Applied Deep Learning

Chapter 4: Convolutional Neural Networks

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Fall 2025

Big Picture: *What to Train*

In CNNs we need to train *all learnable parameters*, *i.e.*,

- *weights and biases of output FNN*
- *weights in the filters of convolutional layers*
- *if we use advanced pooling with weights; then, we should find them as well*

To train, we get dataset $\mathbb{D} = \{(\mathbf{X}_b, \boldsymbol{v}_b) : b = 1, \dots, B\}$ *and train the CNN as*

$$\mathbf{w}^\star = \operatorname*{argmin}_{\mathbf{w}} \hat{R}(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{B} \sum_{b=1}^{B} \mathcal{L}(\boldsymbol{\theta}_b, \boldsymbol{v}_b) \qquad \text{(Training)}$$

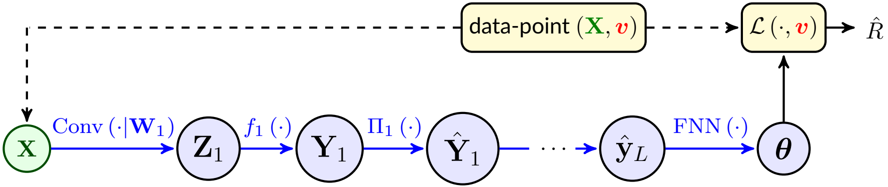
where $\boldsymbol{\theta}_b = \mathrm{CNN}(\mathbf{X}_b | \mathbf{w})$ *for the tensor-type data-point* $\mathbf{X}_b$ *with label* $\boldsymbol{v}_b$

↳ *We solve this optimization via a gradient-based method*

↳ *This means we should be able to pass forward and backward*

Computation Graph

Computation graph for single data-point $\mathbf{X}$ and its true label $\boldsymbol{v}$ is as follows



For simplicity, let's assume that we are doing basic SGD

↳ *We need to pass forward $\mathbf{X}$ over this graph to get loss*

↳ *We should then pass backward to compute gradient*

Let's try both directions

Forward Pass over CNN

The forward pass is what we learned in the last section

- We pass $\mathbf{X}$ through first convolution to get $\mathbf{Z}_1$
- We activate $\mathbf{Z}_1$ to get $\mathbf{Y}_1$
- We pool entries of $\mathbf{Y}_1$ and get $\hat{\mathbf{Y}}_1$

$\vdots$

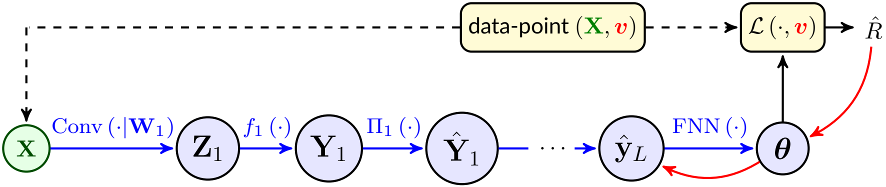
- We flatten and pass forward through the output FNN

Once we are over, we have

↳ *all convolution, activated and pooled values in convolutional layers*

↳ *all affine and activated values in the FNN*

Backward Pass



We now want to backpropagate

↳ *We first compute $\nabla_{\boldsymbol{\theta}} \hat{R}$*

↳ *We the backpropagate over the FNN till we get to its input*

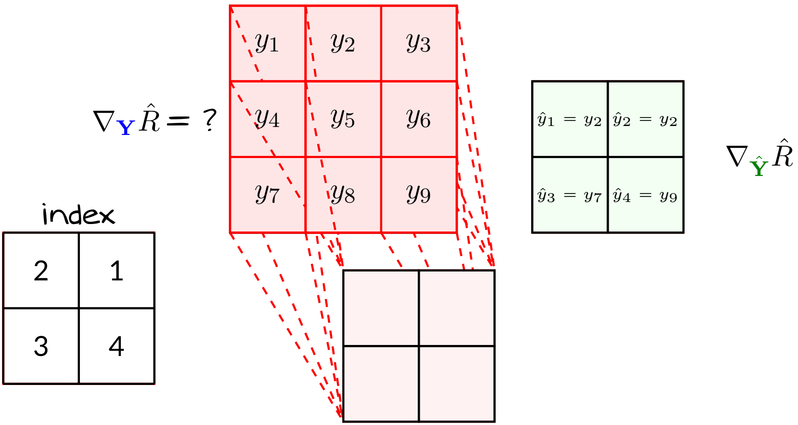
At this point, we sort $\nabla_{\hat{\mathbf{y}}_L} \hat{R}$ in tensor by reversing the flattening

we now have $\nabla_{\hat{\mathbf{Y}}_L} \hat{R}$

We only need to learn how to backpropagate through pooling and convolutional layers and then we can complete the backward pass!

Backpropagate through Pooling: *Max-Pooling*

Let's start with a simple example: *in the following pooling layer, we have partial derivatives with respect to pooled variables and want to compute the partial derivatives with respect to input of pooling layer*



Backpropagation: *Max-Pooling*

We can use chain rule

$$\frac{\partial \hat{R}}{\partial y_1} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_1} = 0$$

$$\frac{\partial \hat{R}}{\partial y_2} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_2} = \frac{\partial \hat{R}}{\partial \hat{y}_1} + \frac{\partial \hat{R}}{\partial \hat{y}_2}$$

$$\frac{\partial \hat{R}}{\partial y_3} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_3} = 0$$

$$\vdots$$

$$\frac{\partial \hat{R}}{\partial y_7} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_7} = \frac{\partial \hat{R}}{\partial \hat{y}_3}$$

