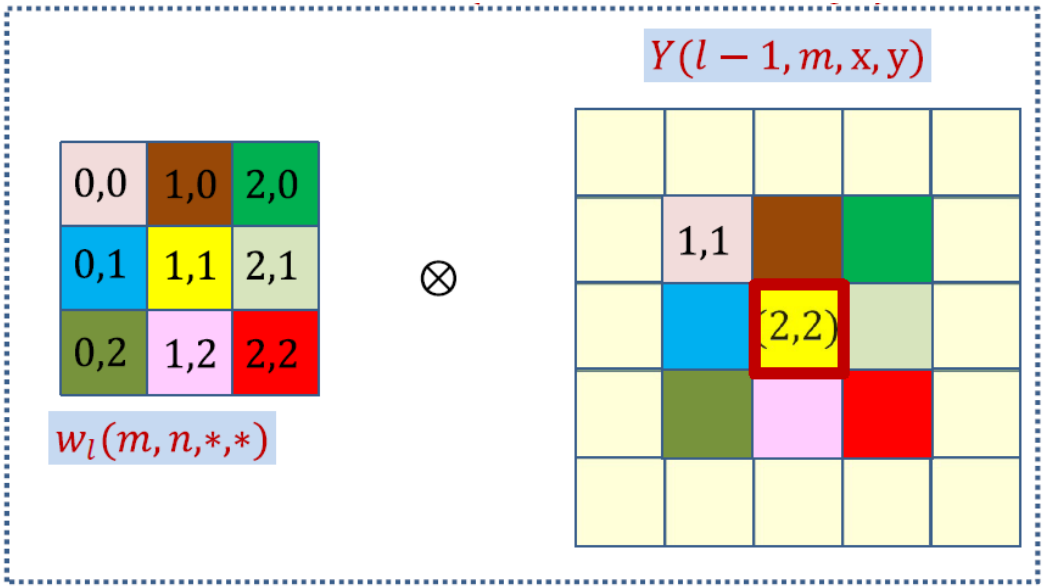


$$z(l,n,0,1) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,2,1)$$

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

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

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

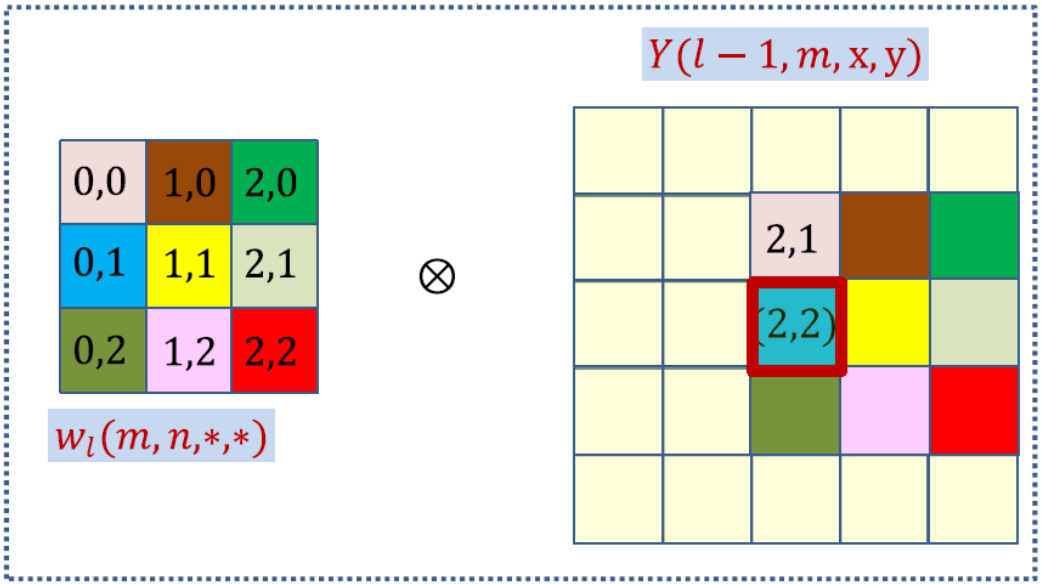


$$z(l,n,1,1) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,1,1)$$

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

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

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

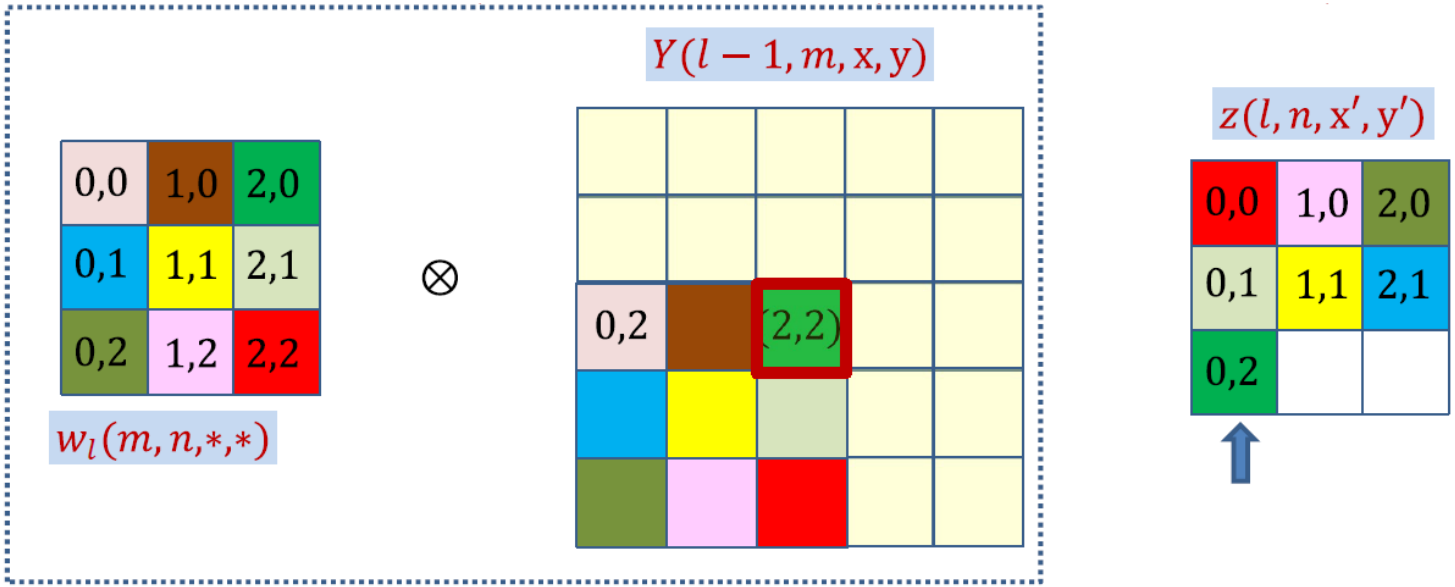


$$z(l,n,2,1) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,0,1)$$

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

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

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

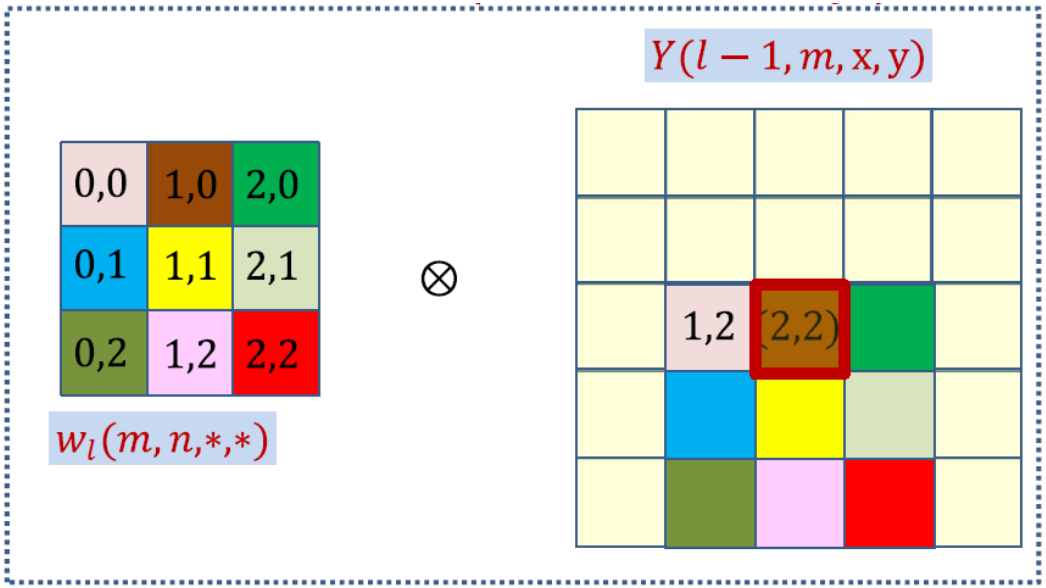


$$z(l,n,0,2) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,2,0)$$

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

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

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

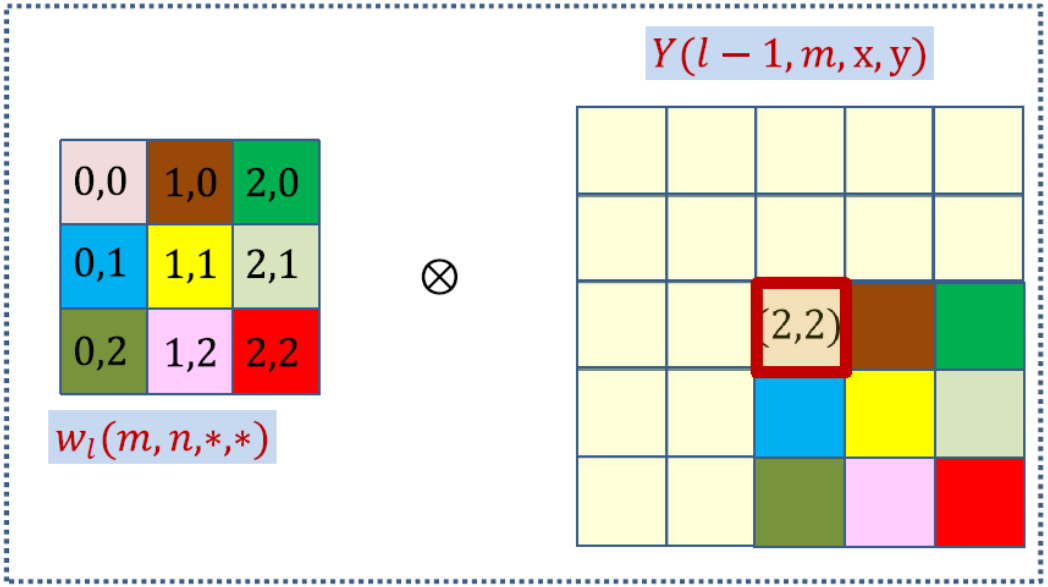


$$z(l,n,1,2) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,2,1)$$

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

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

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

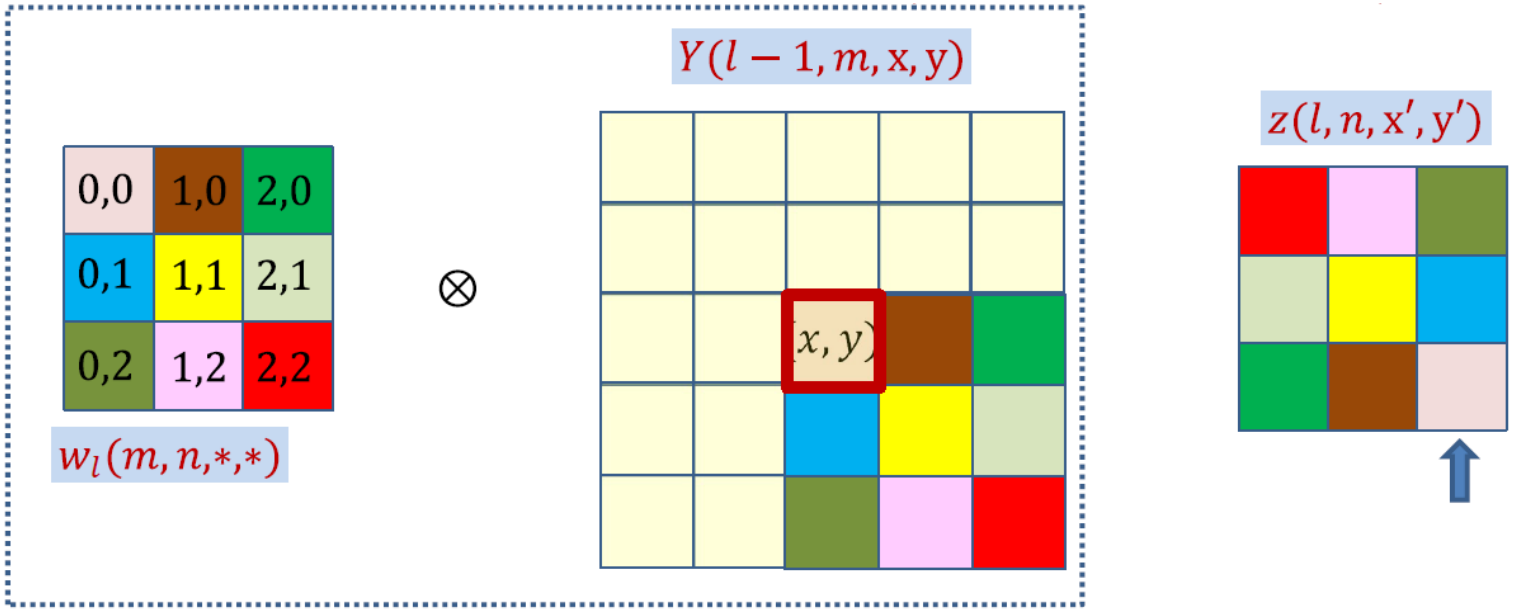


$$z(l,n,2,2) \mathrel{+}= Y(l-1,m,2,2)w_l(m,n,0,0)$$

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

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

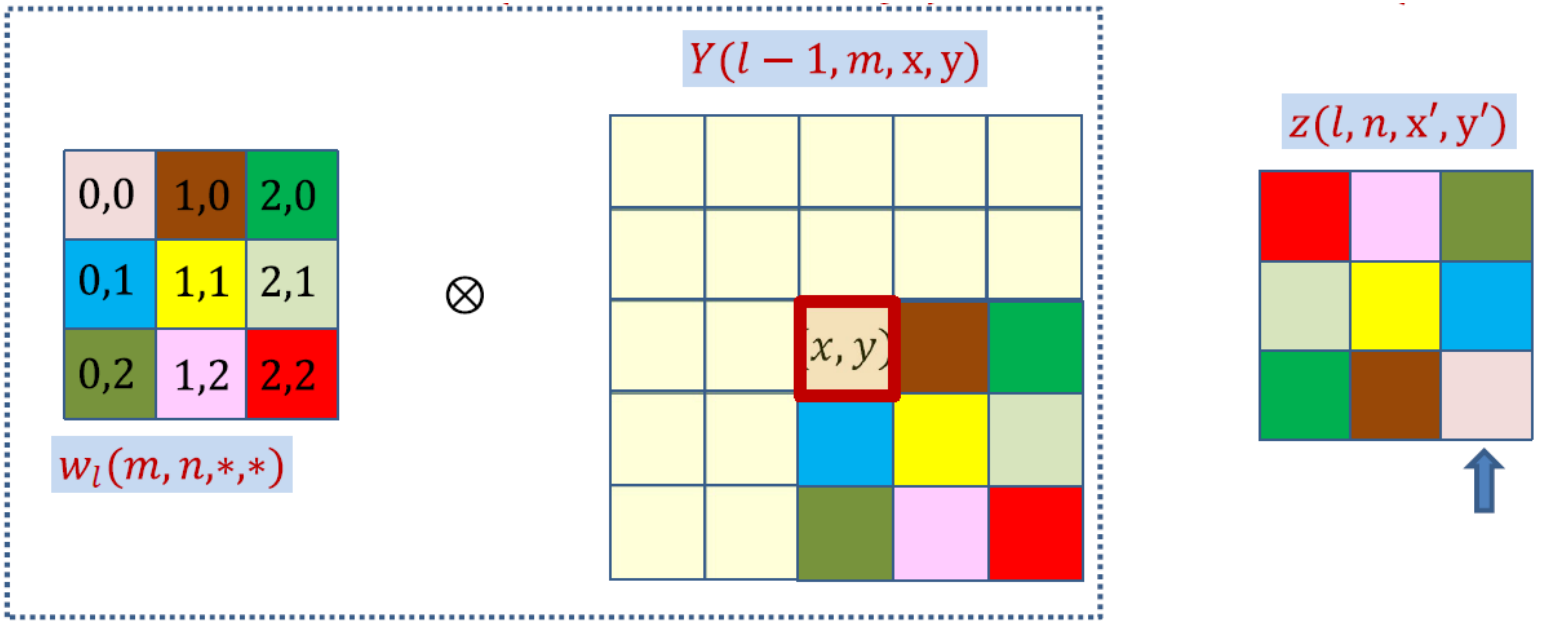
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



