

UNIVERSITY OF
TORONTO

Applied Deep Learning

Lecture 6. *Deep CNNs*

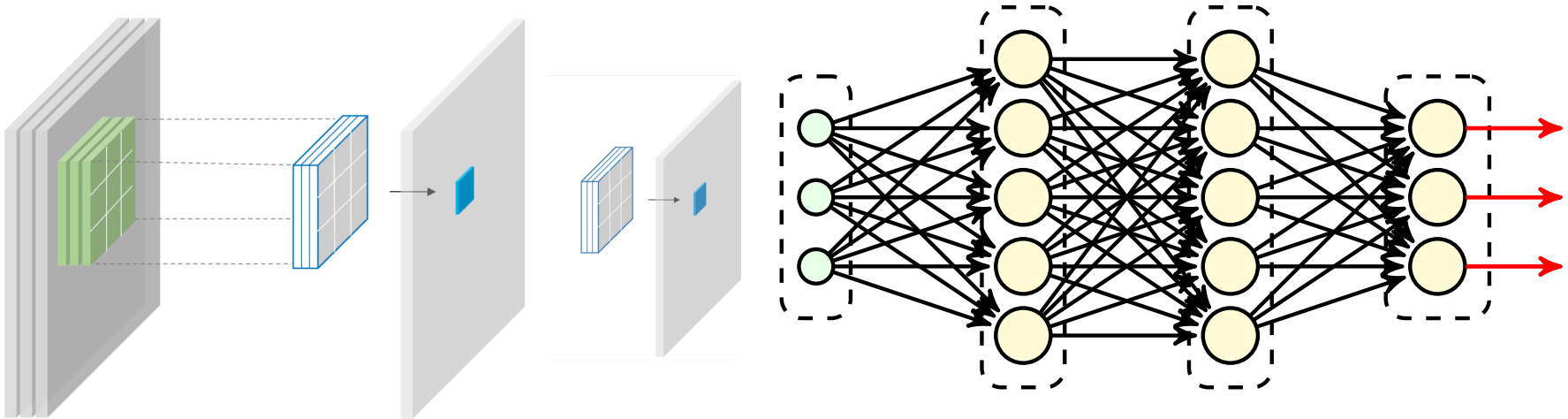
Ali Bereyhi
ECE Department, University of Toronto
Fall 2026



UNIVERSITY OF
TORONTO

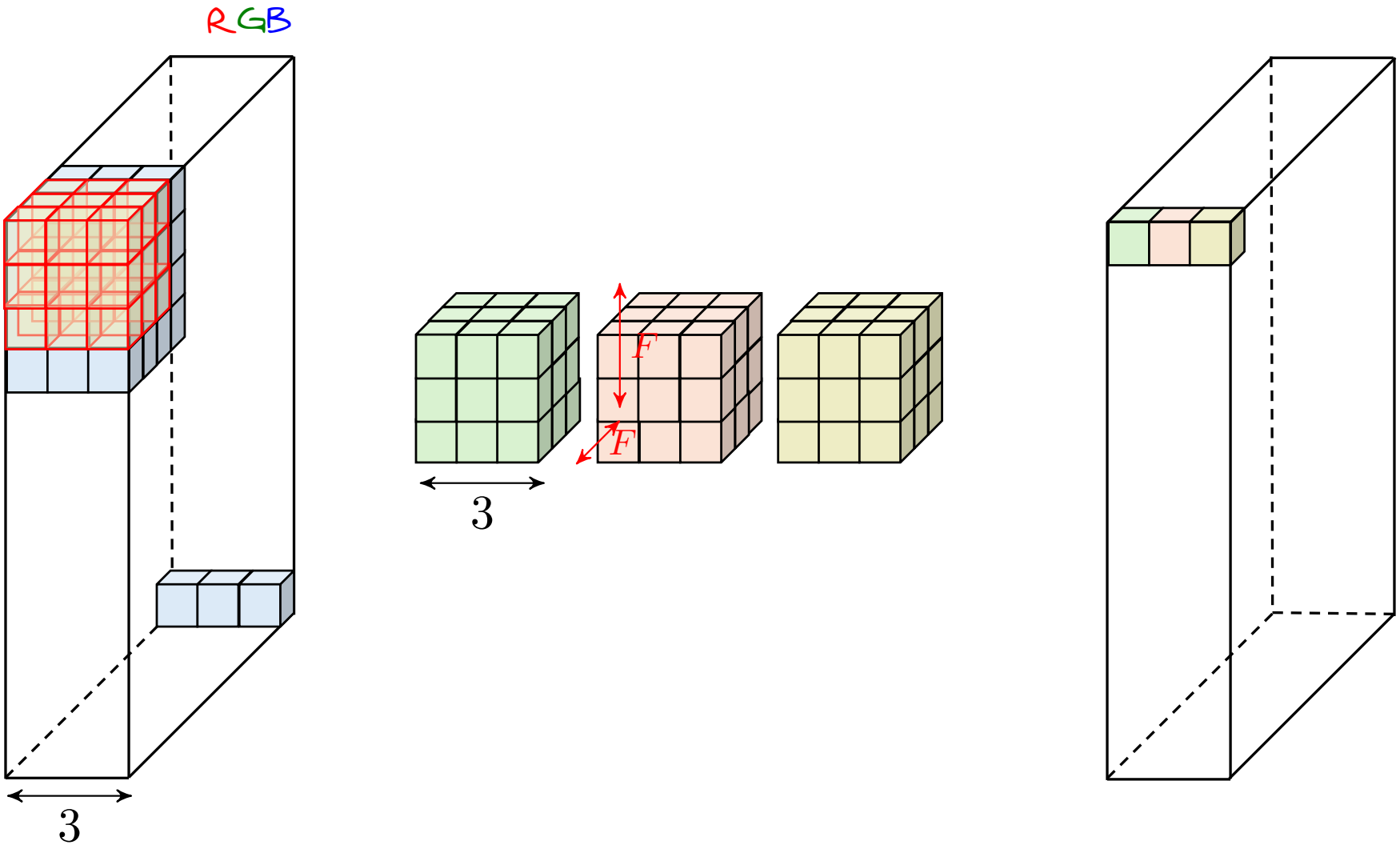
Lecture Overview

CNNs: *General Architectures*

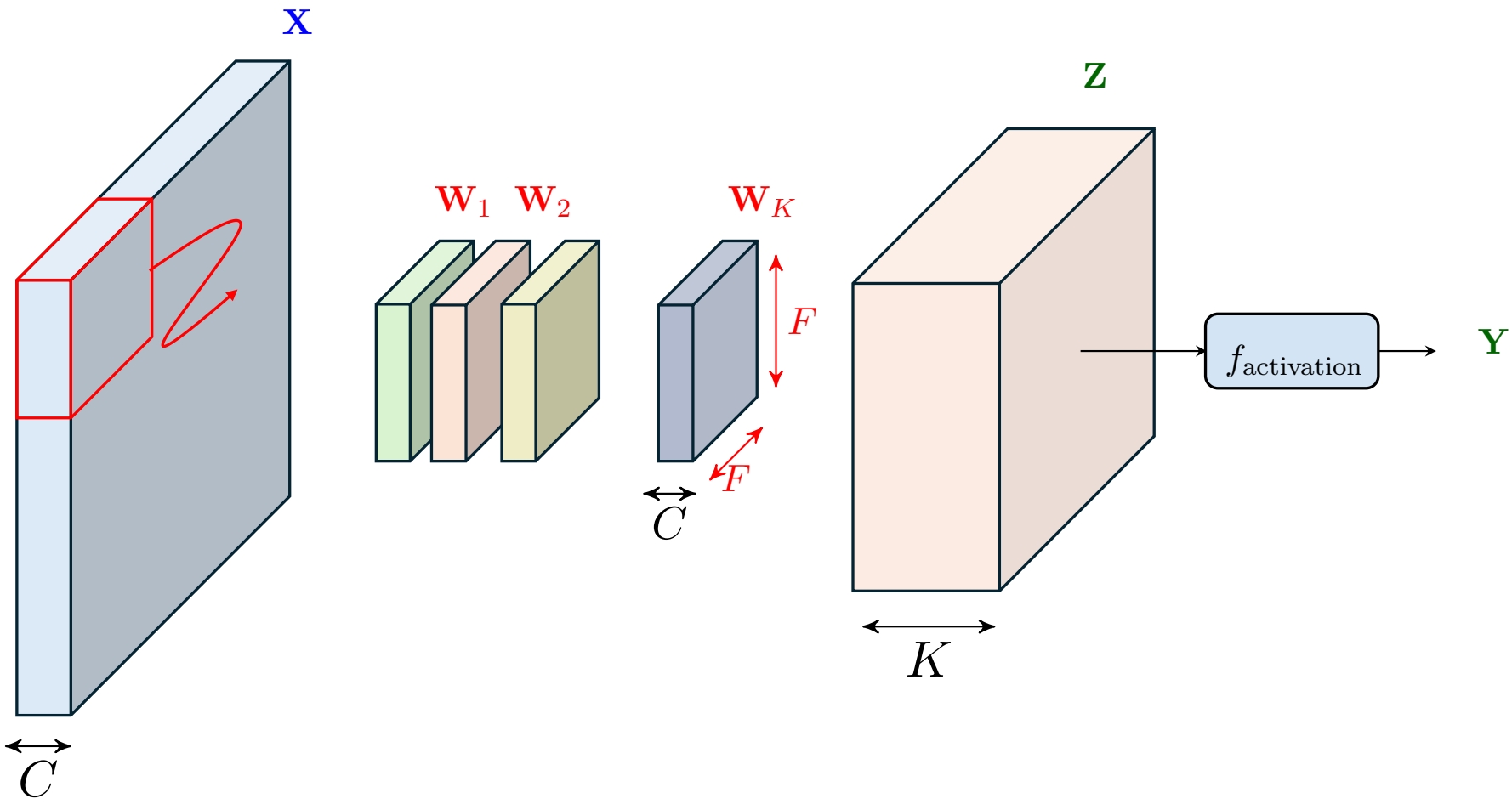


Convolution Pooling Flattening Output MLP

Multi-channel Convolution



Convolution: *Activated Layer*



Lecture 6. *Deep CNNs*

Today, we use our understanding of CNNs to

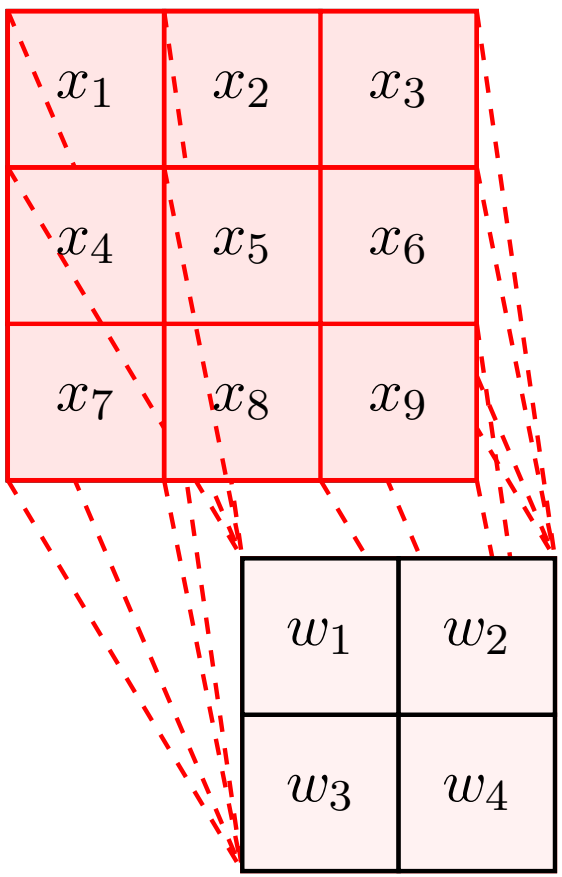
1. design a good deep CNN
   - *How does good architecture look like?*
   - *We take a look at VGG-16*
2. train CNNs
   - *We see that backpropagation is again dual of forward pass*
   - *We learn how to backpropagate*
3. learn the challenge of training very deep FNNs
   - *We learn what vanishing or exploding gradient is*
   - *We learn a bit of history*

