CMU 11-785 Introduction to Deep Learning, Fall 2020

Lecture 12

Backpropagation in CNNs

TAVE Research
DL001
Changdae Oh

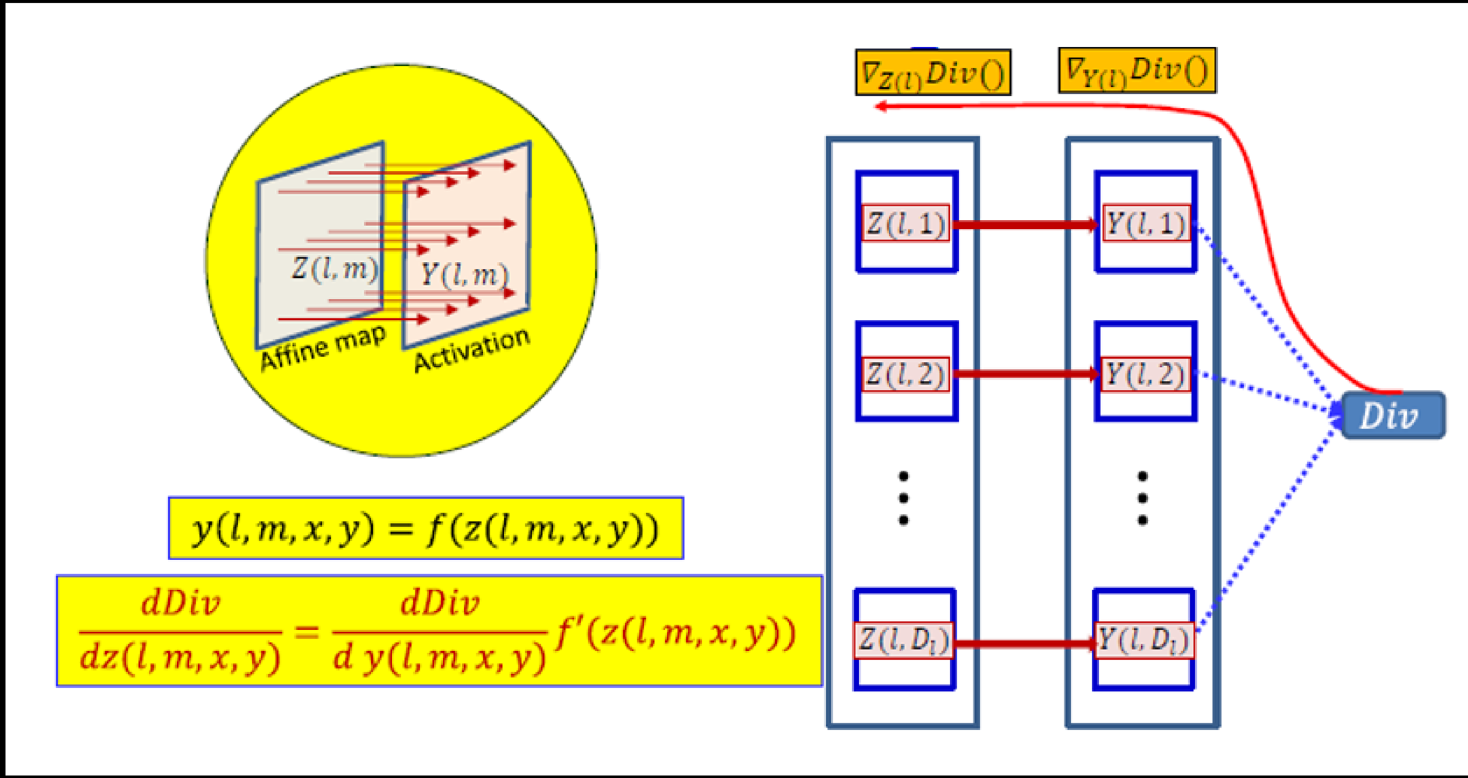
2021.03.14

Topics

- ❖ Backprop in Convolutional Layers
- ❖ Backprop in Pooling Layers
- ❖ Upsampling
- ❖ Transform Invariance
- ❖ Depth-wise Convolution