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

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

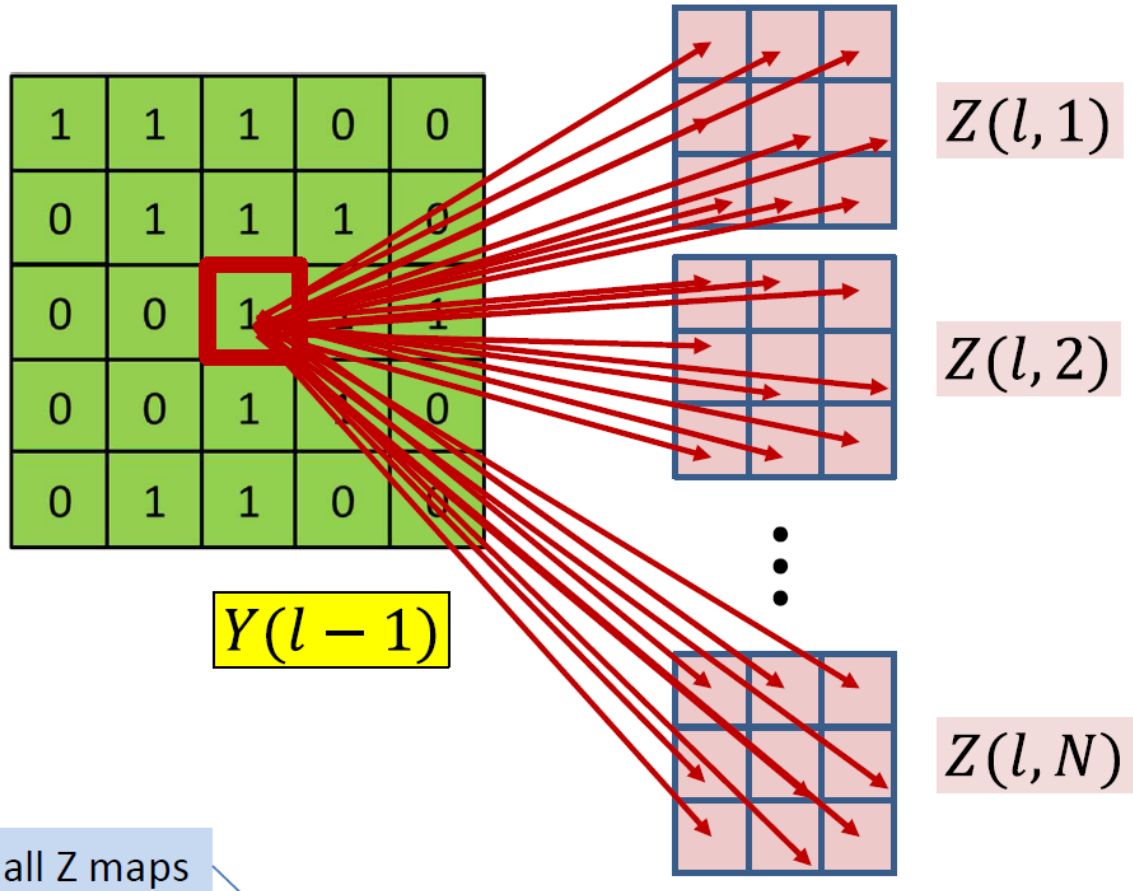
How a single $Y(l-1,m,x,y)$ influences $z(l,n,x',y')$



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

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

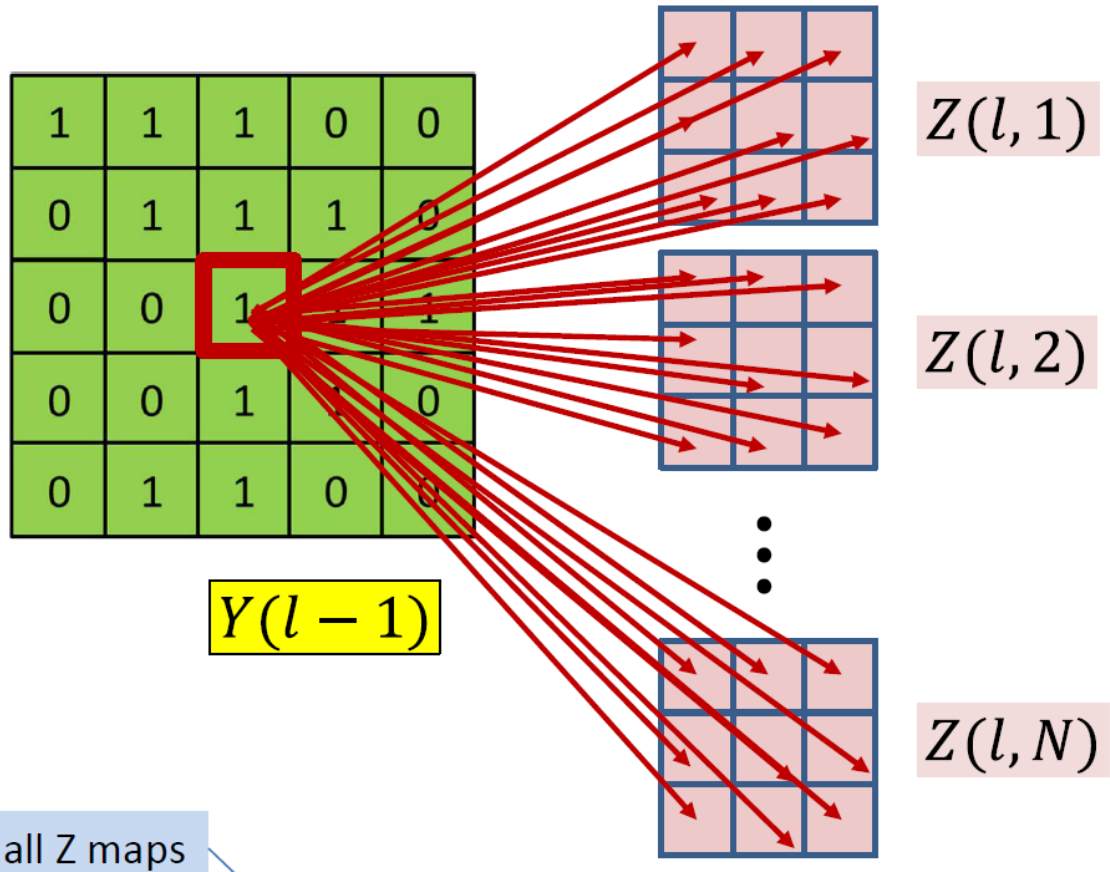
BP: Convolutional layer



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

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

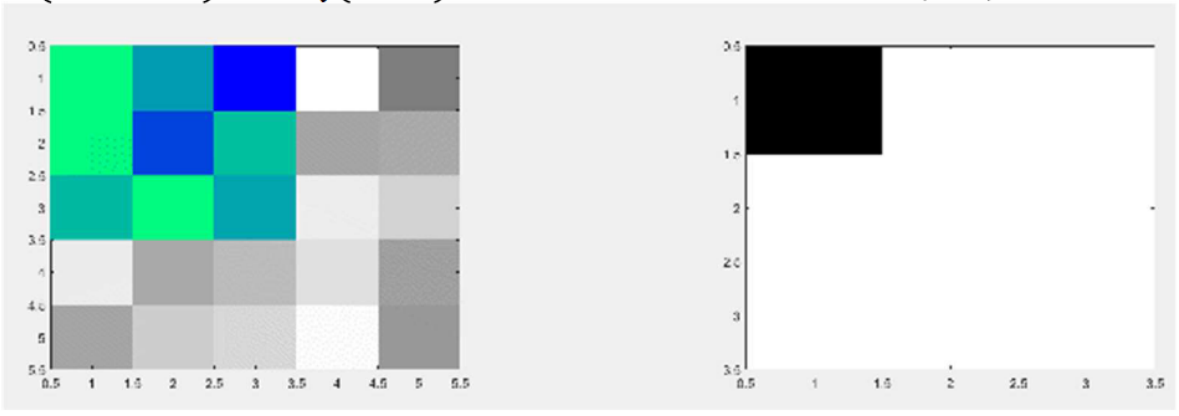
Backpropagating through affine map

- 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

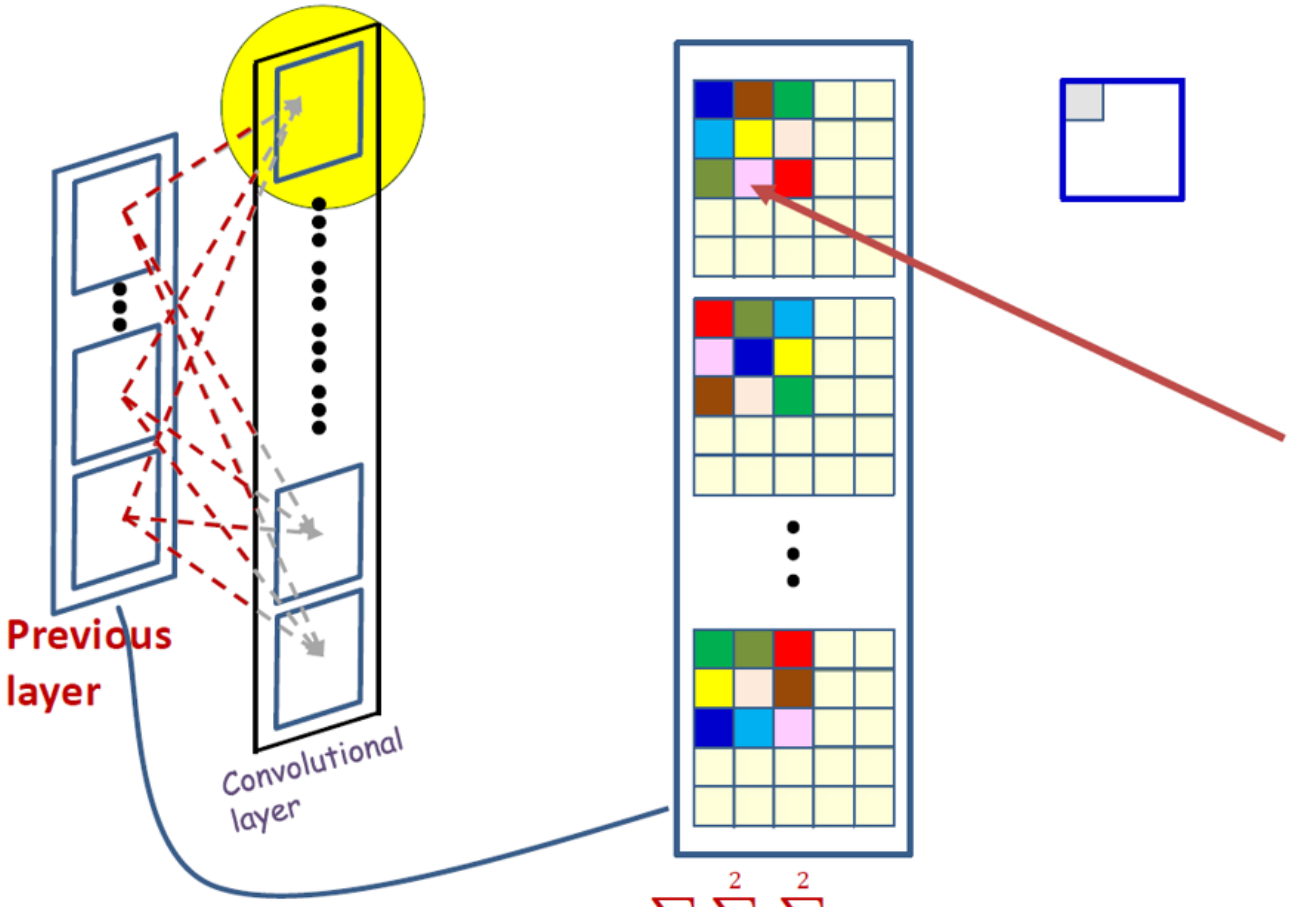
$Y(l-1,m) \otimes w_l(m,n)$ $\quad$ $Z(l,n)$



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

- 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. $w_l(m,n,1,2)$)

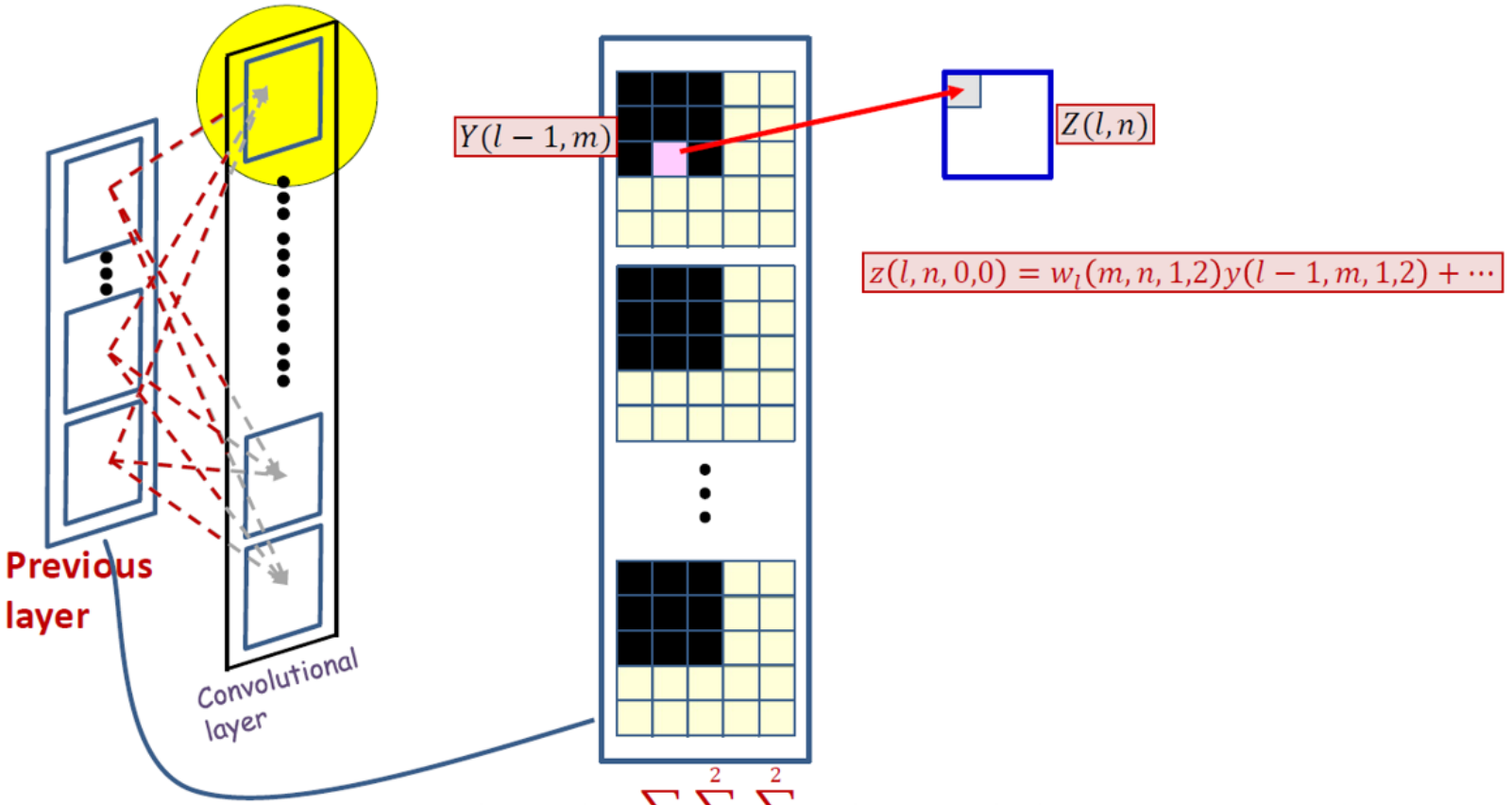
Convolution: the contribution of a single weight



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

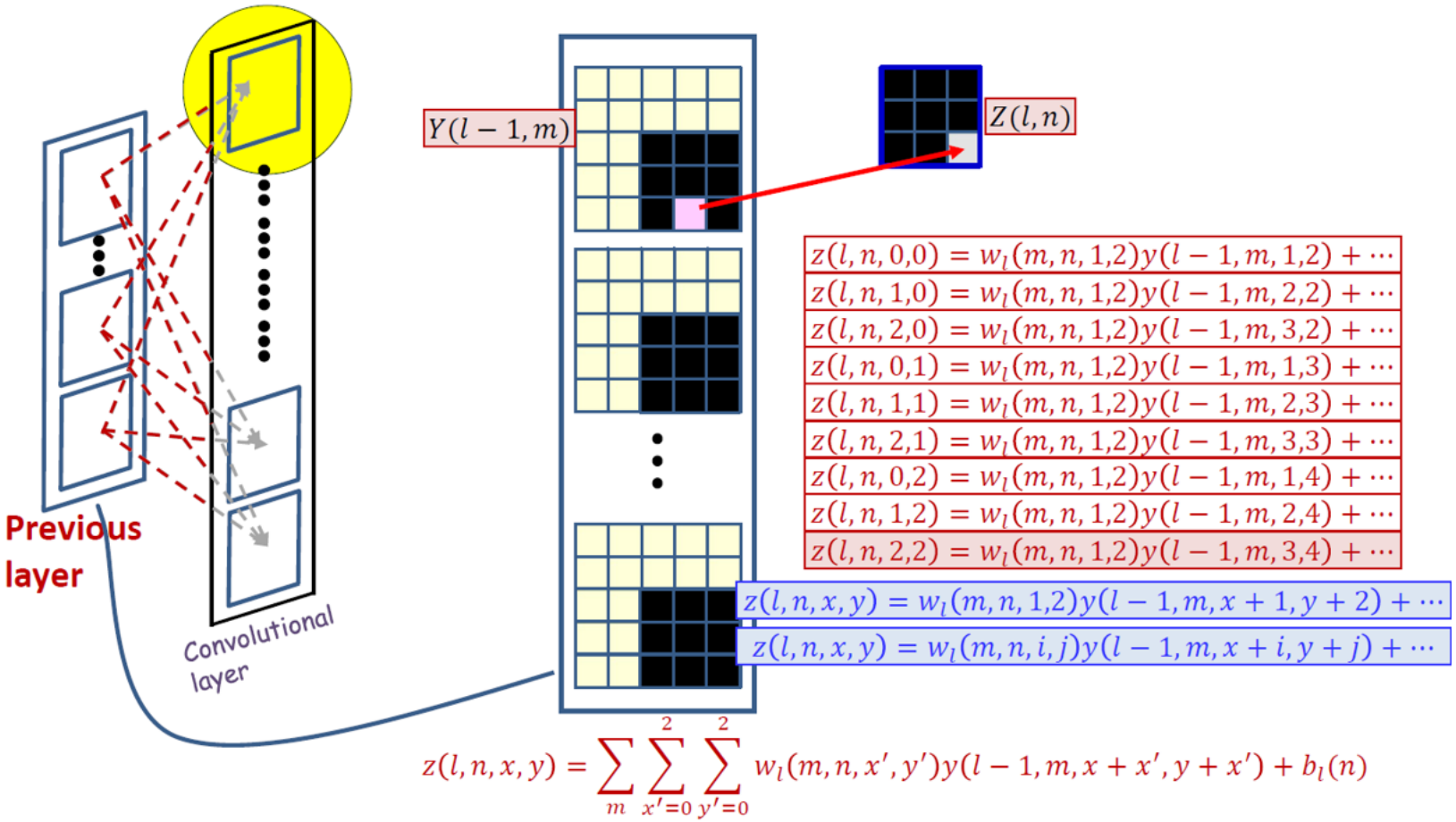
- Each affine output is computed from multiple input maps simultaneously
- Each ***weight*** $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$