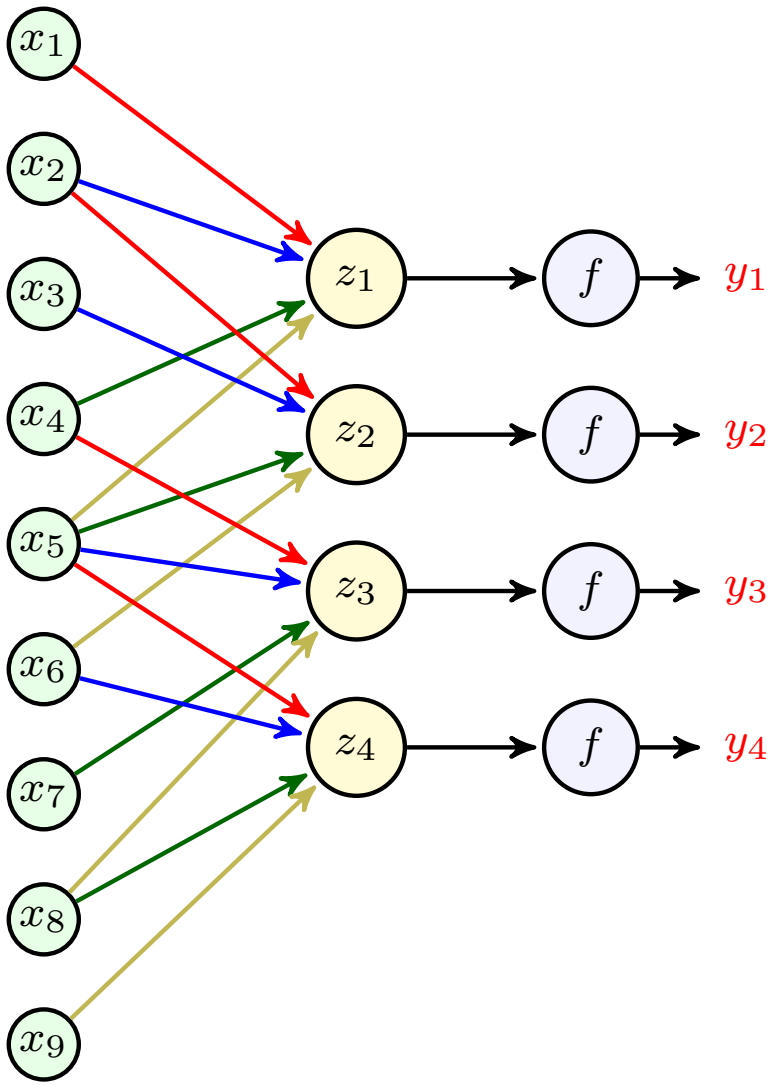


UNIVERSITY OF
TORONTO

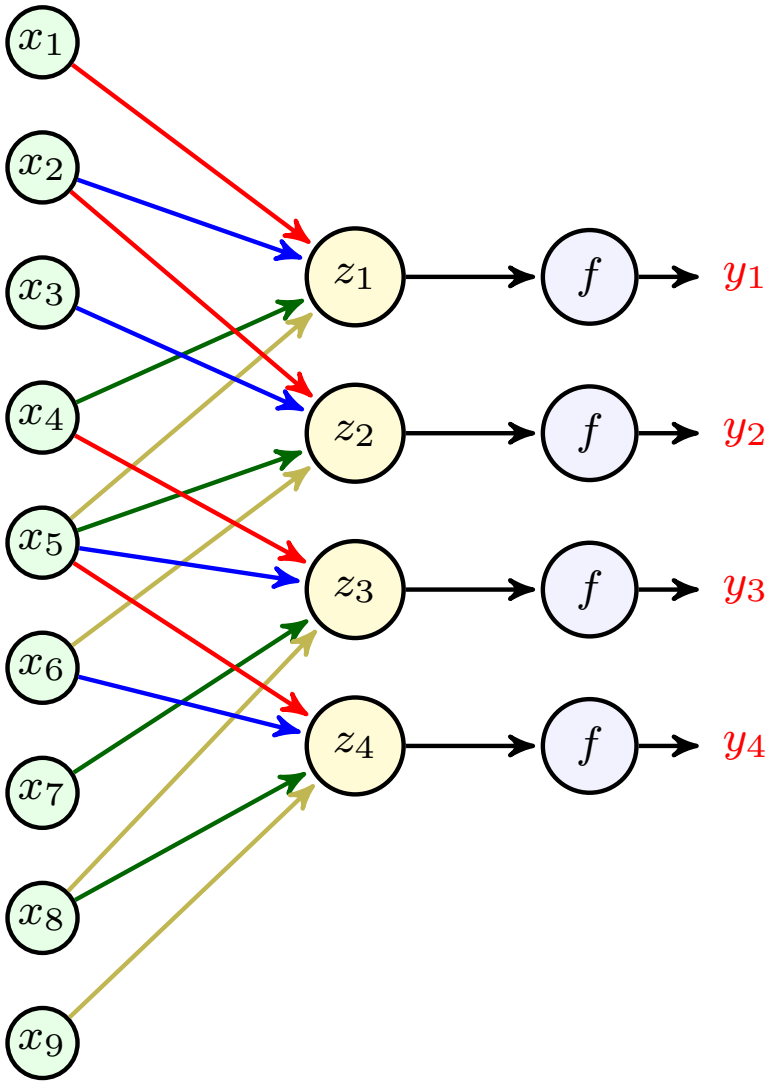
Deep CNNs

Convolutional Layer as *Set of Neurons*





Convolutional Layer as Set of Neurons



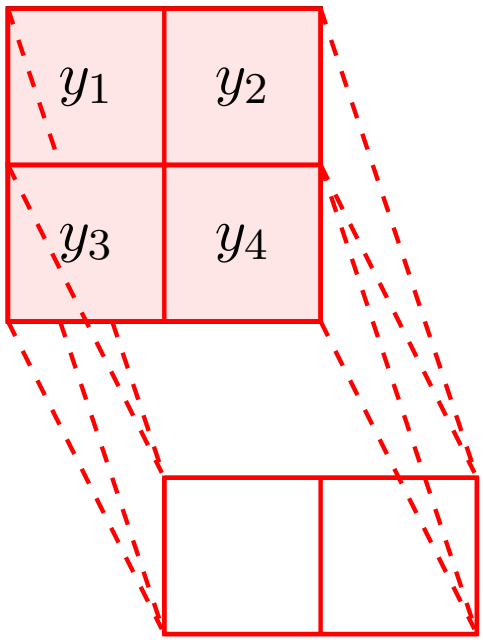
In this viewpoint

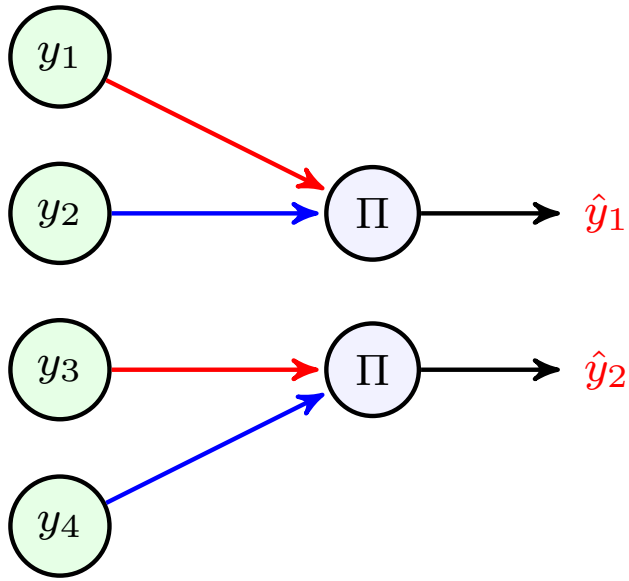
- ↳ *each feature is output of a neuron*
- ↳ *neurons share same weights and bias*
- ↳ *neurons are activated by* $f(\cdot)$

This is however not a fully-connected layer

- ↳ *it is locally-connected*
- ↳ *neurons have shared parameters*

Pooling Layer as *Set of Neurons*

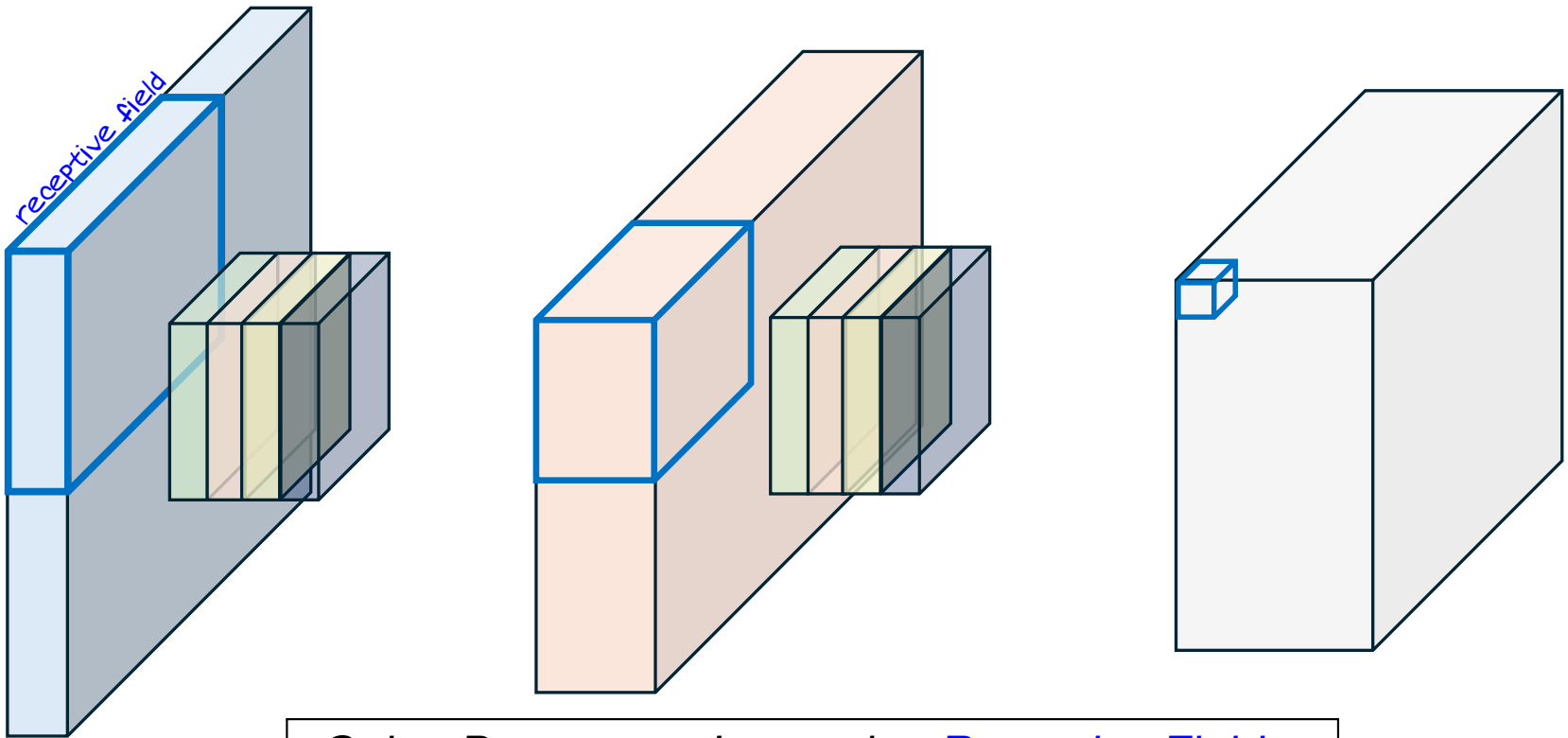




Pooling as a Layer

Similar to convolutional layer, pooling is a feedforward layer with local connectivity and shared parameters

Going Deep: *From Local to Global Features*



Going Deep ⟷ *Increasing Receptive Field*

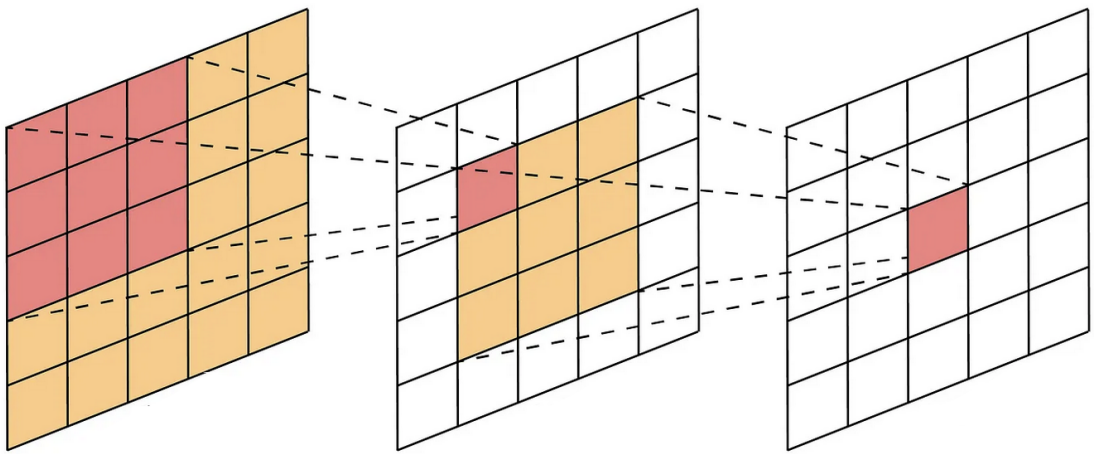
? *But, we end up too much global features!*

✔ *Well, we may use pooling with larger stride*

Going Deep: *Receptive Field*

As we go deep in CNN, we mix features: *features in deeper layers depend on more pixels of input tensor. We often say*

deeper layers have larger receptive field



Receptive Field

The input region that each neuron in a given convolutional layer responds to

General Perspective on Deep CNNs

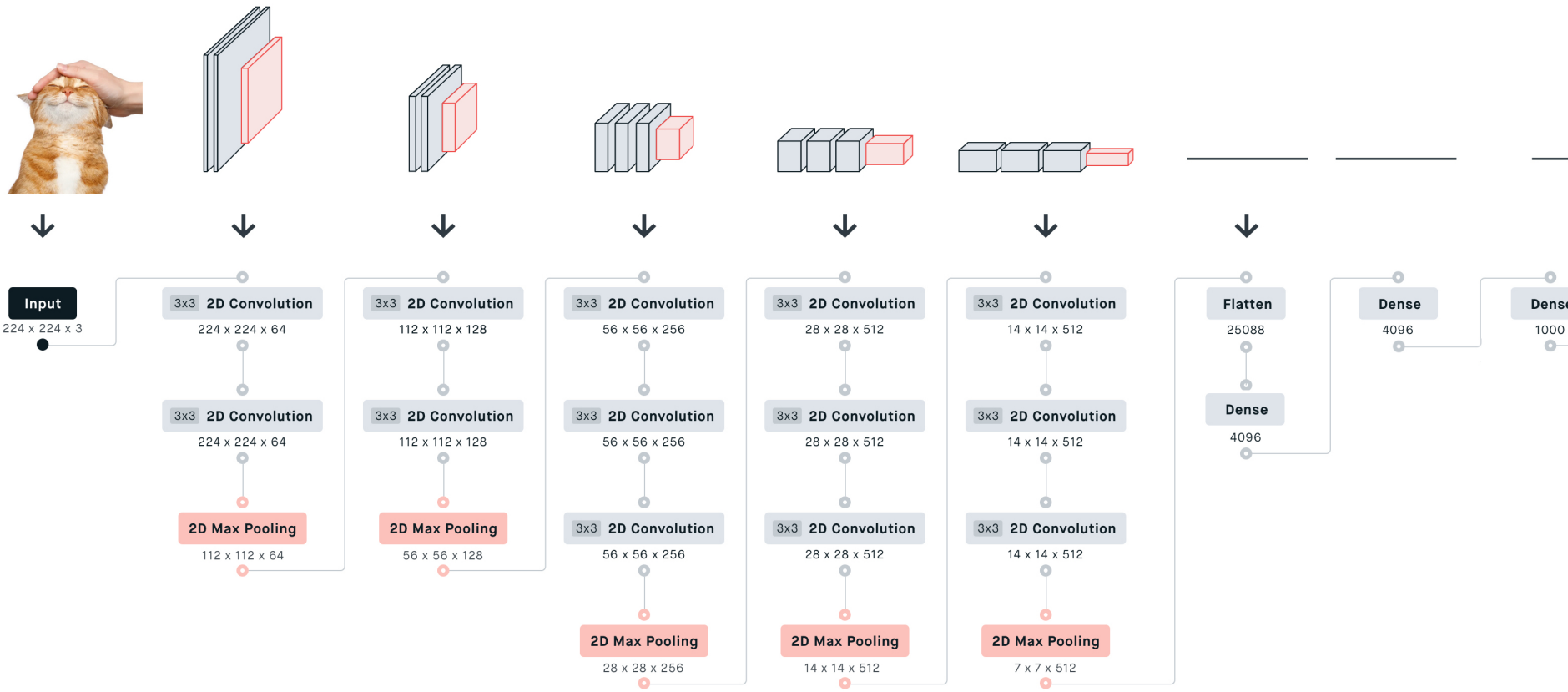
The given example illustrates *an intuitive interpretation of deep CNNs*