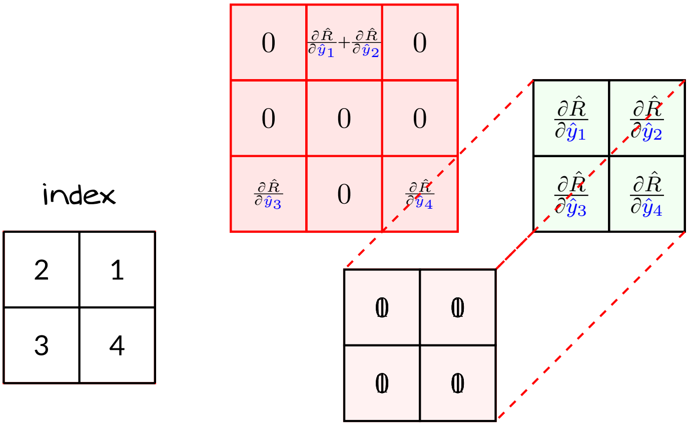
$$\frac{\partial \hat{R}}{\partial y_8} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_8} = 0$$

$$\frac{\partial \hat{R}}{\partial y_9} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_9} = \frac{\partial \hat{R}}{\partial \hat{y}_4}$$

Let's see its visualization

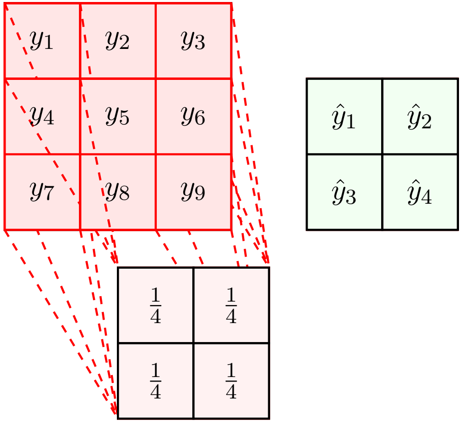
Backpropagation: *Max-Pooling*

Each derivative in green map gets multiplied with its filter and added to corresponding entries on blue map



Backpropagate through Pooling: *Mean-Pooling*

Lets now consider mean-pooling: *we have partial derivatives with respect to pooled variables and want to compute the partial derivatives with respect to input of pooling layer*



Backward Pass: *Mean-Pooling*

We again use chain rule

$$\frac{\partial \hat{R}}{\partial y_1} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_1} = \frac{1}{4} \frac{\partial \hat{R}}{\partial \hat{y}_1}$$

$$\frac{\partial \hat{R}}{\partial y_2} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_2} = \frac{1}{4} \frac{\partial \hat{R}}{\partial \hat{y}_1} + \frac{1}{4} \frac{\partial \hat{R}}{\partial \hat{y}_2}$$

$$\vdots$$

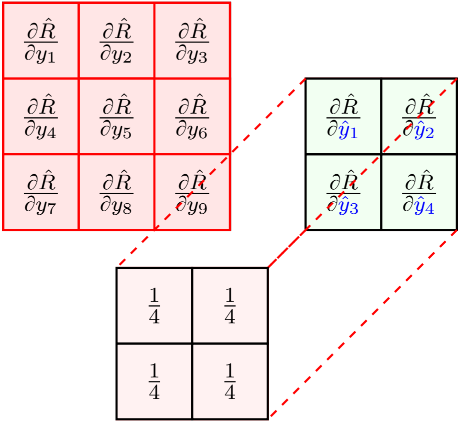
$$\frac{\partial \hat{R}}{\partial y_5} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_5} = \frac{1}{4} \frac{\partial \hat{R}}{\partial \hat{y}_1} + \frac{1}{4} \frac{\partial \hat{R}}{\partial \hat{y}_2} + \frac{1}{4} \frac{\partial \hat{R}}{\partial \hat{y}_3} + \frac{1}{4} \frac{\partial \hat{R}}{\partial \hat{y}_4}$$

$$\vdots$$

$$\frac{\partial \hat{R}}{\partial y_9} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_9} = \frac{1}{4} \frac{\partial \hat{R}}{\partial \hat{y}_4}$$

Backward Pass: *Mean-Pooling*

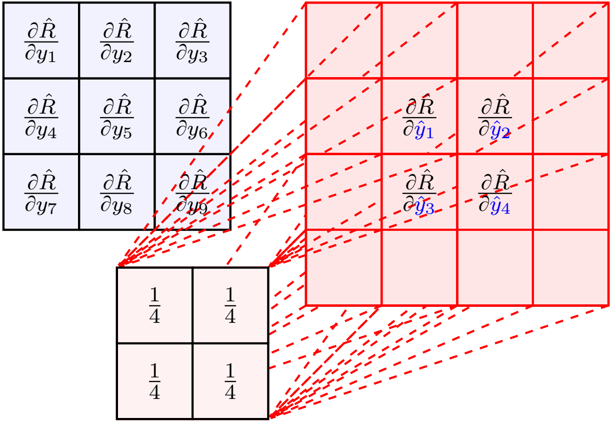
It has same visualization: the filter is however this time fixed



But we can visualize it even better!

