Convolutional Neural Networks 3

Pavlos Protopapas

Outline

1. Regularization for CNN
2. BackProp of MaxPooling layer
3. Layers Receptive Field and dilated convolutions
4. Weights and feature maps visualization

Outline

1. **Regularization for CNN**
2. BackProp of MaxPooling layer
3. Layers Receptive Field and dilated convolutions
4. Weights and feature maps visualization

Regularization for CNN

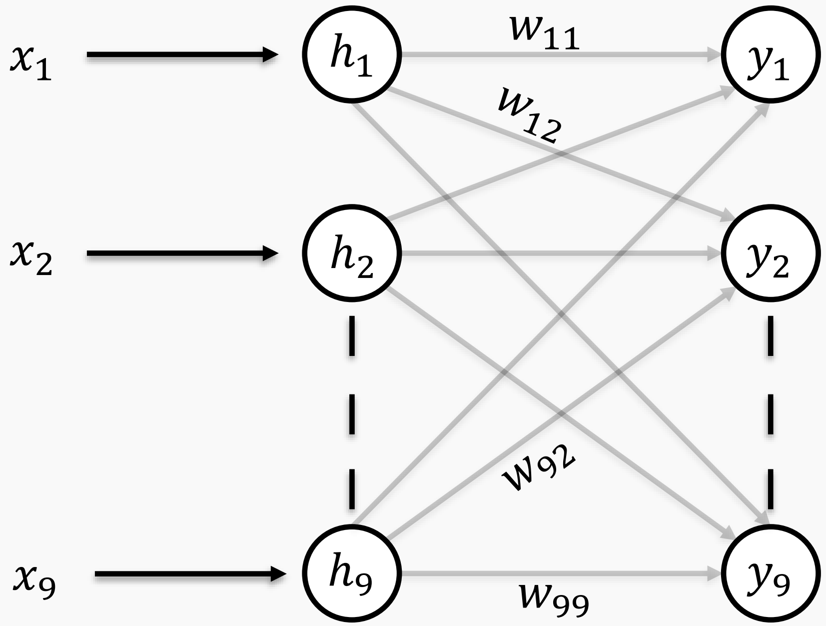
- L2 and L1 work the same way as in FFNN
- Data Augmentation is the same
- Early Stopping same as in FFNN

- Dropout is slightly different – not the same effect as dropout with FFNN.
    - Dropout in CNN still allows the weights in a kernel to be trained.
    - The name is misleading!
    - The effect of dropout on convolutional layers amounts to multiplying Bernoulli noise into the feature maps of the network.

So, if you try adding dropout after a convolutional layer and get bad results, don't be disappointed! There doesn't appear that there is a good reason it *should* provide good results.

Dropout

For an FFNN:



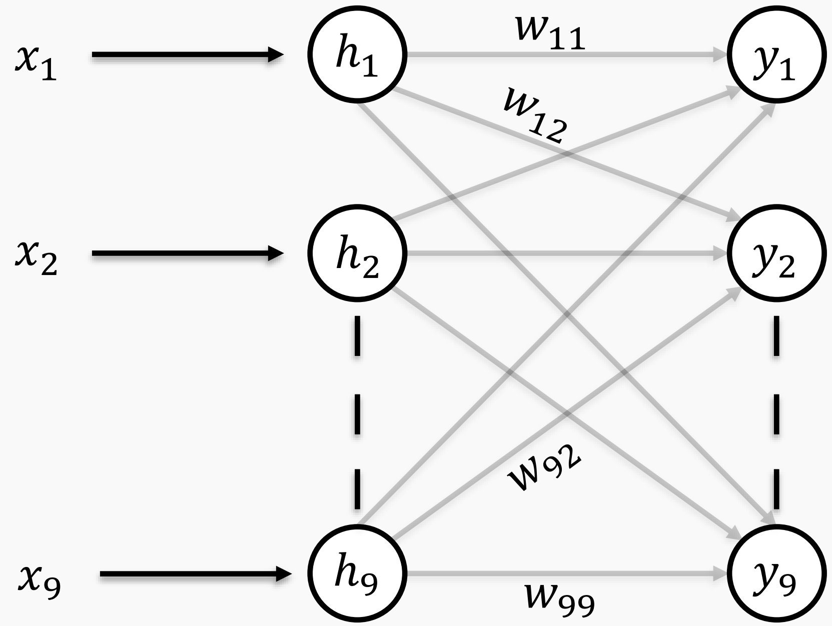
We can write this as:

$$y = Wh$$

$$y = \begin{bmatrix} w_{11} & \cdots & w_{91} \\ w_{12} & \cdots & w_{92} \\ \vdots & \ddots & \vdots \\ w_{19} & \cdots & w_{99} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}$$

Dropout

For an FFNN:



$$y = \begin{bmatrix} w_{11} & \cdots & w_{91} \\ w_{12} & \cdots & w_{92} \\ \vdots & \ddots & \vdots \\ w_{19} & \cdots & w_{99} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}$$

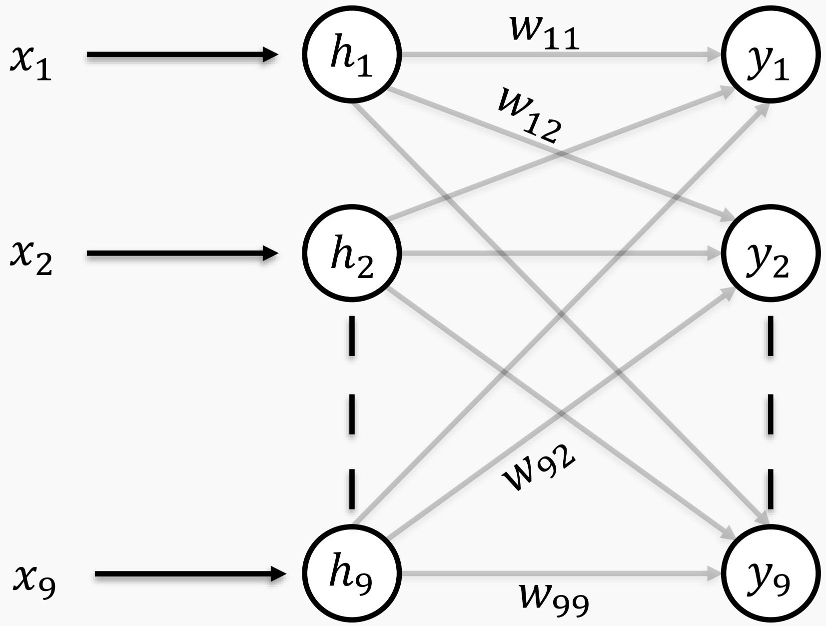
Let's introduce a Boolean diagonal matrix

$$y = \begin{bmatrix} w_{11} & \cdots & w_{91} \\ w_{12} & \cdots & w_{92} \\ \vdots & \ddots & \vdots \\ w_{19} & \cdots & w_{99} \end{bmatrix} \begin{bmatrix} r_1 & \cdots & 0 \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r_9 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}$$

$$y = \begin{bmatrix} r_1 w_{11} & r_2 w_{21} & \cdots & r_9 w_{91} \\ r_1 w_{12} & r_2 w_{22} & \cdots & r_9 w_{92} \\ \vdots & \vdots & \ddots & \vdots \\ r_1 w_{19} & r_2 w_{29} & \cdots & r_9 w_{99} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}$$

Dropout

For an FFNN:



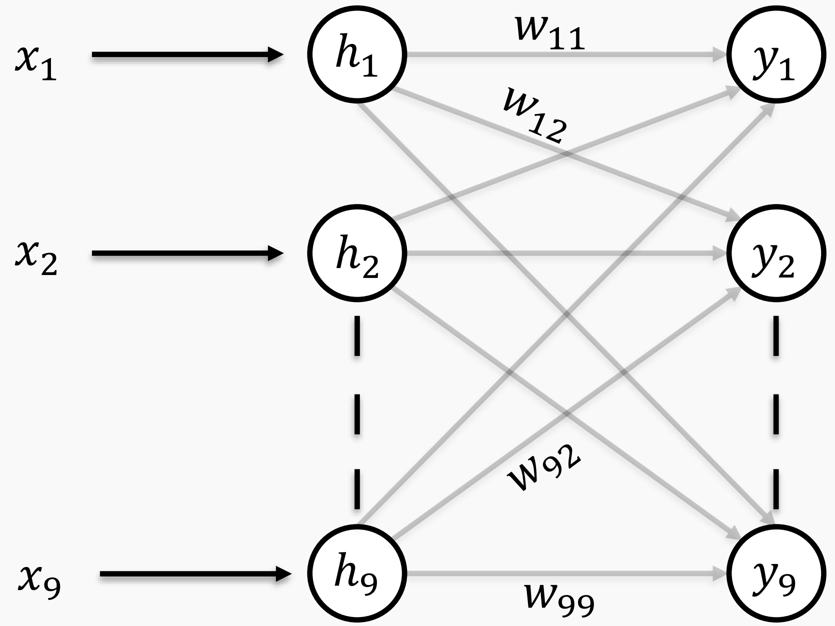
$$y = \begin{bmatrix} r_1 w_{11} & r_2 w_{21} & \cdots & r_9 w_{91} \\ r_1 w_{12} & r_2 w_{22} & \cdots & r_9 w_{92} \\ \vdots & \vdots & \ddots & \vdots \\ r_1 w_{19} & r_2 w_{29} & \cdots & r_9 w_{99} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}$$

Let's suppose, we take r_1 = 1 and r_i = 0
Where $i = 2, 3, \dots, 8, 9$

$$y = \begin{bmatrix} w_{11} & 0 & \cdots & 0 \\ w_{12} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ w_{19} & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}$$

Dropout

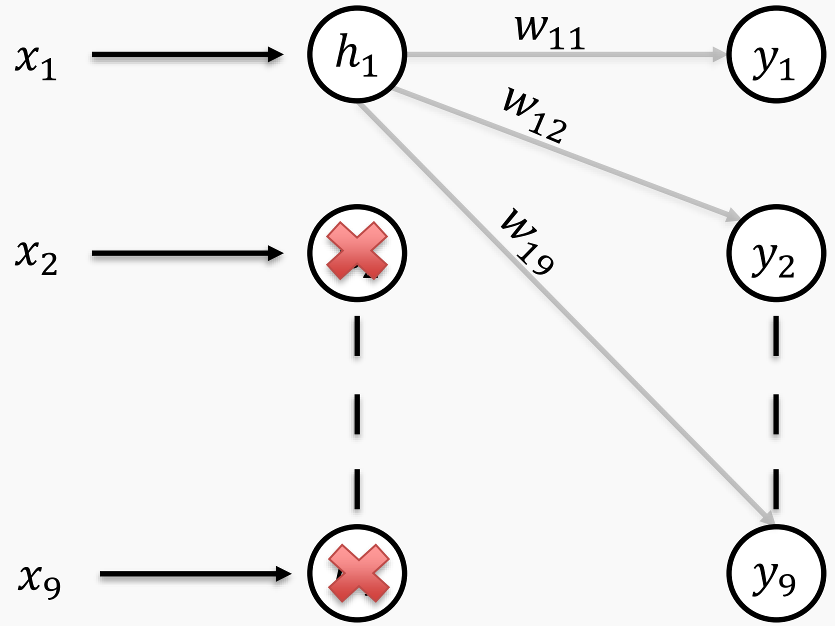
For an FFNN:



$$y = \begin{bmatrix} w_{11} & 0 & \cdots & 0 \\ w_{12} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ w_{19} & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}$$

Dropout

For an FFNN:



$$y = \begin{bmatrix} w_{11} & 0 & \cdots & 0 \\ w_{12} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ w_{19} & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}$$

Dropout

Now before understanding how dropout looks like in CNNs,

Let's understand how Convolution operation can be replaced with a matrix multiplication.

Suppose you have to do the following convolution operation:

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} * \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}$$

Dropout

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \quad * \quad \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}$$

This operation can be represented as:

$$\begin{bmatrix} w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

Dropout

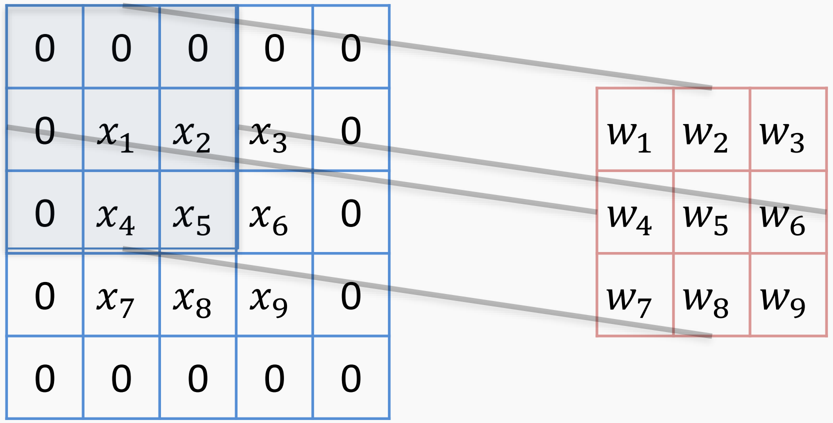
Now let's understand how dropout looks like for a CNN:

We have our padded **image** for simplicity and our **kernel**.

0	0	0	0	0
0	x_1	x_2	x_3	0
0	x_4	x_5	x_6	0
0	x_7	x_8	x_9	0
0	0	0	0	0

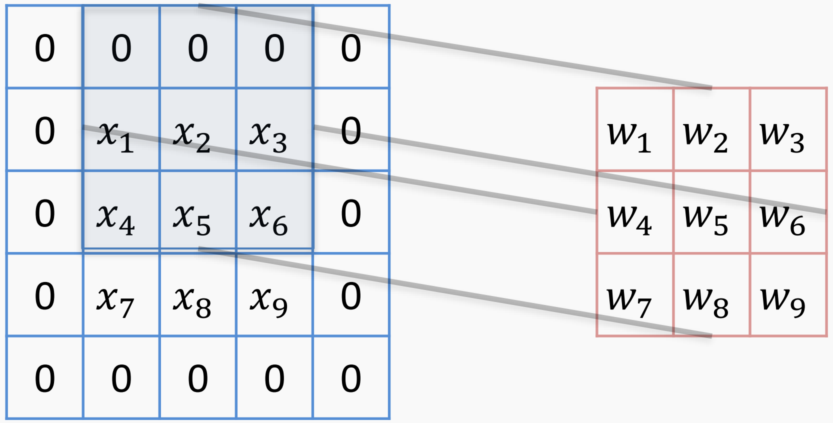
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Dropout



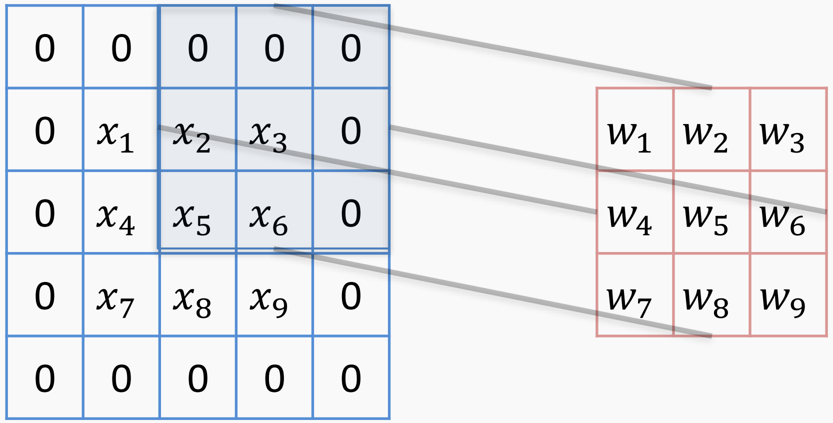
$$\begin{bmatrix} w_5 & w_6 & 0 & w_8 & w_9 & 0 & 0 & 0 & 0 \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