- ↳ *Deep CNNs extract gradually the features of input*
- ↳ *In first layers they extract features of smaller patches of the input*
- ↳ *They gradually expand this region as they go deeper*
  - ↳ *deeper layers look at high-level features*

This gives us a good idea on *how a good deep CNN looks like*

- *First layers focus on understanding local features*
  - ↳ *the computed feature maps are large*
  - ↳ *the number of output channels is small*
- *As we go deeper in the CNN, we get more global understanding*
  - ↳ *the computed feature maps get smaller*
  - ↳ *the number of output channels increases*

Deep CNN: *VGG-16*



Deep CNN: *Example*

Let's count the number of layers with learnable parameters

$$
\begin{aligned}
\text{\# layers with weights} = \quad & 2 && \text{first two convolutions} \\
& + 2 && \text{second two convolutions} \\
& + 3 && \text{third round of convolutions} \\
& + 3 && \text{fourth round of convolutions} \\
& + 3 && \text{fifth round of convolutions} \\
& + 3 && \text{fully-connected network} \\
& = 16
\end{aligned}
$$

This is actually why it's called VGG-16



UNIVERSITY OF
TORONTO

Training CNNs



UNIVERSITY OF
TORONTO

Training CNNs

Big Picture: *What to Train*

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

- *weights and biases of output MLP*
- *weights in the filters of convolutional layers*

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

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

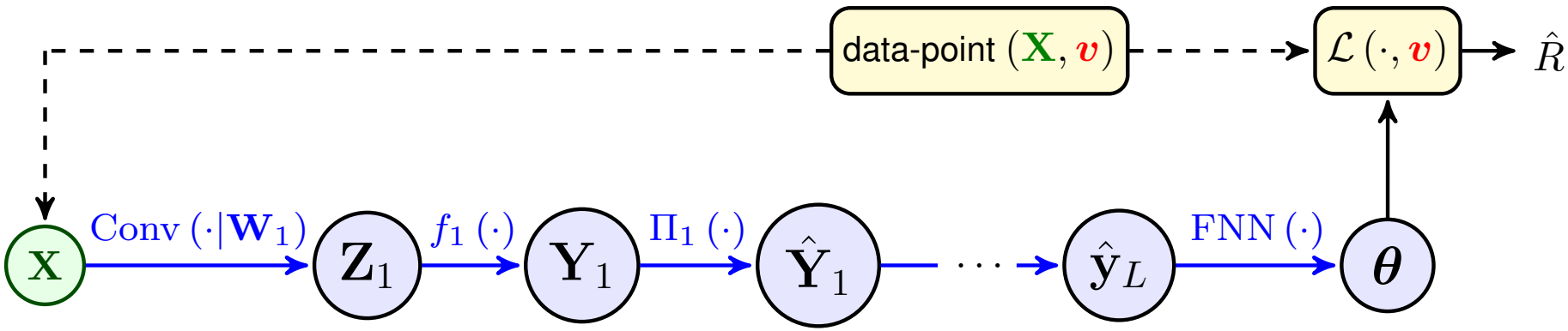
where $\boldsymbol{\theta}_b = \mathrm{CNN}(\mathbf{X}_b | \mathbf{w})$ *for the tensor-type data-point* $\mathbf{X}_b$ *with label* $\boldsymbol{v}_b$

↳ *We use a gradient-based optimizer*

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

Computation Graph

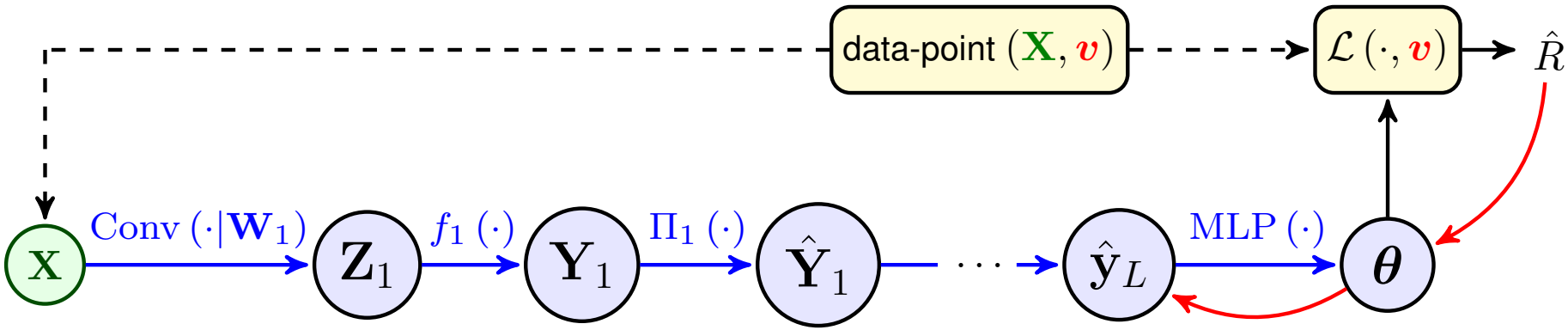
Computation graph for single data-point $\mathbf{X}$ and its true label $\boldsymbol{v}$ is as follows



For each sample $(\mathbf{X}, \boldsymbol{v})$, we need to

1. *pass forward $\mathbf{X}$ over this graph to compute sample loss*
   - ↳ *This is the basic computation in CNNs ⇝ we save all feature maps*
2. *pass backward to compute gradient*
   - ↳ *This is what we plan to do now!*

Backward Pass



The first step in backpropagation is to compute $\nabla_{\boldsymbol{\theta}} \hat{R}$ and

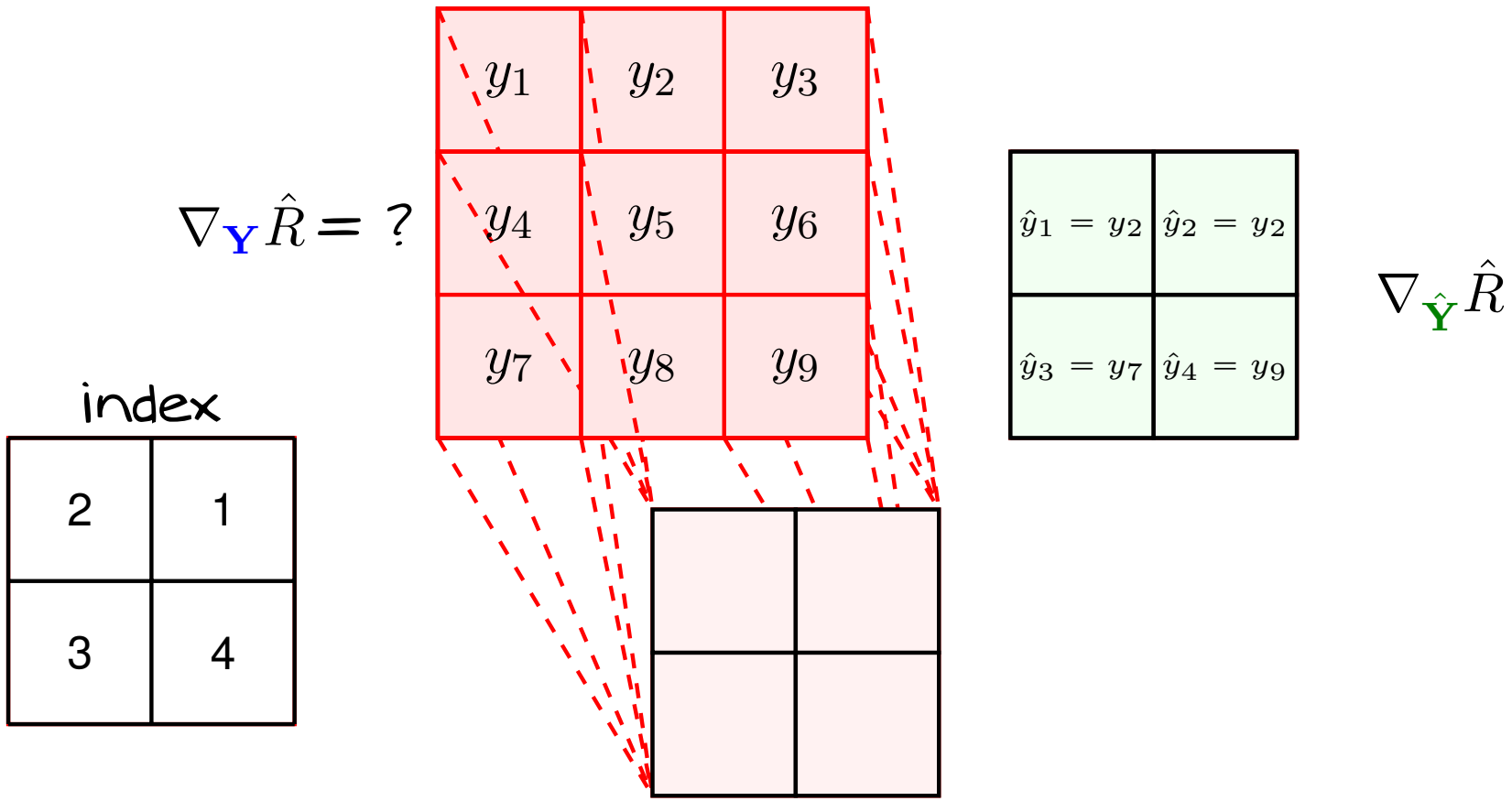
- ✔ *backpropagate over the MLP till we get to its input*

We can sort $\nabla_{\hat{\mathbf{y}}_L} \hat{R}$ as a tensor by reversing the flattening procedure

$$\nabla_{\hat{\mathbf{y}}_L} \hat{R} \leftrightsquigarrow \nabla_{\hat{\mathbf{Y}}_L} \hat{R}$$

Backpropagate through Pooling: *Max-Pooling*

Let's work with an example: *in the following pooling layer, we have the gradient with respect to pooled map and want to compute the gradient with respect to input map*



Backpropagation: *Max-Pooling*

We can use chain rule

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

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

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

$$\vdots$$

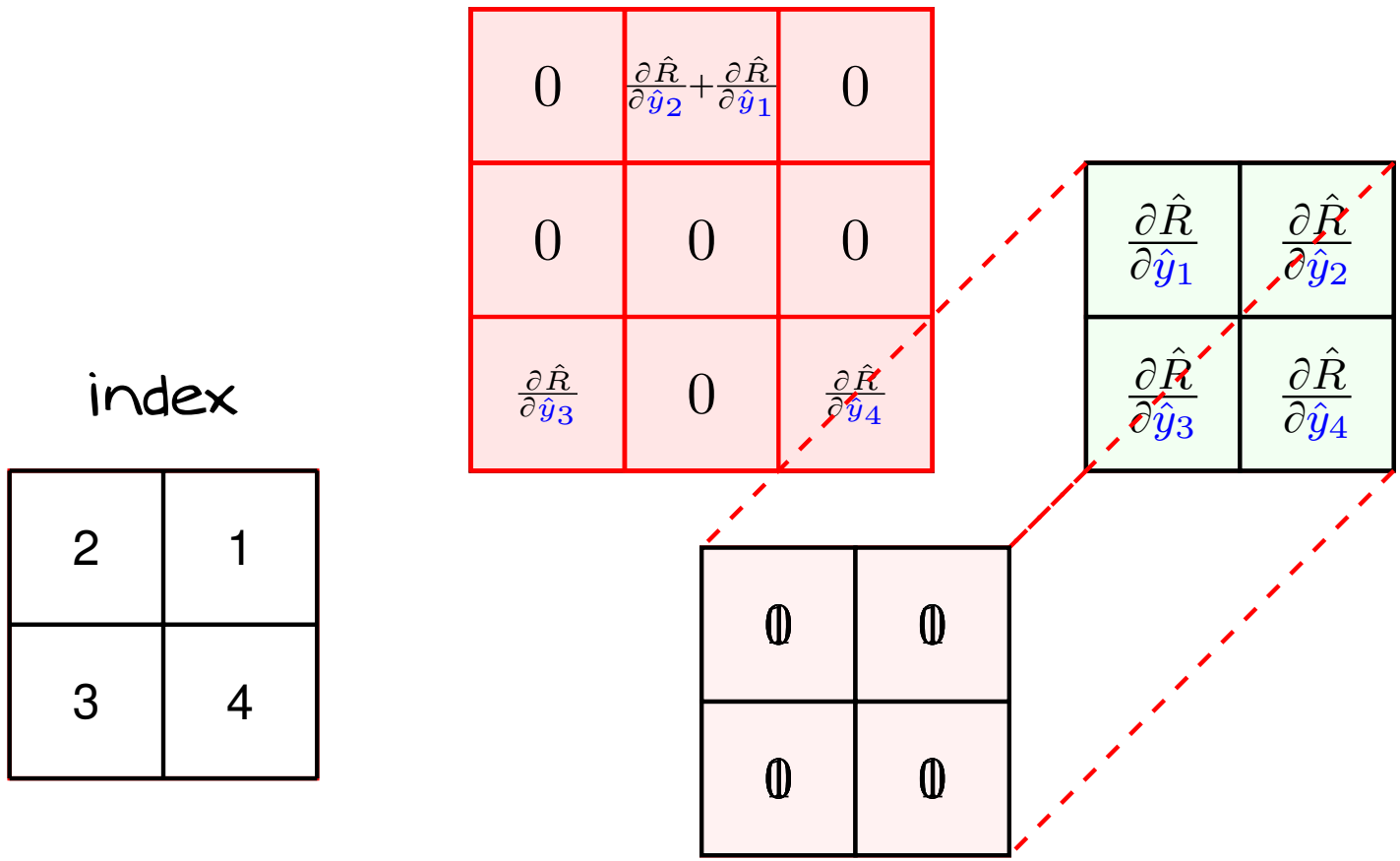
$$\frac{\partial \hat{R}}{\partial y_7} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial y_7} = \frac{\partial \hat{R}}{\partial \hat{y}_3}$$

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

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

Backpropagation: *Max-Pooling*

Each derivative in green map gets multiplied with a filter that is 1 only at the saved index ⇝ it is then added to corresponding entries on blue map



Backpropagation through Pooling: *General Pooling*

Backpropagating Pooling

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

- ? *What about mean pooling?*
- Even easier as we see after learning backpropagation through convolution

Nice Challenge!