Convolution: the contribution of a single weight



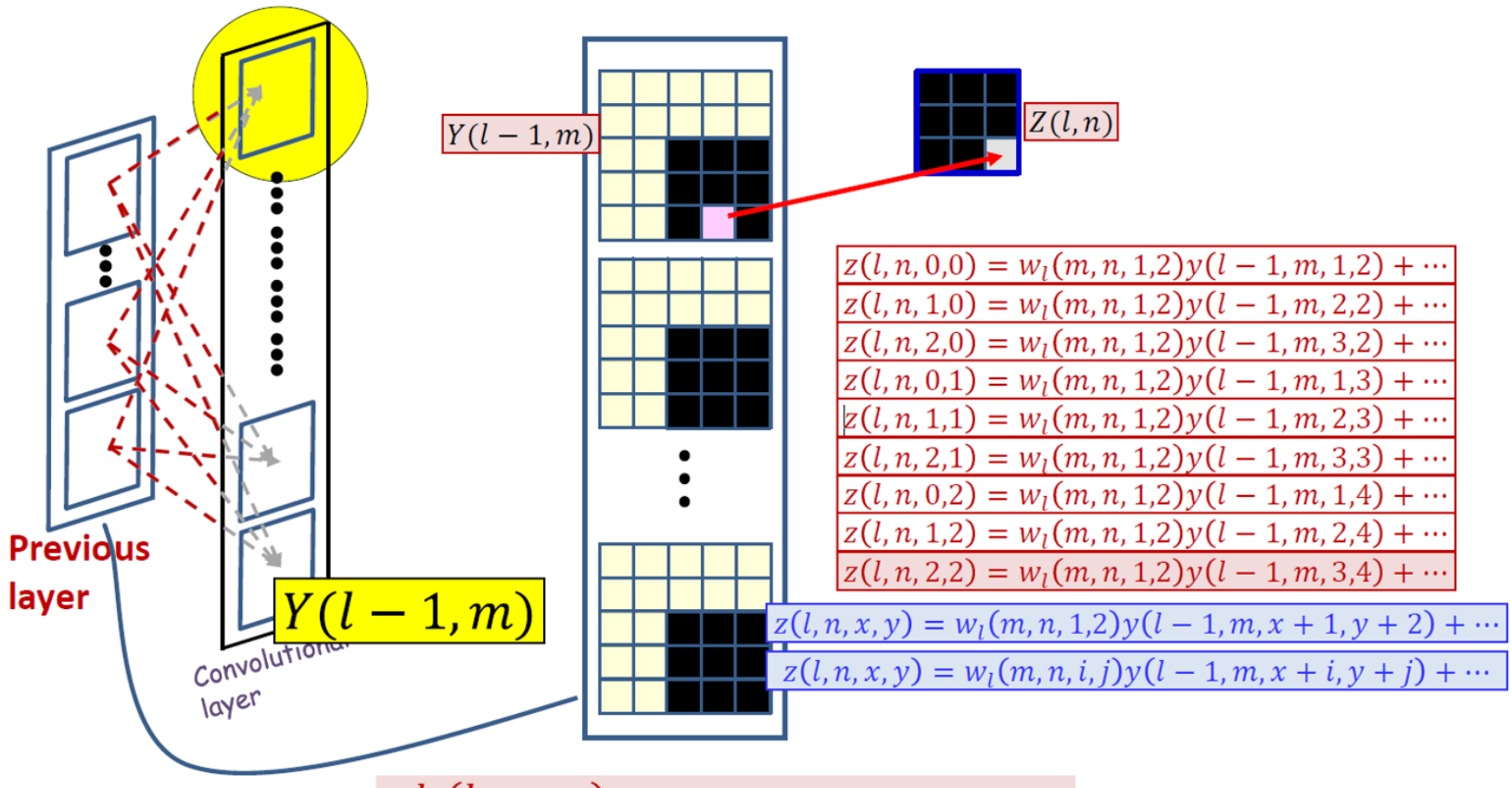
- Each weight $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$
  - Consider the contribution of one filter components: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single weight



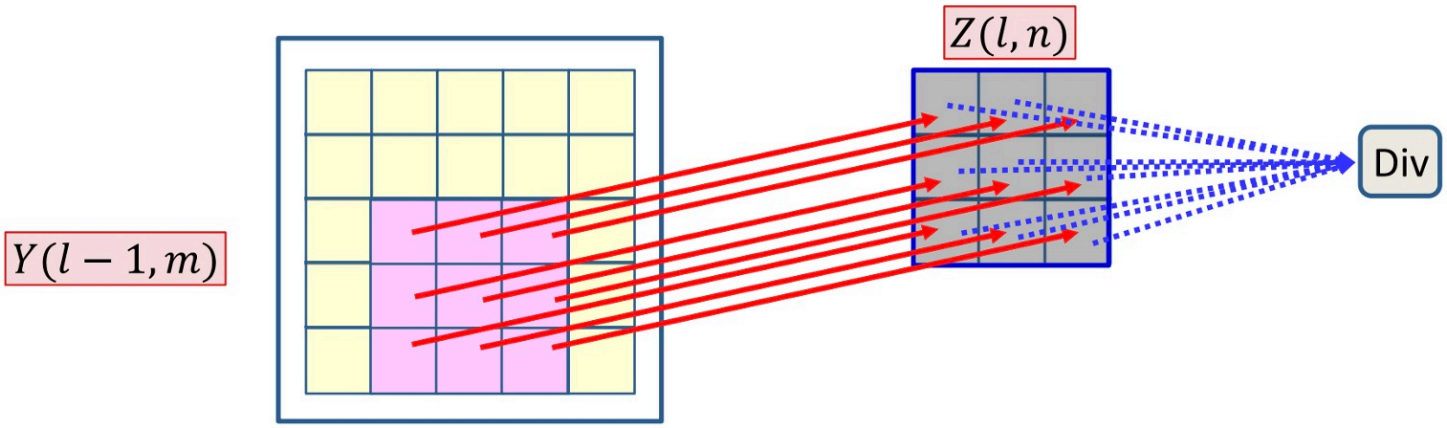
- Each weight $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$
  - Consider the contribution of one filter components: e.g. $w_l(m,n,1,2)$

Convolution: the contribution of a single weight



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

The derivative for a single weight



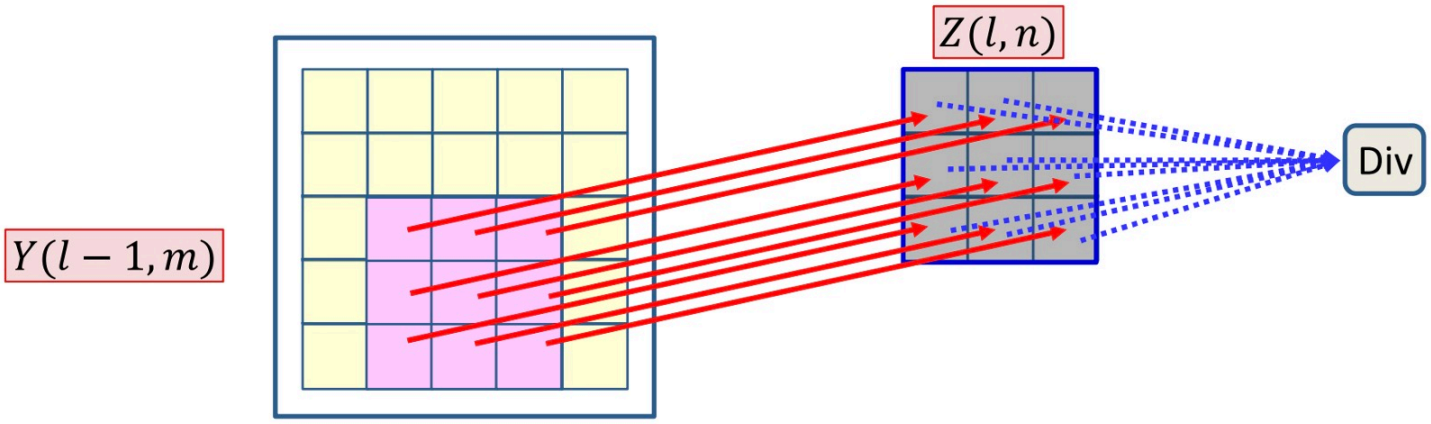
- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$
  - The derivative of each $z(l,n,x,y)$ w.r.t. $w_l(m,n,i,j)$ is given by

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

- 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

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

The derivative for a single weight



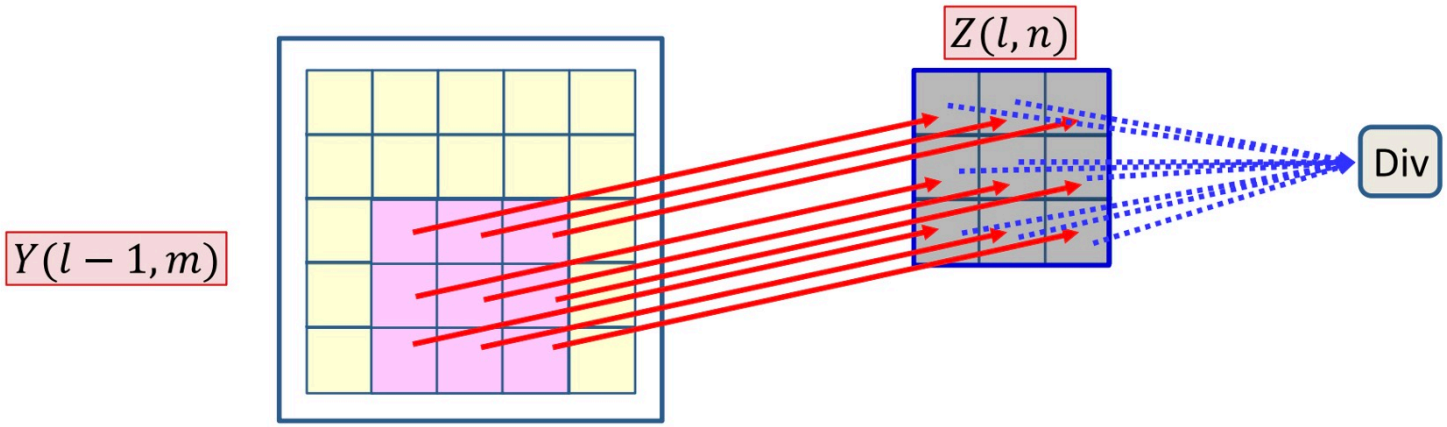
- Each filter component $w_l(m,n,i,j)$ affects several $z(l,n,x,y)$
  - The derivative of each $z(l,n,x,y)$ w.r.t. $w_l(m,n,i,j)$ is given by

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

- The final divergence is influenced by *every* $z(l,n,x,y)$
- The derivati[Already computed]e 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

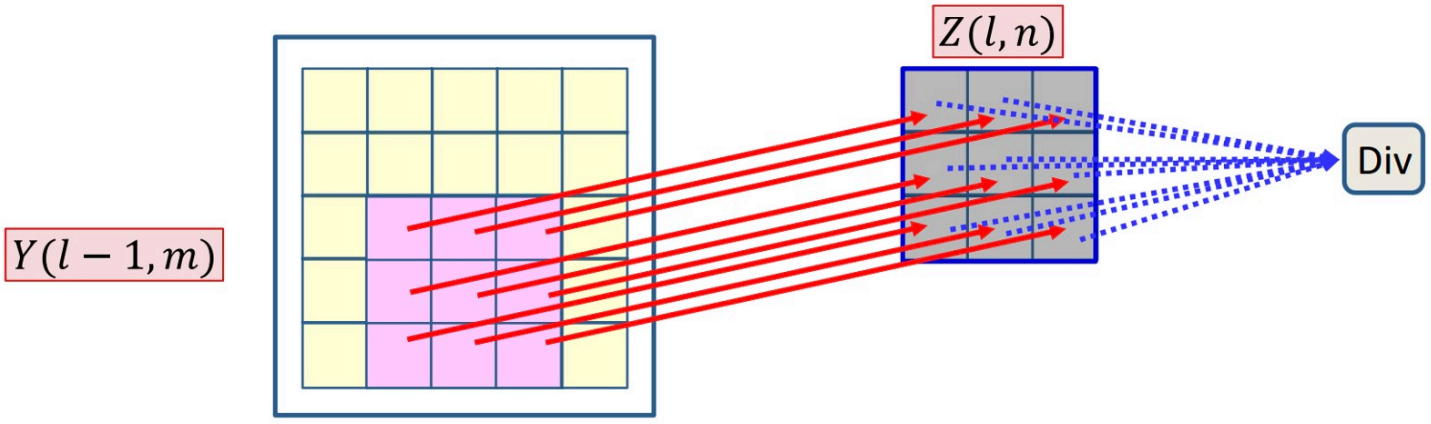
$$\frac{dz(l,n,x,y)}{dw_l(m,n,i,j)} = y(l-1,m,x+i,y+j)$$

- 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

Already computed

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

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 is also a convolution

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

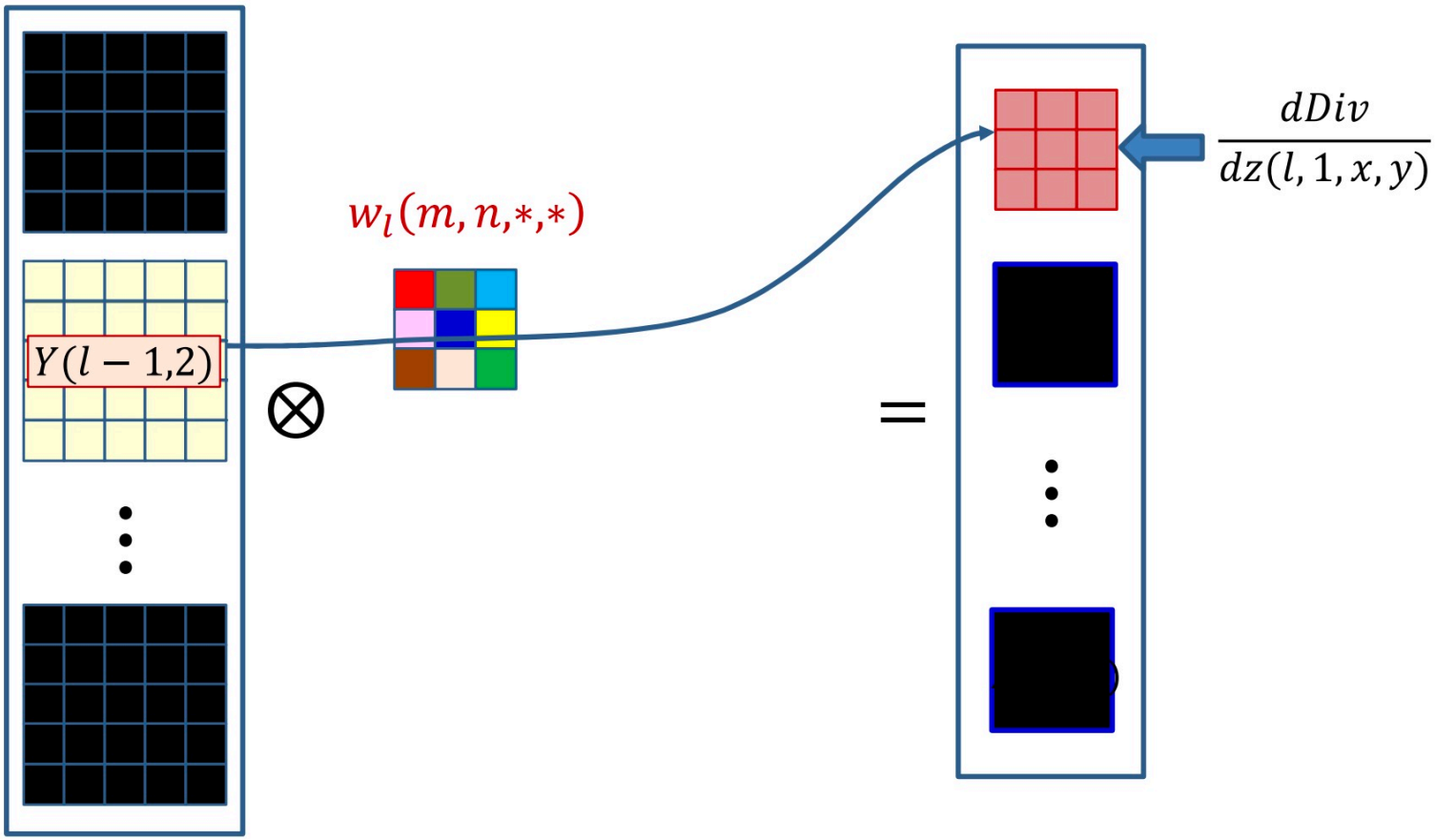
- The derivatives for all components of all filters can be computed directly from the above formula