Backward Pass: *Mean-Pooling*

In fact, this is simply a convolution



Backpropagation through Pooling: *Summary*

Depending on pooling function, backpropagation can be different

↳ *It's however usually described by a convolution*

Moral of Story

Backpropagation through pooling can be done by a convolution-type operation

Backpropagation through Convolution: *Activation*

Convolutional layer has two operations

↳ *linear convolution*

↳ *entry-wise activation*

The *entry-wise activation* is readily backpropagate: *say we have*

$$\mathbf{Y} = f(\mathbf{Z})$$

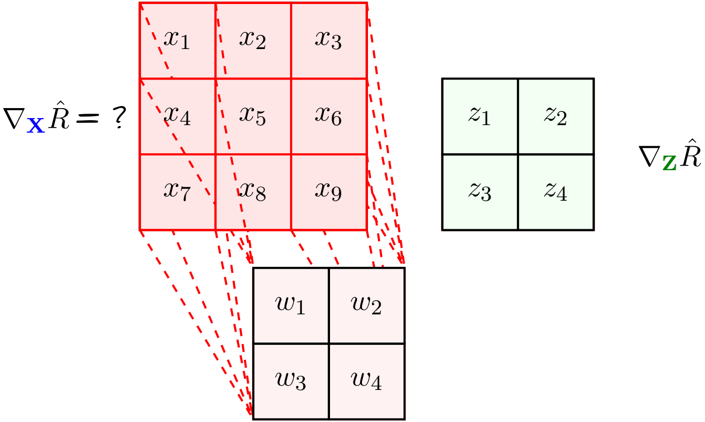
Then, we can backpropagate as in the fully-connected FNNs

$$\nabla_{\mathbf{Z}} \hat{R} = \nabla_{\mathbf{Y}} \hat{R} \odot \dot{f}(\mathbf{Z})$$

Now, let's look at linear convolution!

Backpropagation: *2D Convolution*

Let's find it out through a simple example: *in the following convolutional layer, we have partial derivatives with respect to convolved variables and want to compute the partial derivatives with respect to input of convolutional layer*



Backpropagation: *2D Convolution*

Let's start with chain rule

$$\frac{\partial \hat{R}}{\partial x_1} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_1} = w_1 \frac{\partial \hat{R}}{\partial z_1}$$

$$\frac{\partial \hat{R}}{\partial x_2} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_2} = w_2 \frac{\partial \hat{R}}{\partial z_1} + w_1 \frac{\partial \hat{R}}{\partial z_2}$$

$$\frac{\partial \hat{R}}{\partial x_3} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_3} = w_2 \frac{\partial \hat{R}}{\partial z_2}$$

$$\frac{\partial \hat{R}}{\partial x_4} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_4} = w_3 \frac{\partial \hat{R}}{\partial z_1} + w_1 \frac{\partial \hat{R}}{\partial z_3}$$

$$\frac{\partial \hat{R}}{\partial x_5} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_5} = w_4 \frac{\partial \hat{R}}{\partial z_1} + w_3 \frac{\partial \hat{R}}{\partial z_2} + w_2 \frac{\partial \hat{R}}{\partial z_3} + w_1 \frac{\partial \hat{R}}{\partial z_4}$$

Backpropagation: *2D Convolution*

We can keep on till finish with chain rule

$$\frac{\partial \hat{R}}{\partial x_6} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_6} = w_4 \frac{\partial \hat{R}}{\partial z_2} + w_2 \frac{\partial \hat{R}}{\partial z_4}$$

$$\frac{\partial \hat{R}}{\partial x_7} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_7} = w_3 \frac{\partial \hat{R}}{\partial z_3}$$

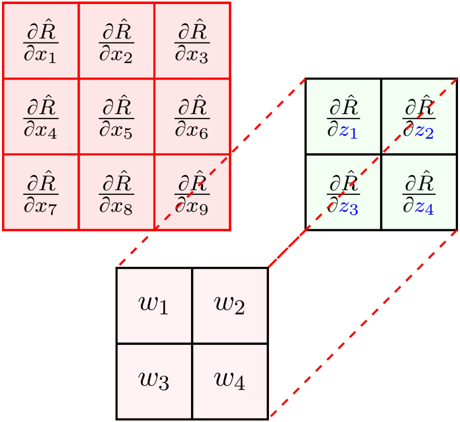
$$\frac{\partial \hat{R}}{\partial x_8} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_8} = w_4 \frac{\partial \hat{R}}{\partial z_3} + w_3 \frac{\partial \hat{R}}{\partial z_4}$$

$$\frac{\partial \hat{R}}{\partial x_9} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_9} = w_4 \frac{\partial \hat{R}}{\partial z_4}$$

Let's look at the visualization

Backpropagation: *2D Convolution*

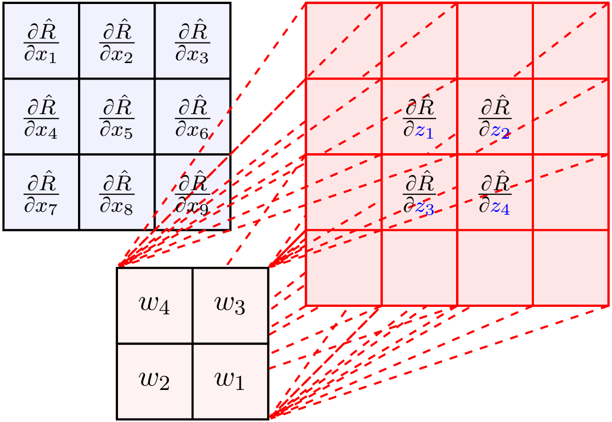
We can use the same visualization as in pooling



Or even a better visualization!