We can indeed visualize the max-pooling backpropagation as convolution of index dependent filter with zero-padded output gradient

- ! *Think about it after you learned how to backpropagate through convolution*

Backpropagation through Convolution: *Activation*

Convolutional layer has two operations

- ✔ *entry-wise activation*
- ? *linear convolution*

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

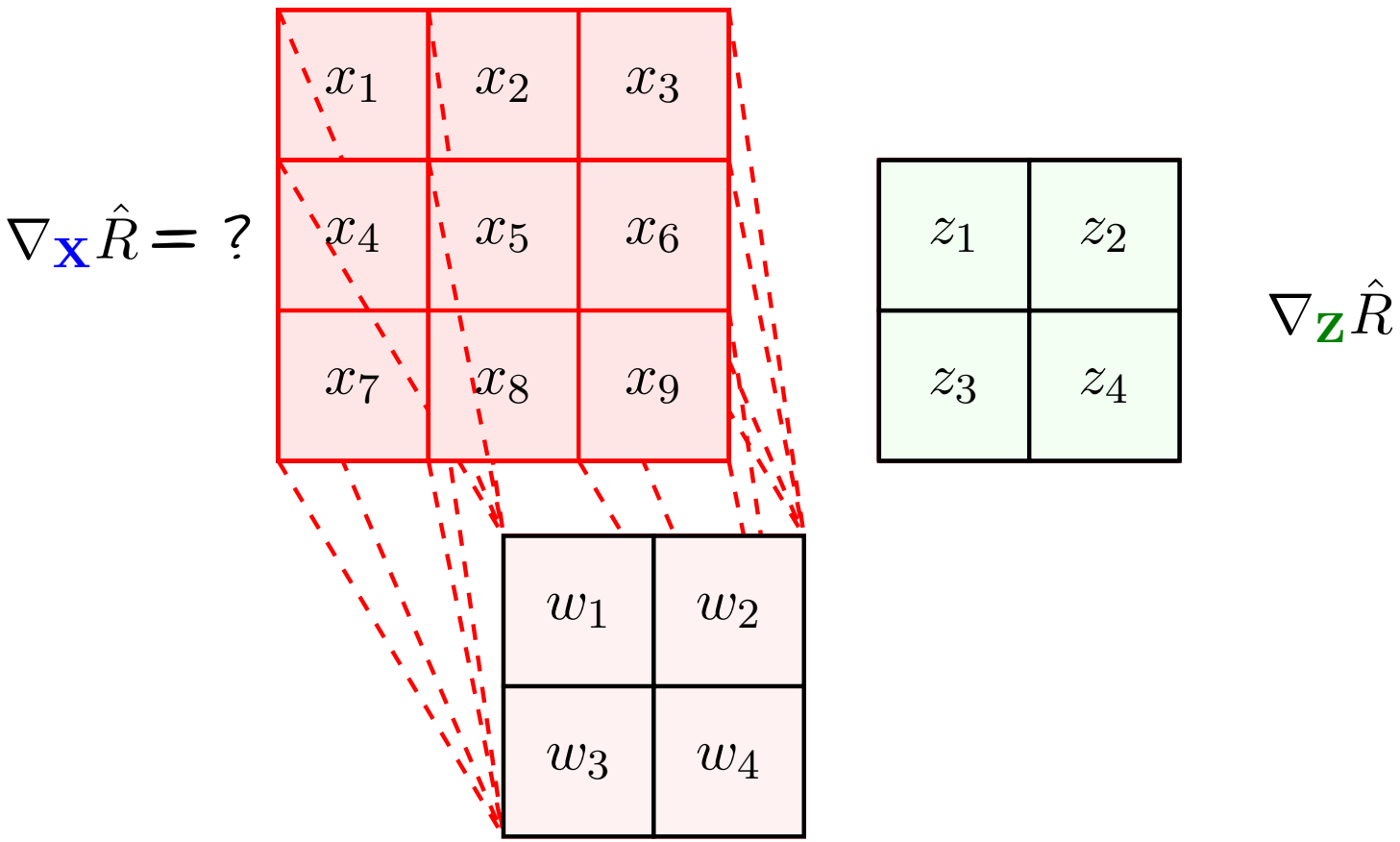
$$\mathbf{Y} = f(\mathbf{Z})$$

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

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

Backpropagation: *Convolution*

Let's work with an example: *we have gradient with respect to output map and want to compute the gradient with respect to input map*



Backpropagation: *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: *Convolution*

We can keep on till finish with chain rule

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

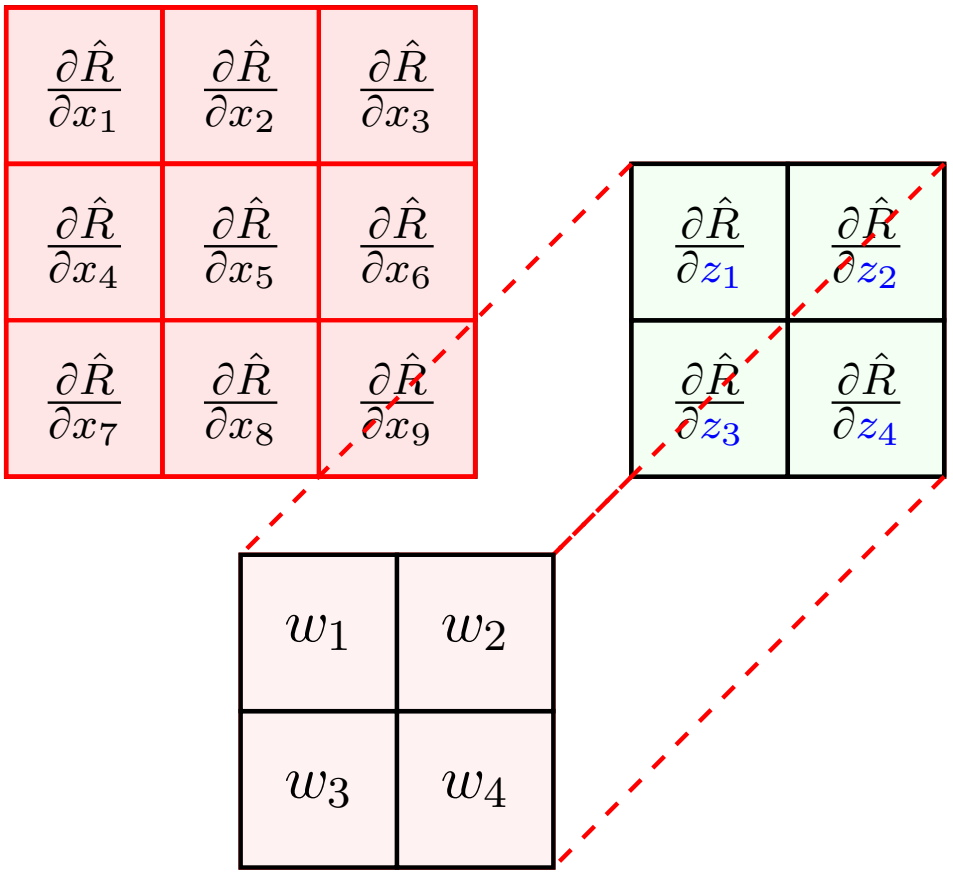
$$\frac{\partial \hat{R}}{\partial x_7} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial x_7} = w_3 \frac{\partial \hat{R}}{\partial z_3}$$

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

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

Backpropagation: *2D Convolution*

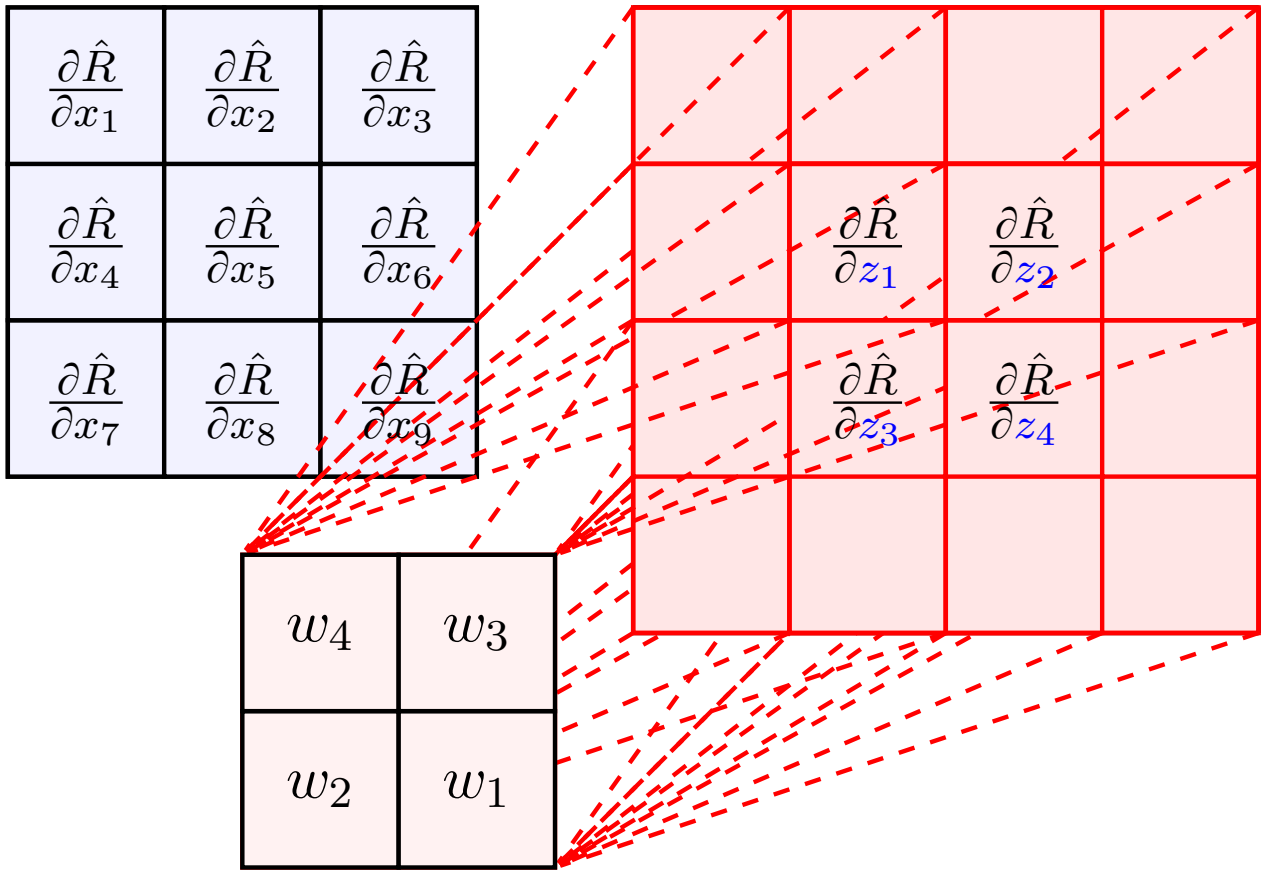
We can use the same visualization as in pooling



Or even a better visualization!

Backpropagation: *Convolution*

It's again simply a convolution



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

Backpropagation: *Convolution*

Backpropagation through Single-channel Convolution

Let $\mathbf{Z} = \mathrm{Conv}\left(\mathbf{X}|\mathbf{W}\right)$*, where* $\mathbf{X}$ *is the input map,* $\mathbf{W}$ *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: *Gradient of Filter*

? *What about the gradient with respect to filter itself?*

$\mathbf{X}$

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

z_1	z_2
z_3	z_4

$\nabla_{\mathbf{Z}} \hat{R}$

w_1	w_2
w_3	w_4

$\nabla_{\mathbf{W}} \hat{R} = \, ?$

Backpropagation: *Gradient of Filter*

As always, we use chain rule

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

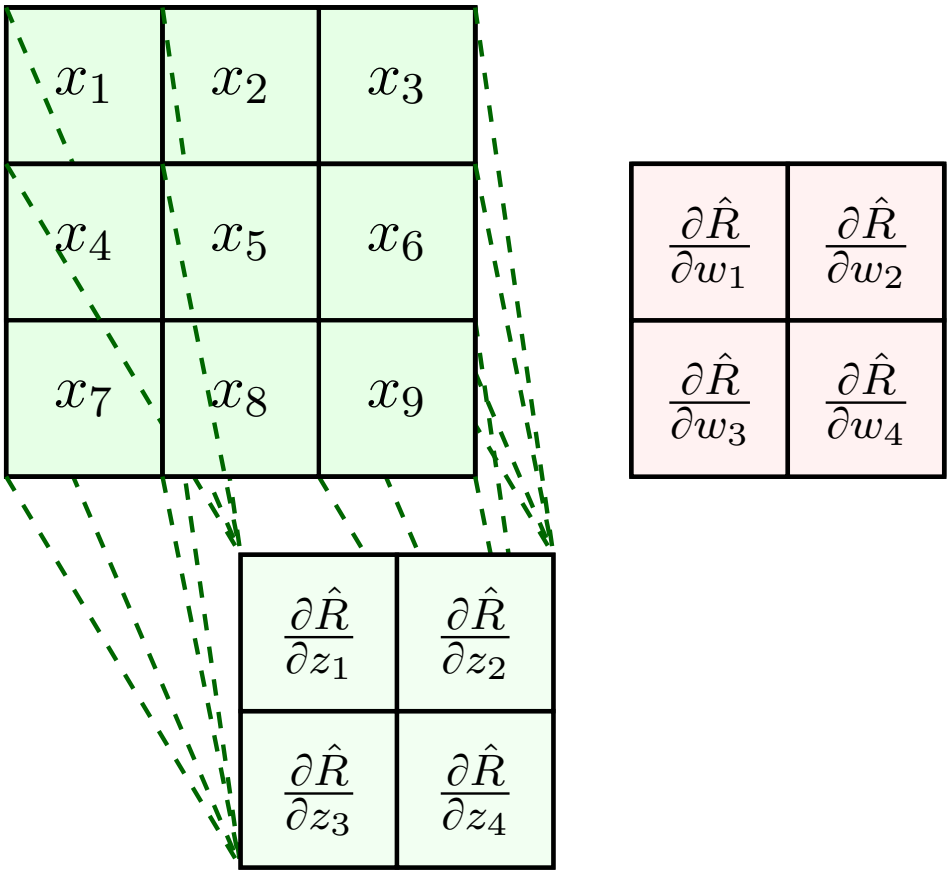
$$\frac{\partial \hat{R}}{\partial w_2} = \sum_{i=1}^{4} \frac{\partial \hat{R}}{\partial z_i} \frac{\partial z_i}{\partial w_2} = x_2 \frac{\partial \hat{R}}{\partial z_1} + x_3 \frac{\partial \hat{R}}{\partial z_2} + x_5 \frac{\partial \hat{R}}{\partial z_3} + x_6 \frac{\partial \hat{R}}{\partial z_4}$$