- In fact, it is just a convolution

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

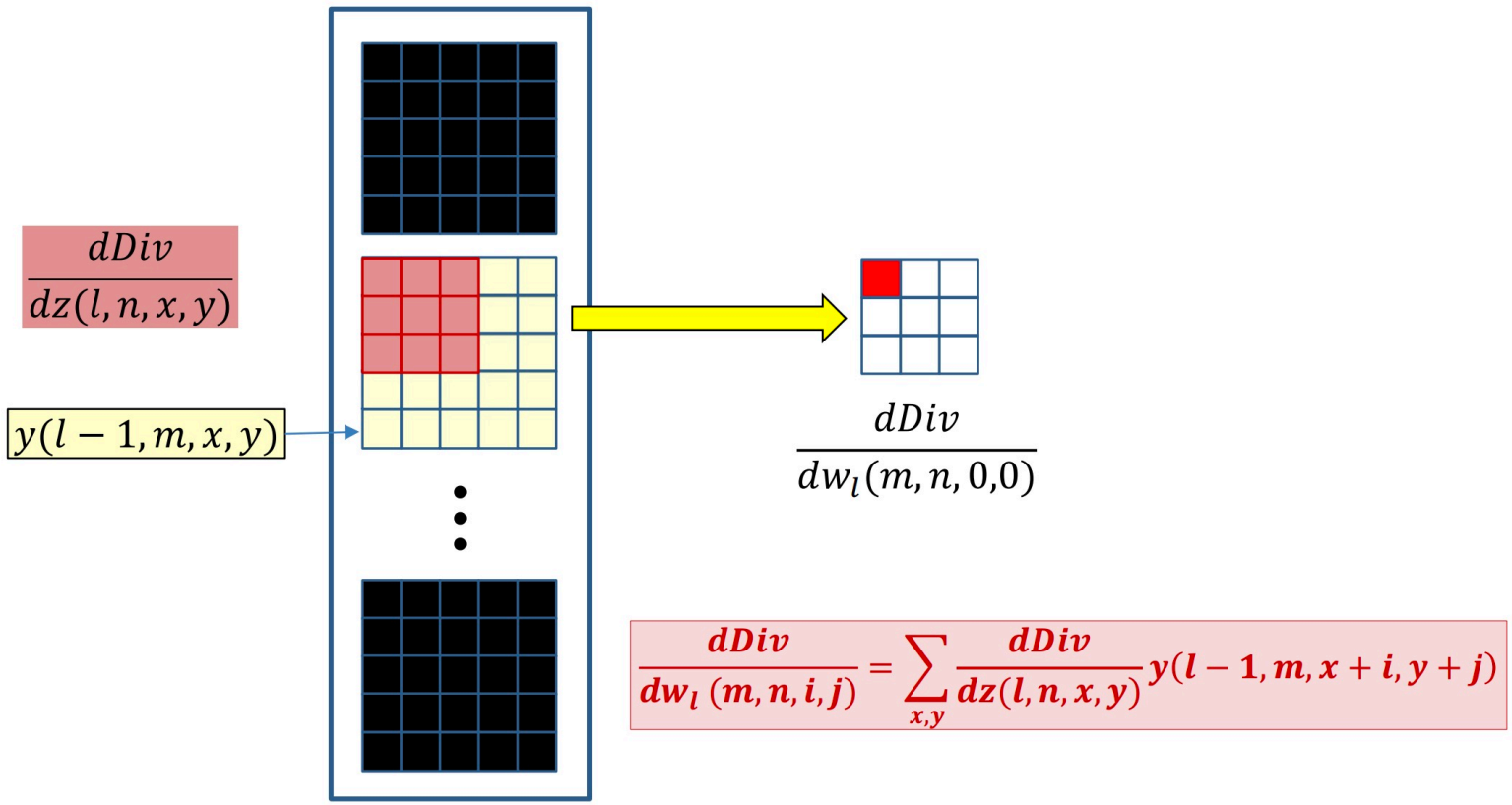
- How?

The filter derivative



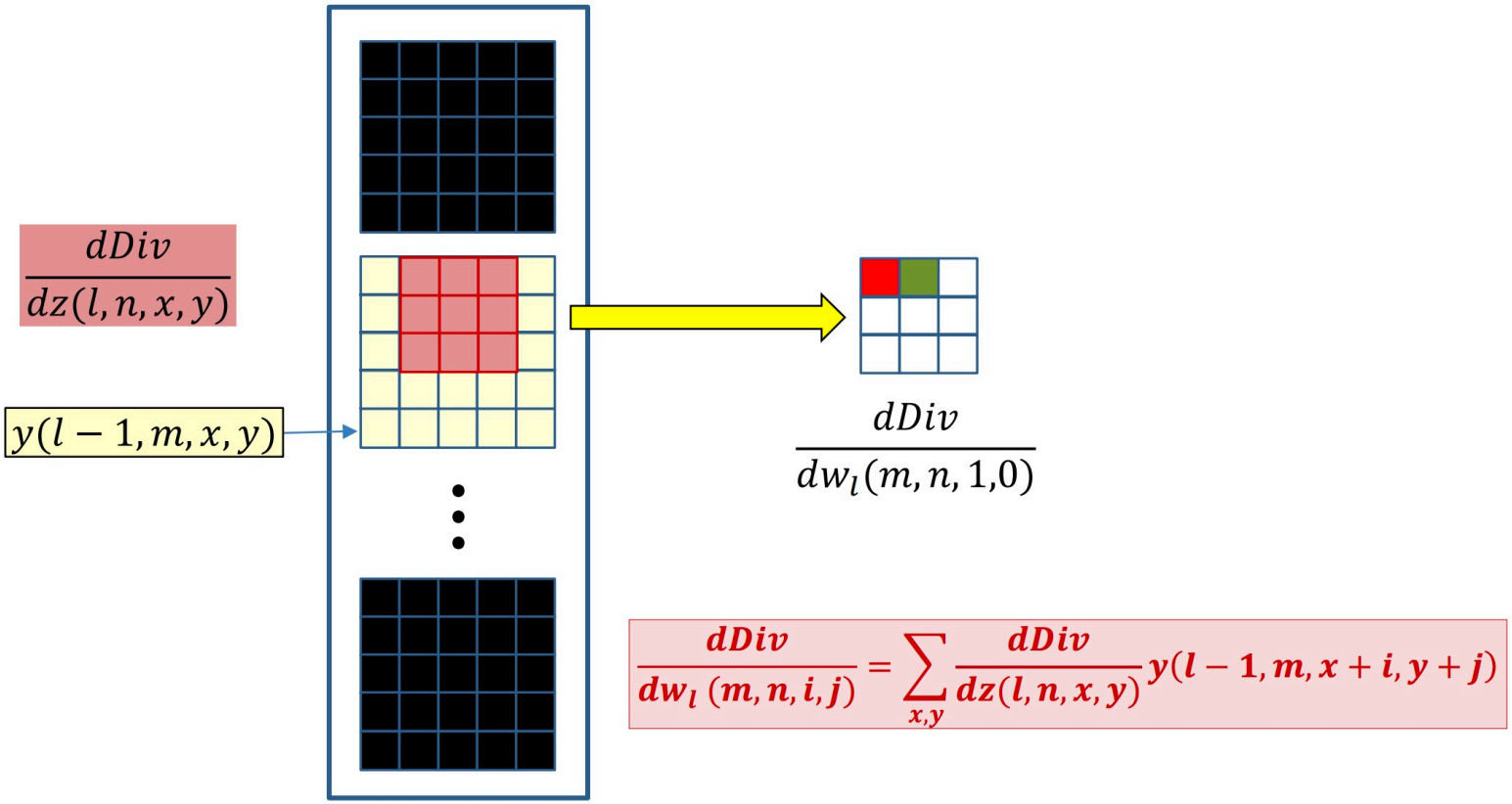
- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

The filter derivative



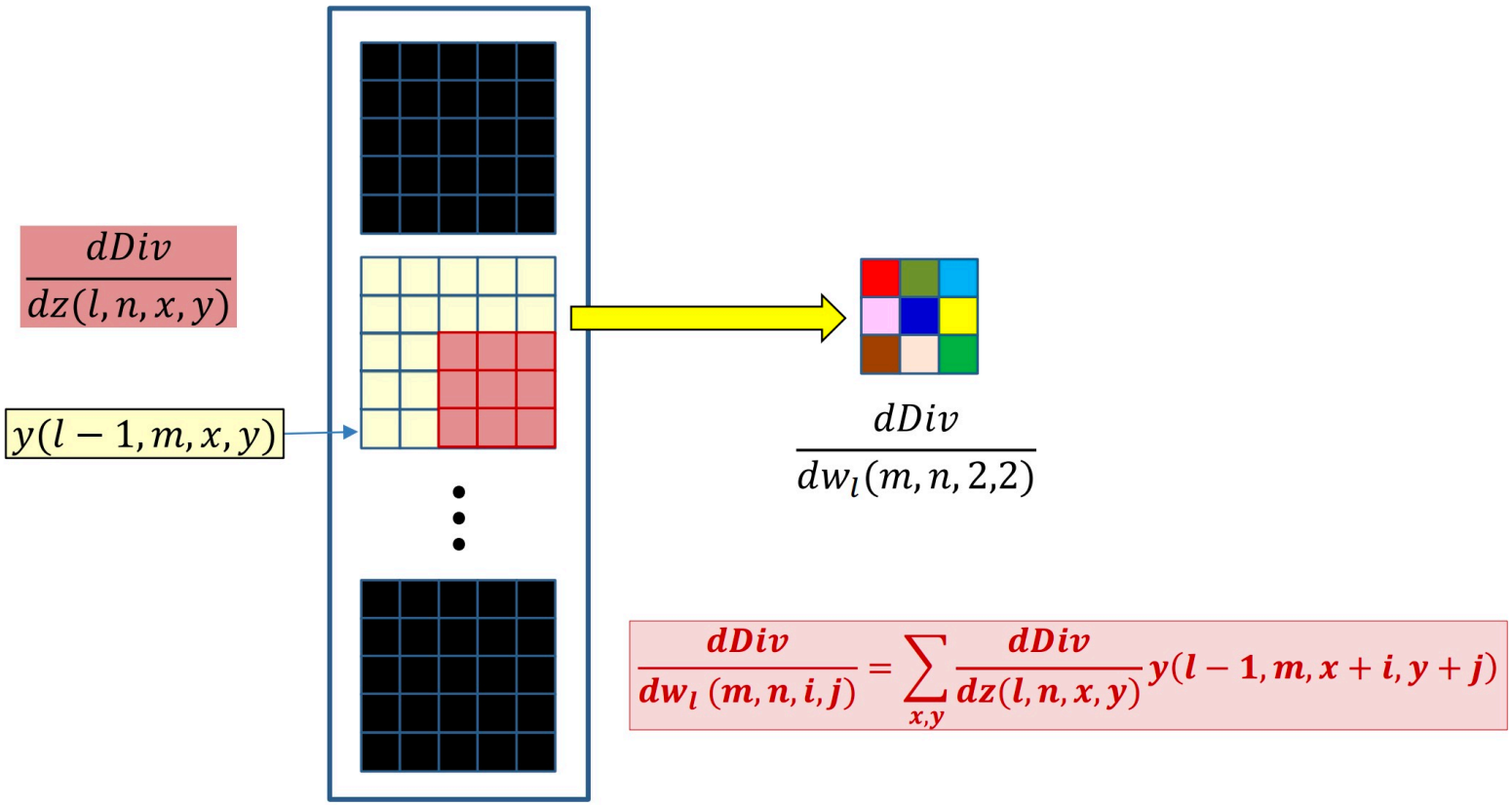
- The derivatives of the divergence w.r.t. every element of $Z(l,n)$ is known
  - Must use them to compute the derivative for $w_l(m,n,*,*)$

The filter derivative



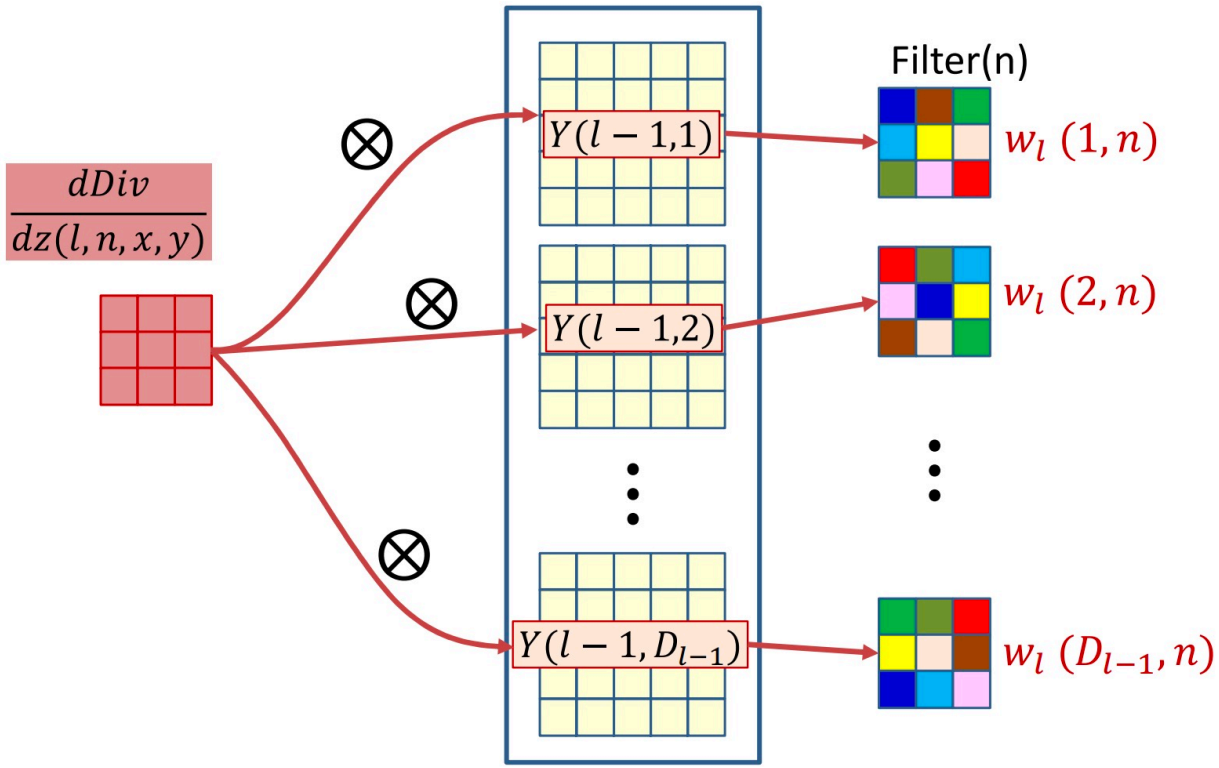
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
  - Must use them to compute the derivative for $w_l(m, n, *, *)$

The filter derivative



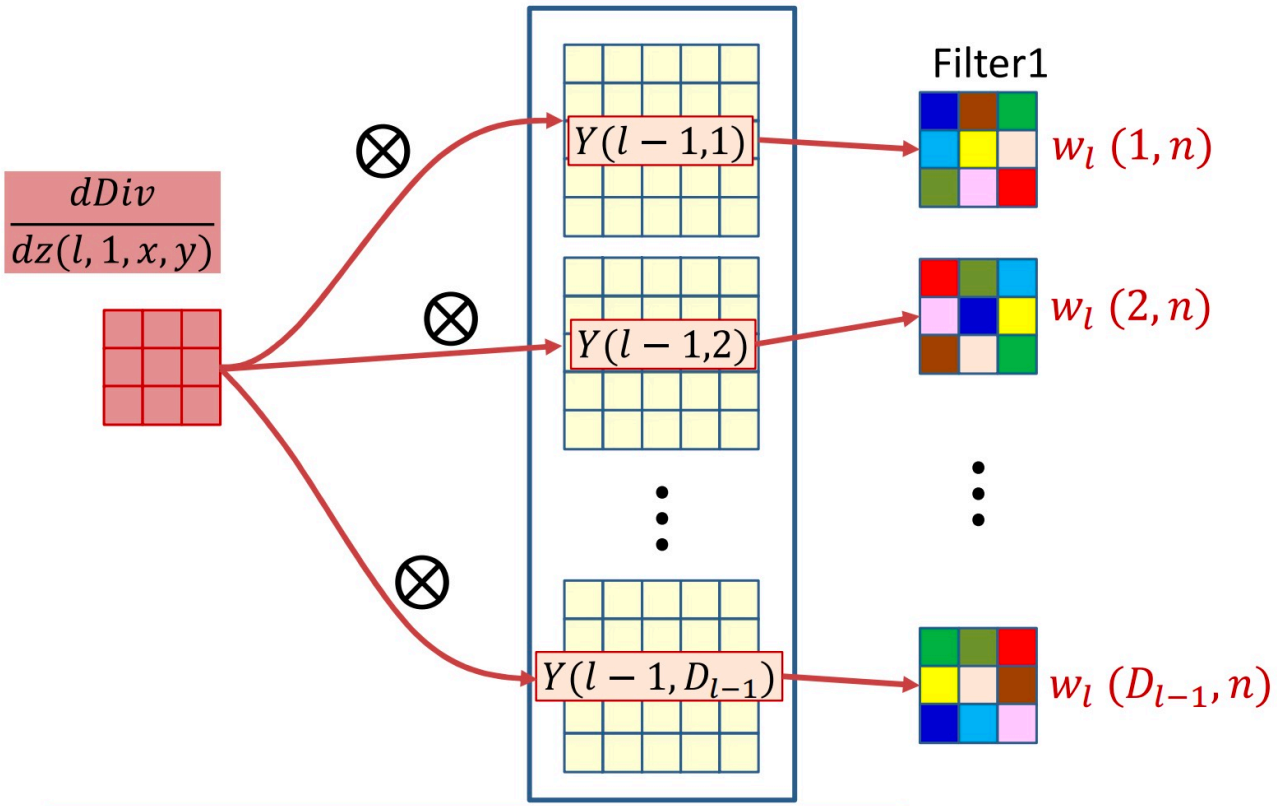
- The derivatives of the divergence w.r.t. every element of $Z(l, n)$ is known
  - Must use them to compute the derivative for $w_l(m, n, *, *)$