Dropout



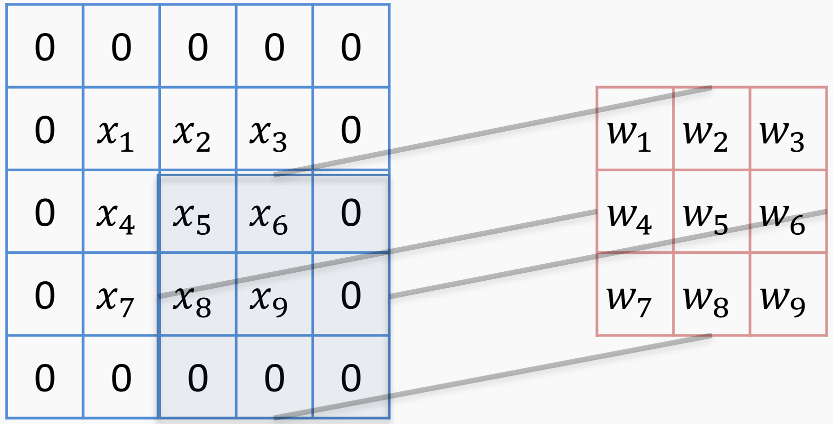
$$\begin{bmatrix} w_5 & w_6 & 0 & w_8 & w_9 & 0 & 0 & 0 & 0 \\ w_4 & w_5 & w_6 & w_7 & w_8 & w_9 & 0 & 0 & 0 \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

Dropout



$$\begin{bmatrix} w_5 & w_6 & 0 & w_8 & w_9 & 0 & 0 & 0 & 0 \\ w_4 & w_5 & w_6 & w_7 & w_8 & w_9 & 0 & 0 & 0 \\ 0 & w_4 & w_5 & 0 & w_7 & w_8 & 0 & 0 & 0 \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

Dropout



$$
\begin{bmatrix}
w_5 & w_6 & 0 & w_8 & w_9 & 0 & 0 & 0 & 0 \\
w_4 & w_5 & w_6 & w_7 & w_8 & w_9 & 0 & 0 & 0 \\
0 & w_4 & w_5 & 0 & w_7 & w_8 & 0 & 0 & 0 \\
w_2 & w_3 & 0 & w_5 & w_6 & 0 & w_8 & w_9 & 0 \\
w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 & w_9 \\
0 & w_1 & w_2 & 0 & w_4 & w_5 & 0 & w_7 & w_8 \\
0 & 0 & 0 & w_2 & w_3 & 0 & w_5 & w_6 & 0 \\
0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \\
0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_4 & w_5
\end{bmatrix}
\begin{bmatrix}
x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9
\end{bmatrix}
$$

Dropout

$$\begin{bmatrix} w_5 & w_6 & 0 & w_8 & w_9 & 0 & 0 & 0 & 0 \\ w_4 & w_5 & w_6 & w_7 & w_8 & w_9 & 0 & 0 & 0 \\ 0 & w_4 & w_5 & 0 & w_7 & w_8 & 0 & 0 & 0 \\ w_2 & w_3 & 0 & w_5 & w_6 & 0 & w_8 & w_9 & 0 \\ w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 & w_9 \\ 0 & w_1 & w_2 & 0 & w_4 & w_5 & 0 & w_7 & w_8 \\ 0 & 0 & 0 & w_2 & w_3 & 0 & w_5 & w_6 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_4 & w_5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

Dropout

Let's introduce the diagonal Boolean matrix, R:

$$\begin{bmatrix} w_5 & w_6 & 0 & w_8 & w_9 & 0 & 0 & 0 & 0 \\ w_4 & w_5 & w_6 & w_7 & w_8 & w_9 & 0 & 0 & 0 \\ 0 & w_4 & w_5 & 0 & w_7 & w_8 & 0 & 0 & 0 \\ w_2 & w_3 & 0 & w_5 & w_6 & 0 & w_8 & w_9 & 0 \\ w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 & w_9 \\ 0 & w_1 & w_2 & 0 & w_4 & w_5 & 0 & w_7 & w_8 \\ 0 & 0 & 0 & w_2 & w_3 & 0 & w_5 & w_6 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_4 & w_5 \end{bmatrix} \begin{bmatrix} r_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & r_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & r_5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & r_6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & r_7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

Dropout

The result of the matrix multiplication will be:

$$\begin{bmatrix} r_1w_5 & r_2w_6 & 0 & r_4w_8 & r_5w_9 & 0 & 0 & 0 & 0 \\ r_1w_4 & r_2w_5 & r_3w_6 & r_4w_7 & r_5w_8 & r_6w_9 & 0 & 0 & 0 \\ 0 & r_2w_4 & r_3w_5 & 0 & r_5w_7 & r_6w_8 & 0 & 0 & 0 \\ r_1w_2 & r_2w_3 & 0 & r_4w_5 & r_5w_6 & 0 & r_7w_8 & r_8w_9 & 0 \\ r_1w_1 & r_2w_2 & r_3w_3 & r_4w_4 & r_5w_5 & r_6w_6 & r_7w_7 & r_8w_8 & r_9w_9 \\ 0 & r_2w_1 & r_3w_2 & 0 & r_5w_4 & r_6w_5 & 0 & r_8w_7 & r_9w_8 \\ 0 & 0 & 0 & r_4w_2 & r_5w_3 & 0 & r_7w_5 & r_8w_6 & 0 \\ 0 & 0 & 0 & r_4w_1 & r_5w_2 & r_6w_3 & r_7w_4 & r_8w_5 & r_9w_6 \\ 0 & 0 & 0 & 0 & r_5w_1 & r_6w_2 & 0 & r_8w_4 & r_9w_5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

Dropout

Let's look at all the terms with w_5 :

$$\begin{bmatrix} r_1w_5 & r_2w_6 & 0 & r_4w_8 & r_5w_9 & 0 & 0 & 0 & 0 \\ r_1w_4 & r_2w_5 & r_3w_6 & r_4w_7 & r_5w_8 & r_6w_9 & 0 & 0 & 0 \\ 0 & r_2w_4 & r_3w_5 & 0 & r_5w_7 & r_6w_8 & 0 & 0 & 0 \\ r_1w_2 & r_2w_3 & 0 & r_4w_5 & r_5w_6 & 0 & r_7w_8 & r_8w_9 & 0 \\ r_1w_1 & r_2w_2 & r_3w_3 & r_4w_4 & r_5w_5 & r_6w_6 & r_7w_7 & r_8w_8 & r_9w_9 \\ 0 & r_2w_1 & r_3w_2 & 0 & r_5w_4 & r_6w_5 & 0 & r_8w_7 & r_9w_8 \\ 0 & 0 & 0 & r_4w_2 & r_5w_3 & 0 & r_7w_5 & r_8w_6 & 0 \\ 0 & 0 & 0 & r_4w_1 & r_5w_2 & r_6w_3 & r_7w_4 & r_8w_5 & r_9w_6 \\ 0 & 0 & 0 & 0 & r_5w_1 & r_6w_2 & 0 & r_8w_4 & r_9w_5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

Regardless of our choice on which r to set to 0, w_5 will be updated in the backpropagation step.

Dropout

Let's look at all the terms with w_5 :

$$\begin{bmatrix} r_1w_5 & r_2w_6 & 0 & r_4w_8 & r_5w_9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_4w_2 & r_5w_3 & 0 & r_7w_5 & r_8w_6 & 0 \\ 0 & 0 & 0 & r_4w_1 & r_5w_2 & r_6w_3 & r_7w_4 & r_8w_5 & r_9w_6 \\ 0 & 0 & 0 & 0 & r_5w_1 & r_6w_2 & 0 & r_8w_4 & r_9w_5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

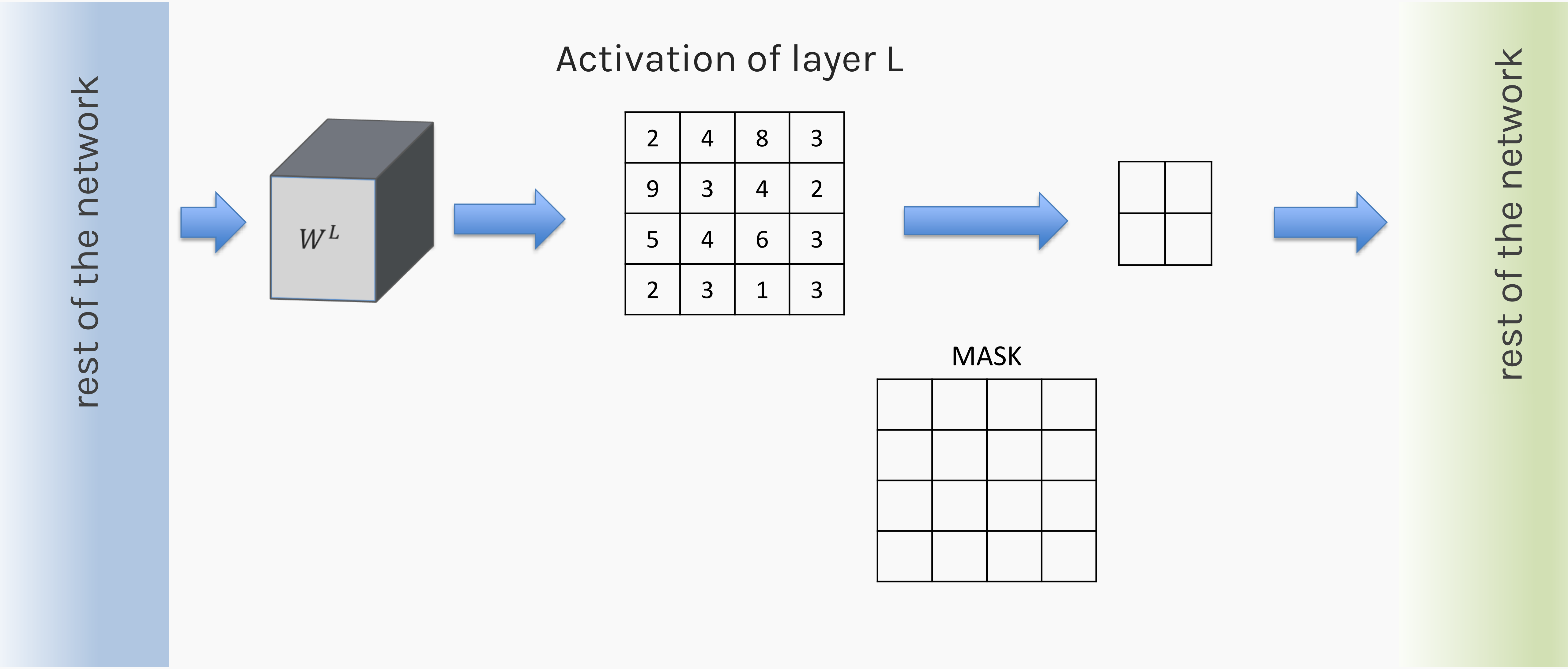
Dropout in CNNs does not turn out the way we expect it to.

Regardless of our choice on which r to set to 0, w_5 will be updated in the backpropagation step.

Outline

1. Regularization for CNN
2. **BackProp of MaxPooling layer**
3. Layers Receptive Field and dilated convolutions
4. Weights and feature maps visualization

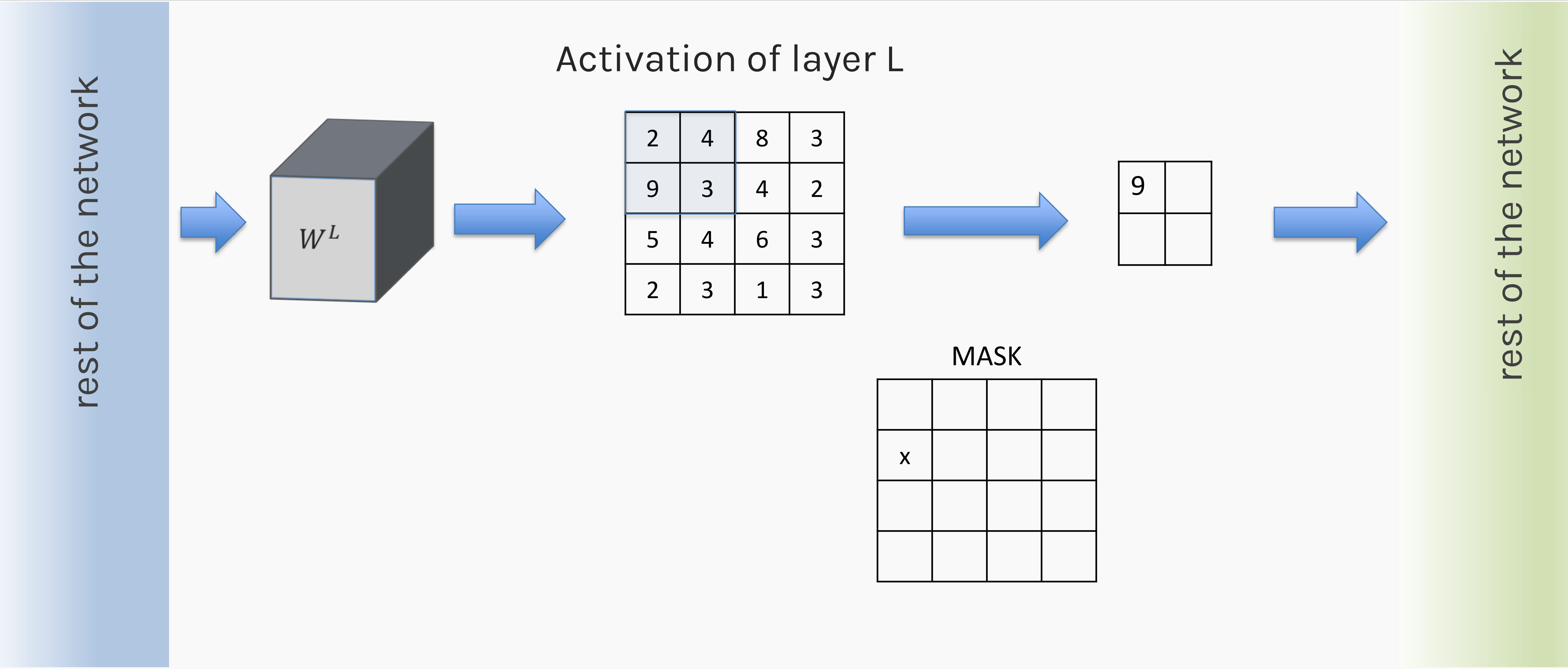
Backward propagation of Maximum Pooling Layer