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

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

Backpropagation: *Gradient of Filter*

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



Backpropagation: *Gradient of Filter*

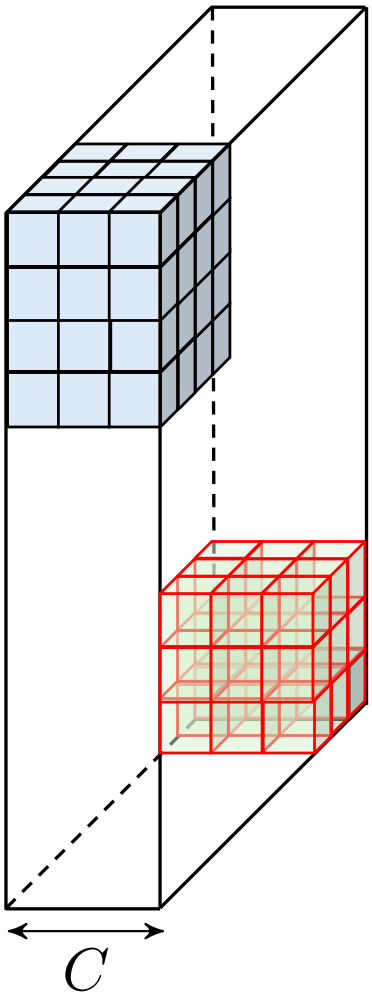
Gradient of Filter

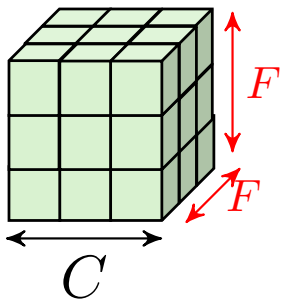
Let $\mathbf{Z} = \mathrm{Conv}\left(\mathbf{X}|\mathbf{W}\right)$, *where* $\mathbf{X}$ *is the input map,* $\mathbf{W}$ *is filter, and* $\mathbf{Z}$ *is the output map. Assume that we know gradient of loss* $\hat{R}$ *with respect to the output map, i.e.,* $\nabla_{\mathbf{Z}}\hat{R}$. *Then, gradient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

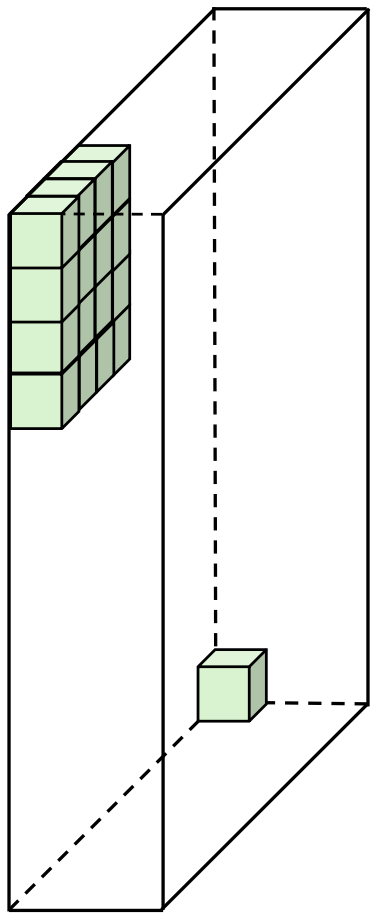
$$\nabla_{\mathbf{W}}\hat{R} = \mathrm{Conv}\left(\mathbf{X}|\nabla_{\mathbf{Z}}\hat{R}\right)$$

+ *But we checked only single-channel case! What about multi-channel case?!*
– Well, we can simply do it by chain rule

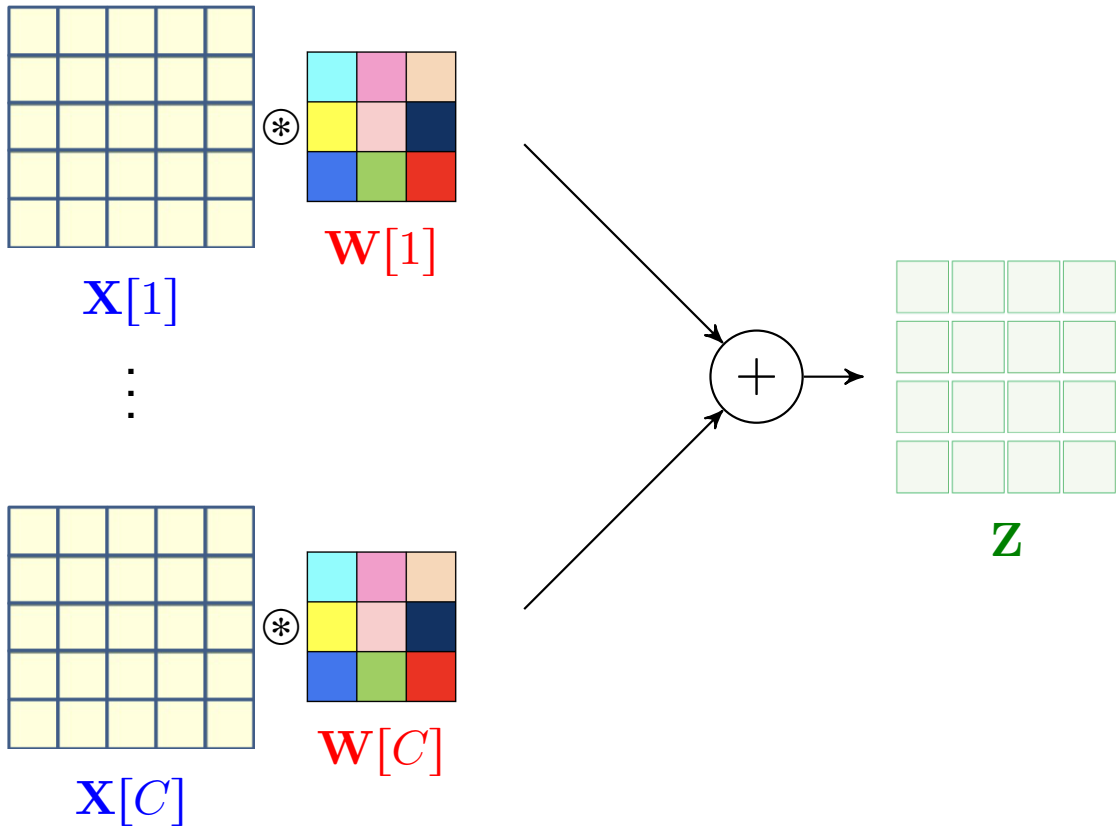
Recap: *Multi-channel Convolution*







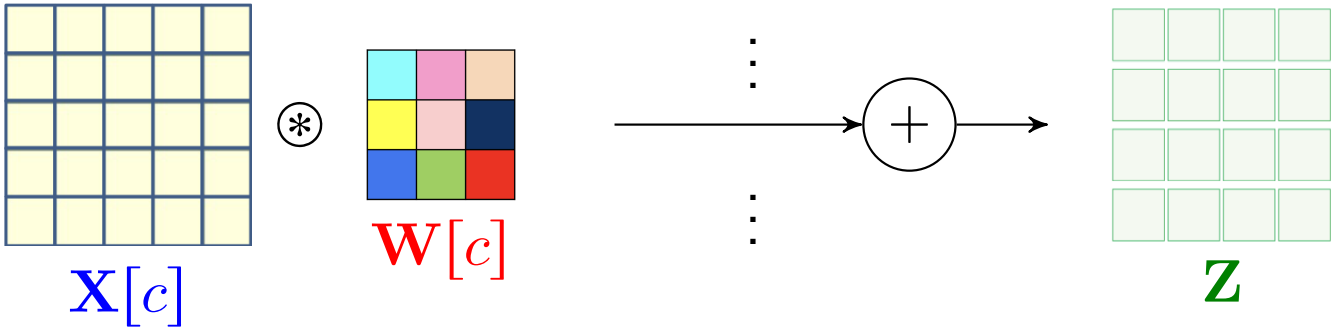
Alternative Representation: *Sum of Convolutions*



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

Backpropagation: *Multi-Channel Convolution*

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



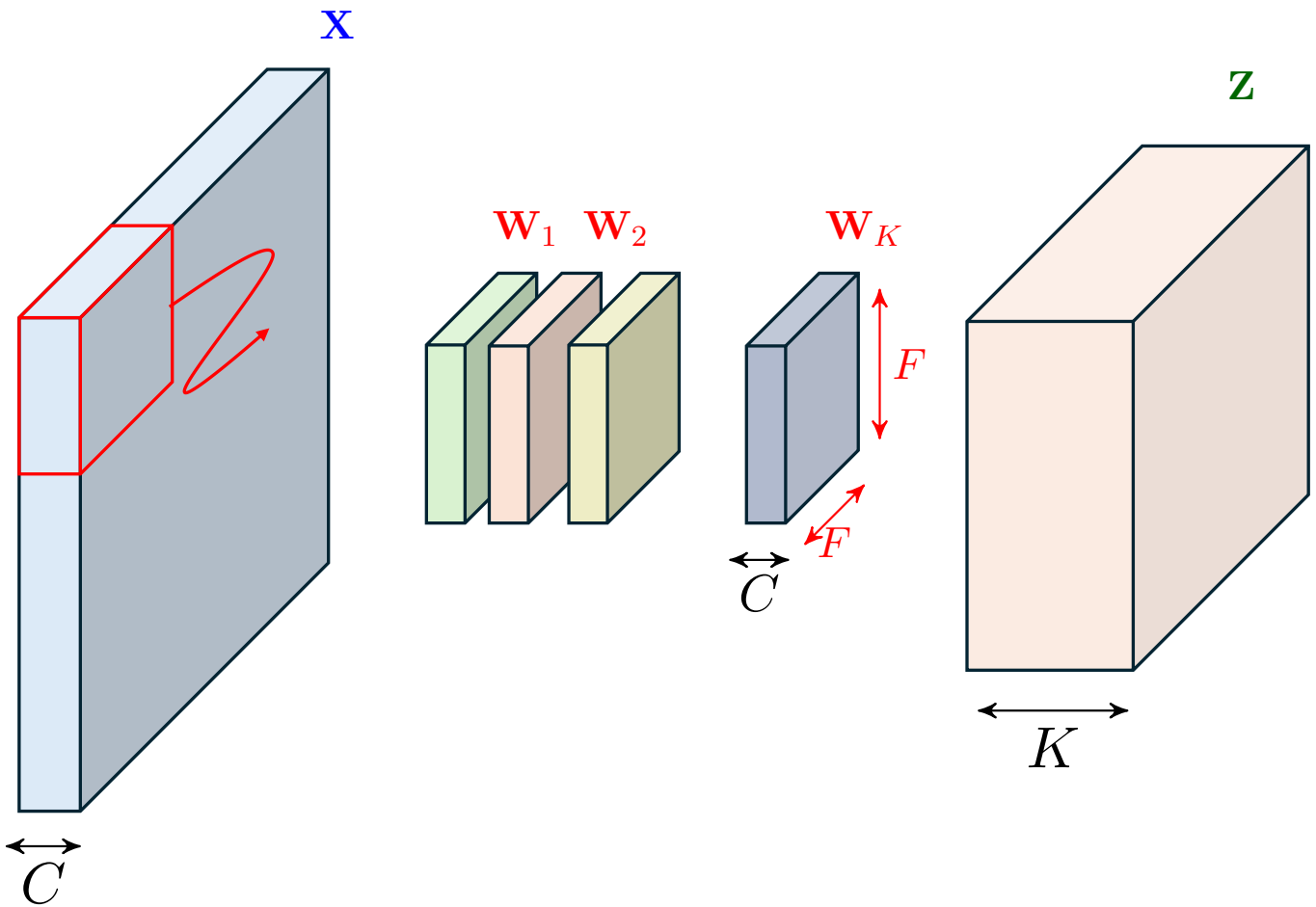
So, we could say

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

Or as a tensor

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

Recap: *Multi-channel Output*



$$\mathbf{Z} = [\text{Conv}(\mathbf{X}|\mathbf{W}_1), \ldots, \text{Conv}(\mathbf{X}|\mathbf{W}_K)]$$

Backpropagation: *Multi-Channel Convolution*

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

$$\mathbf{Z}[1] = \text{Conv}\left(\mathbf{X}|\mathbf{W}_1\right) \qquad \ldots \qquad \mathbf{Z}[K] = \text{Conv}\left(\mathbf{X}|\mathbf{W}_K\right)$$

So, the chain rule can be written as

$$\nabla_{\mathbf{X}}\hat{R} = \sum_{k=1}^{K} \underbrace{\nabla_{\mathbf{Z}[k]}\hat{R} \circ \nabla_{\mathbf{X}}\mathbf{Z}[k]}_{\text{what we calculated for single output map}}$$

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

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

Backpropagation: *Multi-Channel Convolution*

Let's look at a particular input channel c

$$\sum_{k=1}^{K} \text{Conv}\left(\nabla_{\mathbf{Z}[k]} \hat{R} | \breve{\mathbf{W}}_k[c]\right) = \text{Conv}\left(\nabla_{\mathbf{Z}} \hat{R} | \mathbf{W}_c^{\dagger}\right)$$