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

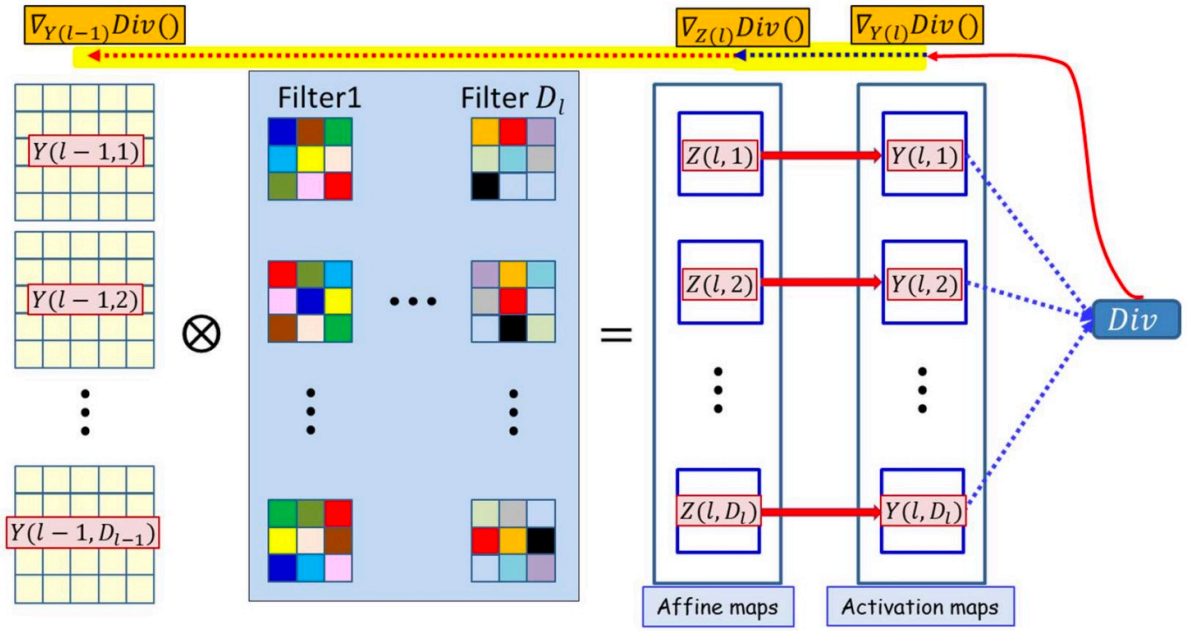


$$\frac{dDiv}{dw_l\ (m,n,i,j)} = \sum_{x,y} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

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

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

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. $Z(l)$

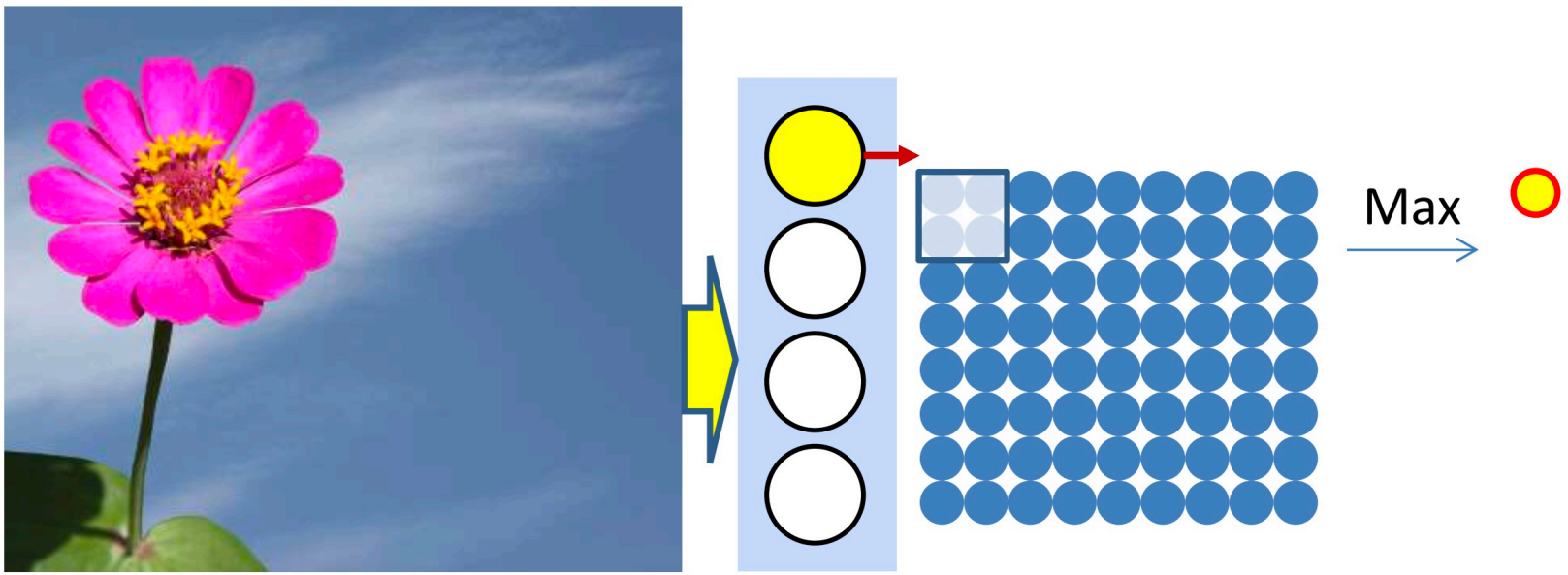
Backpropagation: Convolutional and Pooling layers

- **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. $Y(l)$

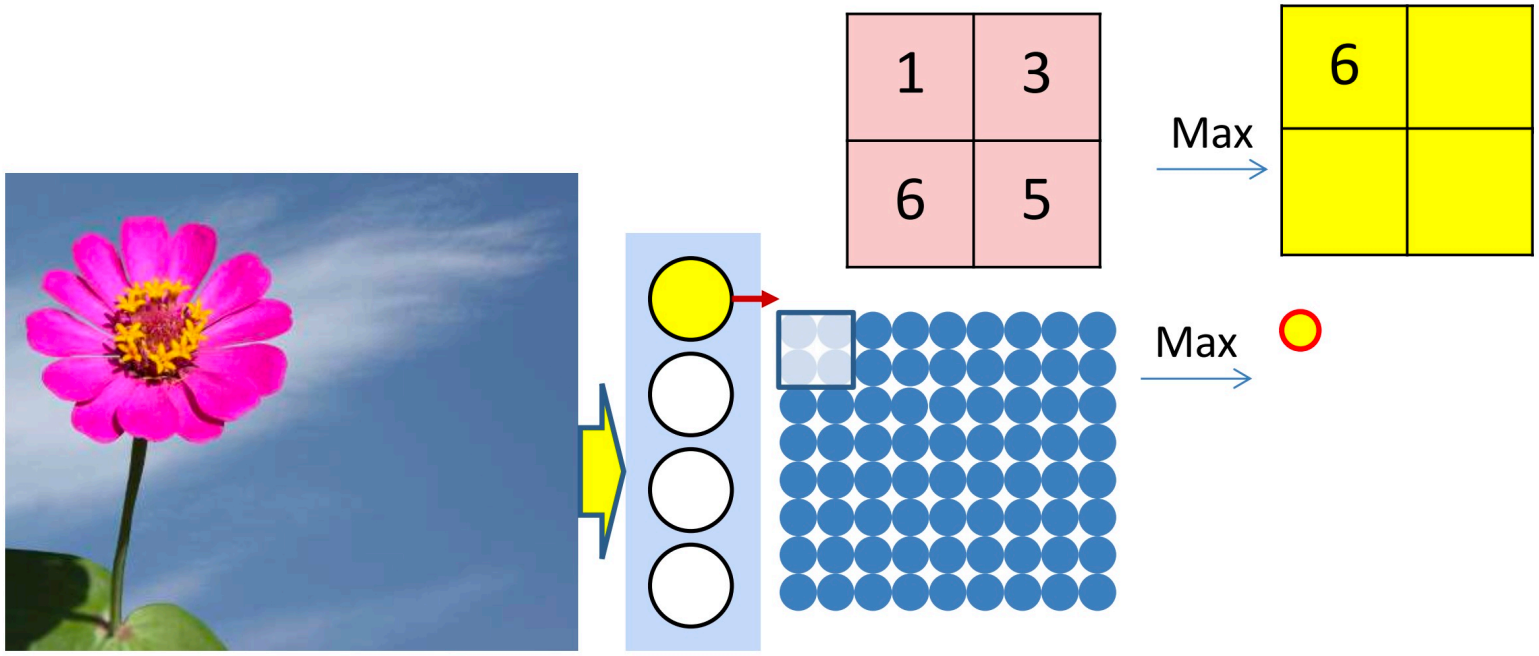
Lecture 7.3 Derivative of Pooling Layer

Pooling



- Pooling "pools" groups of values to reduce jitter-sensitivity
  - Scanning with a "pooling" filter
- The most common pooling is "Max" pooling