2	4	8	3
9	3	4	2
5	4	6	3
2	3	1	3

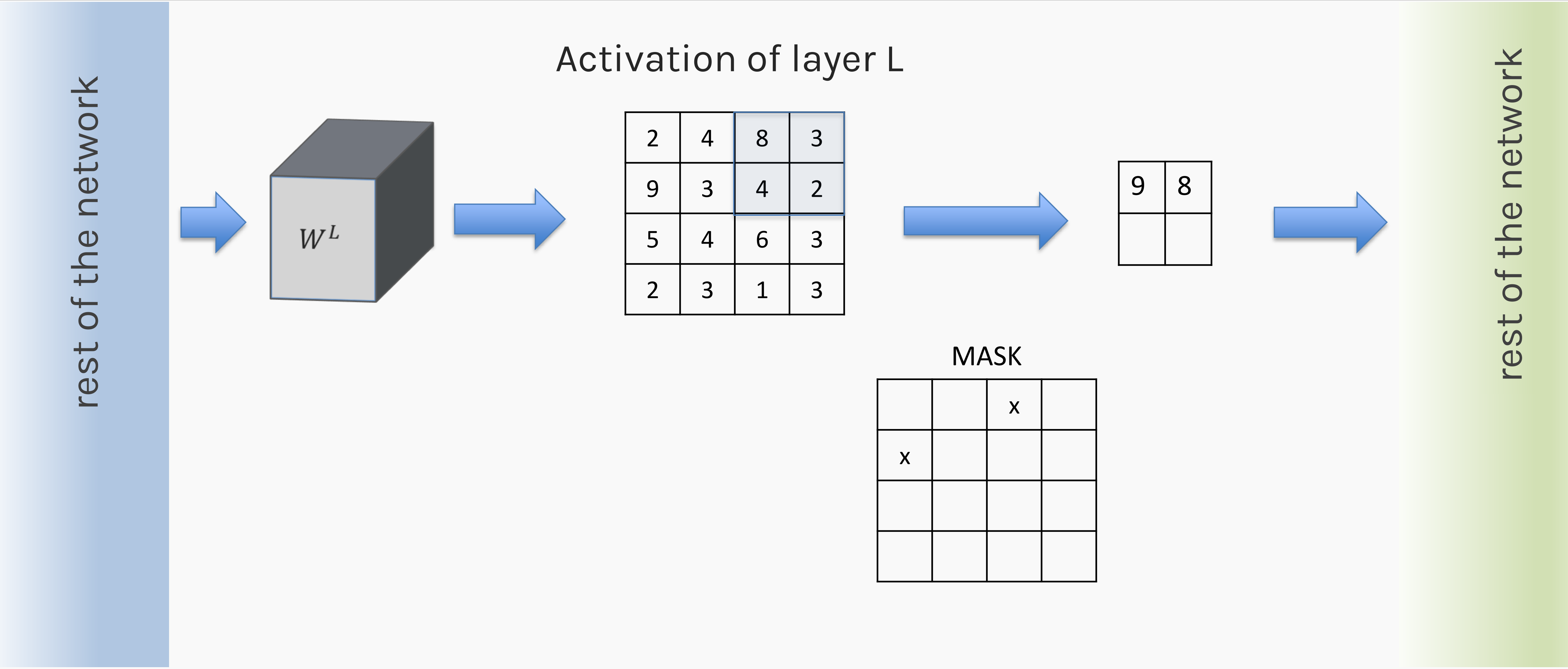
Forward mode, 2x2 stride 2x2

Backward propagation of Maximum Pooling Layer



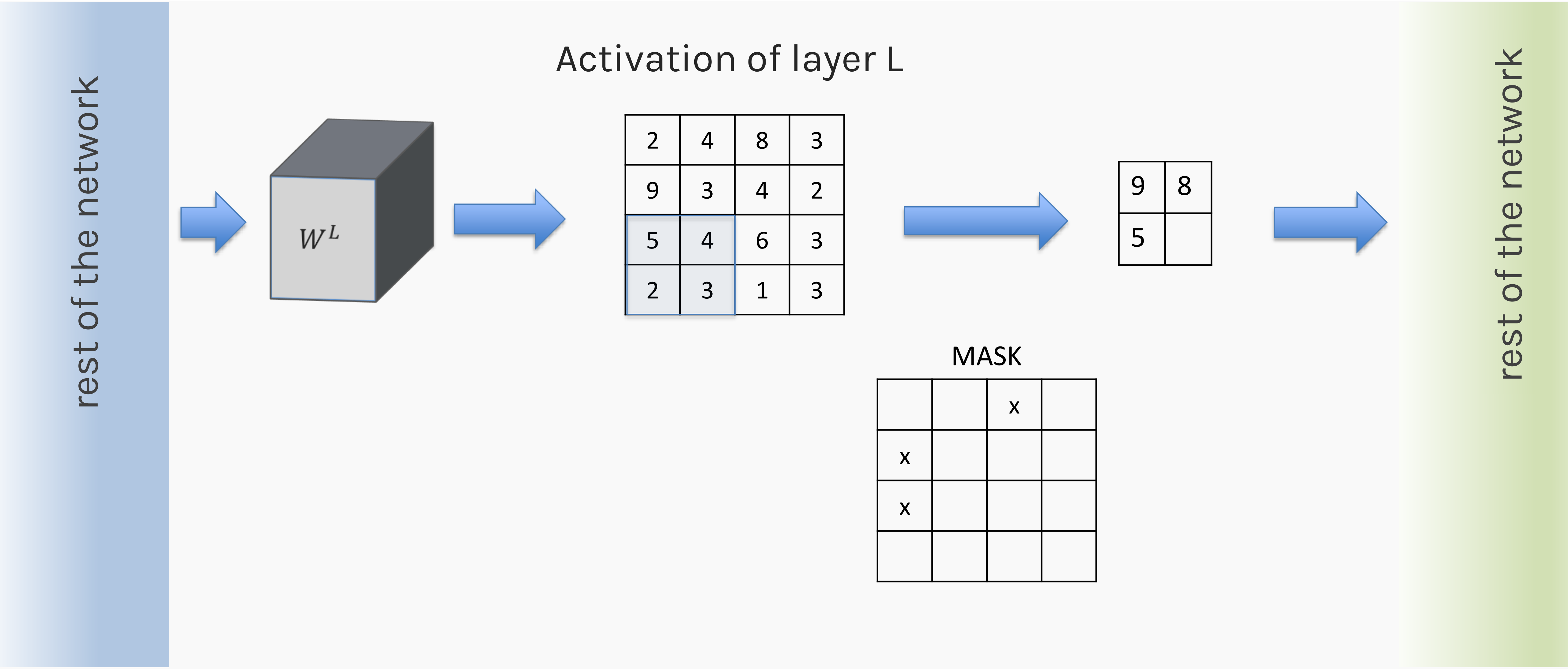
Forward mode, 2x2 stride 2x2

Backward propagation of Maximum Pooling Layer



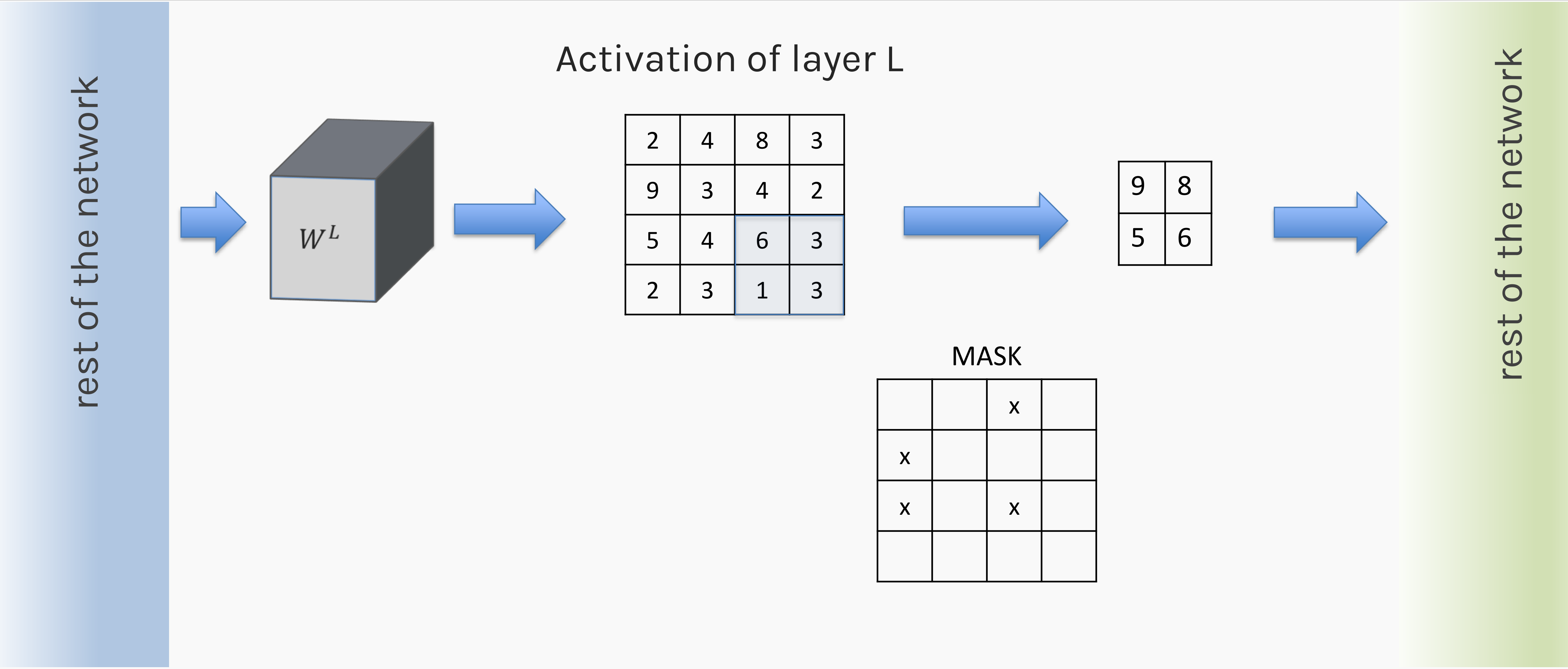
Forward mode, 2x2 stride 2x2

Backward propagation of Maximum Pooling Layer



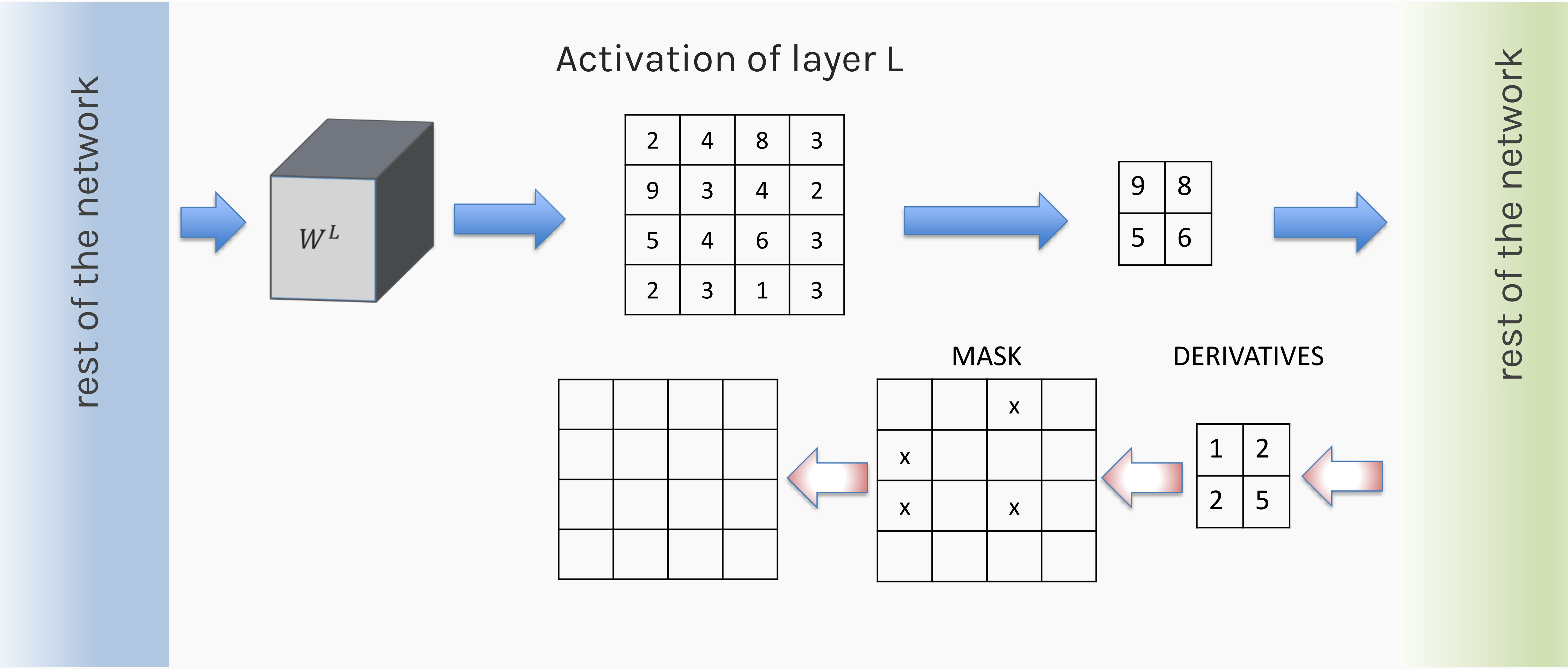
Forward mode, 2x2 stride 2x2

Backward propagation of Maximum Pooling Layer



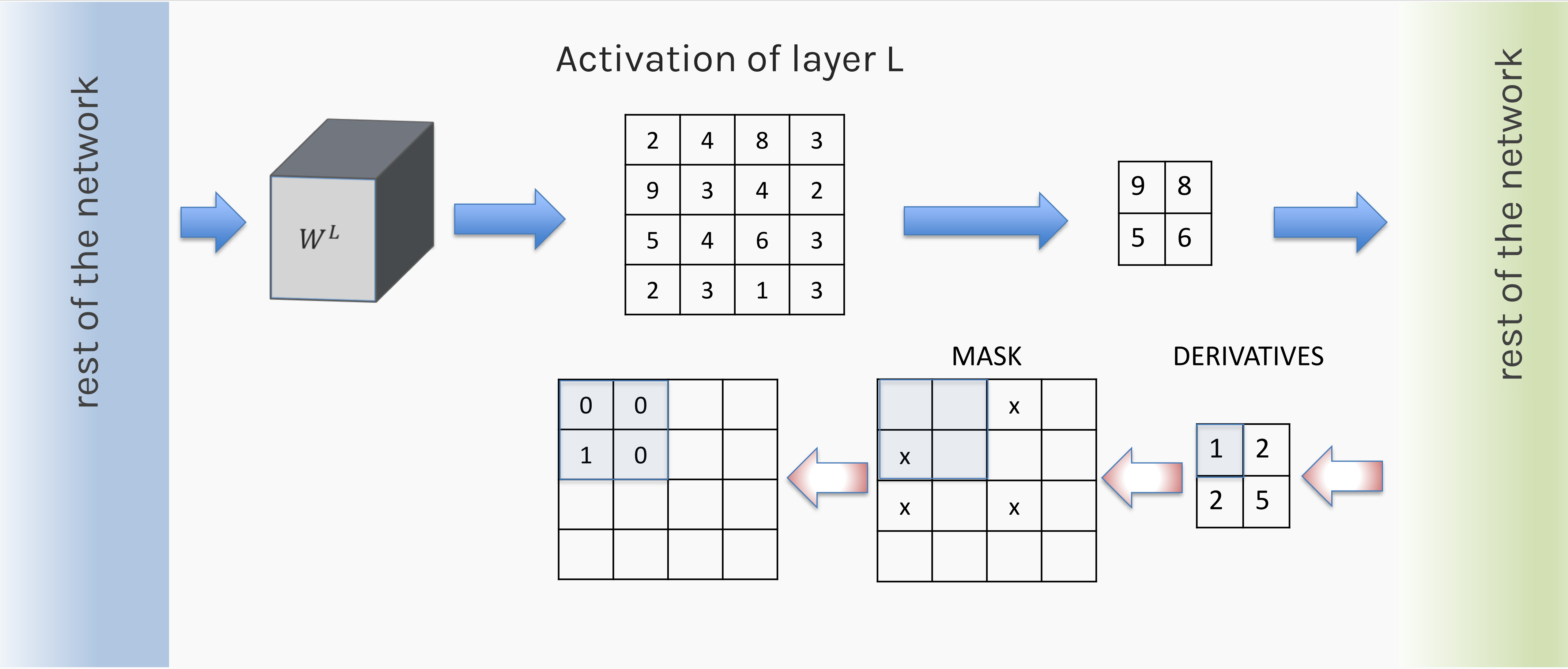
Forward mode, 2x2 stride 2x2

Backward propagation of Maximum Pooling Layer



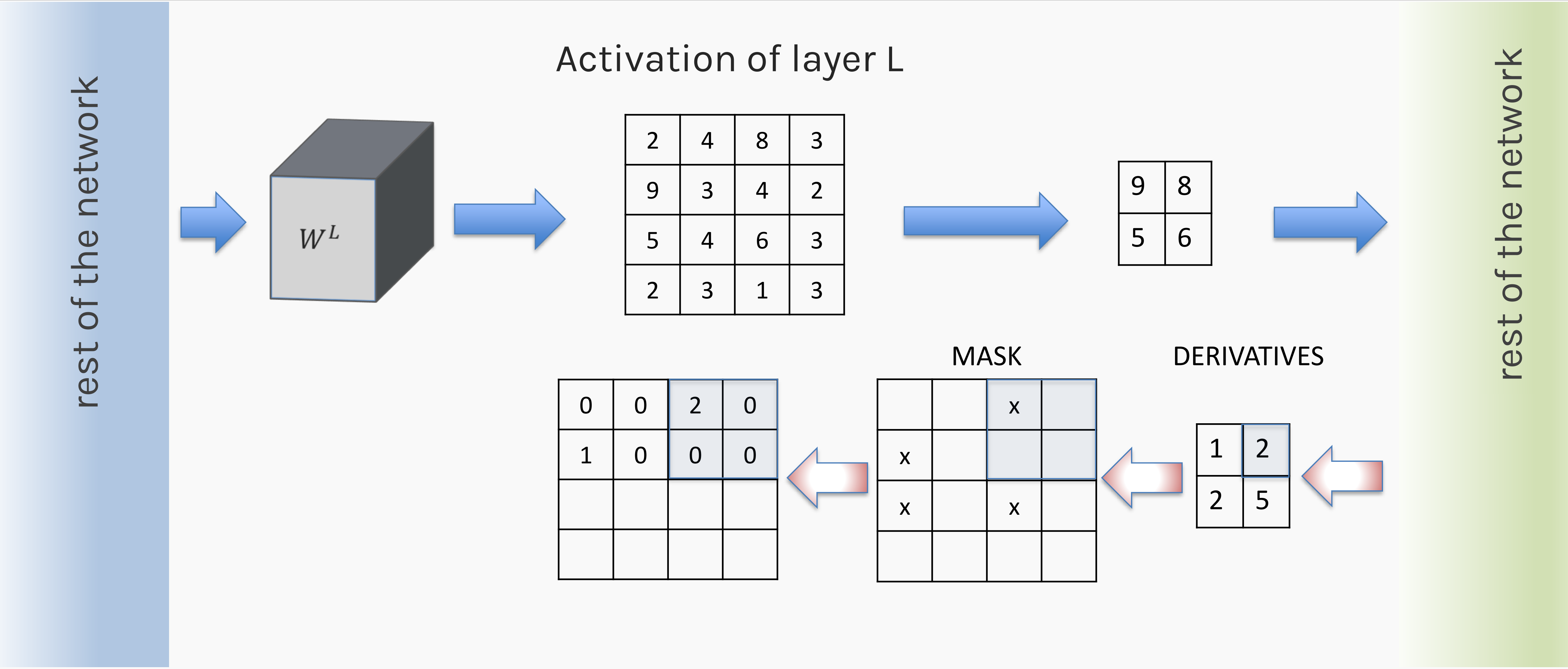
Reverse mode

Backward propagation of Maximum Pooling Layer



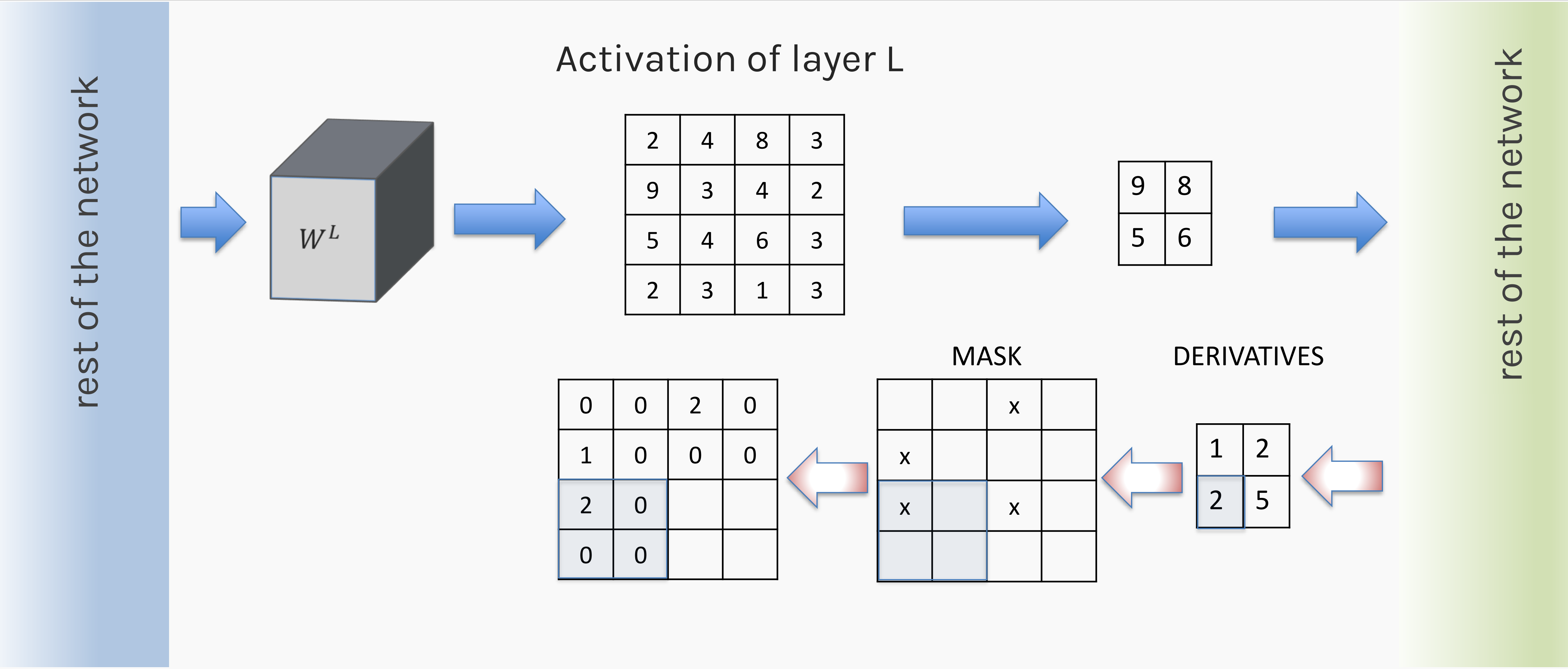
Reverse mode

Backward propagation of Maximum Pooling Layer



Reverse mode

Backward propagation of Maximum Pooling Layer

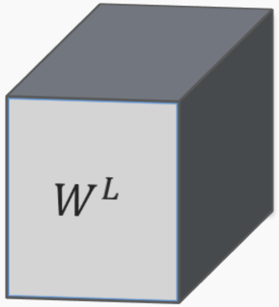


Reverse mode

Backward propagation of Maximum Pooling Layer

rest of the network