Backpropagation: *2D Convolution*

It's again simply a convolution



with filter being reversed up-to-down and right-to-left

Backpropagation: *2D Convolution*

To backpropagate a convolutional layer, we convolve the output gradient with the filter being reversed up-to-down and right-to-left. To match the dimension, we simply apply zero-padding

Backpropagation through 2D Convolution

Let $\mathbf{Z} = \mathrm{Conv}\left(\mathbf{X}|\mathbf{W}\right)$, *where* $\mathbf{X}$ *is the input map, i.e., a matrix,* $\mathbf{W}$ *is filter and* $\mathbf{Z}$ *is the output map. Assume that we know the gradient of loss* $\hat{R}$ *with respect to the output map, i.e.,* $\nabla_{\mathbf{Z}}\hat{R}$. *Then, the gradient of loss* $\hat{R}$ *with respect to the input map is*

$$\nabla_{\mathbf{X}}\hat{R} = \mathrm{Conv}\left(\nabla_{\mathbf{Z}}\hat{R}|\breve{\mathbf{W}}\right)$$

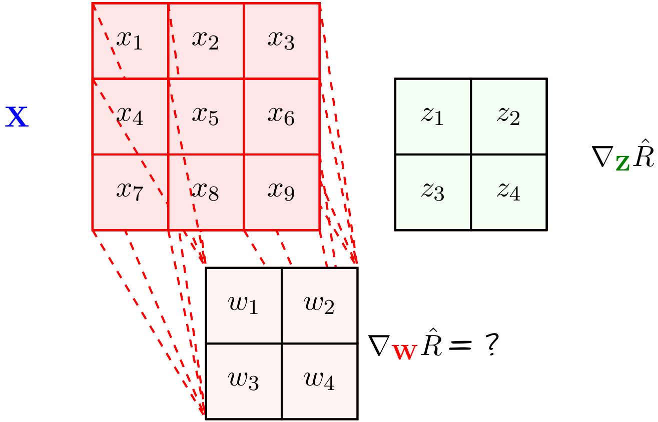
where $\breve{\mathbf{W}}$ *is the up-to-down and left-to-right reverse of filter* $\mathbf{W}$

Backpropagation: *2D Convolution*

- + *What if the dimensions do not match?*
- – The dimensions can only not match if
  1. *we do the forward convolution with a stride different from one*
     - ↳ *We said that this is simply resampling*
     - ↳ *We are going to discuss it later*
     - ↳ *For now assume that we do the convolution withstride one*
  2. *we did no zero-padding in the forward convolution*
     - ↳ *We simply do zero-padding in backward convolution to match the dimensions*
- + *What about the gradient with respect to filter itself? Don't we need it?*
- – Let's check it out

Backpropagation: *Convolution Filter*

We try it for our example: *we have partial derivatives with respect to convolved variables and want to compute the partial derivatives with respect to weights in the filter*



Backpropagation: *Convolution Filter*

As always, we use chain rule

$$\frac{\partial \hat{R}}{\partial w_1} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial w_1} = x_1 \frac{\partial \hat{R}}{\partial z_1} + x_2 \frac{\partial \hat{R}}{\partial z_2} + x_4 \frac{\partial \hat{R}}{\partial z_3} + x_5 \frac{\partial \hat{R}}{\partial z_4}$$

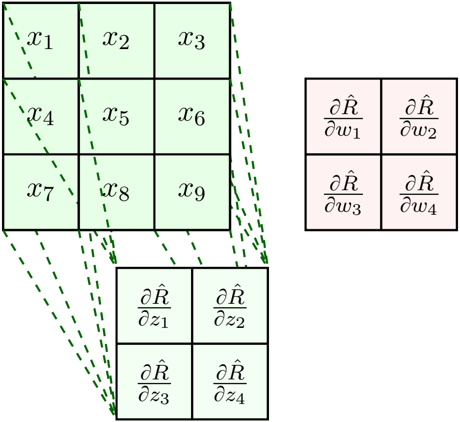
$$\frac{\partial \hat{R}}{\partial w_2} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial w_2} = x_2 \frac{\partial \hat{R}}{\partial z_1} + x_3 \frac{\partial \hat{R}}{\partial z_2} + x_5 \frac{\partial \hat{R}}{\partial z_3} + x_6 \frac{\partial \hat{R}}{\partial z_4}$$

$$\frac{\partial \hat{R}}{\partial w_3} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial w_3} = x_4 \frac{\partial \hat{R}}{\partial z_1} + x_5 \frac{\partial \hat{R}}{\partial z_2} + x_7 \frac{\partial \hat{R}}{\partial z_3} + x_8 \frac{\partial \hat{R}}{\partial z_4}$$

$$\frac{\partial \hat{R}}{\partial w_4} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial w_4} = x_5 \frac{\partial \hat{R}}{\partial z_1} + x_6 \frac{\partial \hat{R}}{\partial z_2} + x_8 \frac{\partial \hat{R}}{\partial z_3} + x_9 \frac{\partial \hat{R}}{\partial z_4}$$

Backpropagation: *Convolution Filter*

It's another convolution with $\nabla_{\mathbf{z}} \hat{R}$ being the filter!



Backpropagation: *Convolution Filter*

Once we backpropagate to the output of a convolutional layer, then we can find the gradient with respect to convolution filter by convolving output gradient with the input map

Gradient w.r.t. Filters

Let $\mathbf{Z} = \mathrm{Conv}\left(\mathbf{X}|\mathbf{W}\right)$, where $\mathbf{X}$ is the input map, i.e., a matrix, $\mathbf{W}$ is filter and $\mathbf{Z}$ is the output map. Assume that we know gradient of loss $\hat{R}$ with respect to the output map, i.e., $\nabla_{\mathbf{Z}}\hat{R}$. Then, gradient of loss $\hat{R}$ with respect to the filter is

$$\nabla_{\mathbf{W}}\hat{R} = \mathrm{Conv}\left(\mathbf{X}|\nabla_{\mathbf{Z}}\hat{R}\right)$$