Max Pooling

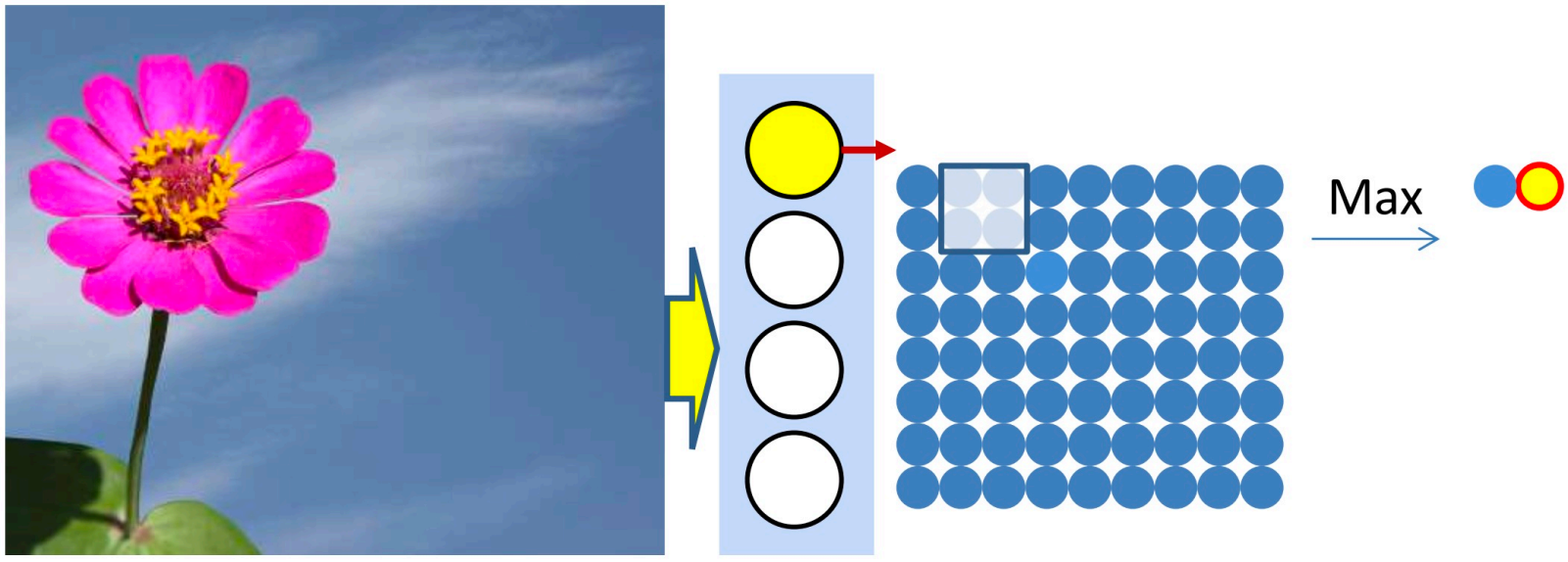


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

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

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

Max Pooling

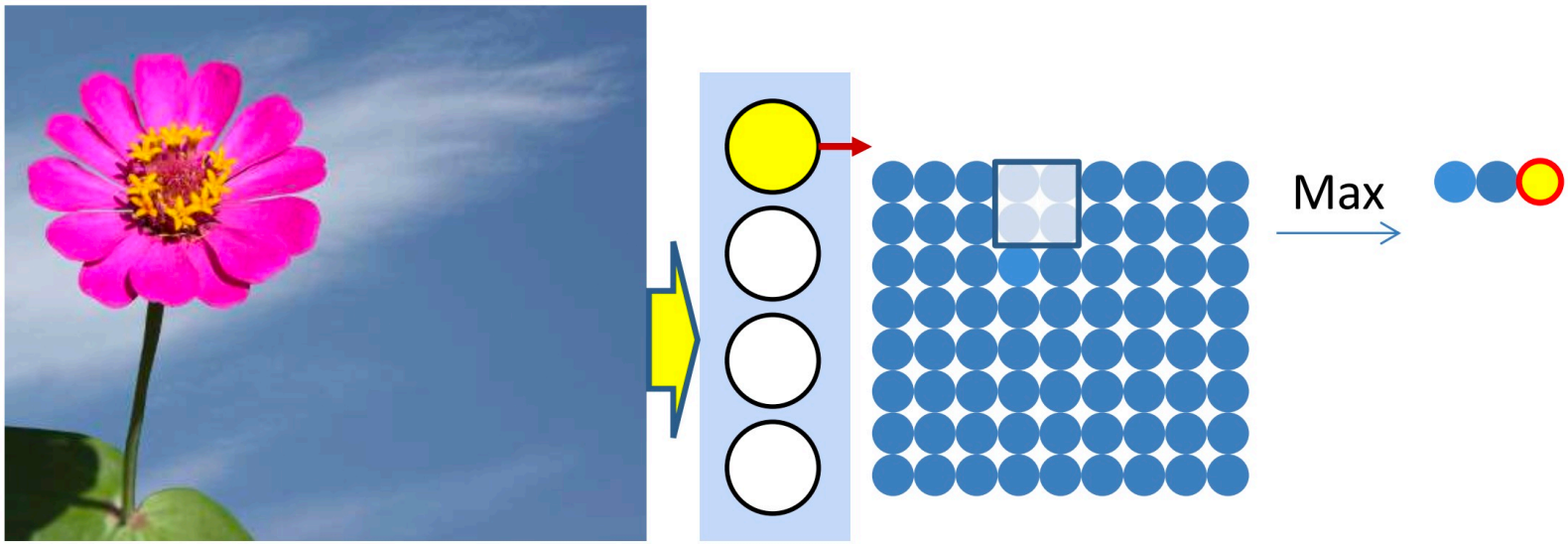


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

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

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

Max Pooling

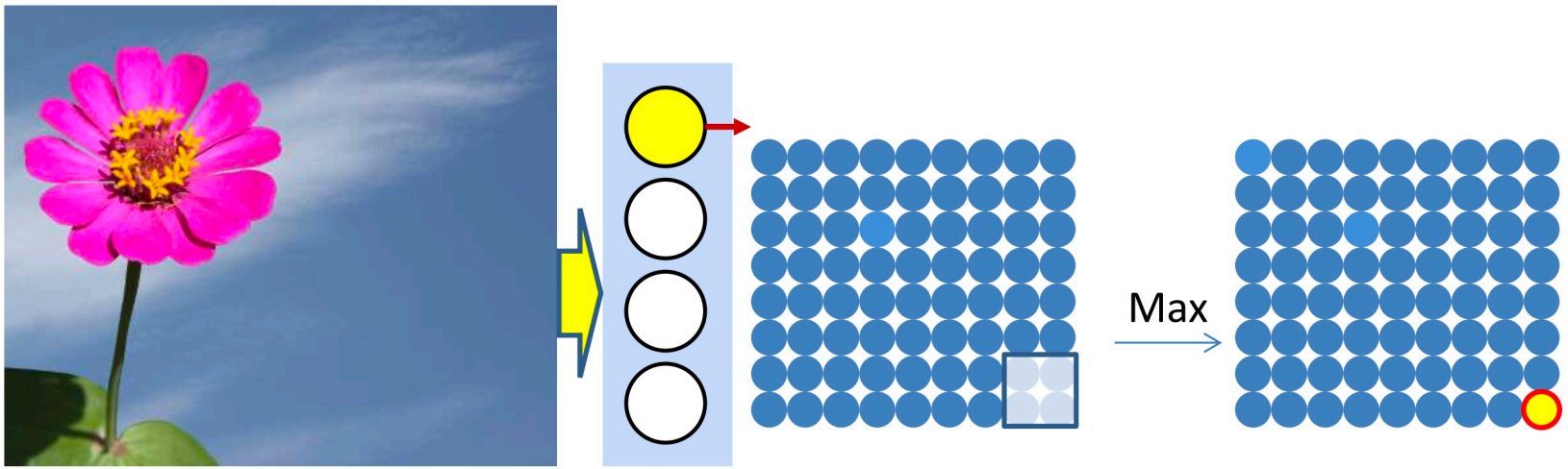


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

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

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

Max Pooling

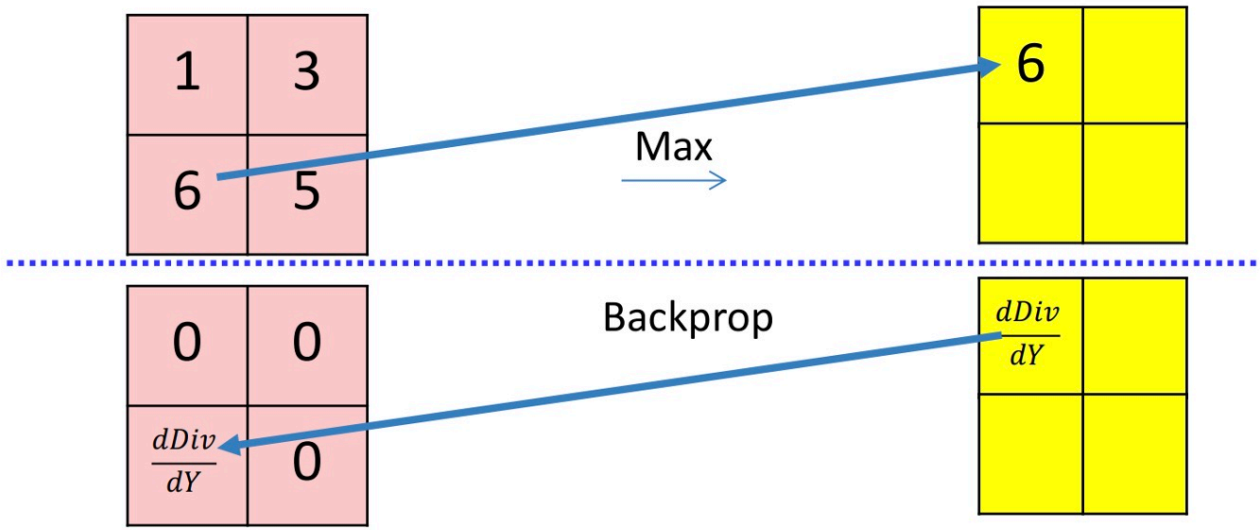


- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

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

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

Max Pooling



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

- Max pooling selects the largest from a pool of elements

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

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

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:D_l
  for x = 1:W_{l-1}-K_l+1
    for y = 1:H_{l-1}-K_l+1
      pidx(l,j,x,y) = maxidx(y(l-1,j,x:x+K_l-1,y:y+K_l-1))
      y(l,j,x,y) = y(l-1,j,pidx(l,j,x,y))
```

Derivative of Max Pooling layer at layer l

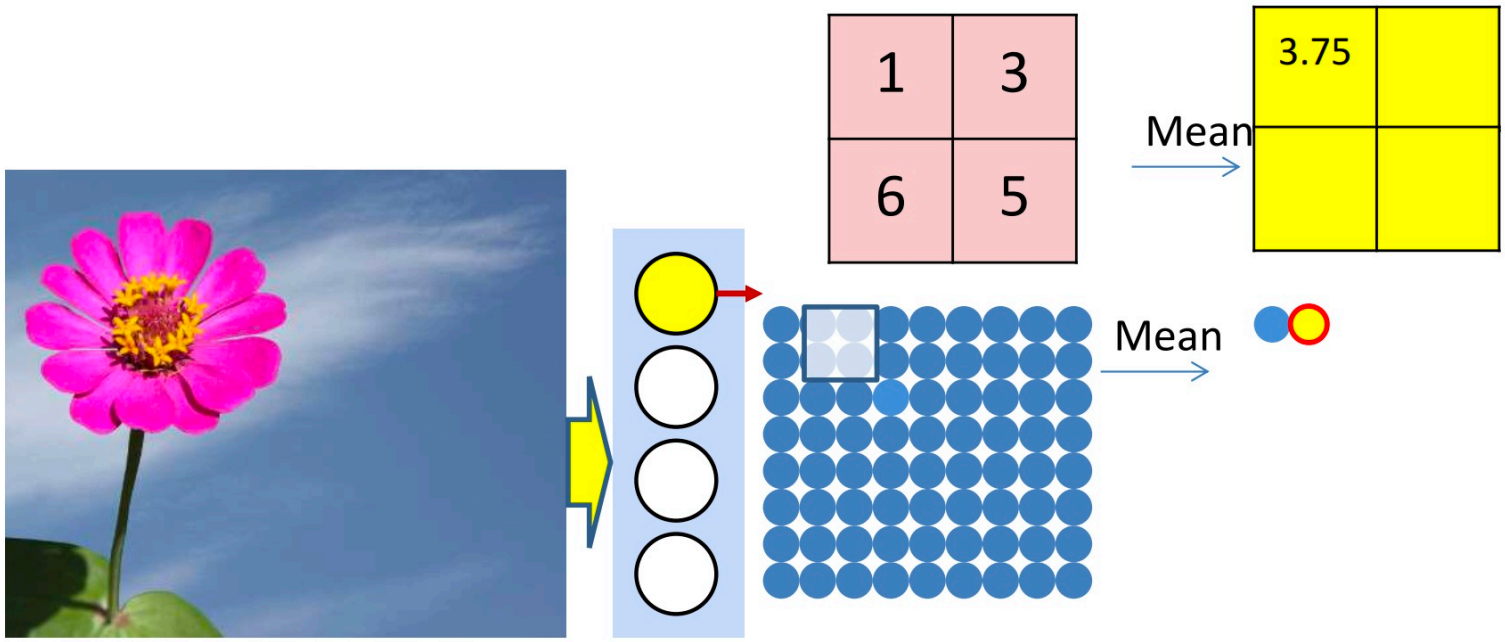
a) Performed separately for every map (j).
*) *Not* combining multiple maps within a single max operation.
b) Keeping track of location of max

Max pooling

```
dy(:,:,:) = zeros(D_l x W_l x H_l)
for j = 1:D_l
    for x = 1:W_l
        for y = 1:H_l
            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

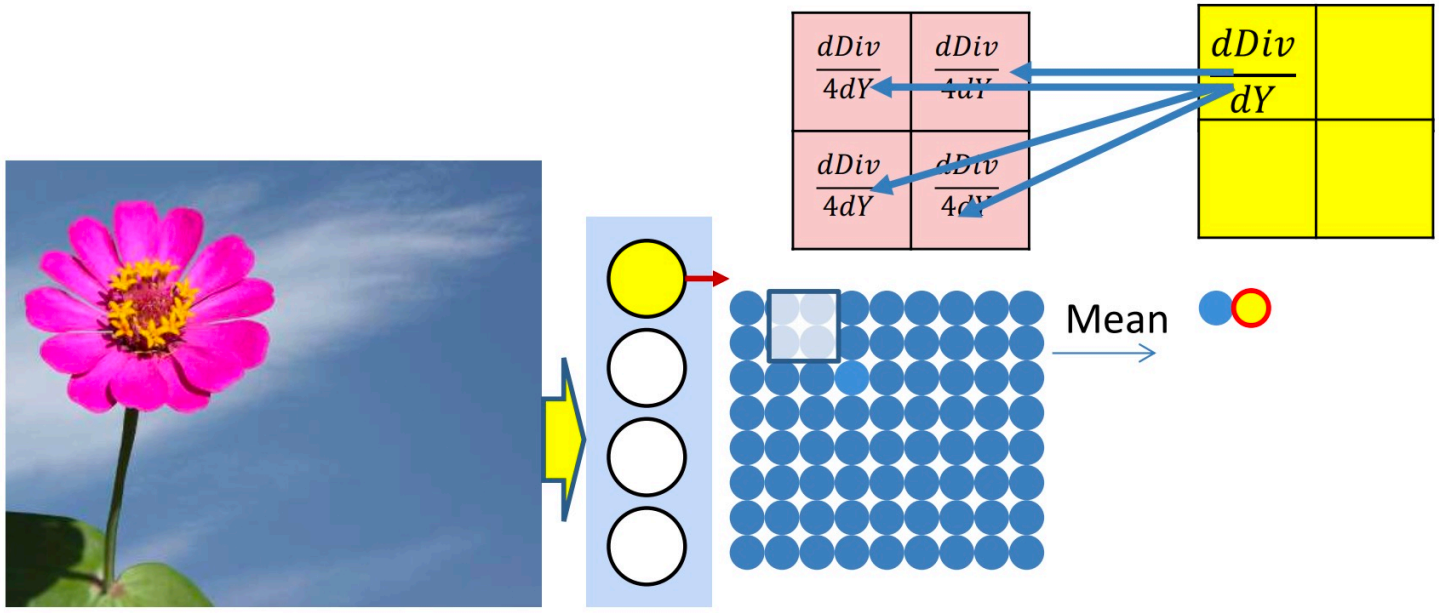
Mean pooling



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

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

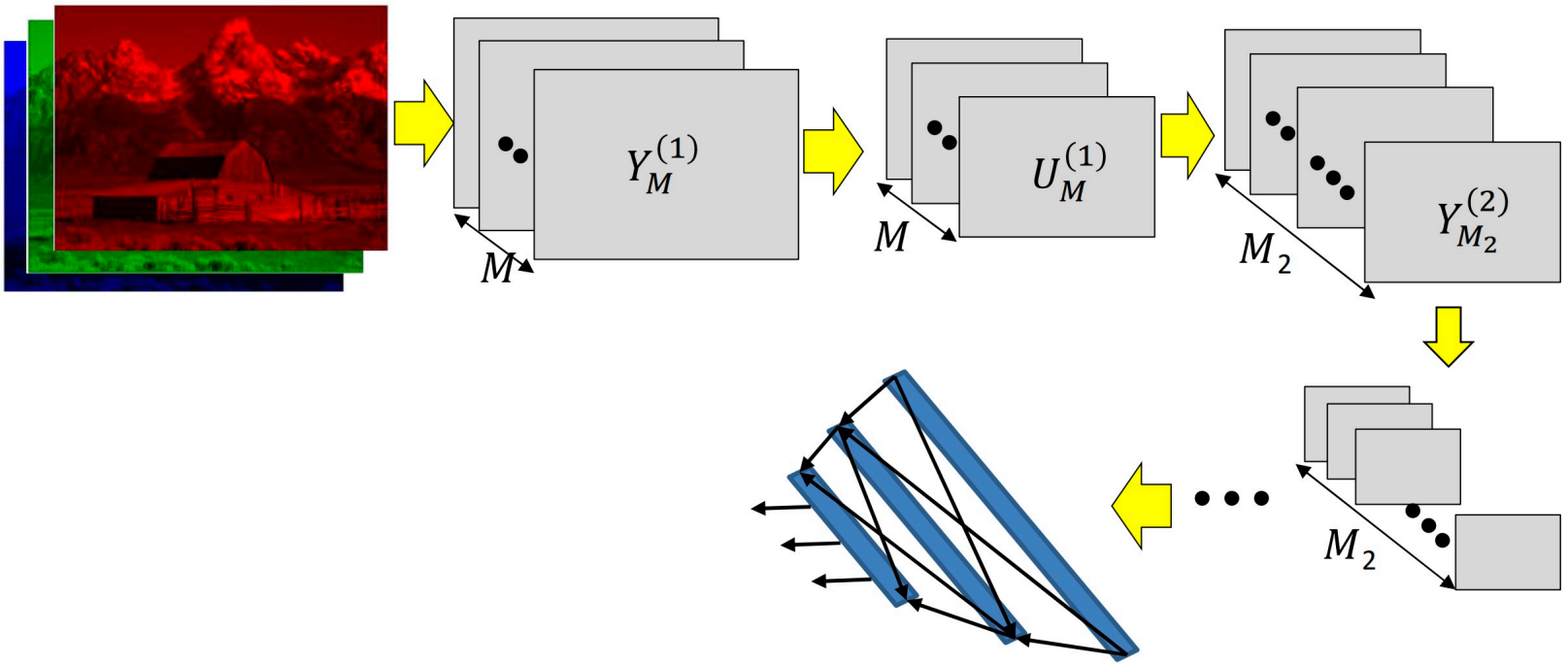
Derivative of mean pooling



- The derivative of mean pooling is distributed over the pool

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

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

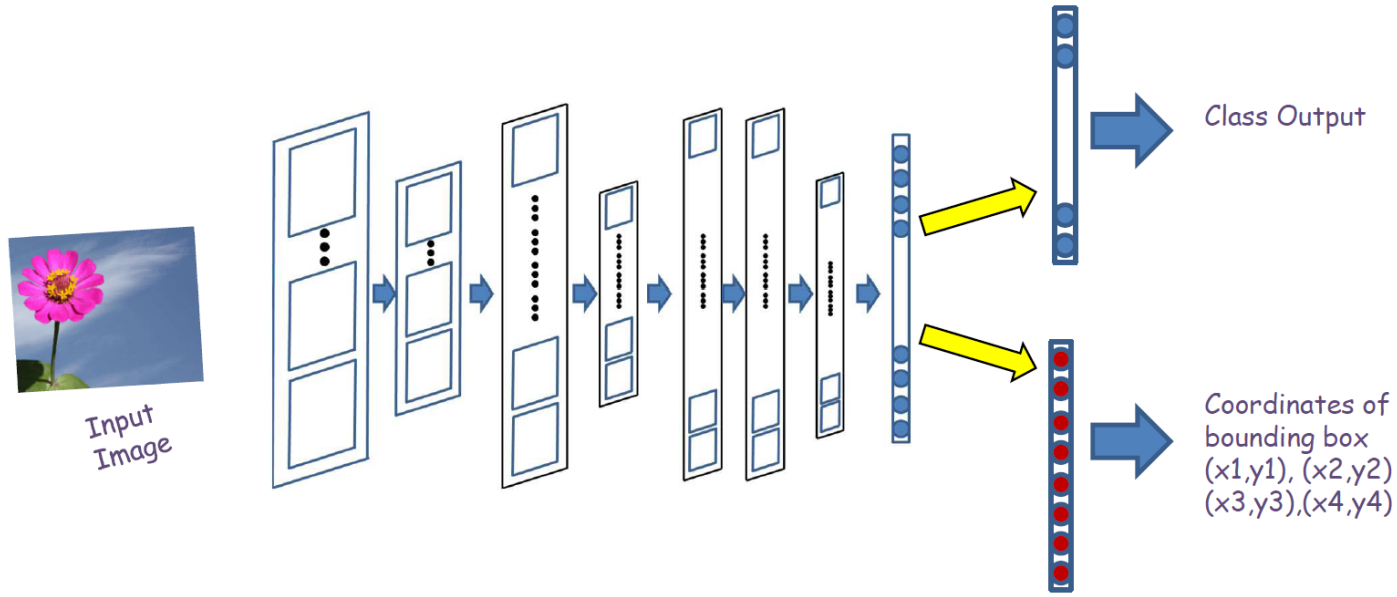
- The convolutional neural network is a supervised version of a computational model of mammalian vision
- It includes
  - Convolutional layers comprising learned filters that scan the outputs of the previous layer
  - Pooling layers that operate over groups of outputs from the convolutional layer to 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

Lecture 7.4 More Issues

What about the exact location?

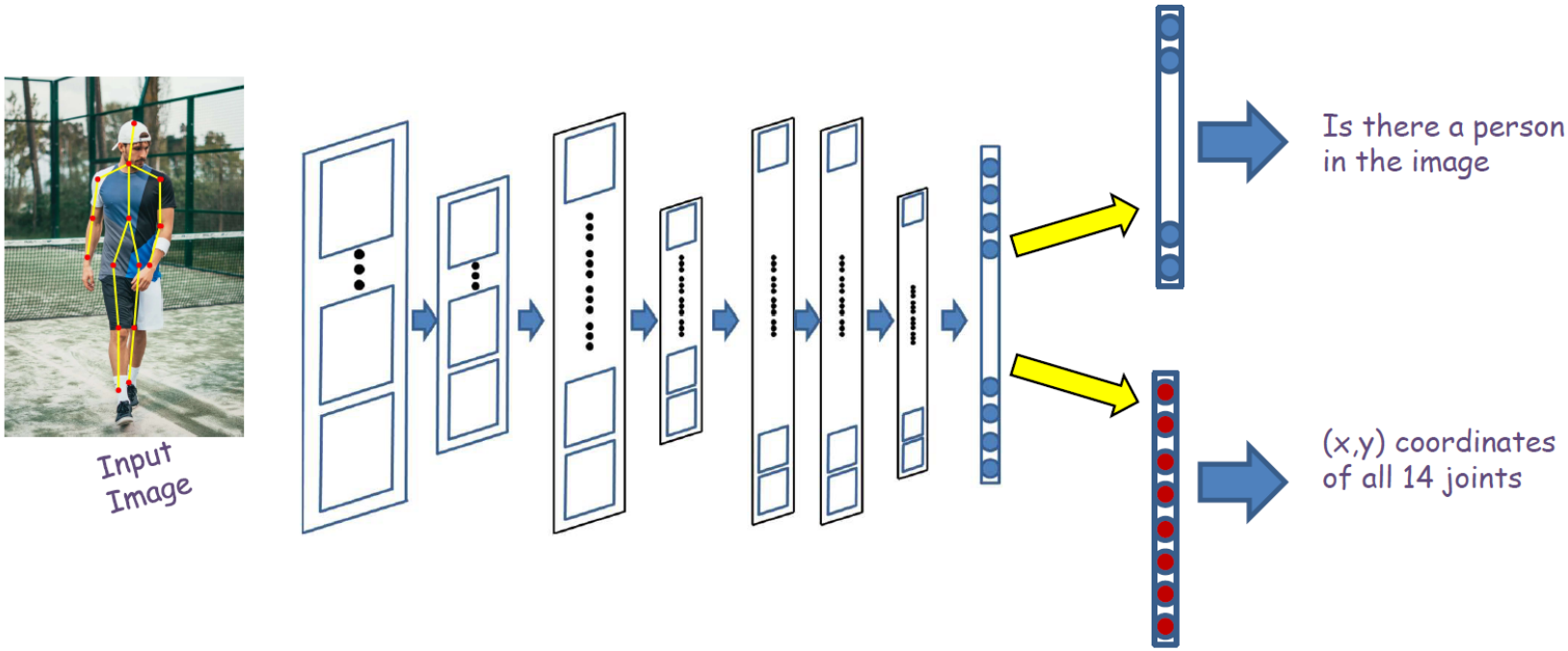
- We began with the desire to identify the picture as containing a flower, regardless of the position of the flower
  - Or more generally the class of object in the picture
- But can we detect the position of the main object?