Activation of layer L



2	4	8	3
9	3	4	2
5	4	6	3
2	3	1	3

9	8
5	6

rest of the network

MASK

		x	
x			
x		x	

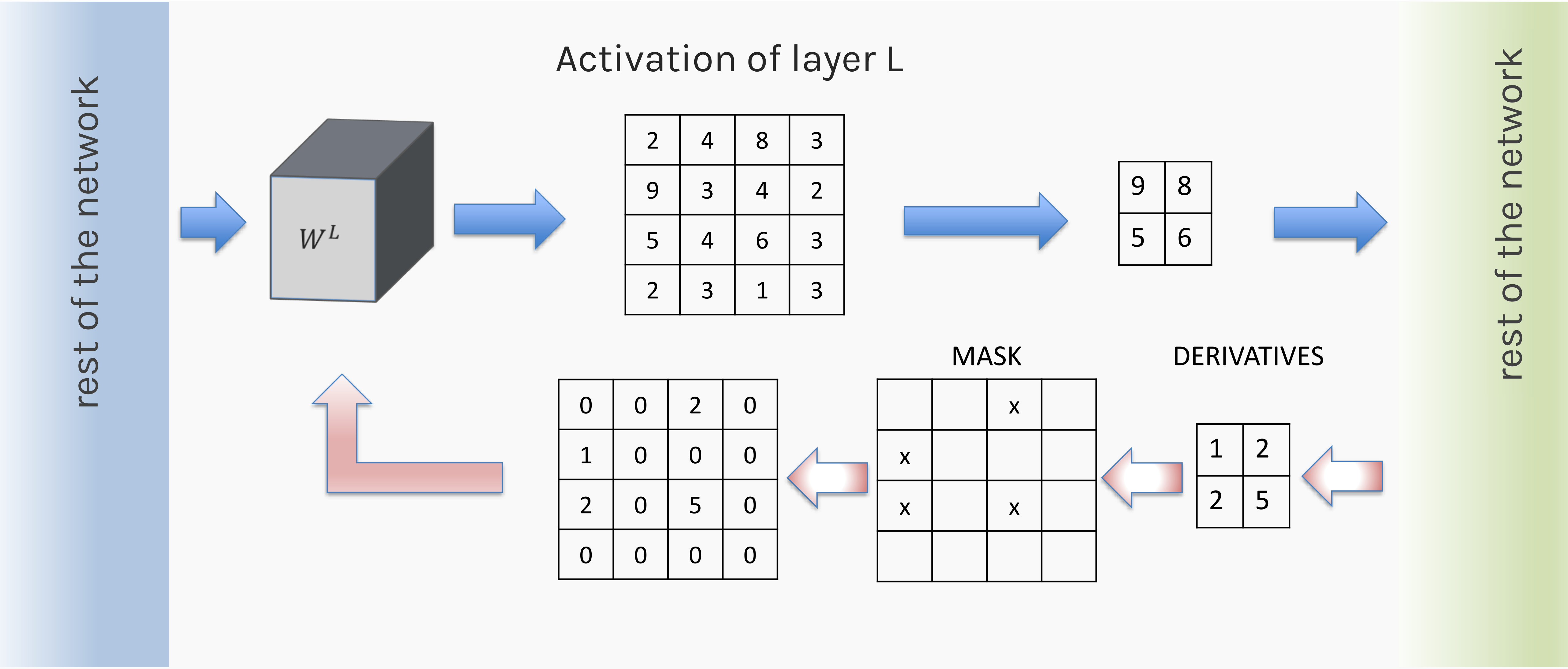
DERIVATIVES

1	2
2	5

0	0	2	0
1	0	0	0
2	0	5	0
0	0	0	0

Reverse mode

Backward propagation of Maximum Pooling Layer



Reverse mode

Outline

1. Regularization for CNN
2. BackProp of MaxPooling layer
3. **Layers Receptive Field and dilated convolutions**
4. Weights and feature maps visualization

Layers Receptive Field

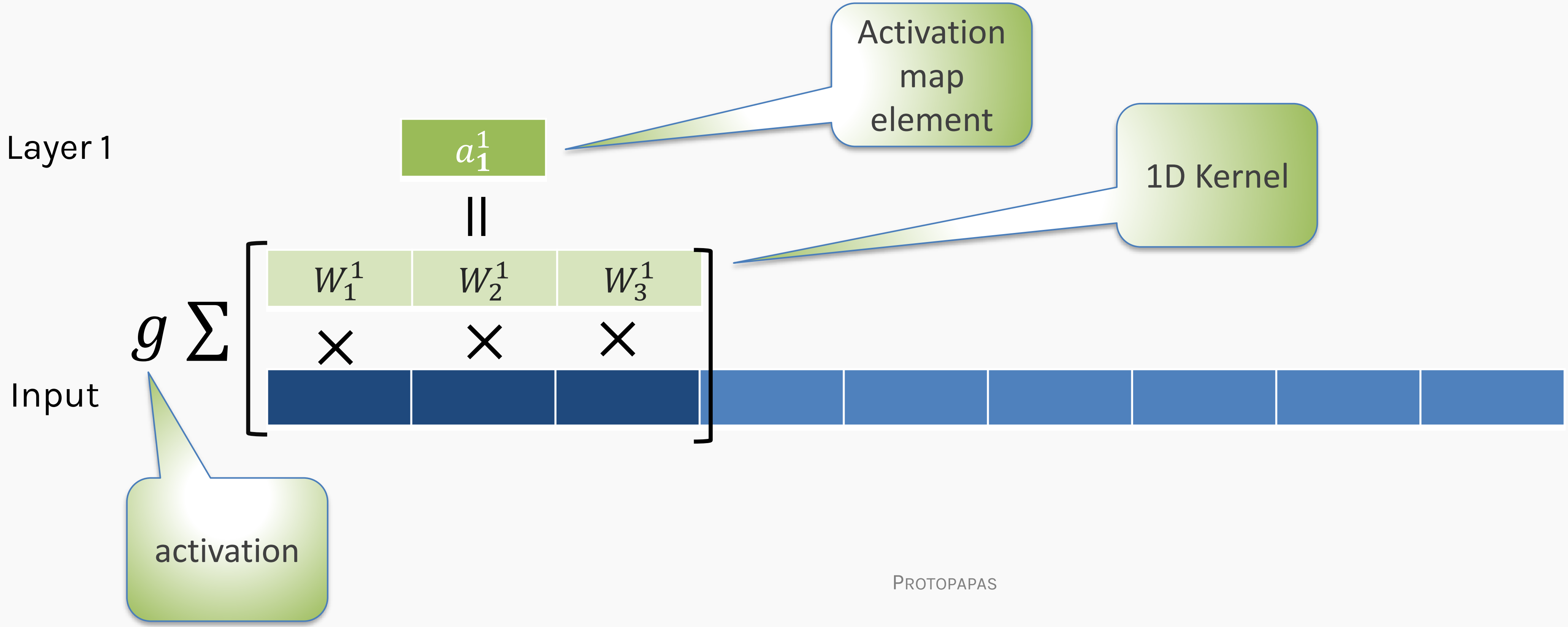
The **receptive field** is defined as the region in the input space that a particular CNN's feature (or activation) is looking at (i.e. be affected by).

The receptive field size is a crucial issue in many visual tasks, as the output must respond to large enough areas in the image to capture information about large objects.

Layers Receptive Field

The **receptive field** is defined as the region in the input space that a particular CNN's feature (or activation) is looking at (i.e. be affected by).