Backprop in Convolutional Layers

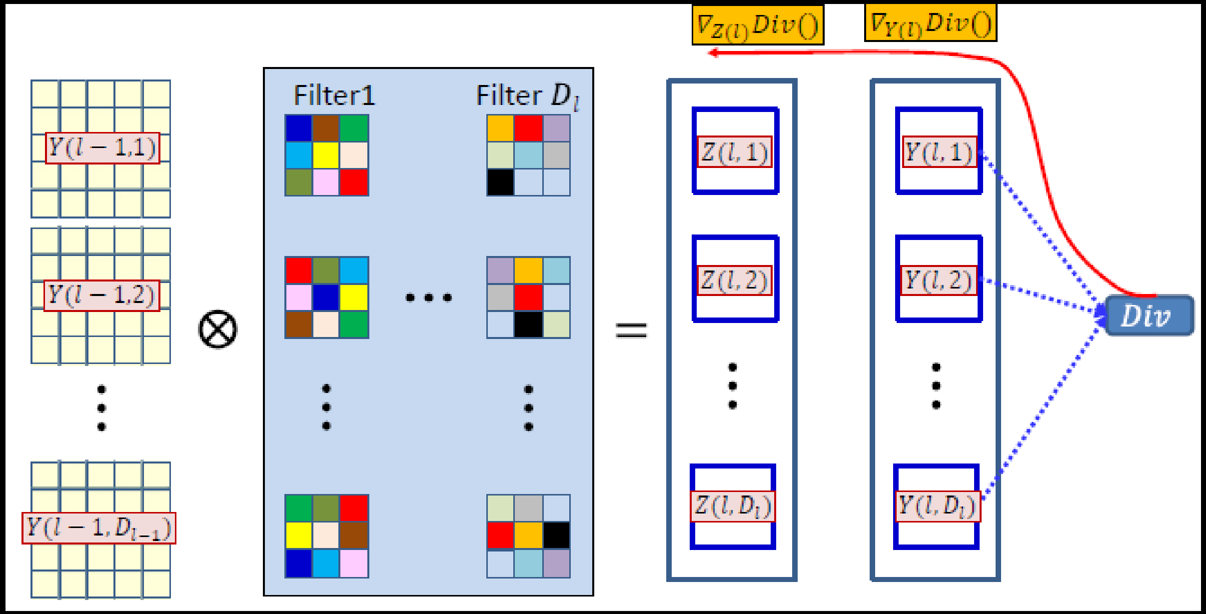
- Backprop through the activation

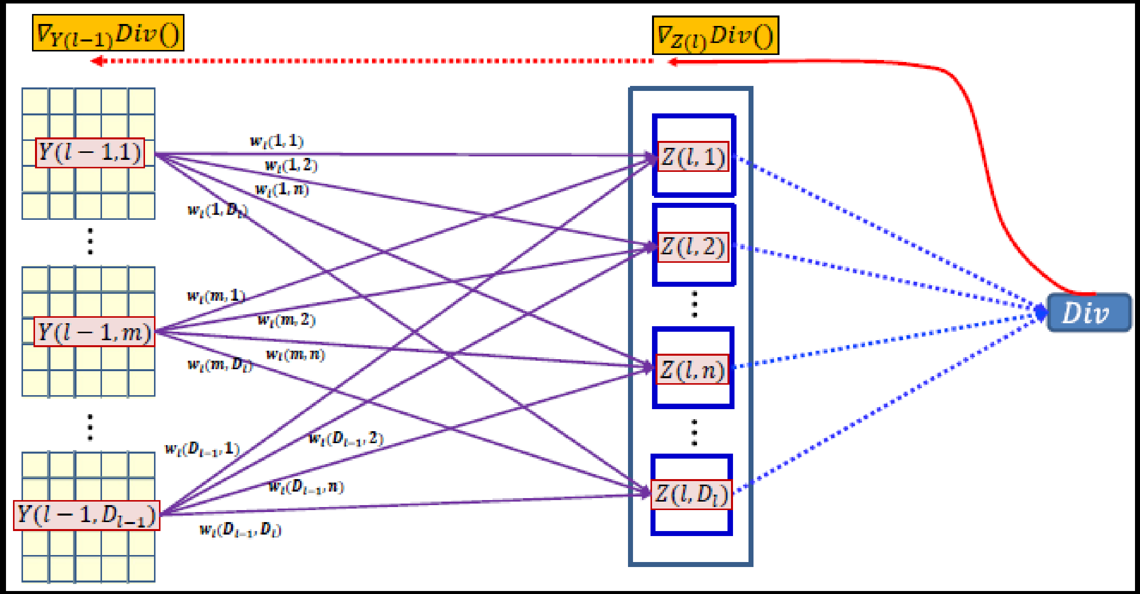


Forward : The activation maps are obtained by point-wise application of the function to affine map
Backward : Just multiply upstream gradient and derivative of activate function

Backprop in Convolutional Layers

- Backprop through the affine map





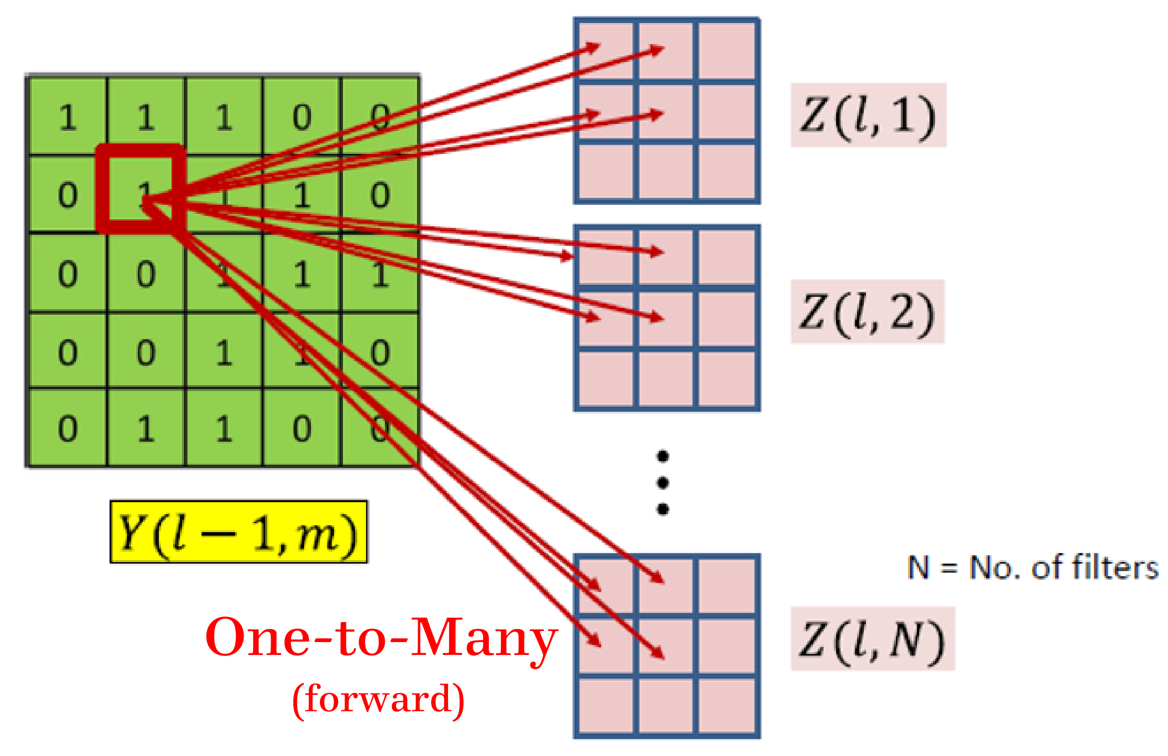
Object :

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

need to compute

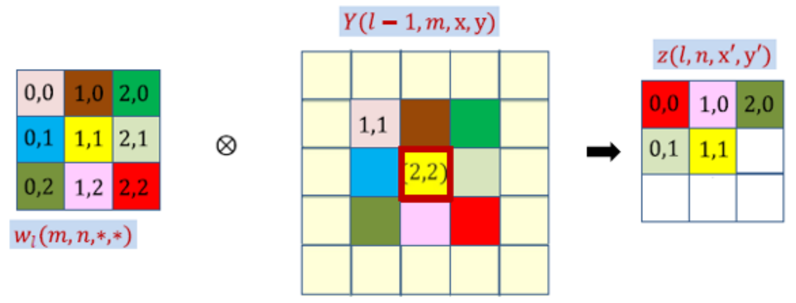
Backprop in Convolutional Layers

- Backprop through the affine map



✓ Each Y(x , y) affects several Z(x' , y') terms.