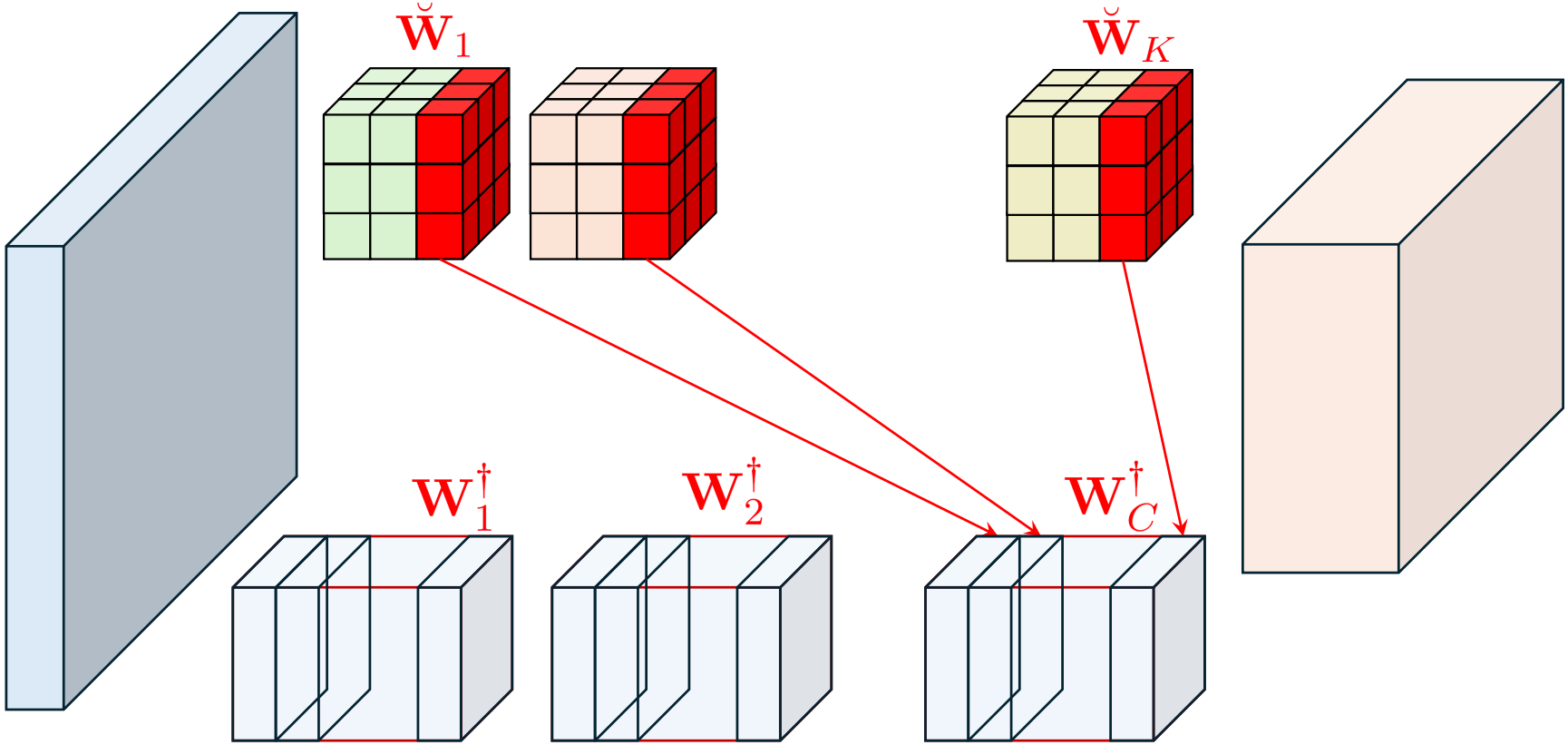
where we define K-channel filter $\mathbf{W}_c^{\dagger} \in \mathbb{R}^{K \times F \times F}$ as

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

So, we can write

$$\nabla_{\mathbf{X}} \hat{R} = \text{Conv}\left(\nabla_{\mathbf{Z}} \hat{R} | \mathbf{W}_1^{\dagger}, \mathbf{W}_2^{\dagger} \ldots, \mathbf{W}_C^{\dagger}\right)$$

Backpropagation: *Multi-channel Convolution*



Backpropagation: *Multi-Channel Convolution*

Backpropagation through Multi-Channel Convolution

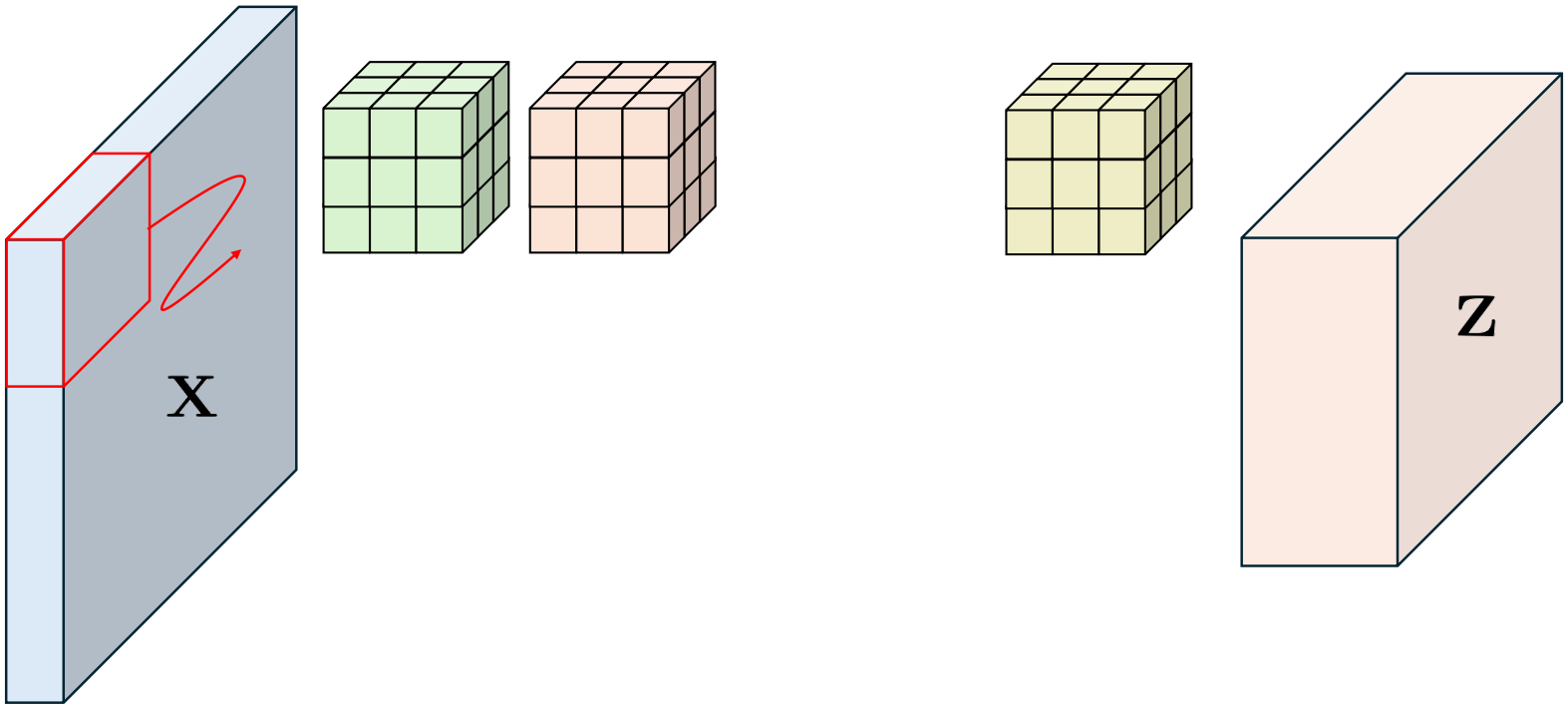
Let $\mathbf{Z} = \mathrm{Conv}\left(\mathbf{X}|\mathbf{W}_1, \ldots, \mathbf{W}_K\right)$, *where* $\mathbf{X}$ *is a* C*-channel input tensor,* $\mathbf{W}_k$ *is* C*-channel filter and* $\mathbf{Z}$ *is a* K*-channel output tensor. Assume that we know the gradient of loss* $\hat{R}$ *with respect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

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

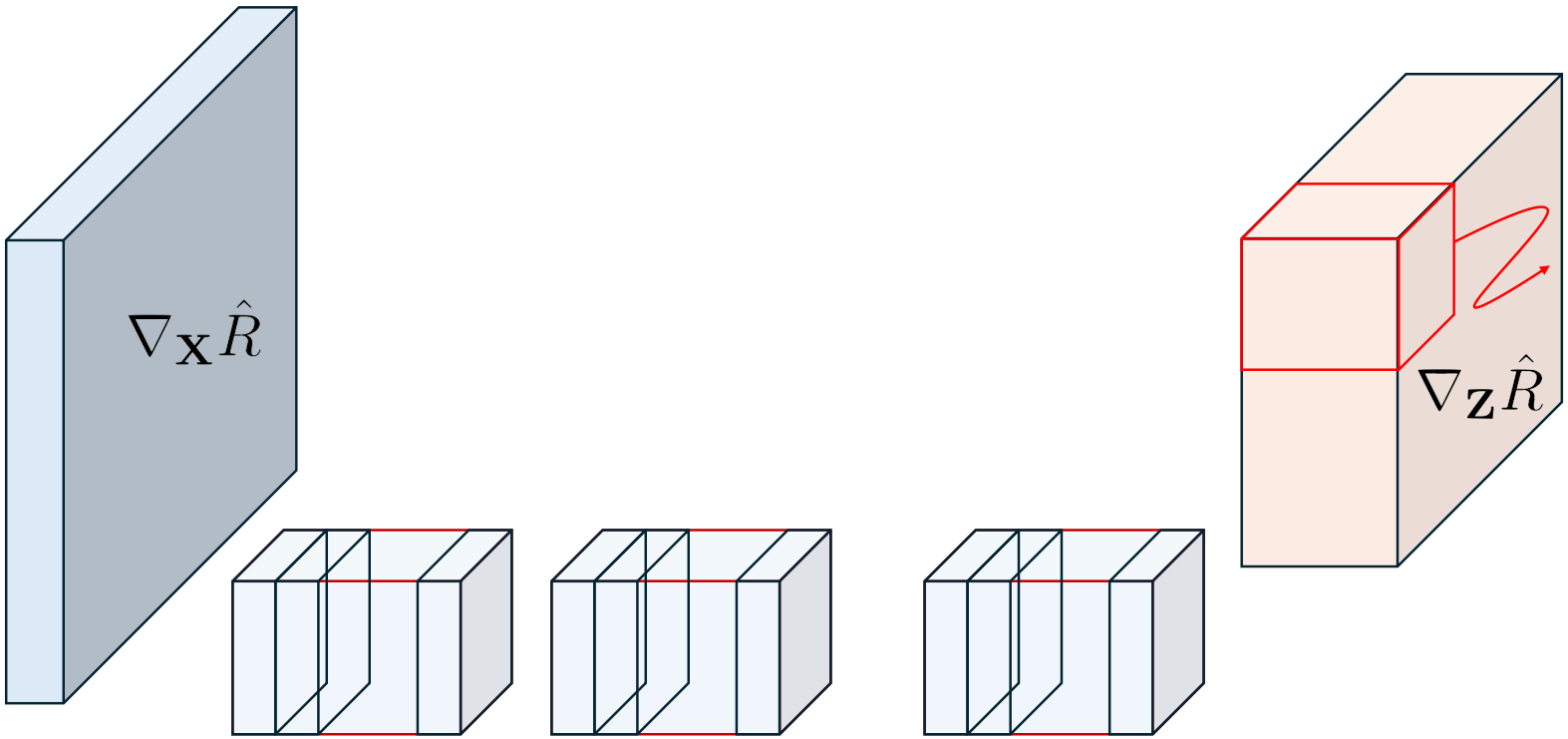
where $\mathbf{W}_c^{\dagger} = \left[\breve{\mathbf{W}}_1[c] \ldots \breve{\mathbf{W}}_K[c]\right]$ *is a* K*-channel filter*

- ? *What about the the gradient with respect to filter?*
- This will be again a convolution $\rightsquigarrow$ Think about it as a homework!

Summary: *Forward and Backward Pass*



Summary: *Forward and Backward Pass*



? *What if stride* $\neq 1$ *?*

We saw this is just a resampling

Backpropagation: *Downsampling*

x_1	x_2	x_3
x_4	x_5	x_6
x_7	x_8	x_9

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

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

$\frac{\partial \hat{R}}{\partial z_1}$	0	$\frac{\partial \hat{R}}{\partial z_2}$
0	0	0
$\frac{\partial \hat{R}}{\partial z_3}$	0	$\frac{\partial \hat{R}}{\partial z_4}$

$$\text{uSample}\left(\nabla_{\mathbf{Z}} \hat{R}|2\right) \leftsquigarrow$$

$\frac{\partial \hat{R}}{\partial z_1}$	$\frac{\partial \hat{R}}{\partial z_2}$
$\frac{\partial \hat{R}}{\partial z_3}$	$\frac{\partial \hat{R}}{\partial z_4}$

Backpropagation: *Upsampling*

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

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

x_1	x_2
x_3	x_4

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

$$\text{dSample}\left(\nabla_{\mathbf{Z}} \hat{R}|2\right)$$

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

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*

In forward pass we do

- *forward MLP*
- *pooling*
- *convolution*
- *upsampling*
- *downsampling*

In backward pass we do

- *backward MLP*
- *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 and update all weights as

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \hat{R}$$

Other Forms of Convolution

The convolution operation we considered in this chapter is often called

2D Convolution

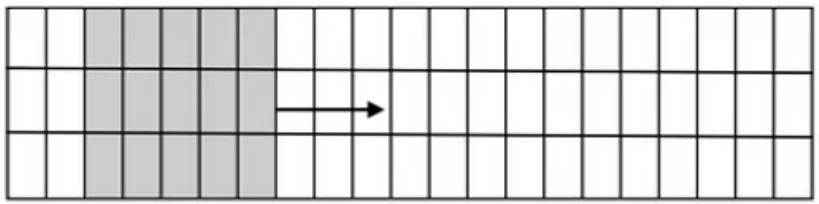
because it screens input in a 2D fashion, i.e., left-to-right and up-to-down

---

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

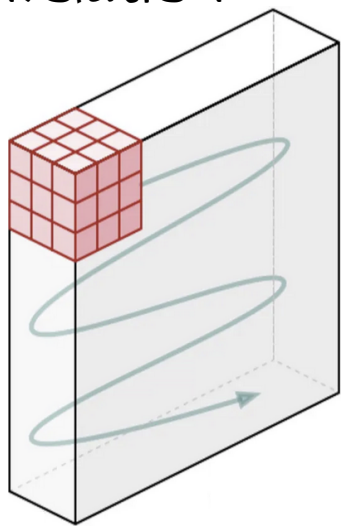
- *we could have 1D convolutions*
  - ↳ *Filter only slides in one direction, i.e., left-to-right or up-to-down*
  - ↳ *This is useful with audio data where we slide the filter over time*
- *we could also have 3D convolutions*
  - ↳ *Filter slides in three direction, i.e., left-to-right, up-to-down and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