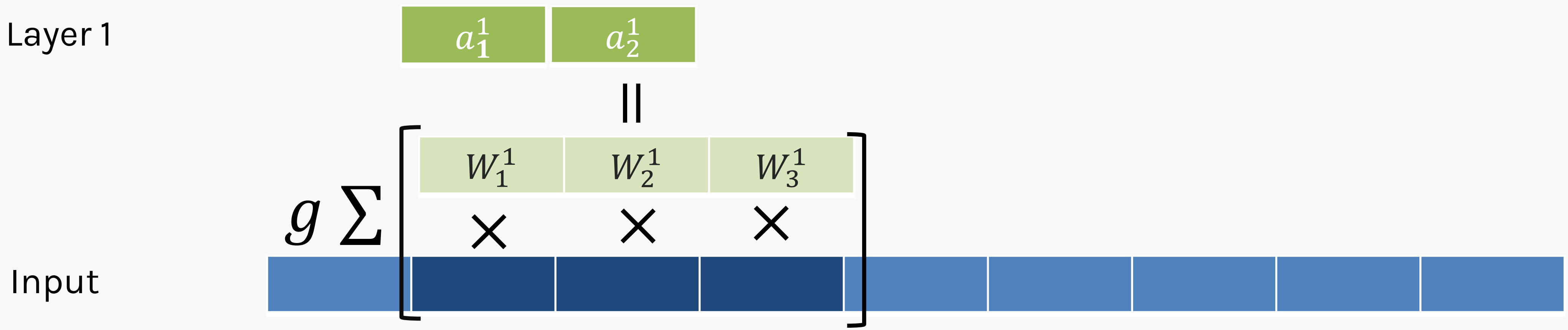
Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



Layers Receptive Field

The **receptive field** is defined as the region in the input space that a particular CNN's feature (or activation) is looking at (i.e. be affected by).

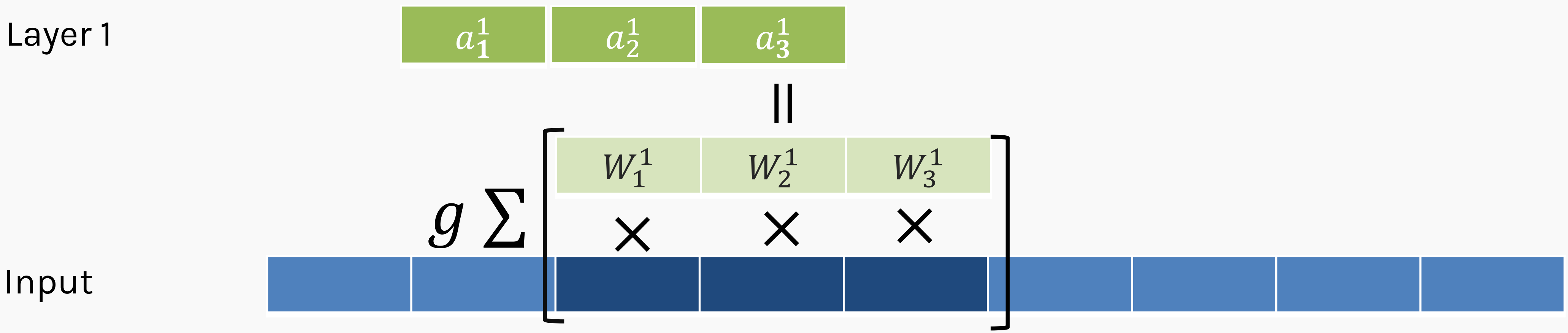
Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



Layers Receptive Field

The **receptive field** is defined as the region in the input space that a particular CNN's feature (or activation) is looking at (i.e. be affected by).

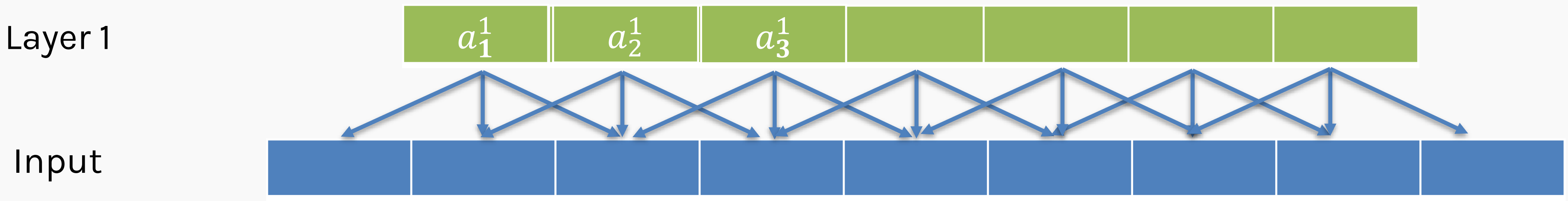
Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



Layers Receptive Field

The **receptive field** is defined as the region in the input space that a particular CNN's feature (or activation) is looking at (i.e. be affected by).

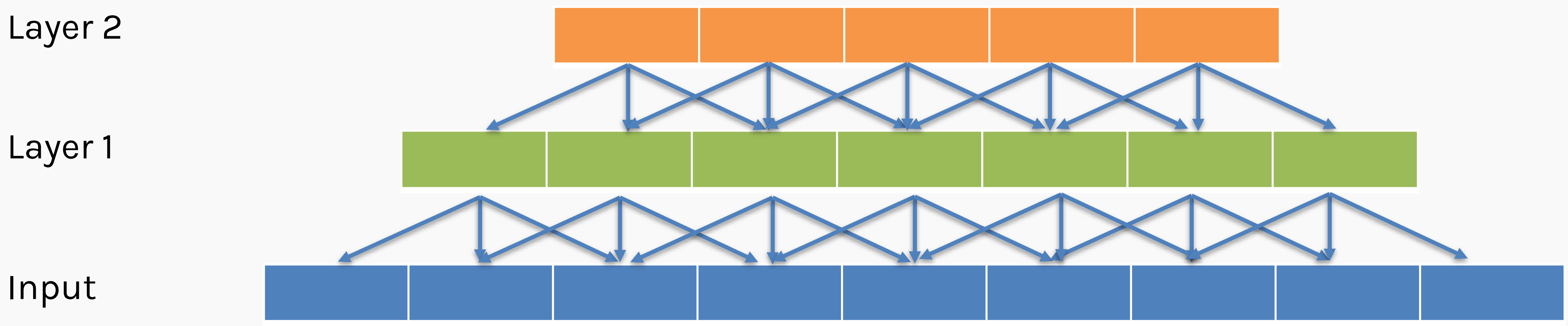
Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



Layers Receptive Field

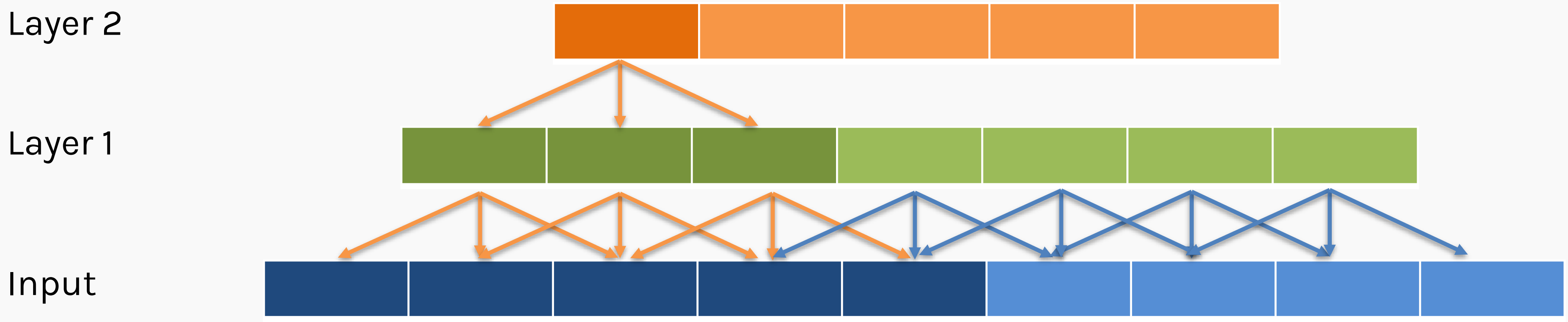
The **receptive field** is defined as the region in the input space that a particular CNN's feature (or activation) is looking at (i.e. be affected by).

Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



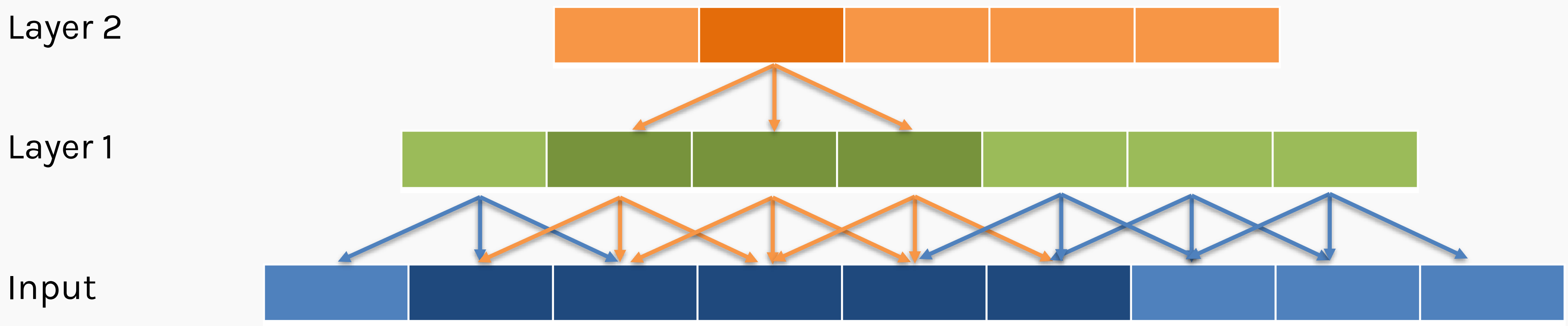
Layers Receptive Field

The receptive field for each element of layer’s 2 is shown below.



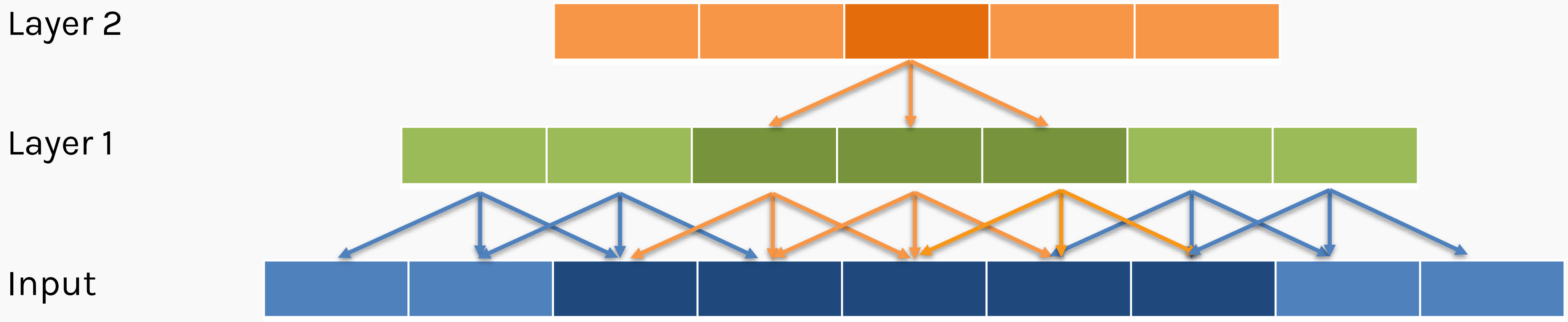
Layers Receptive Field

The receptive field for each element of layer's 2 is shown below.



Layers Receptive Field

The receptive field for each element of layer’s 2 is shown below.

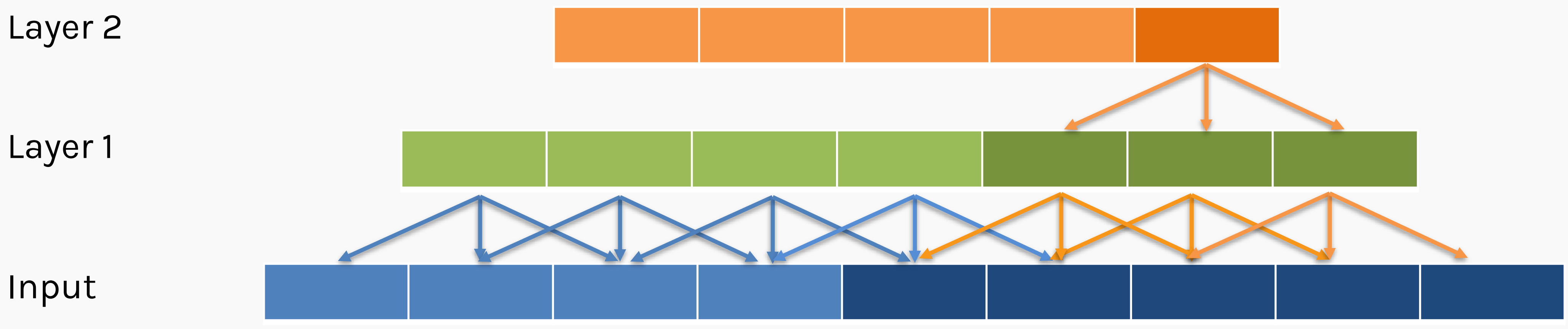


Layers Receptive Field

The receptive field for each element of layers 1 and 2 are shown below.

Layers Receptive Field

The receptive field for each element of layers 1 and 2 are shown below.

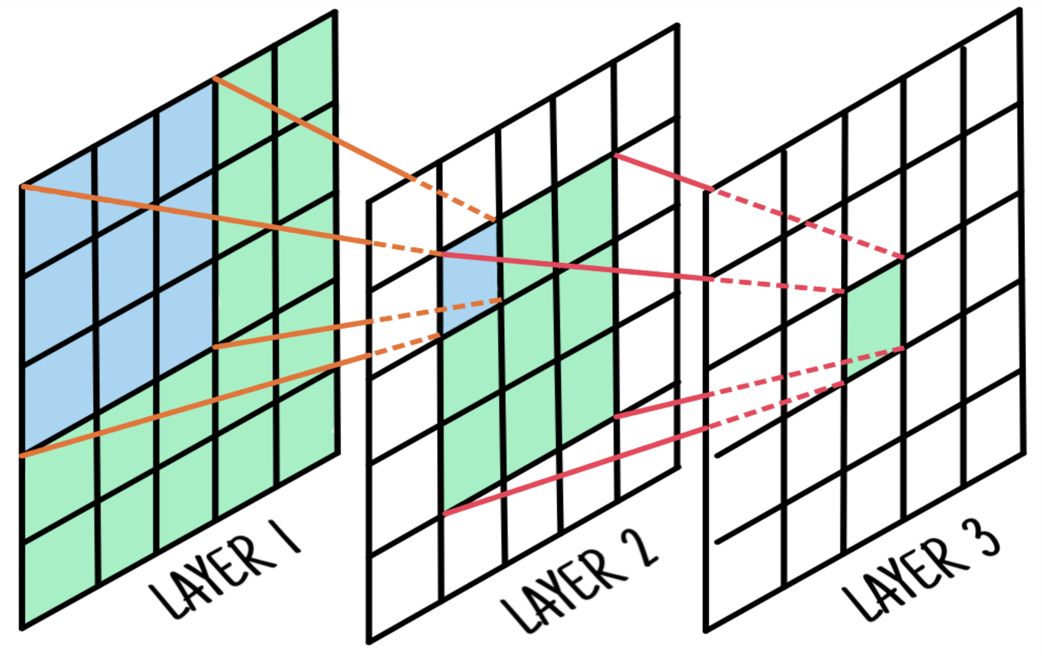


Layers Receptive Field

In 2D, it works the same way.