+ *But we checked only 2D case! What about multi-channel case?!*

– Well, we can simply do it by chain rule

Backpropagation: *Multi-Channel Convolution*

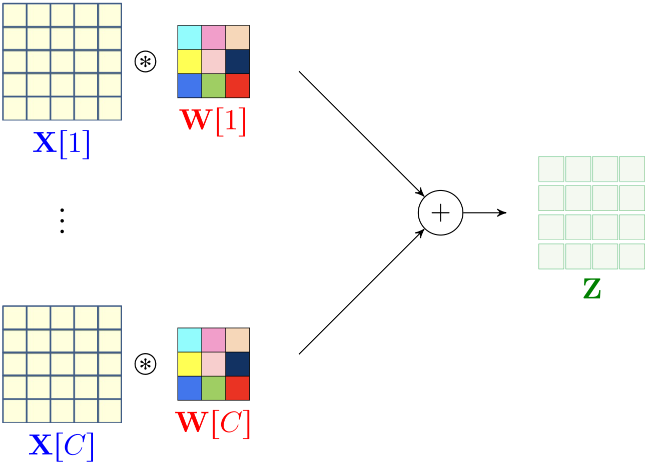
Lets start with a simple case: *we have a C-channel input*, *i.e., a tensor input*, $\mathbf{X}$ *and we compute a single feature map, i.e., a matrix,* $\mathbf{Z}$

↳ *Filter* $\mathbf{W}$ *has also* C *channels*

Let's denote channel c *of input and filter by* $\mathbf{X}[c]$ *and* $\mathbf{W}[c]$*; then, we can write*

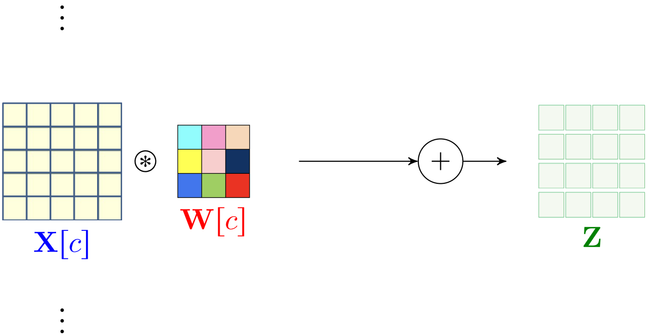
$$\mathbf{Z} = \sum_{c=1}^{C} \mathrm{Conv}\left(\mathbf{X}[c] | \mathbf{W}[c]\right)$$

Backpropagation: *Multi-Channel Convolution*



Backpropagation: *Multi-Channel Convolution*

Channel c of input is connected to the output map by a 2D convolution



So, we could say

$$\nabla_{\mathbf{X}[c]} \hat{R} = \mathrm{Conv}\left(\nabla_{\mathbf{Z}} \hat{R} | \breve{\mathbf{W}}[c] \right)$$

Backpropagation: *Multi-Channel Convolution*

To backpropagate a convolutional layer with tensor input and matrix output, we convolve the output gradient with the filter of each channel being reversed up-to-down and right-to-left

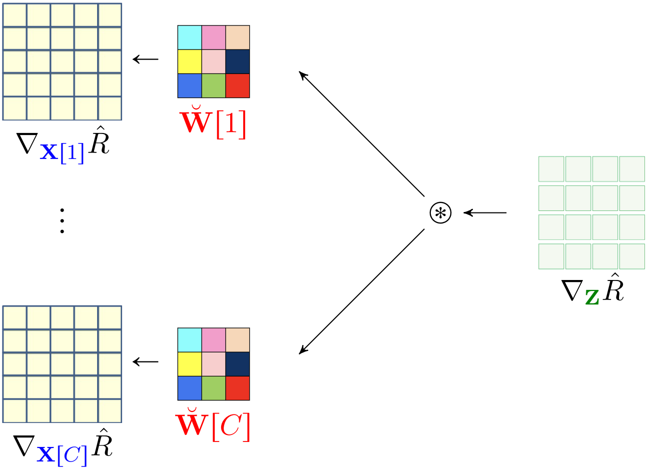
Backpropagation through 2D Convolution

Let $\mathbf{Z} = \mathrm{Conv}\left(\mathbf{X}|\mathbf{W}\right)$, *where* $\mathbf{X}$ *is a* C*-channel input tensor,* $\mathbf{W}$ *is* C*-channel filter and* $\mathbf{Z}$ *is a single-channel output map, i.e., a matrix. Assume that we know the gradient of loss* $\hat{R}$ *with respect to the output map, i.e.,* $\nabla_{\mathbf{Z}}\hat{R}$*. Then, the gradient of loss* $\hat{R}$ *with respect to input tensor is*

$$\nabla_{\mathbf{X}}\hat{R} = \left[\mathrm{Conv}\left(\nabla_{\mathbf{Z}}\hat{R}|\breve{\mathbf{W}}[1]\right), \ldots, \mathrm{Conv}\left(\nabla_{\mathbf{Z}}\hat{R}|\breve{\mathbf{W}}[C]\right)\right]$$

Note that $\nabla_{\mathbf{X}}\hat{R}$ *is also a* C*-channel tensor*

Backpropagation: *Multi-Channel Convolution*



Backpropagation: *Multi-channel Convolution*

We now go for the general case: *we have a C-channel input tensor $\mathbf{X}$ and we compute K-channel feature tensor $\mathbf{Z}$*

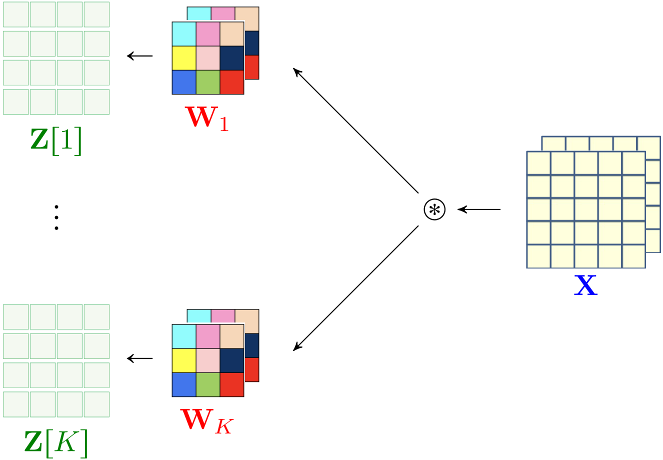
↳ *We have K filters $\mathbf{W}_1, \ldots, \mathbf{W}_K$*