✓ How a single Y(x , y) influences Z(x' , y') ?



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

Contribution of a single position (x', y') :

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

Many-to-One
(backward)

Contribution of the entire nth affine map Z :

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

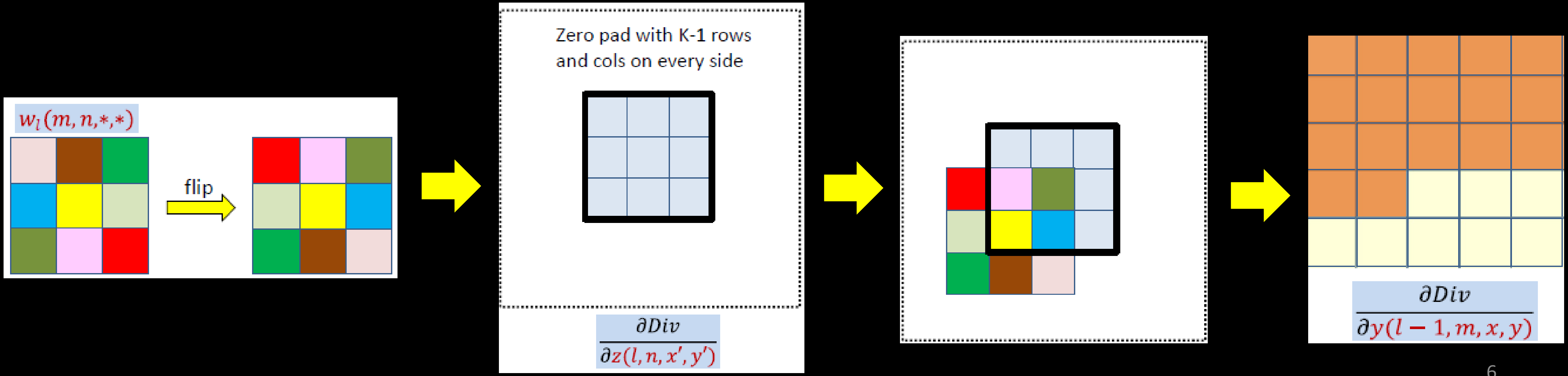
Backprop in Convolutional Layers

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

Summing over all Z maps

The derivative at Y(x, y) is the SUM of point-wise product of the flipped filter and the elements of the derivative at Z map

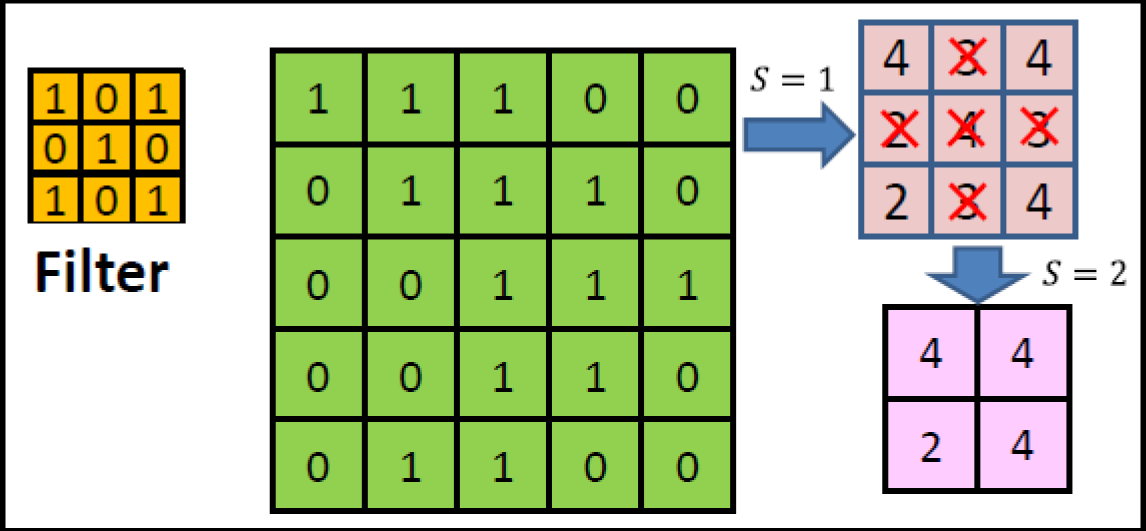
= Convolution !