Finding Bounding Boxes



- The flatten layer outputs to two separate output layers
- One predicts the class of the output
- The second predicts the corners of the bounding box of the object (8 coordinates) in all
- The divergence is the sum of the cross-entropy loss of the classifier layer and L2 loss of the bounding-box predictor
  - Multi-task learning

Pose estimation

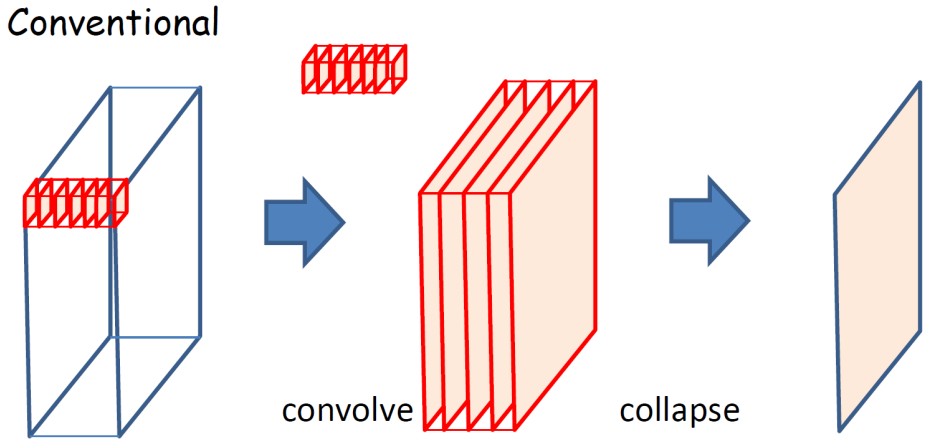


- Can use the same mechanism to predict the joints of a person
  - For pose estimation

Model variations

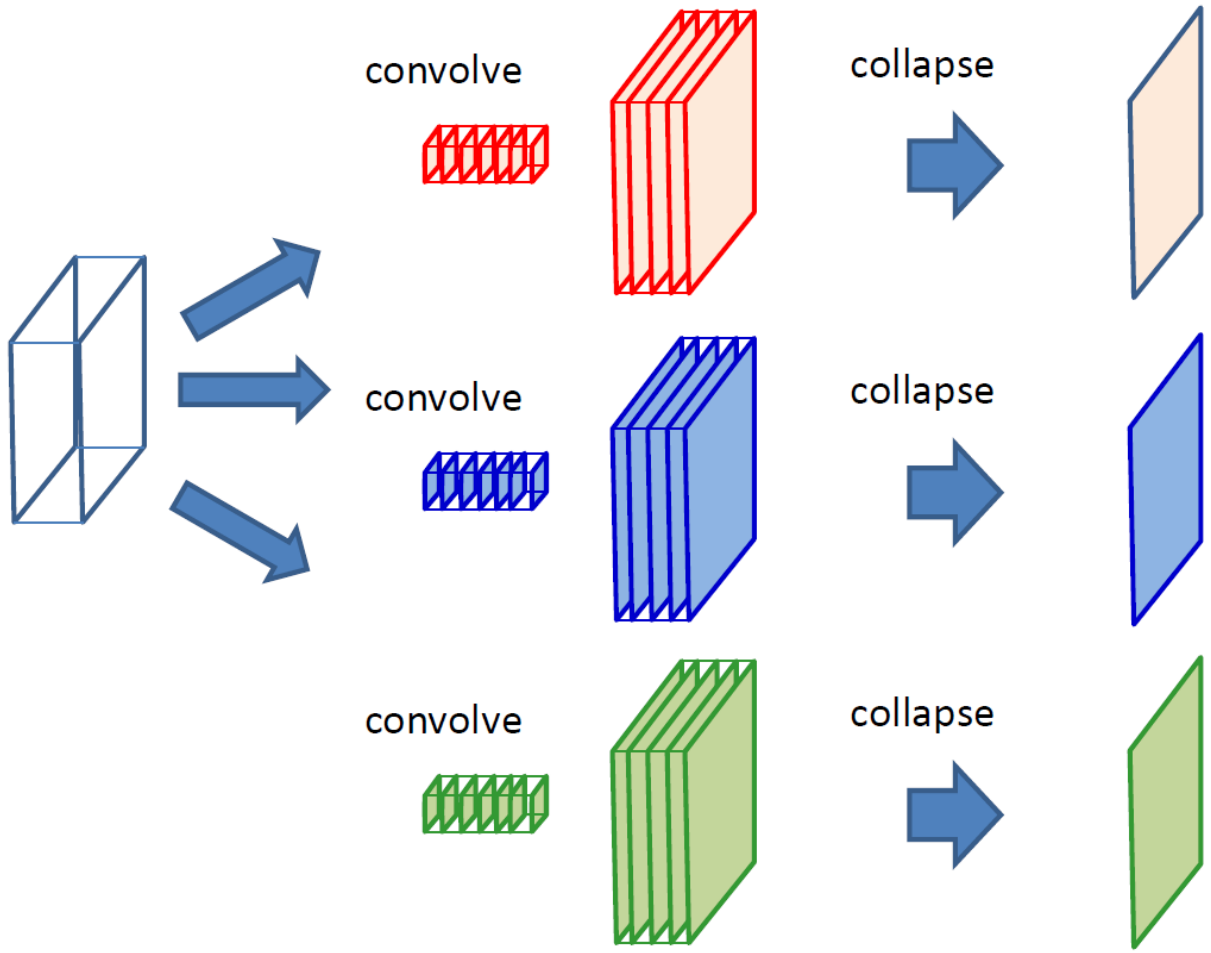
- "Depth-wise" convolutions
  - Instead of multiple independent filters with independent parameters, use common layer-wise weights and combine the layers differently for each filter
- Very deep networks
  - 100 or more layers
  - Formalism called "Resnet"

Conventional convolutions



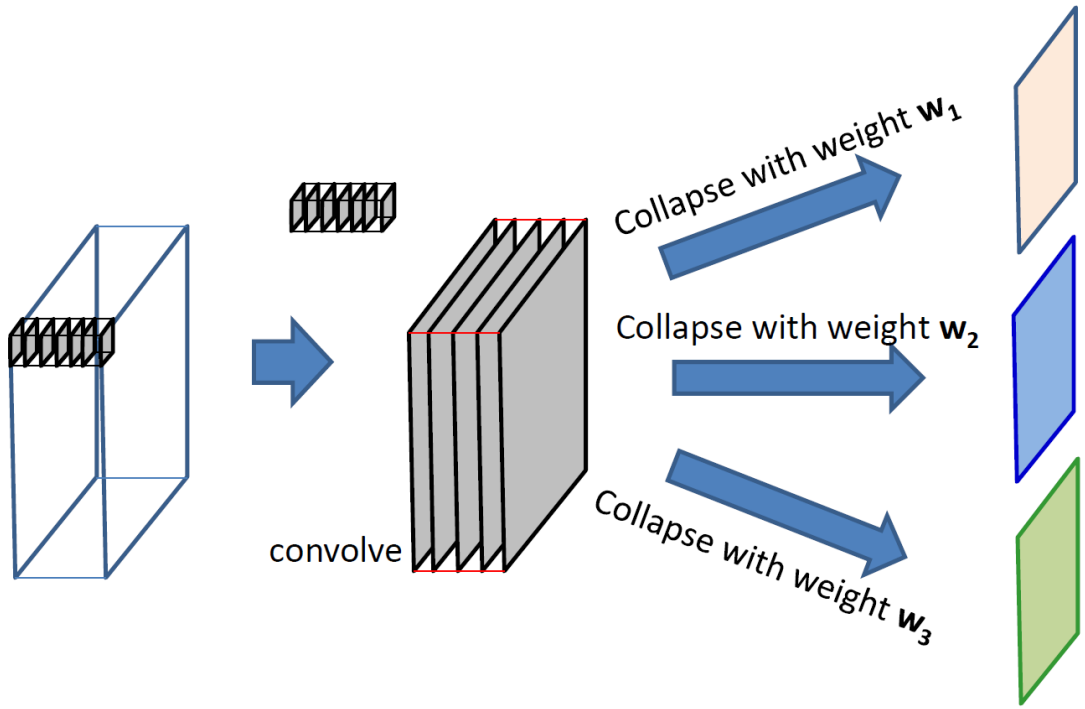
- Alternate view of conventional convolution:
- Each layer of each filter scans its corresponding map to produce a convolved map
- N input channels will require a filter with N layers
- The independent convolutions of each layer of the filter result in N convolved maps
- The N convolved maps are added together to produce the final output map (or channel) for that filter

Conventional convolutions



- This is done separately for each of the M filters producing M output maps (channels)

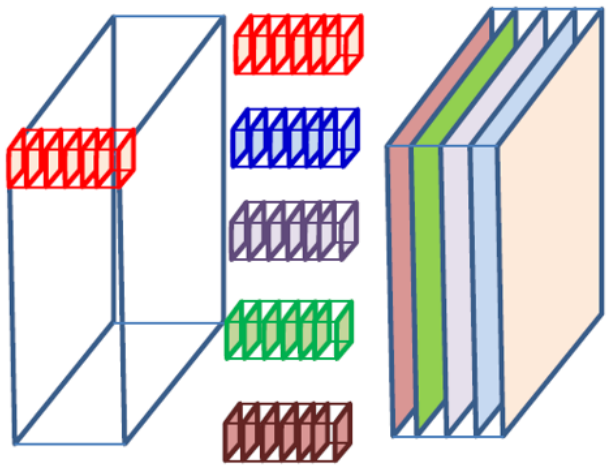
Depth-wise convolution 深度卷积



- In depth-wise convolution the convolution step is performed only once
- The simple summation is replaced by a weighted sum across channels
  - Different weights produce different output channels

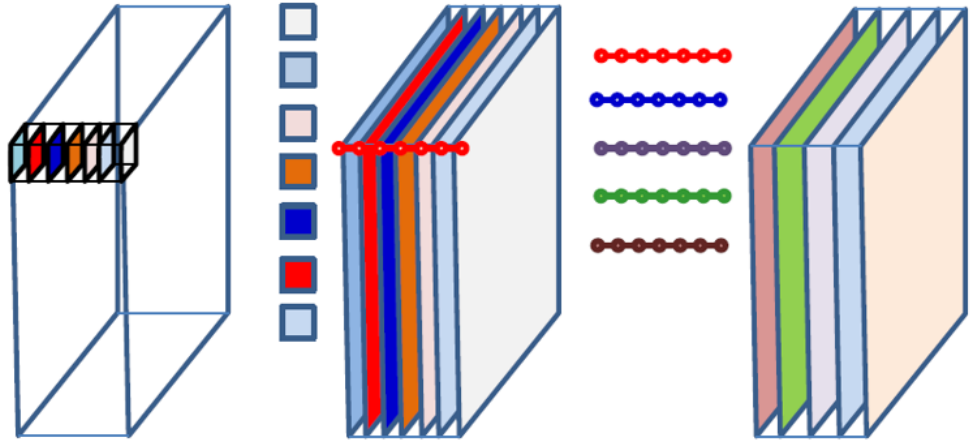
Conventional vs. depth-wise convolution

Conventional



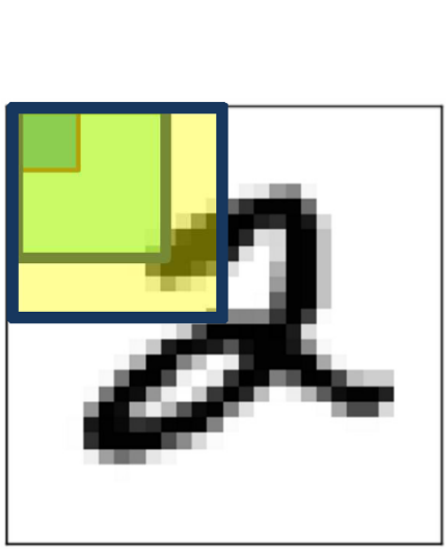
- M input channels, N output channels:
- N independent MxKxK **3D** filters, which span all M input channels
- Each filter produces one output channel
- Total NMK^2 parameters

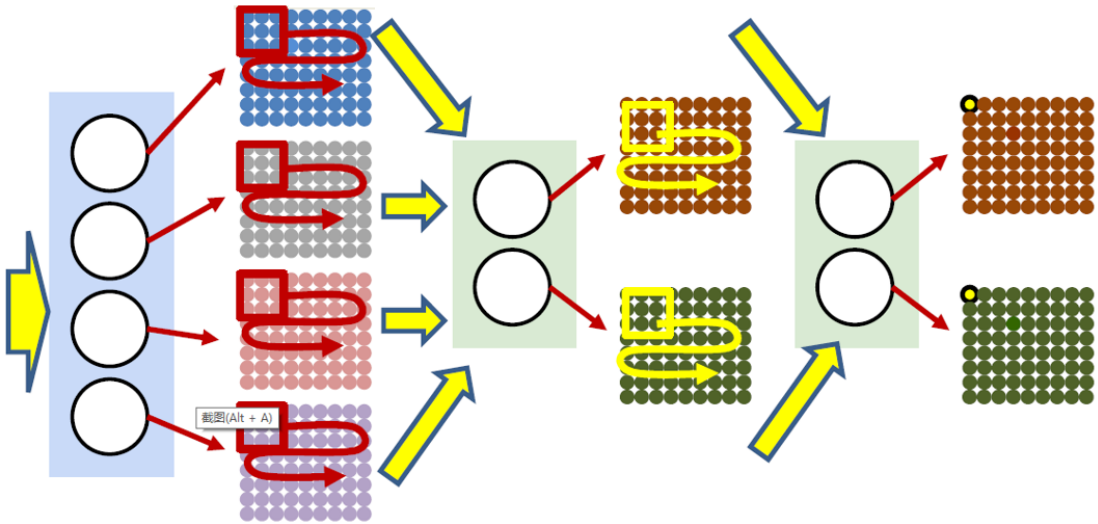
Depth-wise



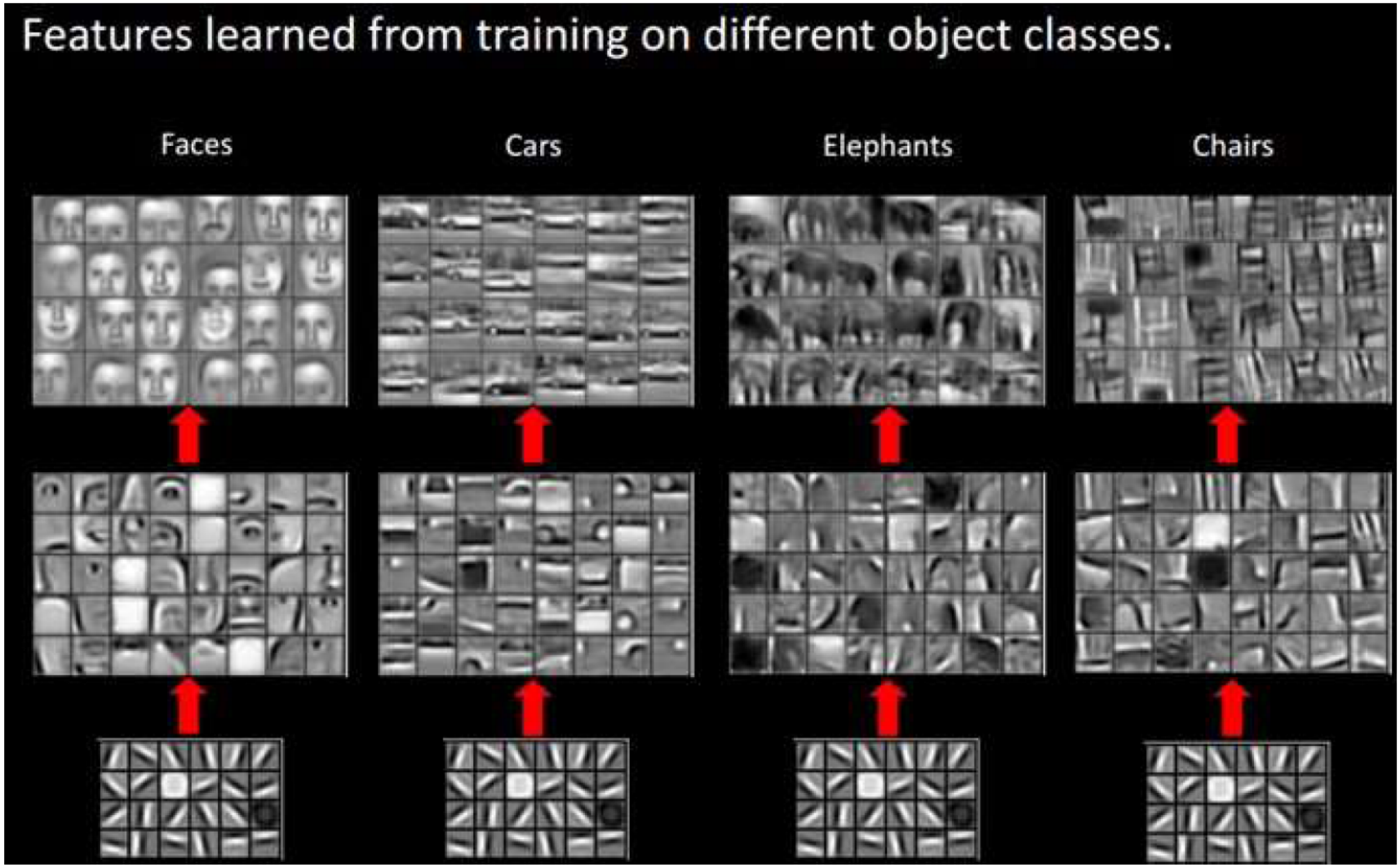
- M input channels, N output channels in 2 stages:
- Stage 1:
  - M independent KxK **2D** filters, one per input channel
  - Each filter applies to only one input channel
  - No. of output channels = no. of input channels
- Stage 2:
  - N Mx1x1 1D filters
  - Each applies to *one* 2D location across all M input channels
- Total $NM + MK^2$ parameters

What do the filters learn? Receptive fields 感受野





- The pattern in the input image that each neuron sees is its "Receptive Field" (感受野)
- The receptive field for a first layer neurons is simply its arrangement of weights
- For the higher level neurons, the actual receptive field is not immediately obvious