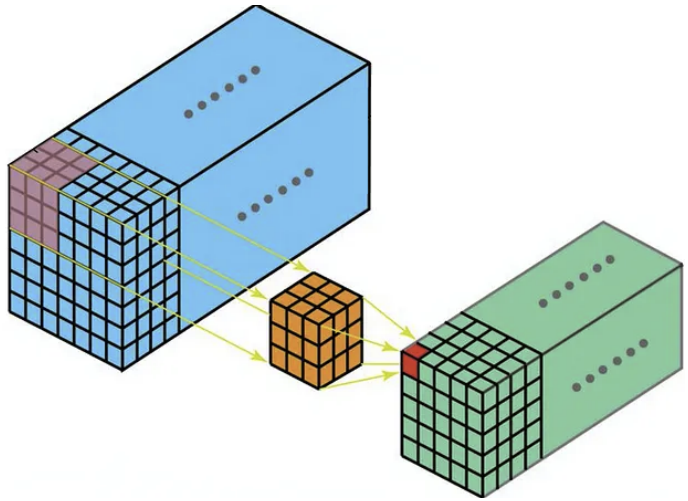
Other Forms of Convolution



1D Convolution

2D Convolution





3D Convolution

You might even deal with some other forms

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

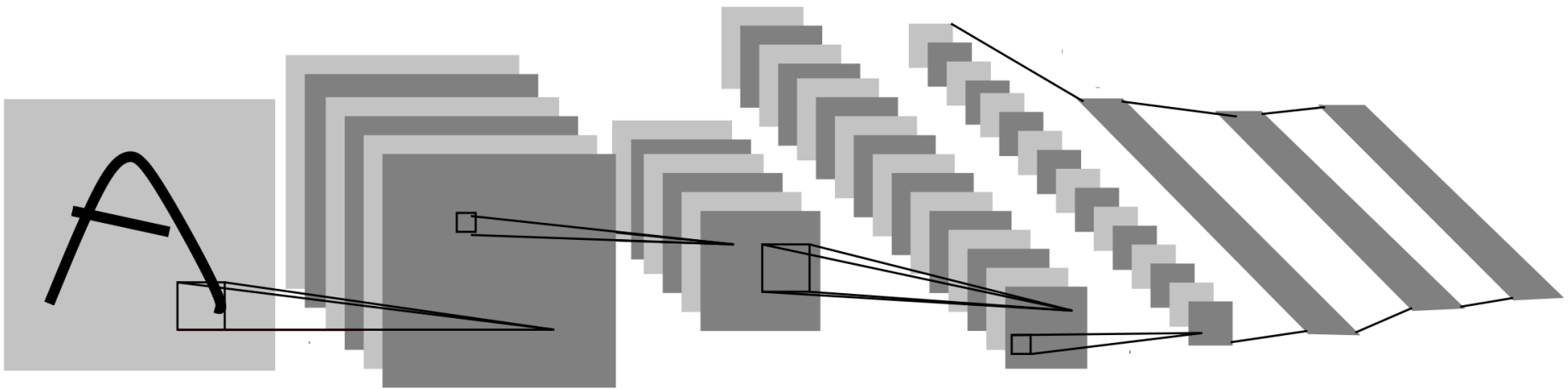


UNIVERSITY OF
TORONTO

Depth and Vanishing Gradient

A Bit of History: *LeNet*

CNNs was first trained via gradient-based algorithms by *Yan LeCun* and his team: *they could develop backpropagation through CNNs and hence they were able to efficiently implement it*[1]



[1] Check out their paper at this link! The diagram is taken from the the paper

ILSVRC: *ImageNet Large Scale Visual Recognition Challenge*

The project ImageNet started a competition in 2010: *it introduced a dataset of 1.2 million images from 1000 different labels. The proposed models are validated by a test dataset. The winner is the model with minimal classification error*

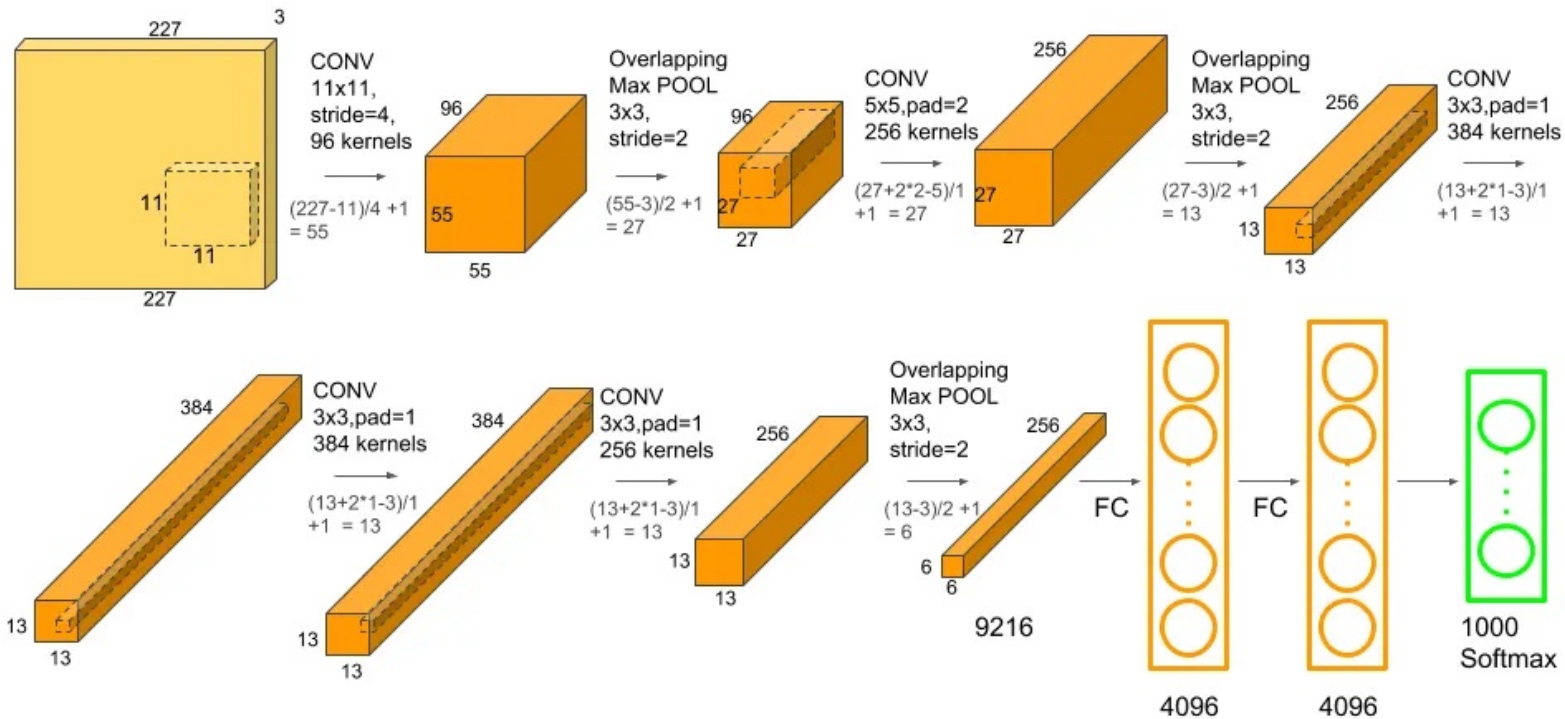
The winners in 2010 and 2011 used basic parameterized models!

- *2010: Team NEC-UIUC*
- *2011: Team XRCE*

In 2012, Supervision from U of T trained a deep CNN and won ILSVRC

First Deep Winner: *AlexNet*

Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton proposed *AlexNet which could greatly reduce the classification error*[2]



[2] Check it out in their paper!

VGG Architectures

Visual Geometry Group at Oxford University developed deeper CNNs: *a class of architectures*[3]

- *VGG-11*
- *VGG-13*
- *VGG-16*
- *VGG-19*

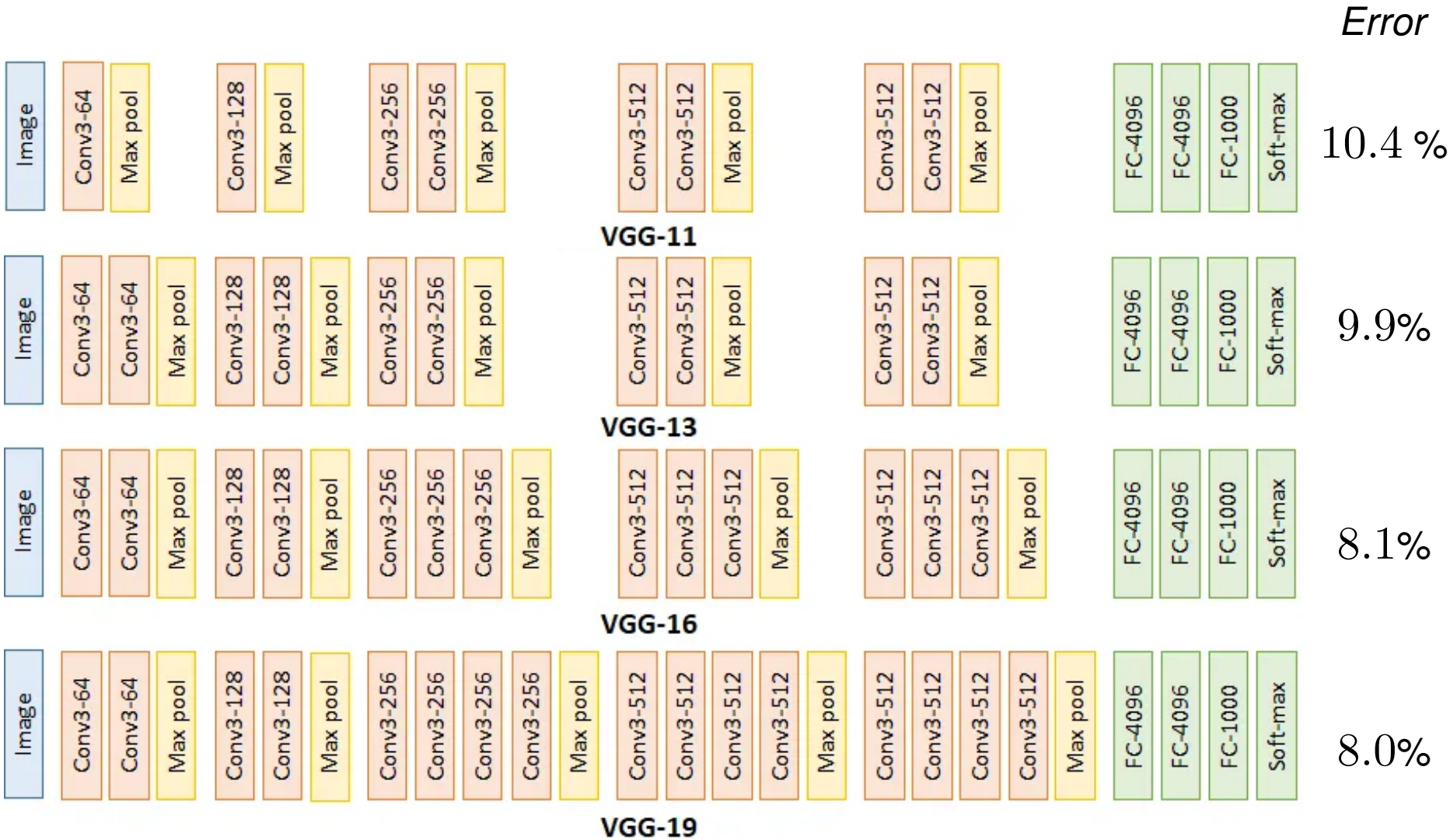
They won in ILSVRC in multiple tasks, but reported an interesting finding as well

Saturation Behaviour in Depth

As we go deeper, the accuracy gets higher notably up to VGG-16; however, VGG-19 can only give marginal improvements!

[3] Check VGG architectures out in their paper

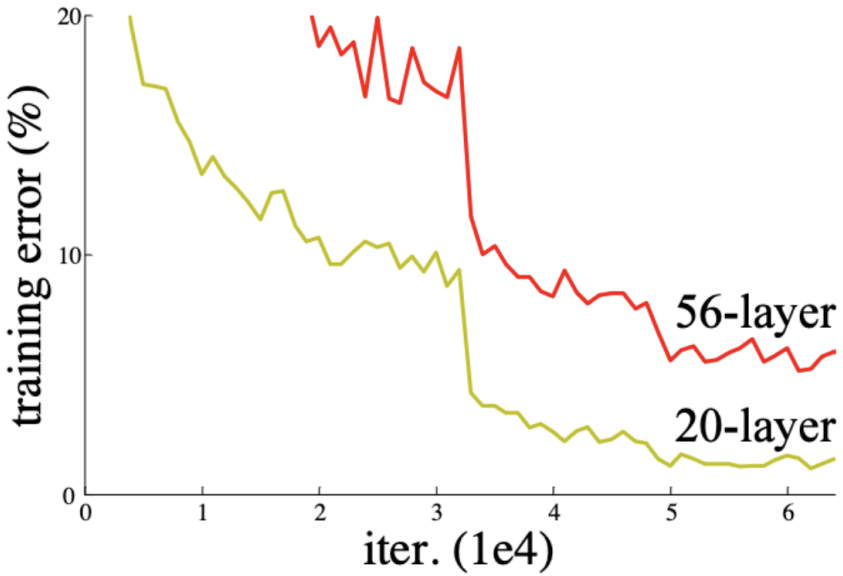
VGG Architectures

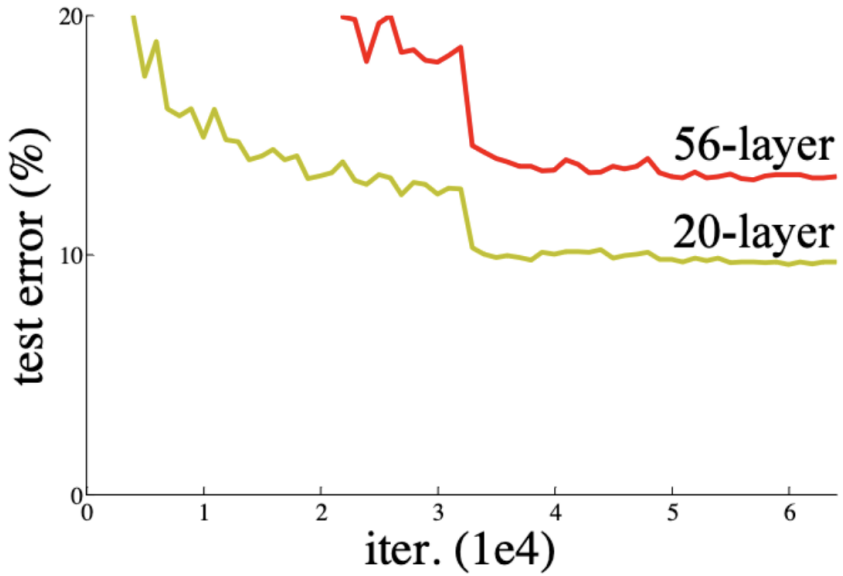


A Hurdle in Going Deeper

Later studies showed that *much deeper CNNs start to perform worse!*

? *Initial guess was overfitting ⇝ ruled out by Microsoft Research Lab*[4]