The receptive field can be calculated using the recursive formula:

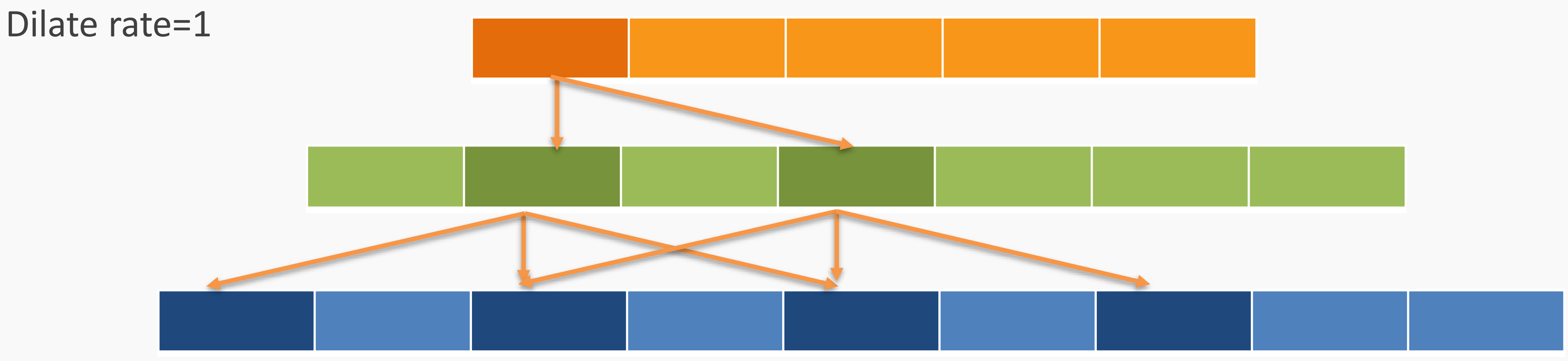
$$r_0 = 1 + \sum_{l=1}^{L} (k_l - 1) \prod_{i=1}^{l-1} s_i$$



- k_l kernel size (positive integer)
- s_l stride (positive integer)

Dilated CNNs

- We can "inflate" the receptive field by inserting holes between the kernel elements.
- These are called **Dilated Convolutions.**
- Dilation rate indicates how much the kernel is widened.

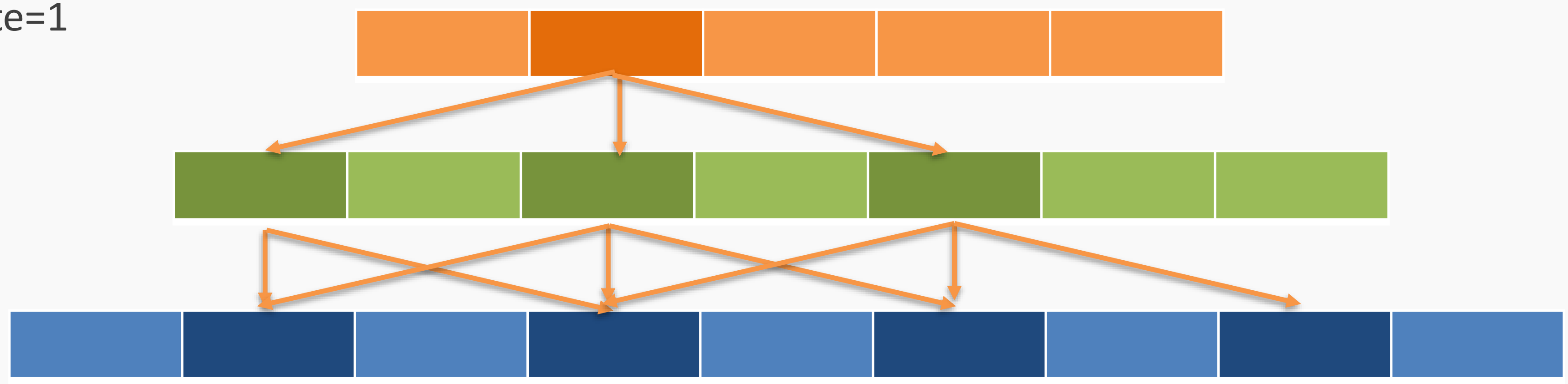


Original Idea: Algorithme a trous, an algorithm for wavelet decomposition (Holschneider et al., **1987**; Shensa, 1992)

Dilated CNNs

- We can "inflate" the receptive field by inserting holes between the kernel elements.
- These are called **Dilated Convolutions.**
- Dilation rate indicates how much the kernel is widened.

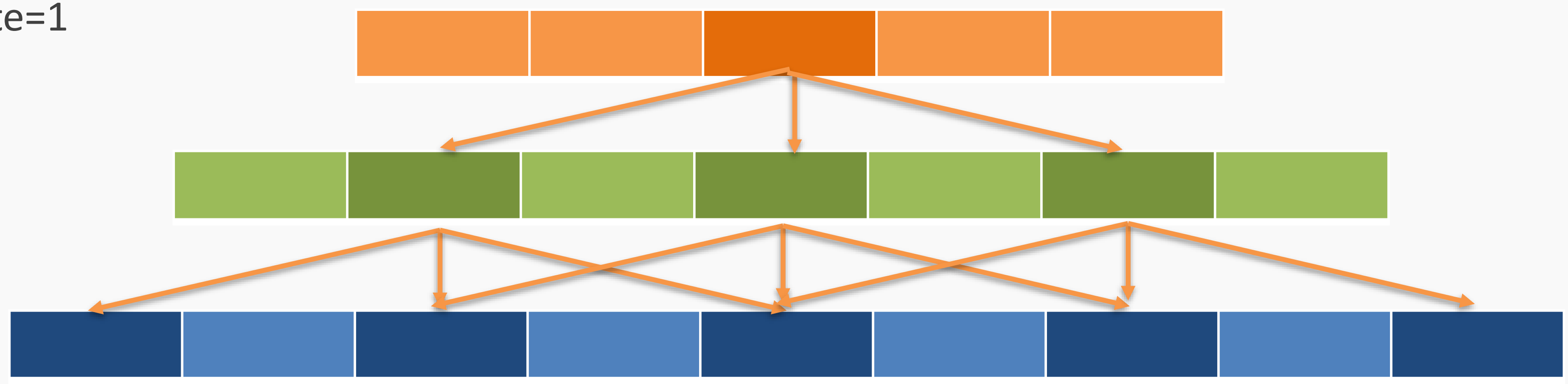
Dilate rate=1



Dilated CNNs

- We can "inflate" the receptive field by inserting holes between the kernel elements.
- These are called **Dilated Convolutions.**
- Dilation rate indicates how much the kernel is widened.

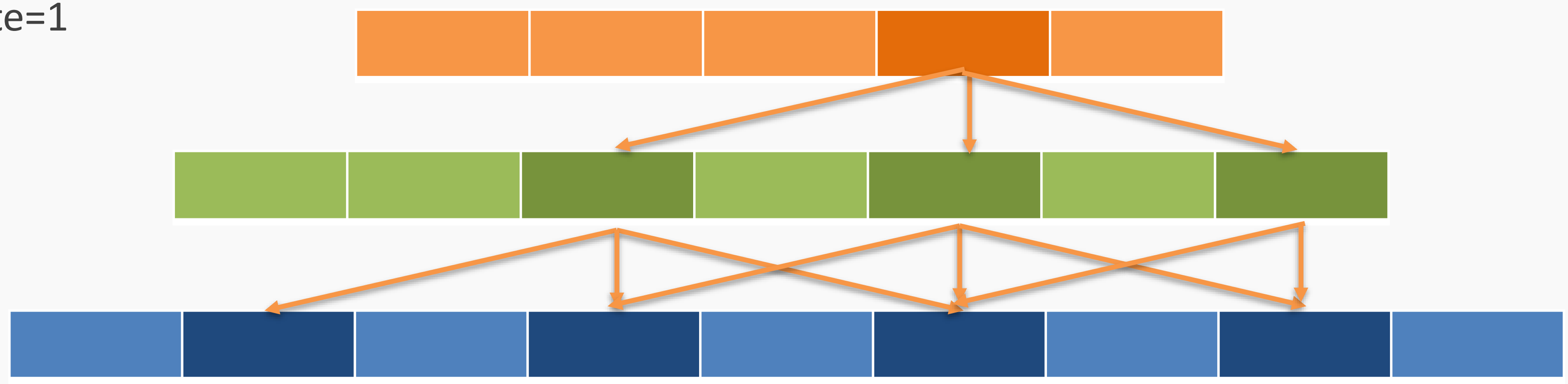
Dilate rate=1



Dilated CNNs

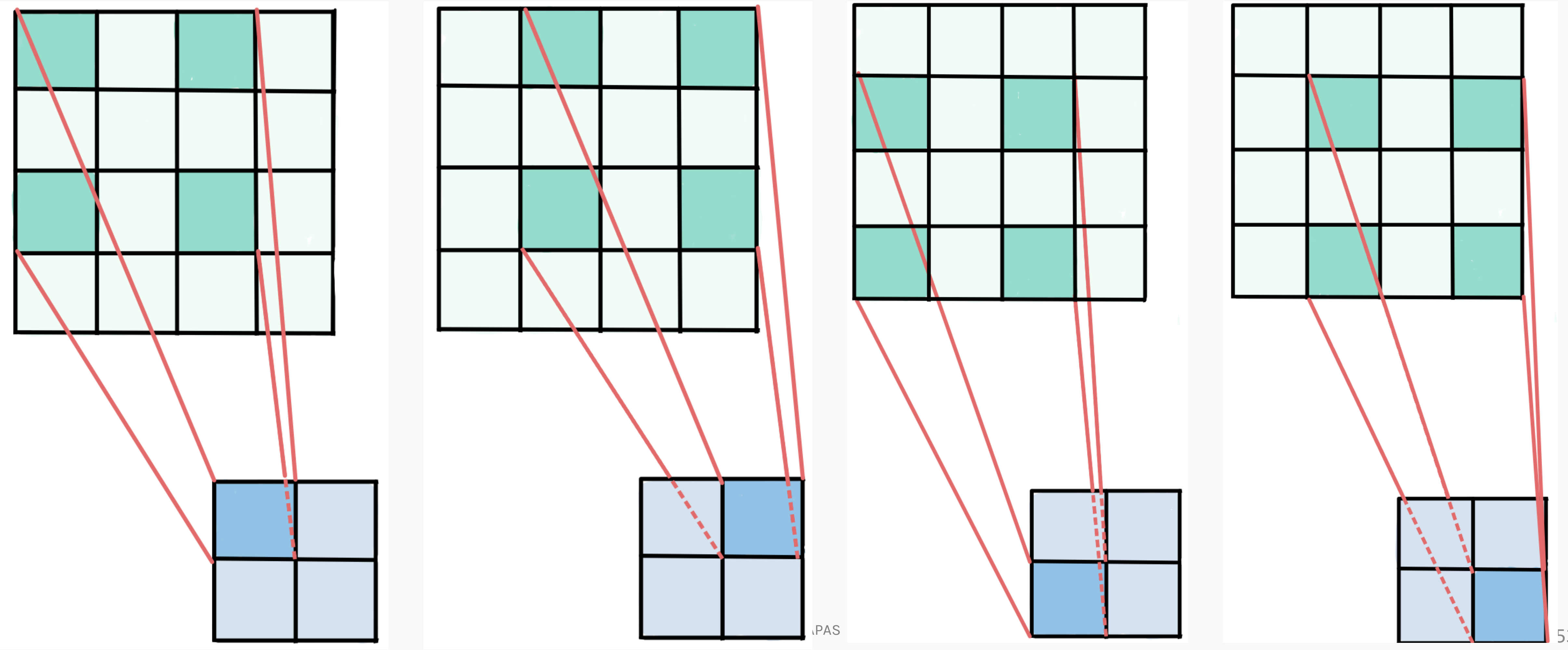
- We can "inflate" the receptive field by inserting holes between the kernel elements.
- These are called **Dilated Convolutions.**
- Dilation rate indicates how much the kernel is widened.

Dilate rate=1



Dilated CNNs

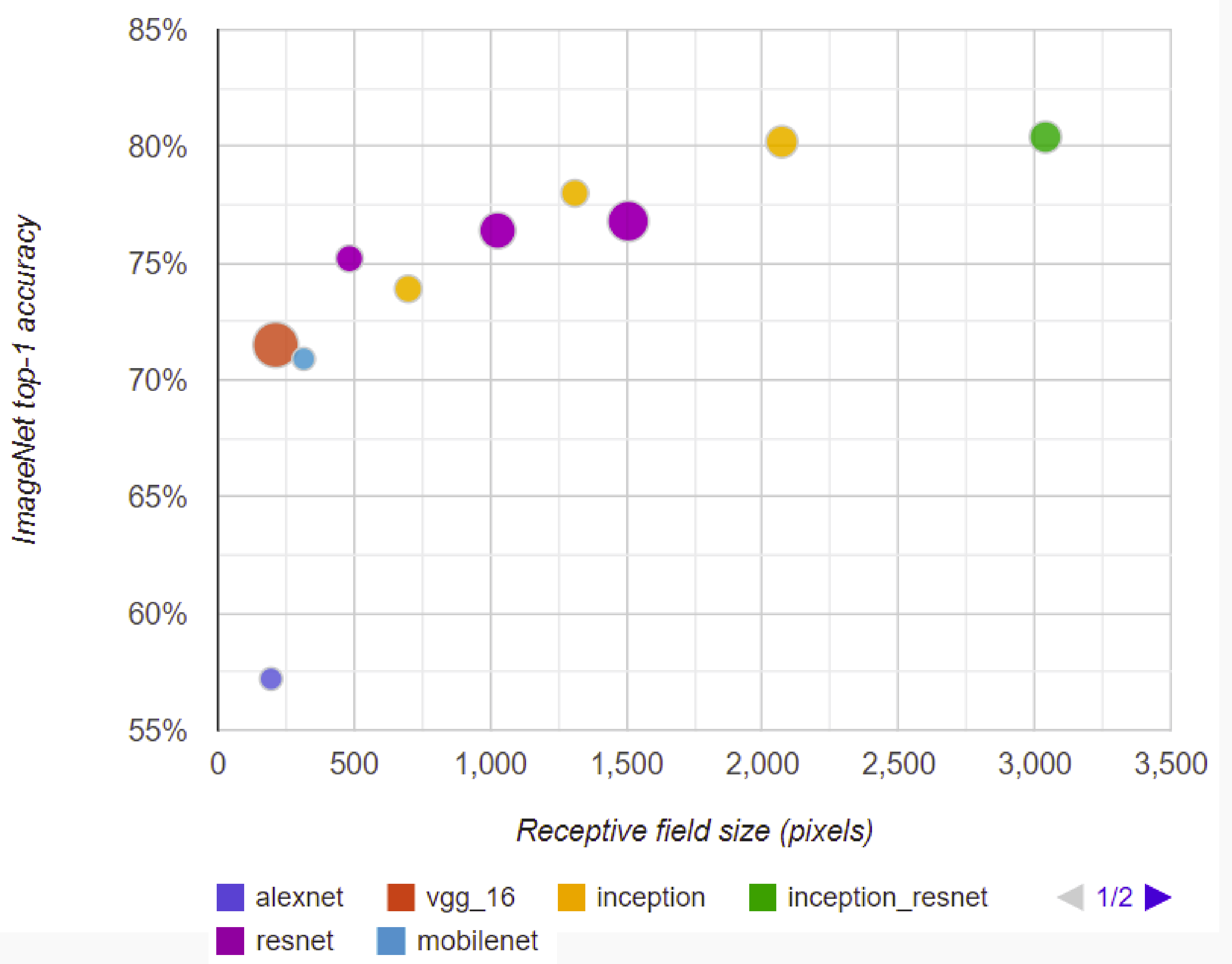
2D Example: 2x2 kernel, stride=1, dilate rate=1



Receptive Field

There is a relationship between classification accuracy and receptive field size.