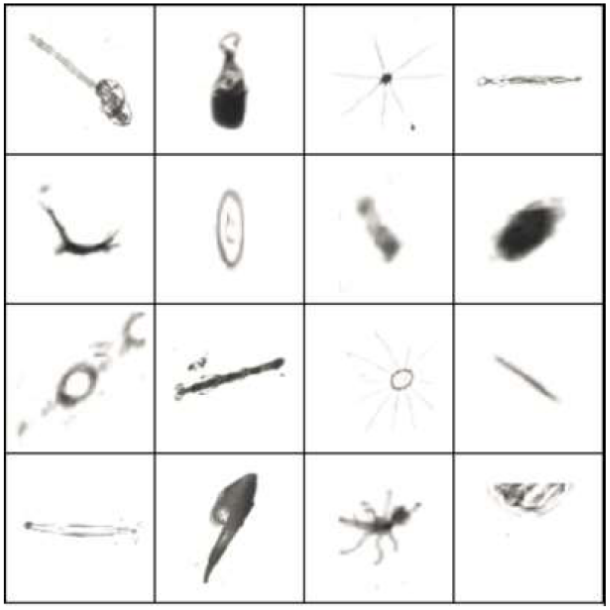

Features learned from training on different object classes.
Faces
Cars
Elephants
Chairs

Training Issues

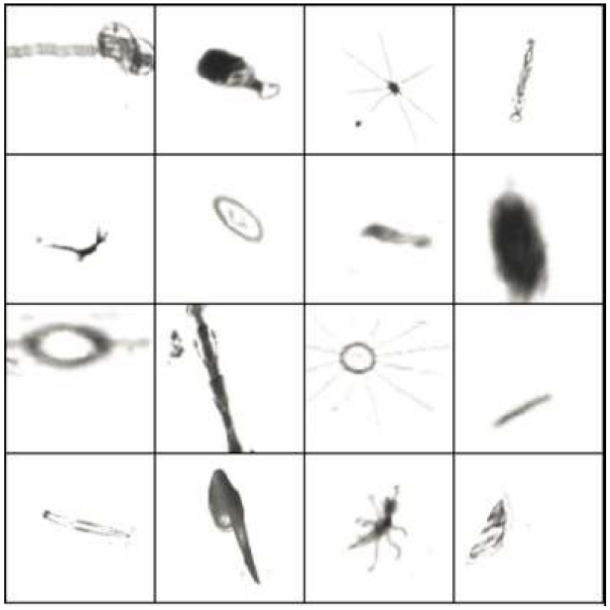
- Standard convergence issues
  - Solution: Adam or other momentum-style algorithms
  - Other tricks such as batch normalization
- The number of parameters can quickly become very large
- Insufficient training data to train well
  - Solution: Data augmentation

Data Augmentation

Original data



Augmented data

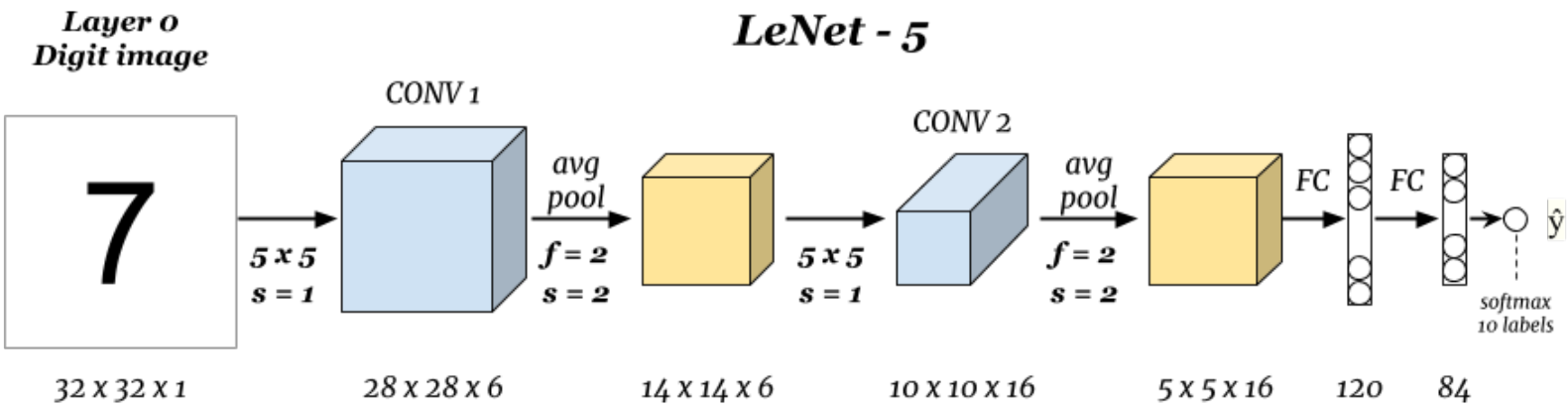


- rotation (旋转): uniformly chosen random angle
- translation (移位): translation between -10 and 10 pixels
- rescaling (缩放): scaling with factor between 1/1.6 and 1.6
- flipping (翻转): flip images horizontally or vertically
- shearing (错切): shearing with angle between -20° and 20°
- stretching(拉长) : stretching between 1/1.3 and 1.3

Convolutional neural nets

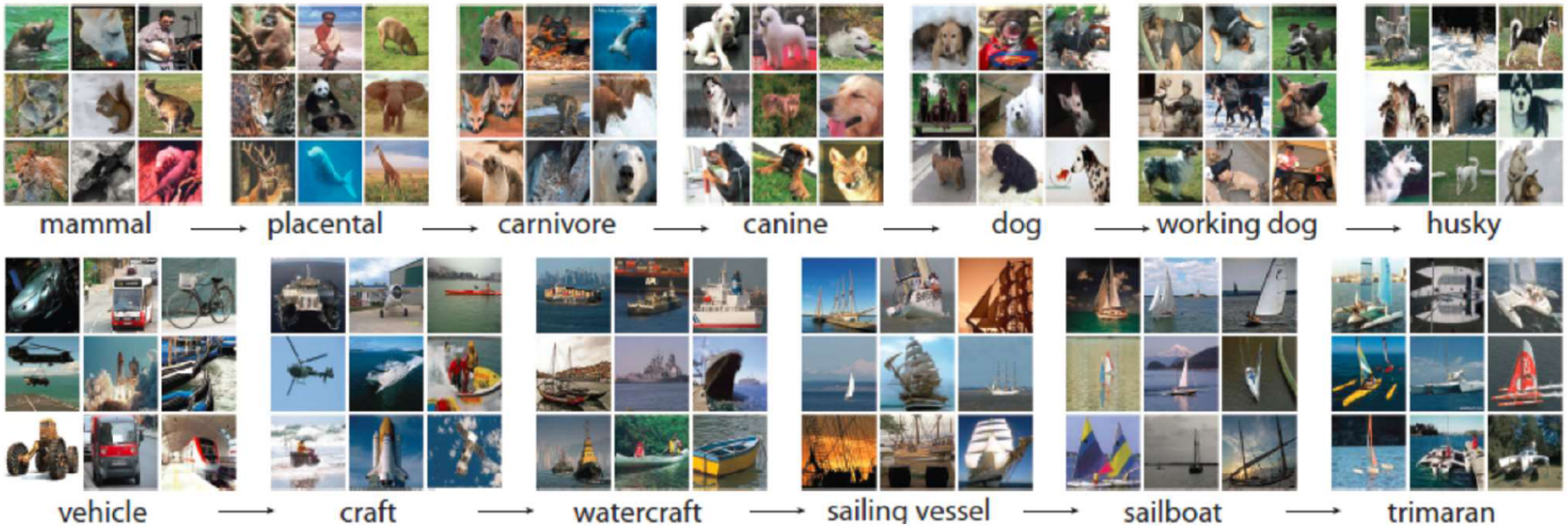
- One of the most frequently used nnet formalism today
- Used everywhere
  - Not just for image classification
  - Used in speech and audio processing
    - Convnets on spectrograms
  - Used in text processing

LeNet-5



- Digit recognition on MNIST (32x32 images)
  - Conv1: 6 5x5 filters in first conv layer (no zero pad), stride 1
    - Result: 6 28x28 maps
  - Pool1: 2x2 max pooling, stride 2
    - Result: 6 14x14 maps
  - Conv2: 16 5x5 filters in 2nd conv layer, stride 1, no zero pad
    - Result: 16 10x10 maps
  - Pool2: 2x2 max pooling with stride 2 for second conv layer
    - Result 16 5x5 maps (400 values in all)
  - FC: Final MLP: 3 layers
    - 120 neurons, 84 neurons, and finally 10 output neurons

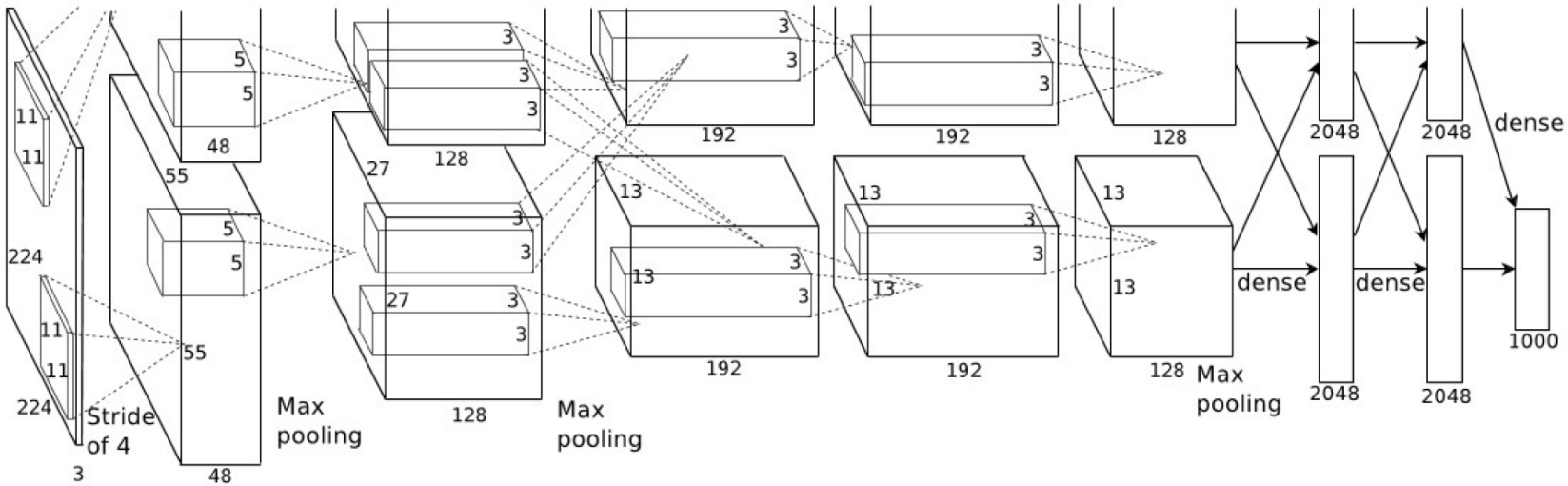
The ImageNet task



- Imagenet Large Scale Visual Recognition Challenge (ILSVRC)
- http://www.image-net.org/challenges/LSVRC/
- Actual dataset: Many million images, thousands of categories
- For the evaluations that follow:
  - 1.2 million pictures
  - 1000 categories

AlexNet

- NN configuration
  - NN contains 60 million parameters and 650,000 neurons,
  - 5 convolutional layers, some of which are followed by max-pooling layers
  - 3 fully-connected layers



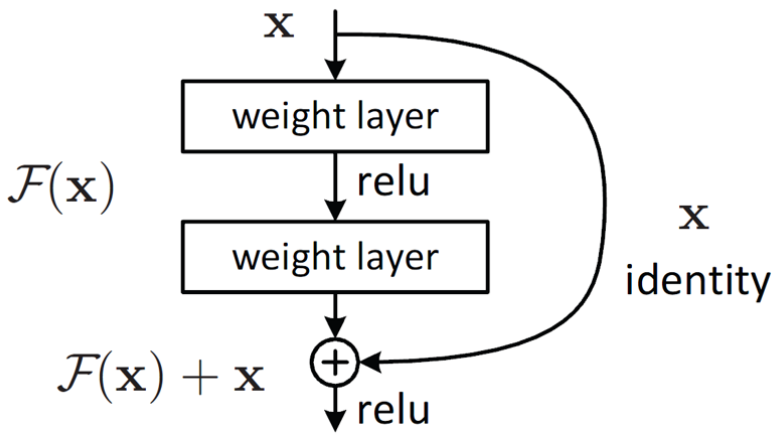
AlexNet

- Input: 227x227x3 images
- Conv1: 96 11x11 filters, stride 4, no zeropad
- Pool1: 3x3 filters, stride 2
- "Normalization" layer [Unnecessary]
- Conv2: 256 5x5 filters, stride 2, zero pad
- Pool2: 3x3, stride 2
- Normalization layer [Unnecessary]
- Conv3: 384 3x3, stride 1, zeropad
- Conv4: 384 3x3, stride 1, zeropad
- Conv5: 256 3x3, stride 1, zeropad
- Pool3: 3x3, stride 2
- FC: 3 layers,
  - 4096 neurons, 4096 neurons, 1000 output neurons

Learning magic in AlexNet

- Activations were RELU
  - Made a large difference in convergence
- "Dropout" : 0.5 (in FC layers only)
- Large amount of data augmentation
- SGD with mini batch size 128
- Momentum, with momentum factor 0.9
- L2 weight decay 5e-4
- Learning rate: 0.01, decreased by 10 every time validation accuracy plateaus (稳定)

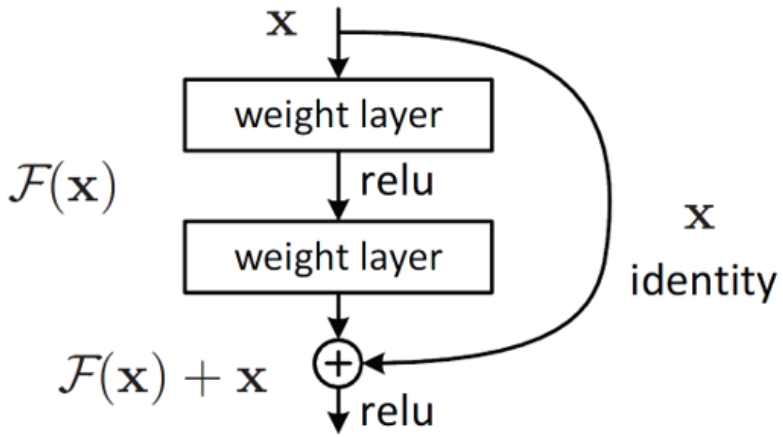
ResNet




```
def forward(self, x):
    residual = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)

    if self.shortcut:
        out += residual

    out = self.relu(out)

    return out
```

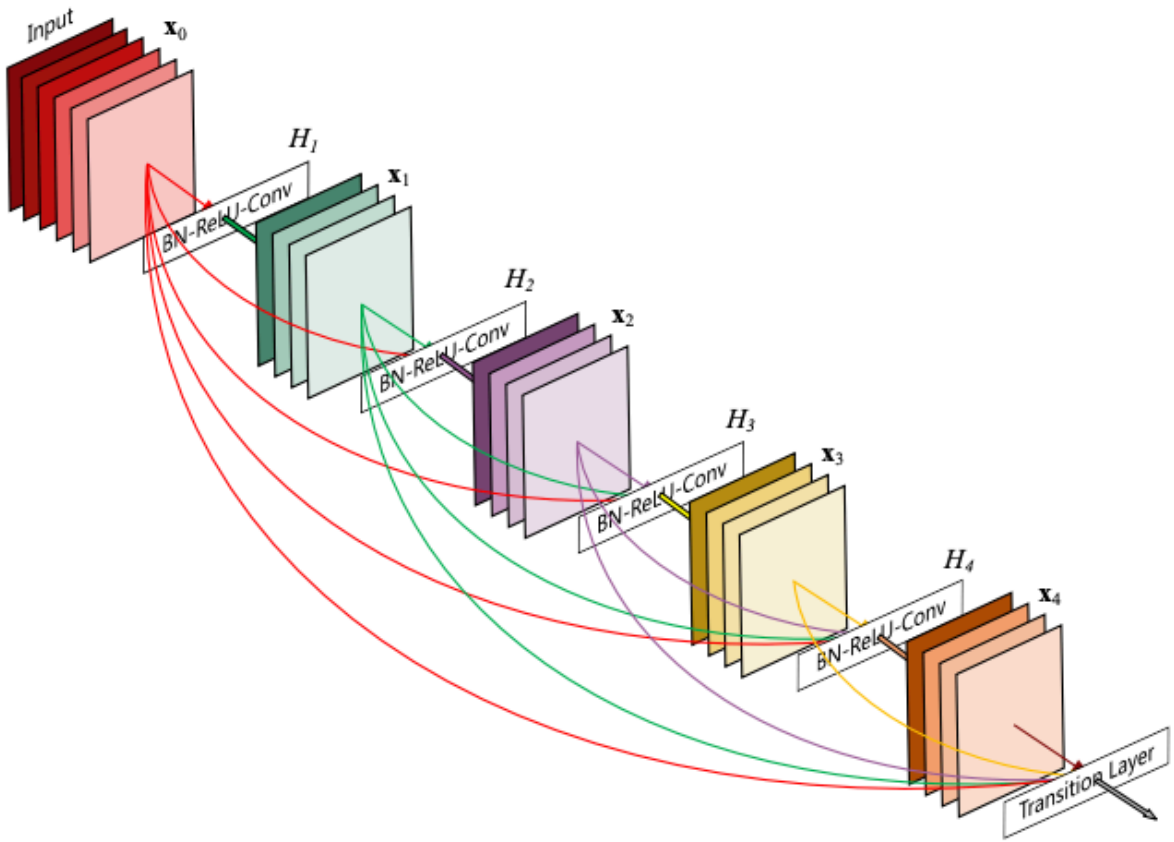
- Resnet: 2015
  - Over 150 layers, with "skip" connections.

ResNet details for the curious..



- Last layer before addition must have the same number of filters as the input to the module
- Batch normalization after each convolution
- SGD + momentum (0.9)
- Learning rate 0.1, divide by 10
- Mini batch 256
- Weight decay 1e-5

DenseNet



- All convolutional
- Each layer looks at the union of maps from all previous layers
  - Instead of just the maps from the immediately previous layer
- Achieved the state of the art (SOTA)

Thank you!