↳ *Each filter has C channels*

Let's denote channel k of the output by $\mathbf{Z}$; then, we can write

$$\mathbf{Z}[k] = \mathrm{Conv}\left(\mathbf{X} | \mathbf{W}_k\right)$$

Backpropagation: *Multi-Channel Convolution*



Backpropagation: *Multi-channel Convolution*

We write the chain rule with a bit cheating: *let's denote the derivative of an object A with respect to object B as $\nabla_B A$*

- *if A and B are both scalars then its simple derivative*
- *if A is a scalar and B is a vector it's gradient*
- *if A and B are both vectors then its Jacobian*
- $\ldots$

When $\mathbf{Z} = f(\mathbf{X})$ and $\hat{R} = g(\mathbf{Z})$: chain rule says

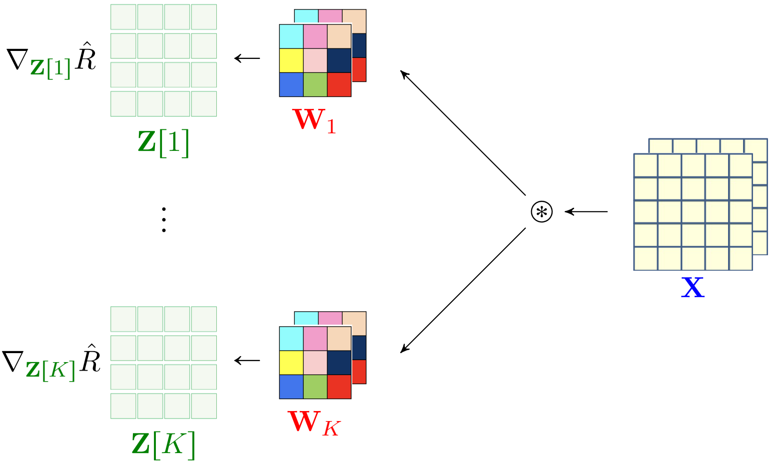
$$\nabla_{\mathbf{X}} \hat{R} = \nabla_{\mathbf{Z}} \hat{R} \circ \nabla_{\mathbf{X}} \mathbf{Z}$$

where $\circ$ is a kind of product

We now write the chain rule with this simplified notation

Backpropagation: *Multi-Channel Convolution*

We do know $\nabla_{\mathbf{Z}} \hat{R} = [\nabla_{\mathbf{Z}[1]} \hat{R}, \ldots, \nabla_{\mathbf{Z}[K]} \hat{R}]$



Backpropagation: *Multi-Channel Convolution*

The output tensor can be seen as K functions of the input tensor

$$\mathbf{Z}[1] = f_1(\mathbf{X}) \qquad \dots \qquad \mathbf{Z}[K] = f_K(\mathbf{X})$$

So, the chain rule can be written as

$$\begin{aligned}
\nabla_{\mathbf{X}} \hat{R} &= \sum_{k=1}^{K} \underbrace{\nabla_{\mathbf{Z}[k]} \hat{R} \circ \nabla_{\mathbf{X}} \mathbf{Z}[k]}_{\text{what we calculated for single output map}} \\
&= \sum_{k=1}^{K} \left[\mathrm{Conv}\left(\nabla_{\mathbf{Z}[k]} \hat{R} | \breve{\mathbf{W}}_k[1] \right), \dots, \mathrm{Conv}\left(\nabla_{\mathbf{Z}[k]} \hat{R} | \breve{\mathbf{W}}_k[C] \right) \right] \\
&= \left[\sum_{k=1}^{K} \mathrm{Conv}\left(\nabla_{\mathbf{Z}[k]} \hat{R} | \breve{\mathbf{W}}_k[1] \right), \dots, \sum_{k=1}^{K} \mathrm{Conv}\left(\nabla_{\mathbf{Z}[k]} \hat{R} | \breve{\mathbf{W}}_k[C] \right) \right]
\end{aligned}$$

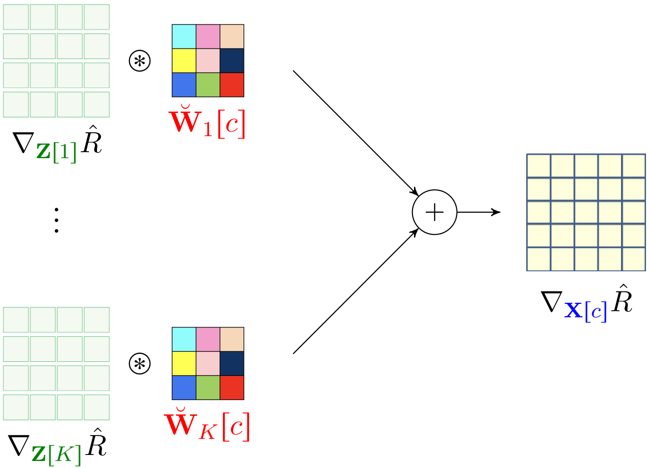
Backpropagation: *Multi-Channel Convolution*

The backward pass is therefore

$$\nabla_{\mathbf{X}} \hat{R} = \left[\sum_{k=1}^{K} \mathrm{Conv}\left(\nabla_{\mathbf{Z}[k]} \hat{R} | \breve{\mathbf{W}}_k[1] \right) \dots \sum_{k=1}^{K} \mathrm{Conv}\left(\nabla_{\mathbf{Z}[k]} \hat{R} | \breve{\mathbf{W}}_k[C] \right) \right]$$