Large receptive fields are necessary for high-level recognition tasks, but with diminishing rewards.

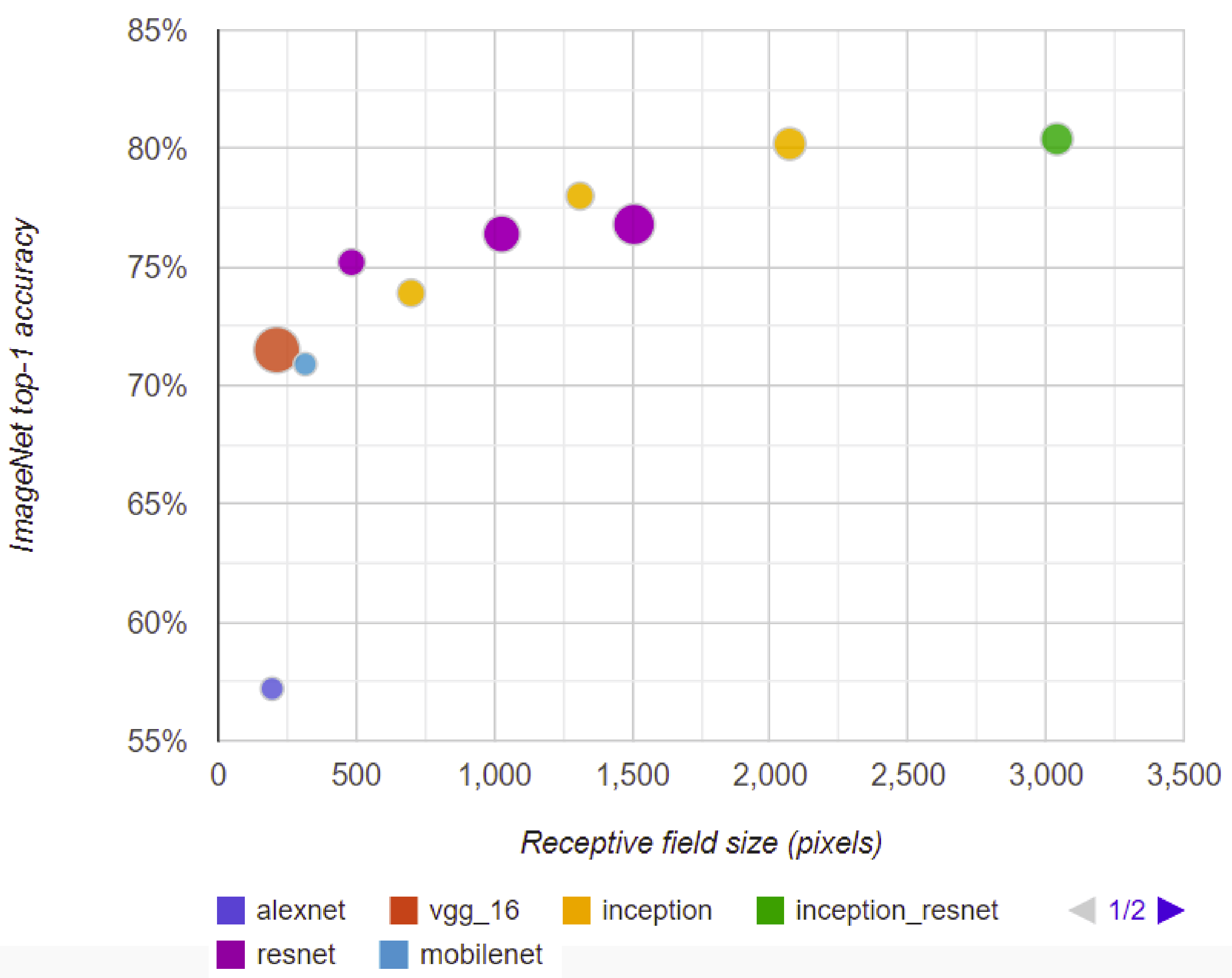


Araujo, A., Norris, W., & Sim, J. (2019). [Computing receptive fields of convolutional neural networks.](#) *Distill*, *4*(11), e21.

Receptive Field

The receptive field is important, as higher-level features generally are bigger than low-level ones.

Moreover, the final convolutions need to extract global features. Otherwise, the final FFCC layers would have problems generalizing.



Araujo, A., Norris, W., & Sim, J. (2019). [Computing receptive fields of convolutional neural networks.](#) *Distill*, *4*(11), e21.

Outline

1. Regularization for CNN
2. BackProp of MaxPooling layer
3. Layers Receptive Field
4. **Weights and feature maps visualization**

Lessons for Visualization

Choosing/designing machine learning visualization requires that we think about:

***Why and for whom* to visualize?**

- are we visualizing to **diagnose** problems with our models?
- are we visualizing to **interpret** our model's meaningfulness?

***What and how to visualize*?**

- do we visualize decision boundaries, weights of our model, and or distributional differences in the data?

Why and for whom to visualize?

1. **Interpretability & Explainability:** understand how deep learning models make decisions and what representations they have learned.

For others

2. **Debugging & Improving Models:** help model developers build and debug their models, with the hope of expediting the iterative experimentation process to ultimately improve performance.

For us

3. **Teaching Deep Learning Concepts:** educate non-expert users about AI

For me

From: [Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers]

What and how to visualize?

What technical components of neural networks could be visualized?

- Computational Graph & Network Architecture
- Learned Model Parameters: weights, filters
- Individual Computational Units: activations, gradients
- Aggregate information: performance metrics

How can they be insightfully visualized?

How depends on the type of data and model as well as our specific investigative goal.

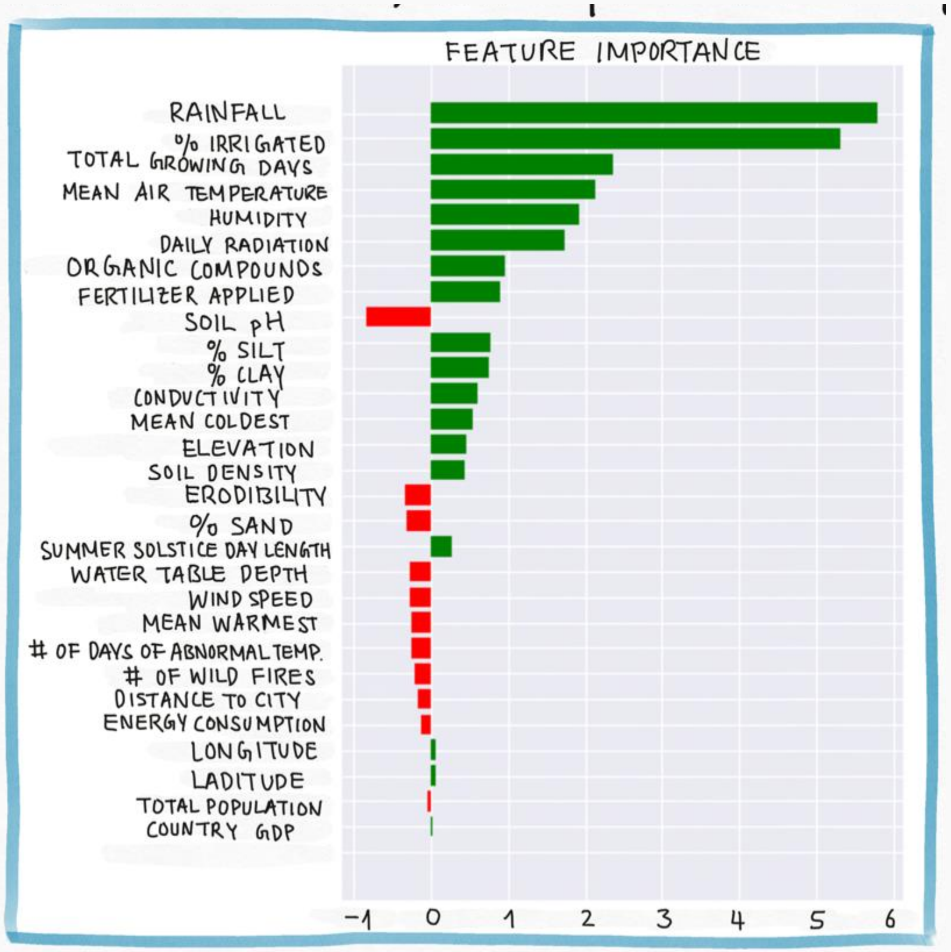
What to Visualize for Neural Network Models?

For logistic regression, $p(y=1|w,x)=\sigma(w^T x)$ we can interrogate the model by printing out the weights of the model.

Recalling from previous lectures, we can visualize the feature importance looking at the coefficients

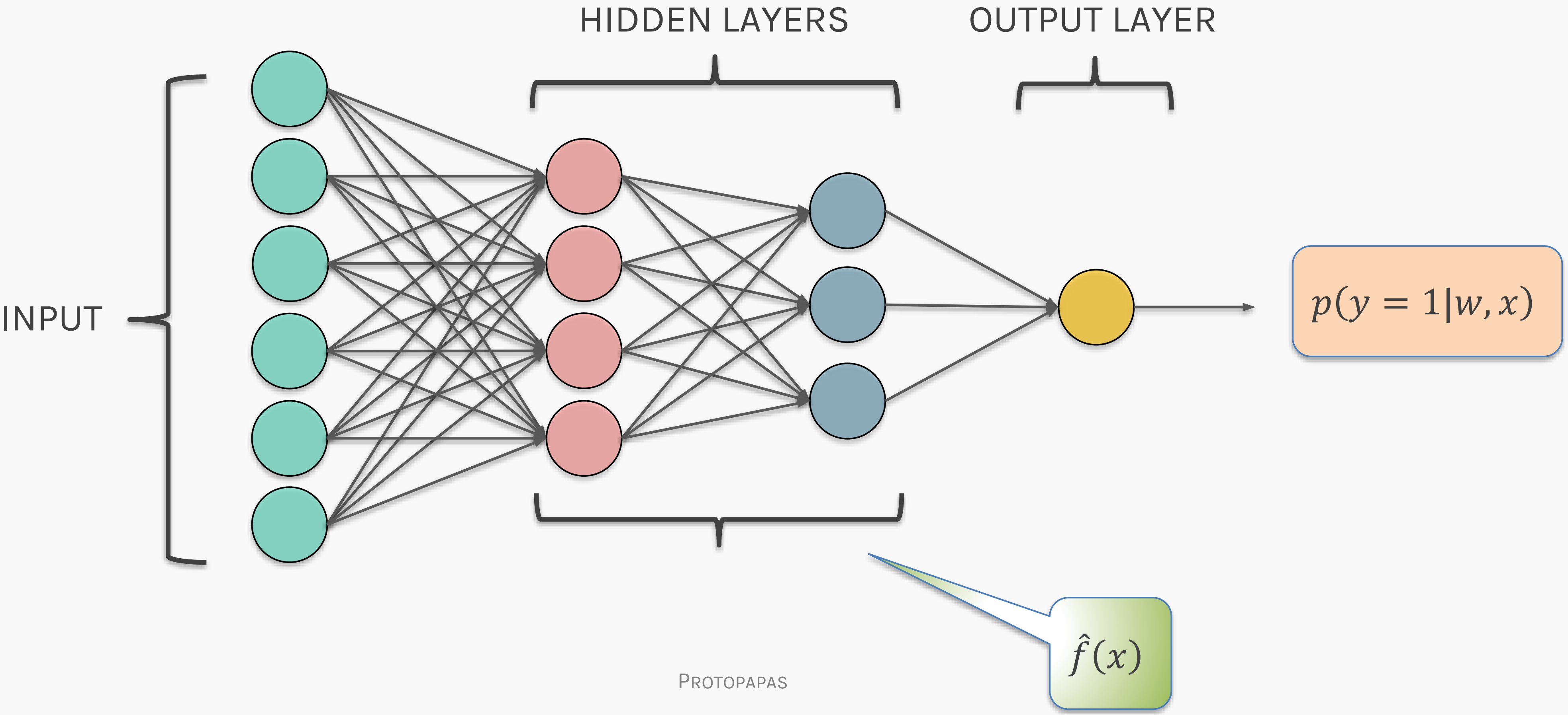
$$\ln\left(\frac{P(y=1)}{P(y=0)}\right)=w^T x$$

log-odds



What to Visualize for Neural Network Models?

For a neural network classifier, $p(y=1|w,x)=\sigma(\hat{f}(x))$ would it be helpful to print out all the weights?



Weight Space Versus Function Space

While it's convenient to build up a complex function by composing simple ones -as in neural networks- understanding the impact of each weight on the outcome is difficult.

In fact, the relationship between weights of a neural network and the function the network represents is extremely complicated:

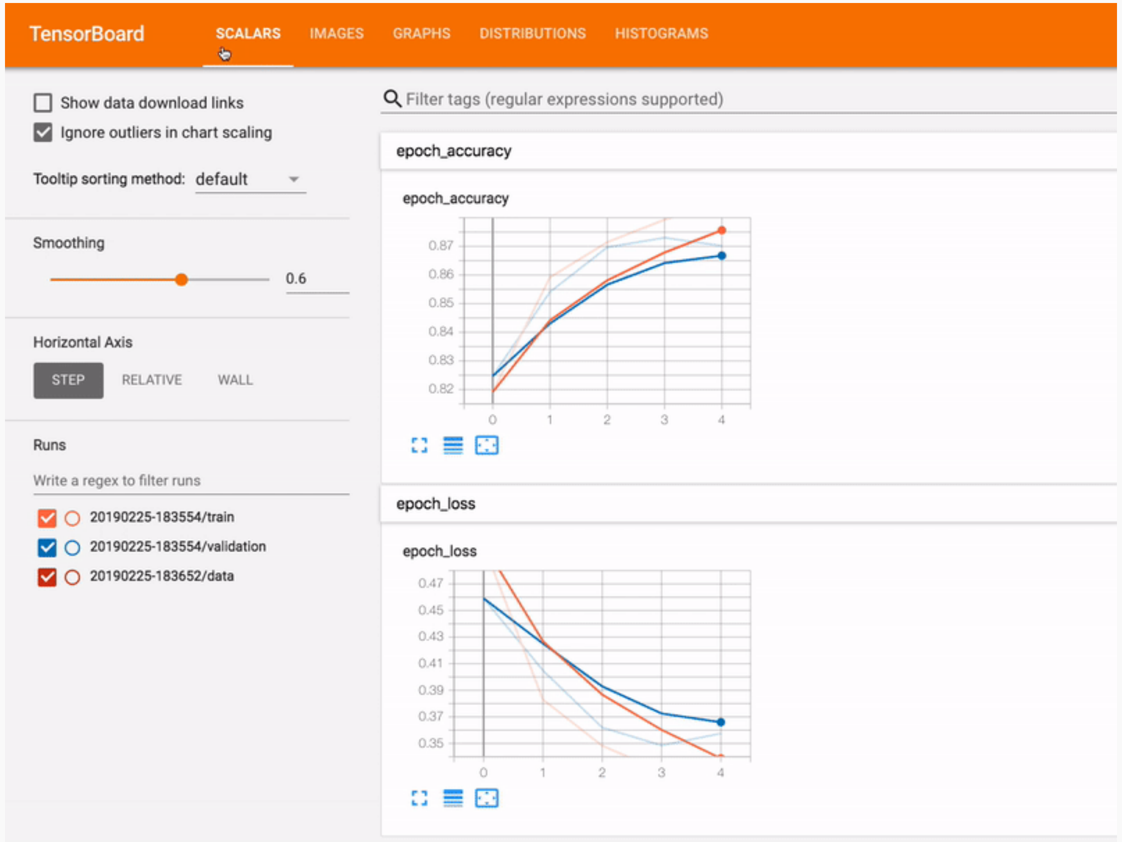
1. the same function may be represented by two very different set of weights for the same architecture.
2. the architecture may be overly expressive - it can express the function $\hat{f}$ using a subset of the weights and hidden nodes (i.e. the trained model can have weights that are zero or nodes that contribute little to the computation).