Backprop in Convolutional Layers

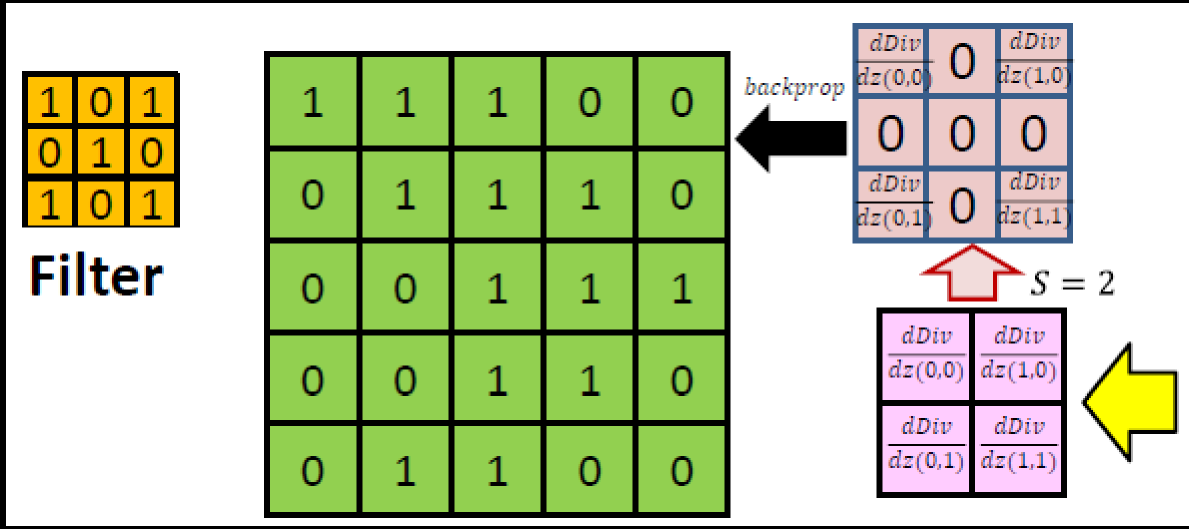
- zero padded map
  - The zero padding regions must be deleted before further backprop

- stride greater than 1



Forward

Downsampling affine map by S

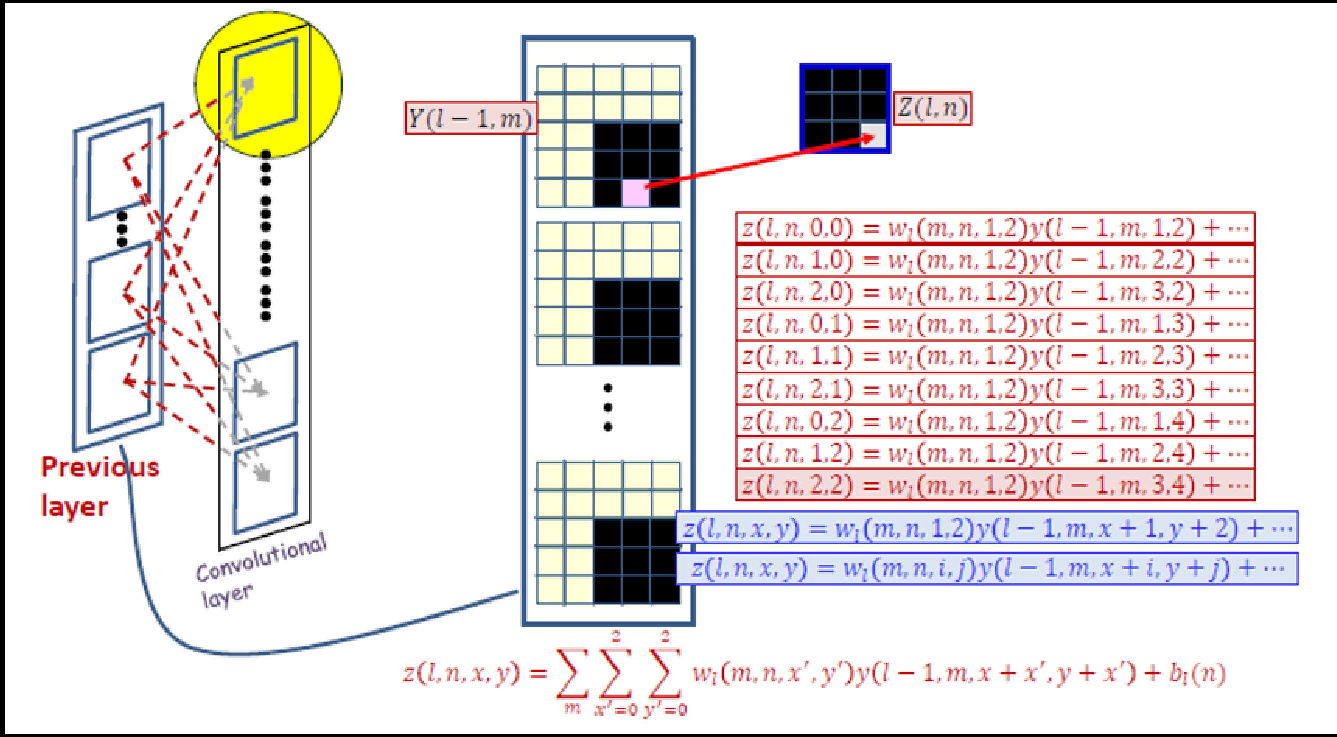


Backward

Upsampling derivative map by S

Backprop in Convolutional Layers

- The derivatives for the weights
  - Each W(x , y) affects several Z(x , y)



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

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

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

The derivative of the divergence w.r.t. W(i, j) must sum over all Z(x, y) terms it influences