Let's look at a particular input channel c

Backpropagation: *Multi-Channel Convolution*



Backpropagation: *Multi-Channel Convolution*

This is a new multi-channel convolution: let's define K-channel filter $\mathbf{W}_c^\dagger$ for $c = 1, \dots, C$ as follows

$$\mathbf{W}_c^\dagger = \left[\breve{\mathbf{W}}_1[c] \dots \breve{\mathbf{W}}_K[c] \right]$$

Then, channel c of $\nabla_{\mathbf{X}} \hat{R}$ is the convolution of $\nabla_{\mathbf{Z}} \hat{R}$ with $\mathbf{W}_c^\dagger$

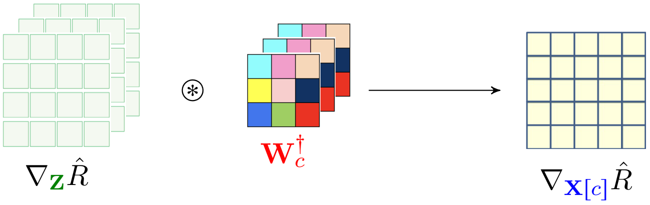
$$\nabla_{\mathbf{X}[c]} \hat{R} = \mathrm{Conv} \left(\nabla_{\mathbf{Z}} \hat{R} | \mathbf{W}_c^\dagger \right)$$

Or shortly, we can write

$$\nabla_{\mathbf{X}} \hat{R} = \mathrm{Conv} \left(\nabla_{\mathbf{Z}} \hat{R} | \mathbf{W}_1^\dagger, \dots, \mathbf{W}_C^\dagger \right)$$

Backpropagation: *Multi-Channel Convolution*

This means that we can look at channel c *of* $\nabla_{\mathbf{x}} \hat{R}$ *as*



Backpropagation: *Multi-Channel Convolution*

> *To backpropagate to channel c of tensor input, we convolve the output gradient tensor with the K-channel filter tensor that is constructed by collecting the channel c of all K forward filters, each being reversed up-to-down and right-to-left*

Backpropagation through Multi-Channel Convolution

Let $\mathbf{Z} = \operatorname{Conv}\left(\mathbf{X}|\mathbf{W}_1, \ldots, \mathbf{W}_K\right)$, where $\mathbf{X}$ is a C-channel input tensor, $\mathbf{W}_k$ is C-channel filter and $\mathbf{Z}$ is a K-channel output tensor. Assume that we know the gradient of loss $\hat{R}$ with respect to the output tensor, i.e., $\nabla_{\mathbf{Z}} \hat{R}$. Then, the gradient of loss $\hat{R}$ with respect to input tensor is

$$\nabla_{\mathbf{X}} \hat{R} = \operatorname{Conv}\left(\nabla_{\mathbf{Z}} \hat{R} | \mathbf{W}_1^{\dagger}, \ldots, \mathbf{W}_C^{\dagger}\right)$$

where $\mathbf{W}_c^{\dagger} = \left[\breve{\mathbf{W}}_1[c] \ldots \breve{\mathbf{W}}_K[c]\right]$ *is a K-channel filter*

Backpropagation through Convolution: *Summary*

Moral of Story

We can always backpropagate through a convolutional layer with C input channels and K output channels by

1. *multiply output gradient entry-wise with derivative of activation function*
2. *convolve output gradient with C different K-channel filters computed by rearranging of the K forward filters*

To compute the gradient with respect to weights of channel c in filter k

↳ *convolve channel c of input tensor with channel k of output gradient*

+ *Sounds great! But what about cases with stride $\neq$ 1?!*
– Well! we said *we can decompose them as convolution/pooling with stride 1 + a resampling unit. We just need to learn how to backpropagate through a resampling unit*

Backpropagation: *Downsampling*

Let's again try an example: *we have partial derivatives with respect to down-sampled variables and want to compute the partial derivatives with respect to input variables*

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

$$\mathbf{Z} = \mathrm{dSample}\left(\mathbf{X}|2\right) \rightsquigarrow$$

$z_1 = x_1$	$z_2 = x_3$
$z_3 = x_7$	$z_4 = x_9$

Backpropagation: *Downsampling*

Let's write with chain rule

$$\frac{\partial \hat{R}}{\partial x_1} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_1} = \frac{\partial \hat{R}}{\partial z_1}$$

$$\frac{\partial \hat{R}}{\partial x_2} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_2} = 0$$

$$\frac{\partial \hat{R}}{\partial x_3} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_3} = \frac{\partial \hat{R}}{\partial z_2}$$

$$\frac{\partial \hat{R}}{\partial x_4} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_4} = 0$$

$$\vdots$$

$$\frac{\partial \hat{R}}{\partial x_7} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_7} = \frac{\partial \hat{R}}{\partial z_3}$$

$$\frac{\partial \hat{R}}{\partial x_8} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_8} = 0$$

$$\frac{\partial \hat{R}}{\partial x_9} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_9} = \frac{\partial \hat{R}}{\partial z_4}$$

Backpropagation: *Downsampling*

This is simply an upsampling with the same factor

$$
\begin{bmatrix}
\frac{\partial \hat{R}}{\partial z_1} & 0 & \frac{\partial \hat{R}}{\partial z_2} \\
0 & 0 & 0 \\
\frac{\partial \hat{R}}{\partial z_3} & 0 & \frac{\partial \hat{R}}{\partial z_4}
\end{bmatrix}
\qquad
\underset{\leftsquigarrow}{\mathrm{uSample}\left(\nabla_{\mathbf{z}} \hat{R} | 2\right)}
\qquad
\begin{bmatrix}
\frac{\partial \hat{R}}{\partial z_1} & \frac{\partial \hat{R}}{\partial z_2} \\
\frac{\partial \hat{R}}{\partial z_3} & \frac{\partial \hat{R}}{\partial z_4}
\end{bmatrix}
$$