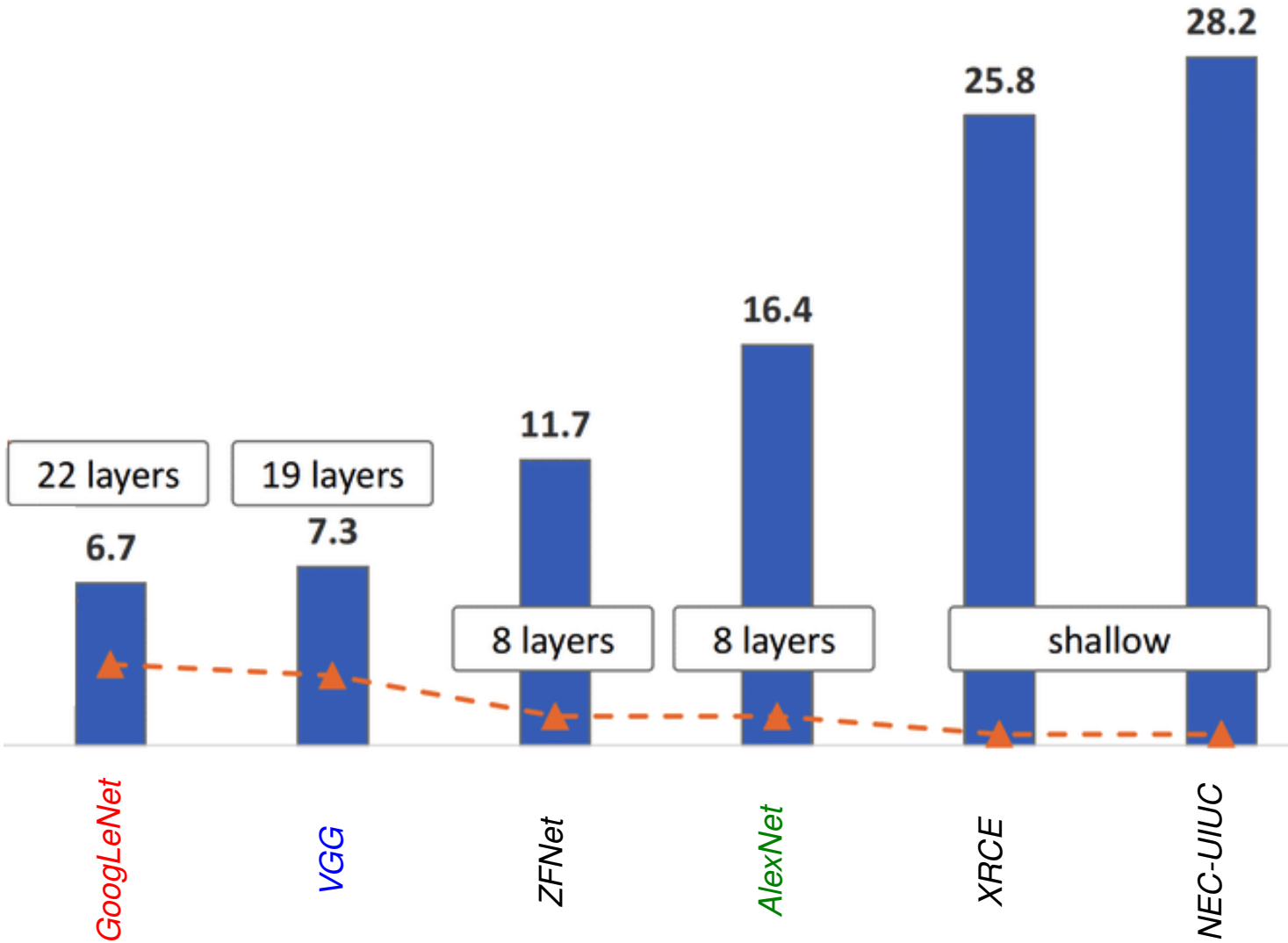


! *If it's coming from overfitting we should see better training and worst test risk!*

↳ *People started to blame the vanishing gradient behaviour of deep NNs*

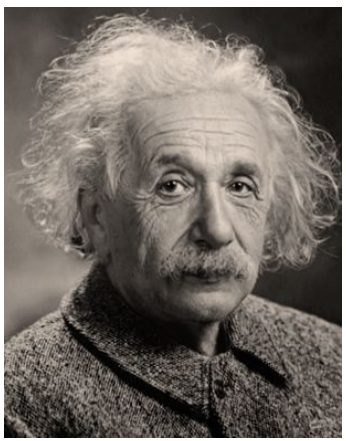
[4] Check it in this paper from which the figure is taken

Depth vs Accuracy: *ILSVRC Winners till 2014*



Problem of Depth

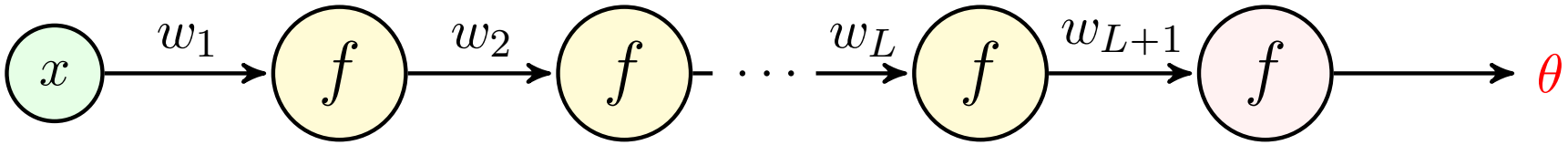
+ *But, what is the problem of vanishing gradient?*
– This is a general behaviour in deep NNs: *as we go deeper, the gradients determined by backpropagation at initial layers get smaller and smaller, such that at some point they stop getting updated*
+ *How does it come?*
– Let's see it! But we follow Albert Einstein advice!



"Everything should be made as simple as possible, but not simpler!"

Vanishing Gradients: *Simple Example*

Consider the following dummy FNN: *an FNN has a single scalar input*

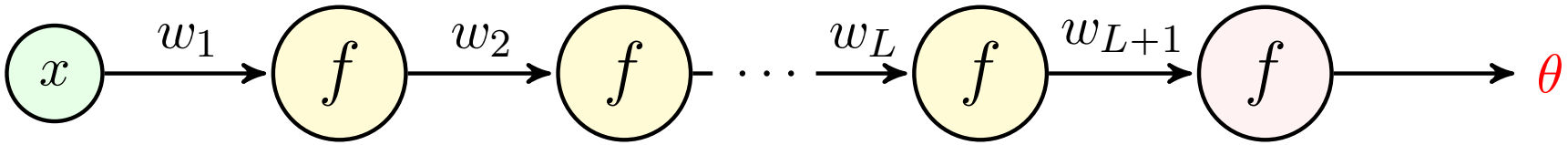


Let's write forward pass

- $y_1 = f(w_1 x)$
- $y_2 = f(w_2 y_1)$

  $\cdots$
- $y_L = f(w_L y_{L-1})$
- $\theta = f(w_{L+1} y_L) \rightsquigarrow \hat{R} = \mathcal{L}(\theta, v)$

Vanishing Gradients: *Simple Example*



Now, we go for backward pass: we start with $\overleftarrow{\theta} = \mathrm{d}\hat{R}/\mathrm{d}\theta$ *and go backward*

- $\theta = f\left(w_{L+1} y_L\right)$

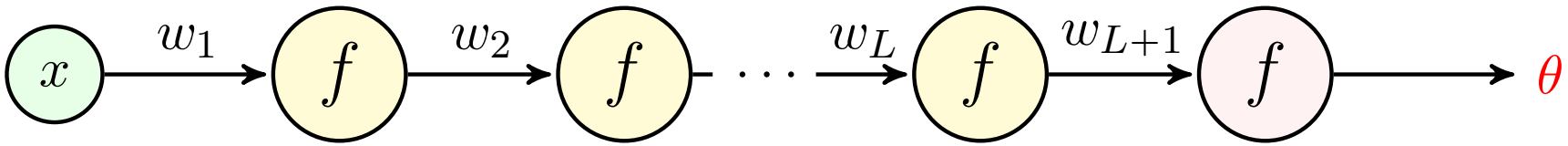
$$\overleftarrow{y}_L = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}y_L} = \overleftarrow{\theta}\, w_{L+1} \dot{f}\left(w_{L+1} y_L\right)$$

- $y_L = f\left(w_L y_{L-1}\right)$

$$\overleftarrow{y}_{L-1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_{L-1}} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_L}\frac{\mathrm{d}y_L}{\mathrm{d}y_{L-1}} = \overleftarrow{y}_L w_L \dot{f}\left(w_L y_{L-1}\right)$$

$$= \overleftarrow{\theta}\, w_{L+1} w_L \dot{f}\left(w_L y_{L-1}\right) \dot{f}\left(w_{L+1} y_L\right)$$

Vanishing Gradients: *Simple Example*



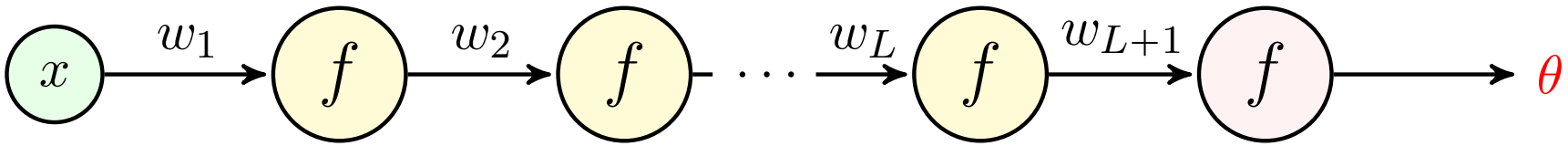
As we keep on going backward, the multiplication terms expand

- $y_2 = f\left(w_2 y_1\right)$

$$\overleftarrow{y}_1 = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_2}\frac{\mathrm{d}y_2}{\mathrm{d}y_1} = \overleftarrow{y_2} w_2 \dot{f}\left(w_2 y_1\right)$$

$$= \overleftarrow{\theta} \prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)$$

Vanishing Gradients: *Simple Example*



Now, let's compute derivative of loss with respect to the first weight w_1

- $y_1 = f(w_1 x)$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \frac{\mathrm{d}\hat{R}}{\mathrm{d}y_1}\frac{\mathrm{d}y_1}{\mathrm{d}w_1} = \overleftarrow{y_1} x \dot{f}(w_1 x)$$

$$= \overleftarrow{\theta}\, x \dot{f}(w_1 x) \prod_{\ell=2}^{L+1} w_\ell \dot{f}(w_\ell y_{\ell-1})$$

Vanishing Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta}\ \underbrace{x\dot{f}\left(w_1 x\right)\prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in Backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a sigmoid activation and all weight are smaller than 1*
  - ↳ *Note that for sigmoid* $\dot{f}\left(x\right) < 1$ *for any* x
  - ↳ *There is* $a < 1$ *that all derivatives are smaller than, e.g.,* $a = 1 - 10^{-10}$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta} x\dot{f}\left(w_1 x\right)\prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right) < \overleftarrow{\theta} x a^{2L+1}$$

Exploding Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta}\ \underbrace{x\dot{f}\left(w_1 x\right)\prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in Backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a sigmoid activation and all weight are smaller than 1*

$$\lim_{L\uparrow\infty} \frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = 0$$

↳ *Backpropagation accumulation concentrates at zero!*

Gradient with respect to first layer vanishes as the network gets too deep

Exploding Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta}\ \underbrace{x\dot{f}\left(w_1 x\right)\prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in Backpropagation}}$$

Now, consider the following cases

- ***Case II:*** *We have a ReLU activation and all weight are larger than 1*
  - ↳ *Assume* $x > 0$*; then,* $\dot{f}\left(w_\ell y_{\ell-1}\right) = 1$ *since all the sequence is positive*
  - ↳ *There is* $a > 1$ *that all weights are larger than, e.g.,* $a = 1 + 10^{-10}$

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta}\, x\dot{f}\left(w_1 x\right)\prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right) > \overleftarrow{\theta}\, x a^L$$

Vanishing Gradients: *Simple Example*

$$\frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} = \overleftarrow{\theta}\ \underbrace{x\dot{f}\left(w_1 x\right)\prod_{\ell=2}^{L+1} w_\ell \dot{f}\left(w_\ell y_{\ell-1}\right)}_{\text{accumulated in Backpropagation}}$$

Now, consider the following cases

- ***Case I:*** *We have a ReLU activation and all weight are larger than 1*

$$\lim_{L\uparrow\infty} \frac{\mathrm{d}\hat{R}}{\mathrm{d}w_1} \to \infty$$

↳ *Backpropagation accumulation explodes!*

Gradient with respect to first layer explodes as the network gets too deep

Exploding-Vanishing Gradients: *Summary*

Moral of Story

As NNs get very deep, gradients of initial layers can get extremely small or large

- *Weights and derivative of activation function are key deciders*
  - ↳ *We need them to mostly around 1*
- *The vanishing occurs more frequently*
  - ↳ *Many weights in NNs get often small*
  - ↳ *Gradients of activation are also often bounded*

? *How can we resolve this issue?*

There are various solutions and we look at two key ones

1. Normalization
2. Residual Learning



UNIVERSITY OF
TORONTO

Further Read

Further Read

Read More on These Topics

- Goodfellow, et al., *Deep Learning*
  - ↳ Chapter 9
- Bishop & Bishop, *Deep Learning: Foundations and Concepts*
  - ↳ Chapter 10

Nice to Check

1. *Very Deep Convolutional Networks for Large-Scale Image Recognition*
   - ↳ Simonyan and Zisserman (2014)
2. *Deep Residual Learning for Image Recognition*
   - ↳ K. He et al. (2015)