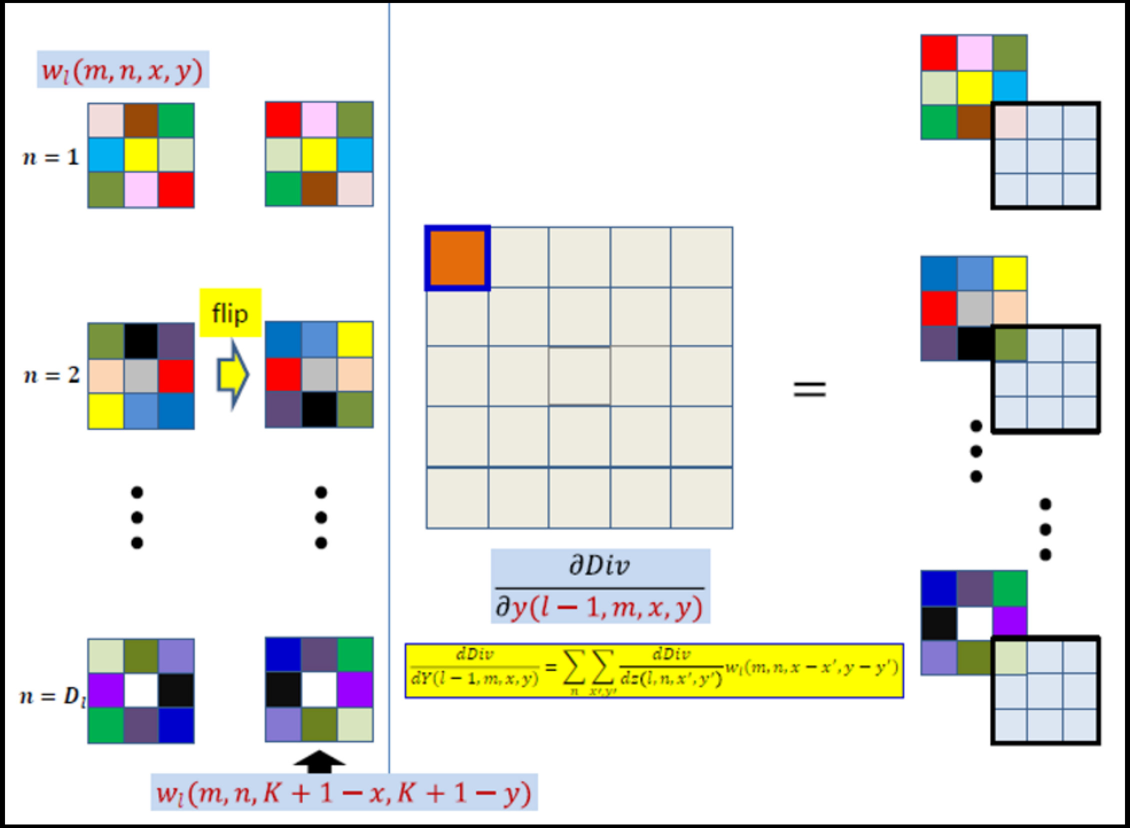
$$\frac{dDiv}{dw_l(m,n,i,j)} = \boxed{\sum_{x,y}} \frac{dDiv}{dz(l,n,x,y)} y(l-1,m,x+i,y+j)$$