Backpropagation: *Upsampling*

Now, let's look into upsampling: *we have partial derivatives with respect to up-sampled variables and want to compute the partial derivatives with respect to input variables*

$z_1{=}x_1$	$z_2{=}0$	$z_3{=}x_2$
$z_4{=}0$	$z_5{=}0$	$z_6{=}0$
$z_7{=}x_3$	$z_8{=}0$	$z_9{=}x_4$

$$\mathbf{Z} = \mathrm{uSample}\left(\mathbf{X}|2\right)$$

x_1	x_2
x_3	x_4

Backpropagation: *Downsampling*

Let's write with chain rule

$$\frac{\partial \hat{R}}{\partial x_1} = \sum_{i=1}^{9} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_1} = \frac{\partial \hat{R}}{\partial z_1}$$

$$\frac{\partial \hat{R}}{\partial x_2} = \sum_{i=1}^{9} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_2} = \frac{\partial \hat{R}}{\partial z_3}$$

$$\frac{\partial \hat{R}}{\partial x_3} = \sum_{i=1}^{9} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_3} = \frac{\partial \hat{R}}{\partial z_7}$$

$$\frac{\partial \hat{R}}{\partial x_4} = \sum_{i=1}^{9} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_4} = \frac{\partial \hat{R}}{\partial z_9}$$

Backpropagation: *Upsampling*

This is downsampling with the same factor

$\frac{\partial \hat{R}}{\partial z_1}$	$\frac{\partial \hat{R}}{\partial z_2}$	$\frac{\partial \hat{R}}{\partial z_3}$
$\frac{\partial \hat{R}}{\partial z_4}$	$\frac{\partial \hat{R}}{\partial z_5}$	$\frac{\partial \hat{R}}{\partial z_6}$
$\frac{\partial \hat{R}}{\partial z_7}$	$\frac{\partial \hat{R}}{\partial z_8}$	$\frac{\partial \hat{R}}{\partial z_9}$

$$\mathrm{dSample}\left(\nabla_{\mathbf{z}} \hat{R} | 2\right)$$

$\frac{\partial \hat{R}}{\partial z_1}$	$\frac{\partial \hat{R}}{\partial z_3}$
$\frac{\partial \hat{R}}{\partial z_7}$	$\frac{\partial \hat{R}}{\partial z_9}$

Backpropagation: *Resampling*

Backpropagation through Downsampling

To backpropagate through a downsampling unit with factor (stride) S we up-sample the output gradient with factor S

Backpropagation through Upsampling

To backpropagate through an upsampling unit with factor (stride) S we down-sample the output gradient with factor S

Forward and Backpropagation in CNNs: *Summary*

To each forward action, there is a backward counterpart

In forward pass we do

- *forward FNN*
- *pooling*
- *convolution*
- *upsampling*
- *downsampling*

In backward pass we do

- *backward FNN*
- *backward pooling* $\sim$ *convolution*
- *convolution with reversed filters*
- *downsampling*
- *upsampling*

Once we over with backward pass, we compute all required gradients from forward and backward variables

We Use Advanced Techniques

For FNN, we looked into advanced techniques such as

- *Dropout and Regularization*
  - ↳ *to reduce the impact of overfitting*
- *Input* Normalization
  - ↳ *to improve training with unbalanced datasets*
- *Batch* Normalization
  - ↳ *to make the training more stable against feature variations*
- *Data Preprocessing*
  - ↳ *to clean our training dataset*

Same goes with CNNs

we should use all these methods for same purposes in CNNs

Other Forms of Convolution

The convolution operation we considered in this chapter is often called

2D Convolution

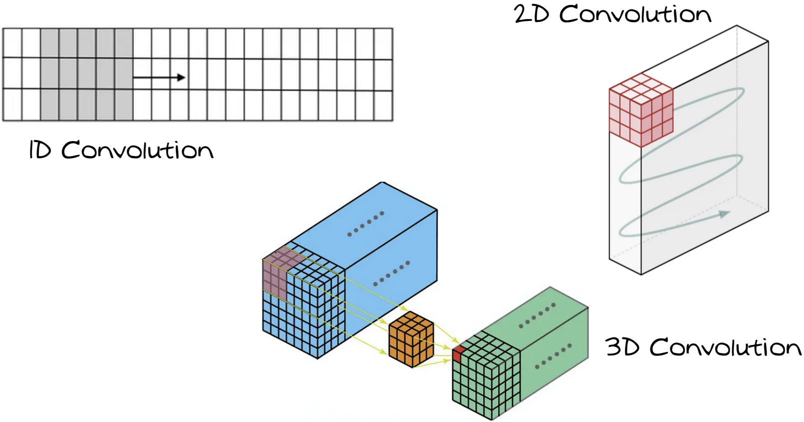
because it screens input in a 2D fashion, i.e., left-to-right and up-to-down

---

2D convolution is the most popular form of convolution; however,

- *we could have 1D convolutions*
  - ↳ *The filter only slides in one direction, i.e., left-to-right or up-to-down*
  - ↳ *This is useful with audio data where we slide the filter over time*
- *we could also have 3D convolutions*
  - ↳ *The filter slides in all three direction, i.e.,left-to-right, up-to-down and front-to-back*
  - ↳ *This is useful with 3D images, e.g., 3D medical image of brain*

Other Forms of Convolution



There are also other forms, e.g., depth-wise convolution

at the end of day, they all slide over input with some filter