=> This too is a **Convolution !**

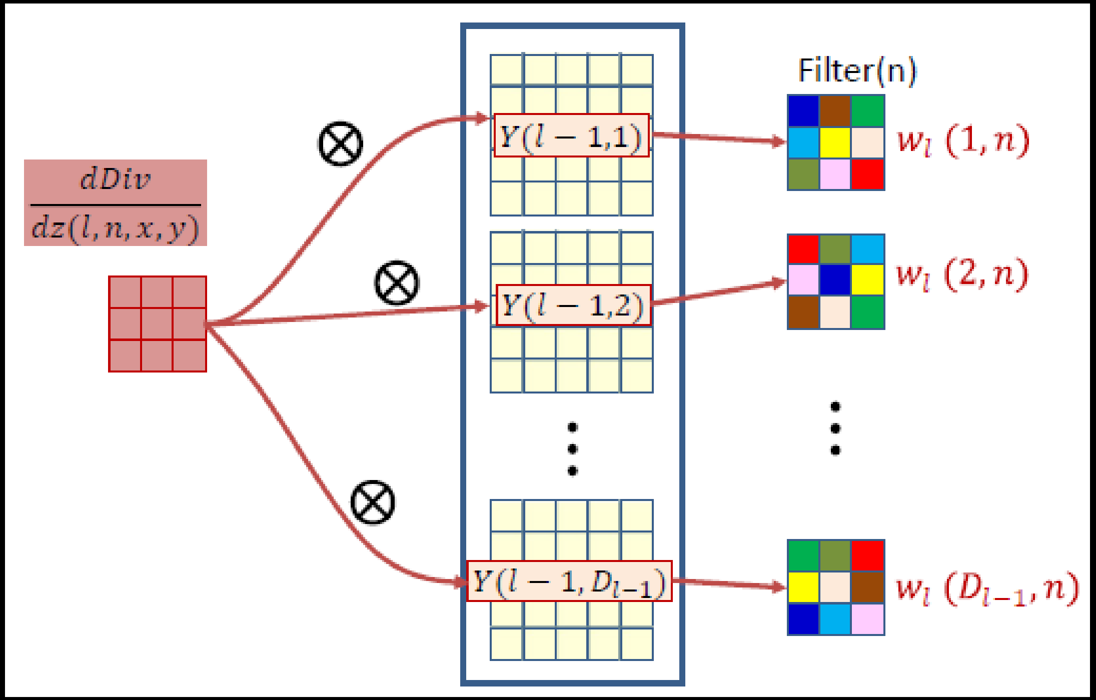
Backprop in Convolutional Layers

Convolutions on Backward pass

< Derivative w.r.t. previous Y map >

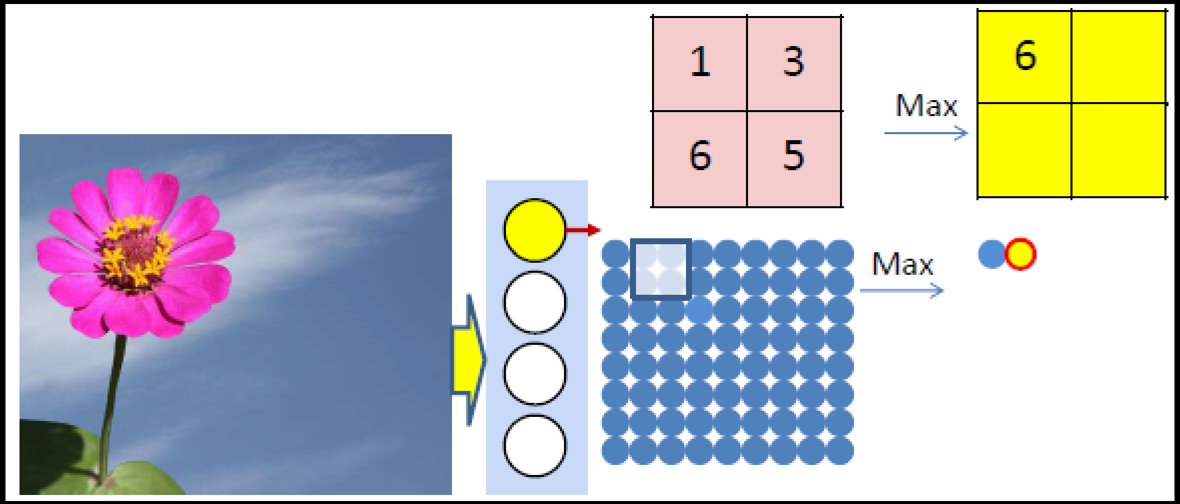


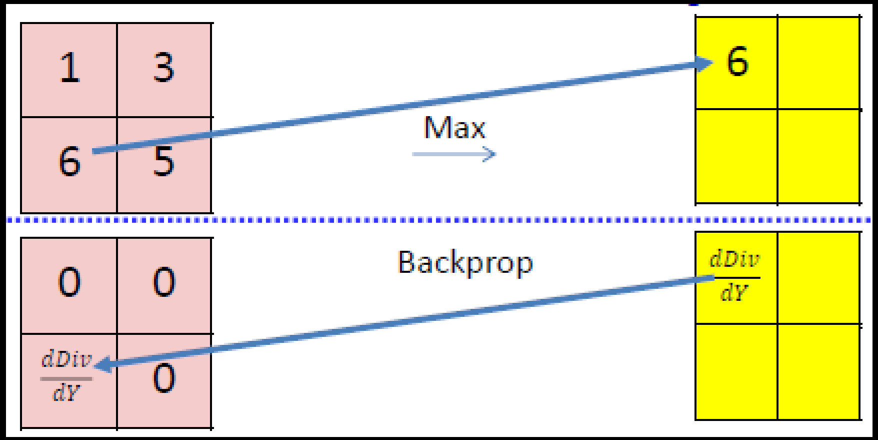
< Derivative w.r.t. weights (filters) >



Backprop in Pooling Layers

- Derivative of Max pooling





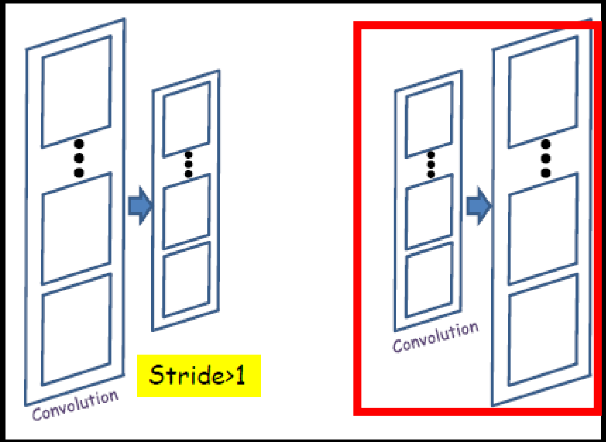
Pseudo code of backprop in max pooling layer

```
dy(:,:,:) = zeros(D_l x W_l x H_l)
for j = 1:D_l
    for x = 1:W_l_downsampled
        for y = 1:H_l_downsampled
            dy(l-1,j,pidx(l,j,x,y)) += dy(l,j,x,y)
```

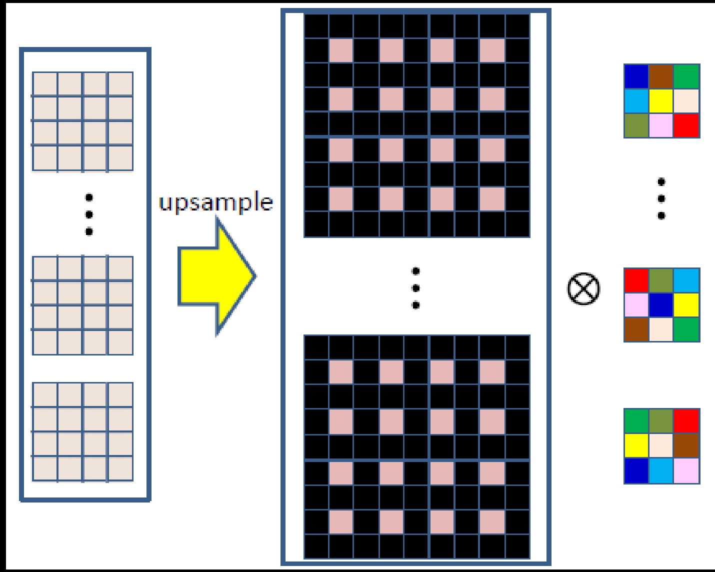
- Keep tracking of location of max
- '+=' for overlapping windows

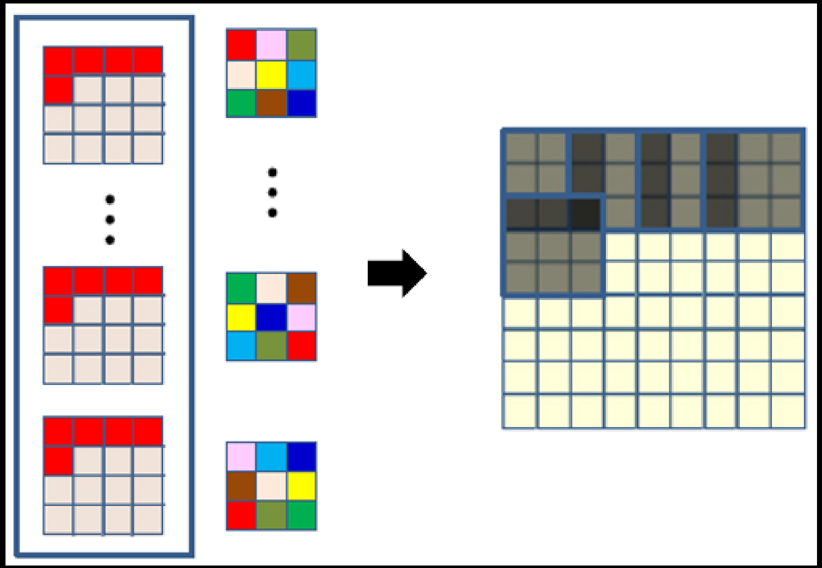
Upsampling

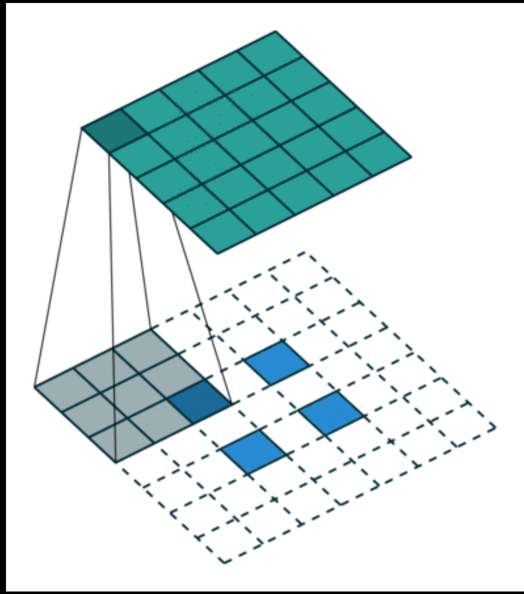
- for the restoration of Resolution



Transposed Convolution

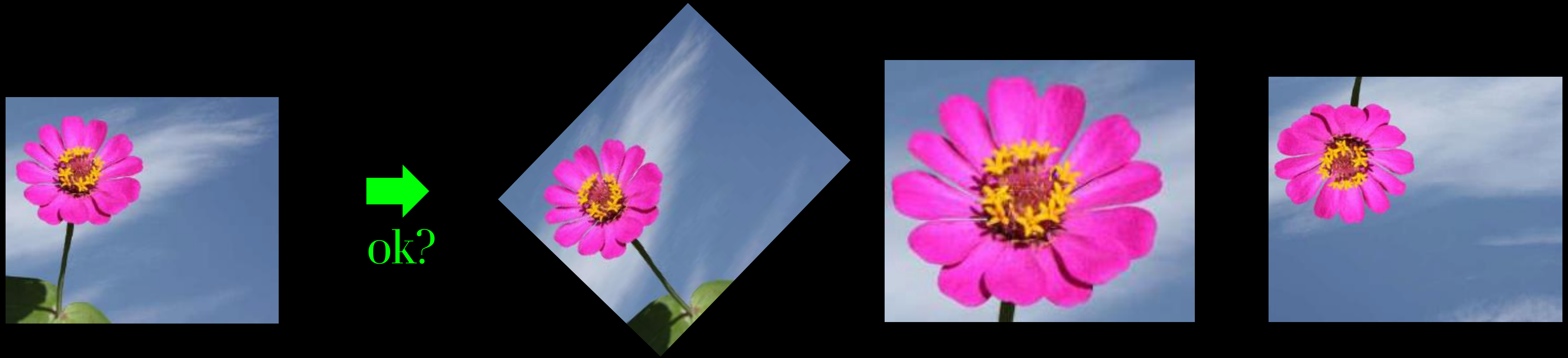




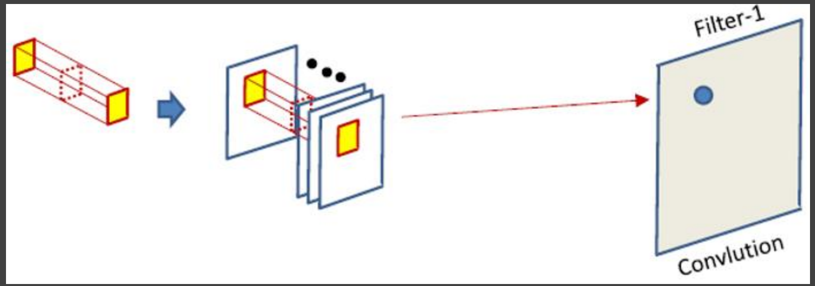


Transform Invariance

- CNNs are shift invariant
- What about rotation, scaling or reflection invariance?



Scan with filter – different perspective

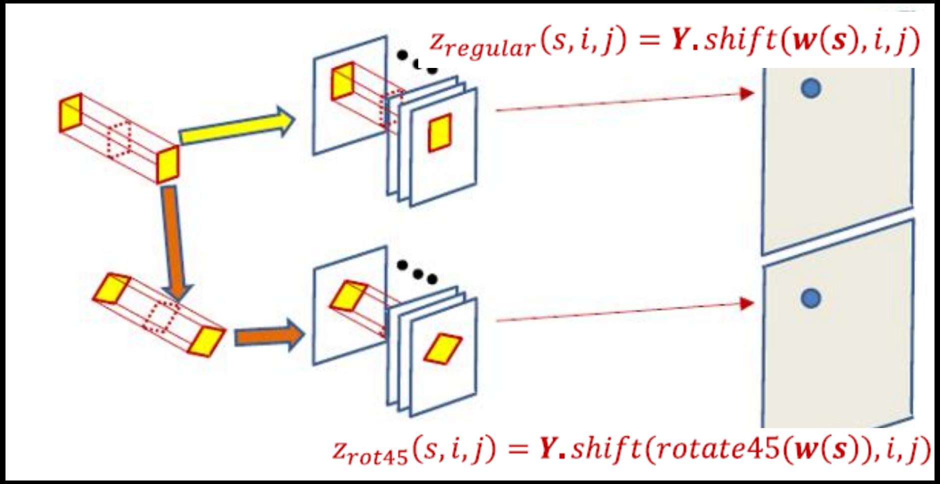


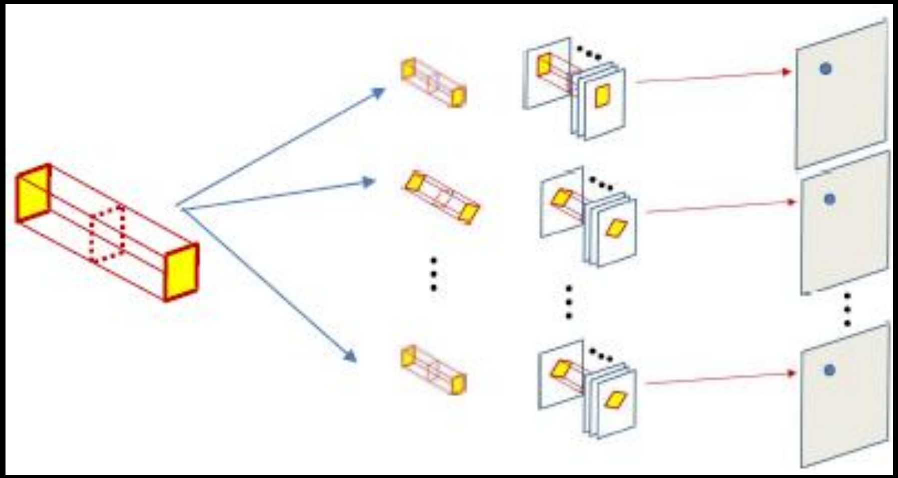
$$z(s,i,j) = \boldsymbol{Y}.shift(\boldsymbol{w}(\boldsymbol{s}), i, j)$$

- Output map of convolution is an inner product of some region of Y map and shifting weights

Transform Invariance

- Generalizing shift-invariance





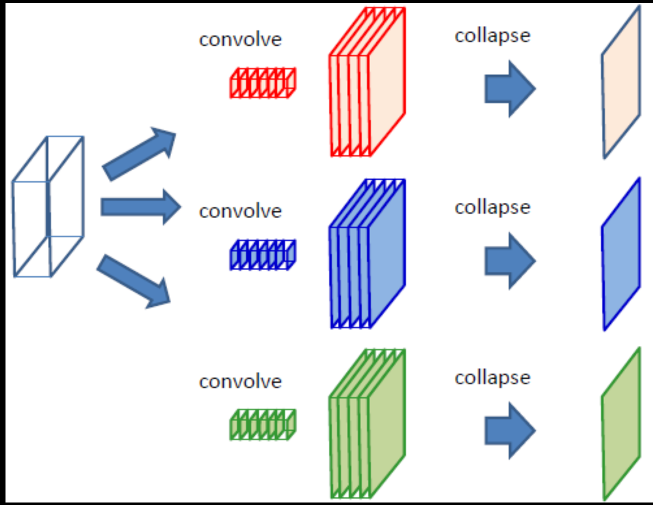
$$z_{T_t}(s,i,j) = \mathbf{Y}.shift(T_t(\mathbf{w}(\mathbf{s})),i,j)$$

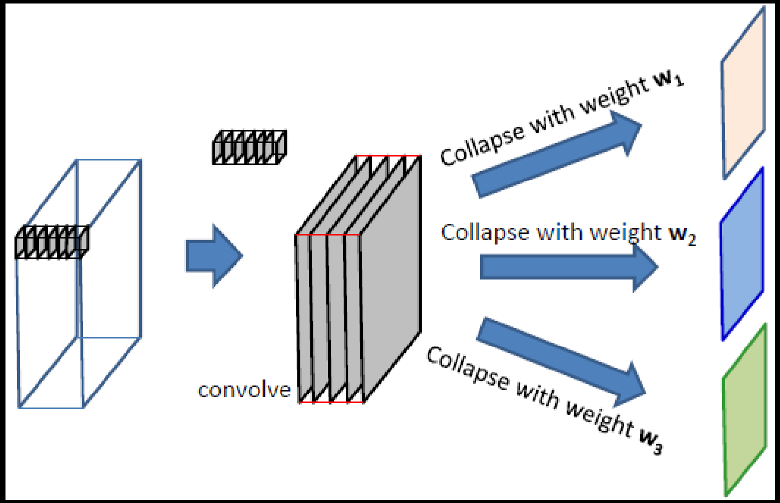
The moving filter is a composite function of Shifting & Transformation(like rotate, scale, reflect …)

✓ Each filter produces a set of transformed (and shifted) maps

✓ The network becomes invariant to all the transforms considered

Depth-wise Convolution

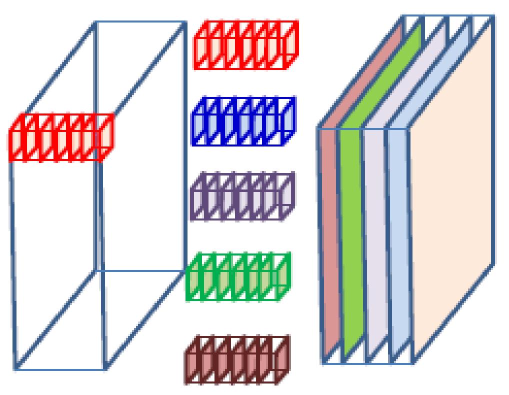




Conventional

tot params

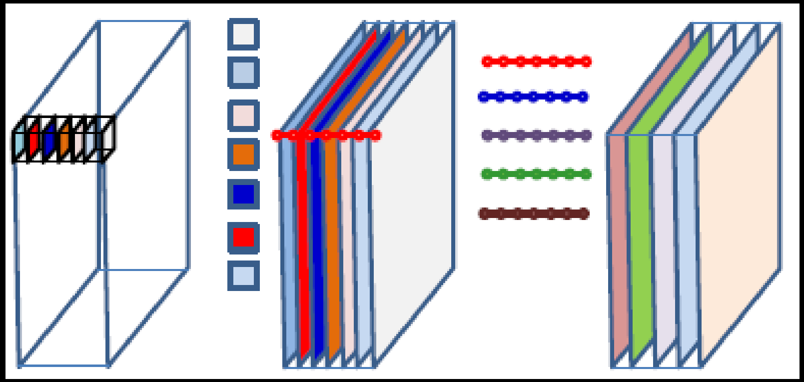
(I * O * K * K)



Depth-wise

tot params

(I * O + I * K * K)



Instead of multiple independent filters with independent parameters,

Use common layer-wise weights and combine the layers differently for each filter

✓ greater parameter efficiency