Debugging & Improving Models : Tensorboard

What happens if we want to know the outputs of a specific hidden layer?

By visualizing the network weights and activations as we train, we can diagnose issues that ultimately impact model performance.

TensorFlow provides a functionality to explore the inner workings of the network.



From: [Tensorboard]()

Debugging & Improving Models : Tensorboard

Callbacks are objects that operate while the network is being trained, evaluated or making predictions. Tensorboard can be used as a callback, calling it directly from the .fit() method.

Path to store the logfile.

Frequency of the records. Can be set as ‘epochs’, ‘batch’ or a specific number of batches.

```
tf.keras.callbacks.TensorBoard(log_dir = "logs",update_freq = "epoch")
```

```
model.fit(x_train, y_train, callbacks=[tb_callback])
```

From: [Tensorboard]

Debugging & Improving Models : Tensorboard

Frequency of the records. Can be set as ‘epochs’, ‘batch’ or a specific number of batches.

Path to store the logfile.

```
tf.keras.callbacks.TensorBoard(log_dir = "logs",update_freq = "epoch")
```

```
model.fit(x_train, y_train, callbacks=[tb_callback])
```

Remember to load the Tensorboard notebook extension.

```
%load_ext tensorboard
```

Or execute it locally in your console.

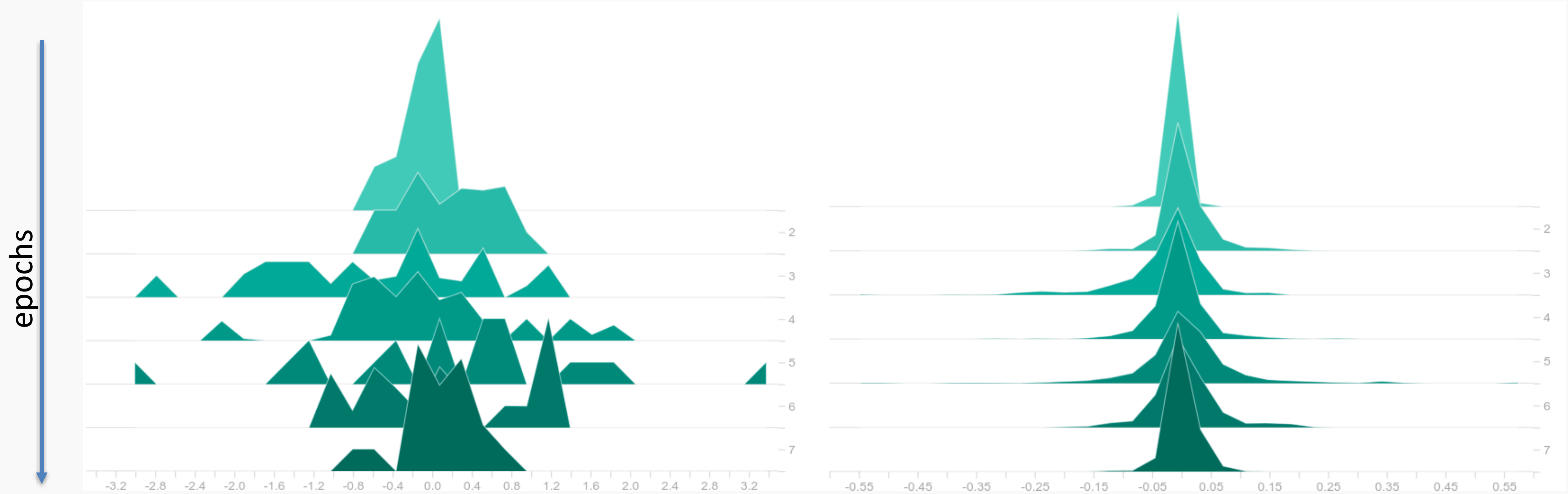
```
tensorboard--logdir = path_to_your_logs
```

From: [Tensorboard]

Debugging & Improving Models : Tensorboard

By visualizing the network weights and activations as we train, we can diagnose issues that ultimately impact model performance.

The following visualizes the distribution of gradients in two hidden layers over the course of training. What problems do we see?

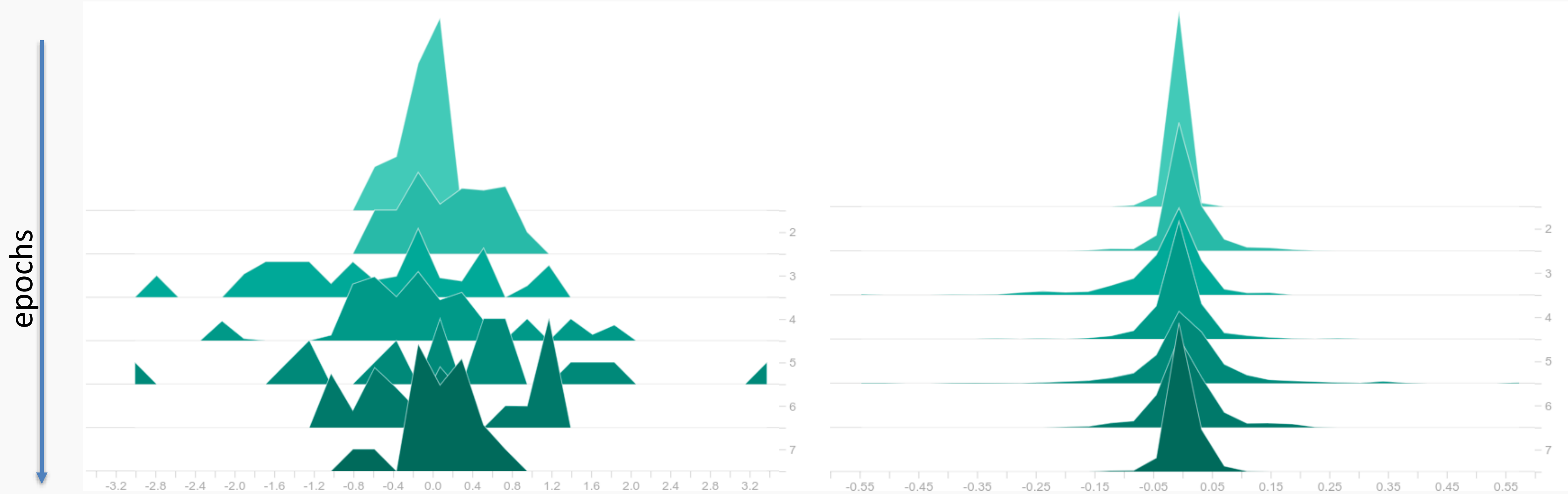


From: [Tensorboard]()

Debugging & Improving Models : Tensorboard

In the first layer, the gradients became big for some weights which might introduce instability in the training.

In the second, the gradients start to became zero. Smaller gradients imply smaller weight updates and slow training speed.

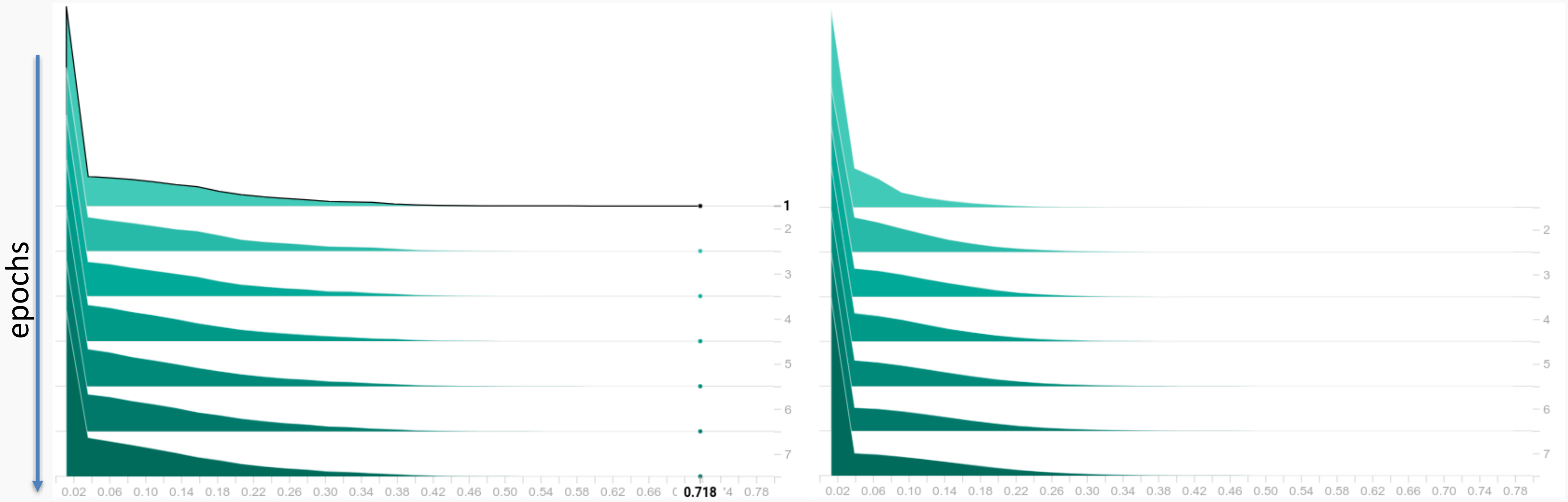


From: [Tensorboard]

Debugging & Improving Models : Tensorboard

The following visualizes the distribution of activations in two convolutional hidden layers over the course of training.

What problems do we see?

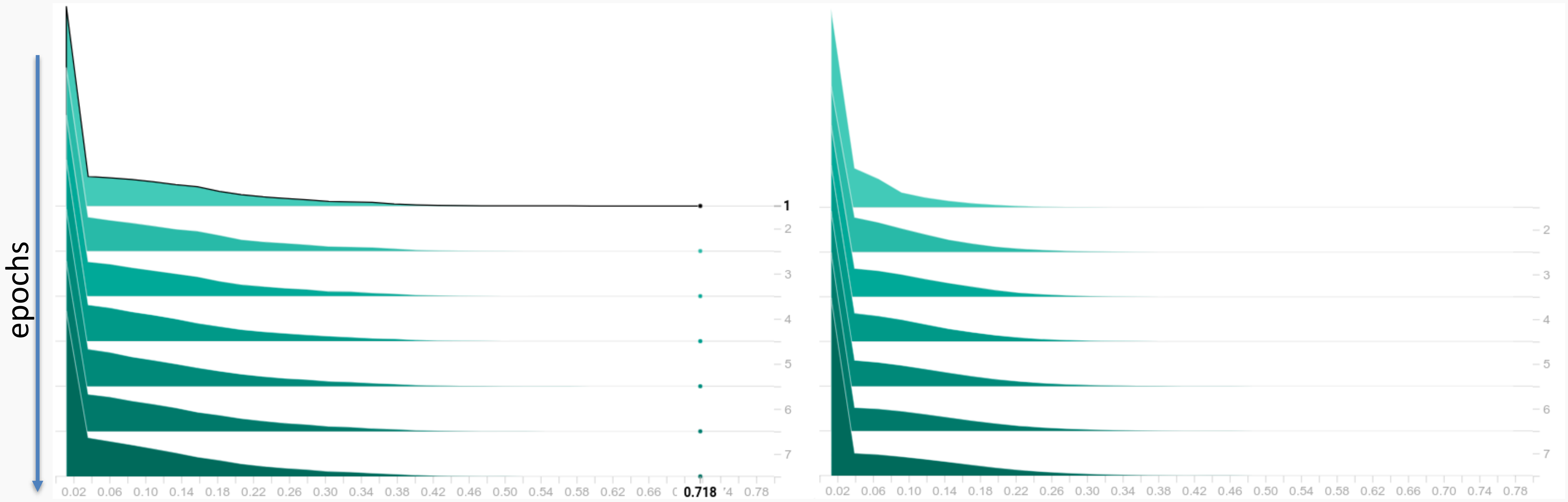


From: [Tensorboard]

Debugging & Improving Models : Tensorboard

The following visualizes the distribution of activations in two convolutional hidden layers over the course of training.

The activations are sparse.



From: [Tensorboard]

CNN Feature Extraction Visualization

We know that CNNs extract features that best helps us to perform our downstream task (e.g. classification).

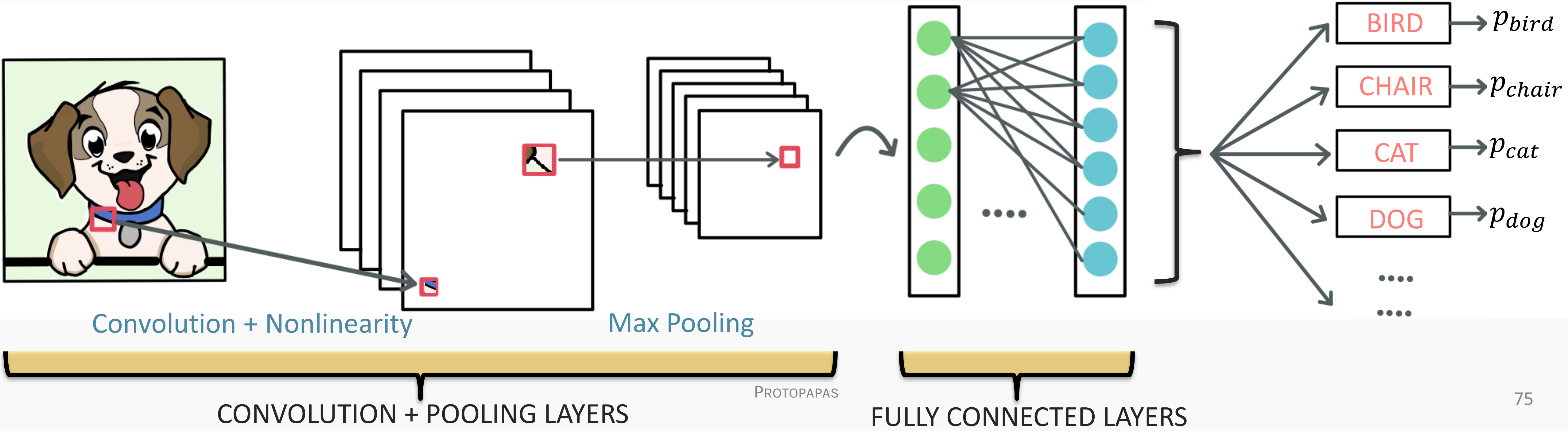
Idea: We train a CNN for feature extraction and a model (e.g. MLP, decision tree, logistic regression) for classification, *simultaneously* and *end-to-end*.

CNN Feature Extraction Visualization

We know that CNNs extract features that best helps us to perform our downstream task (e.g. classification).

Idea: We train a CNN for feature extraction and a model (e.g. MLP, decision tree, logistic regression) for classification, *simultaneously* and *end-to-end*.

The resulting feature maps are matrices, that we can interpret as images. As such, we can analyze them a look for relevant patterns.



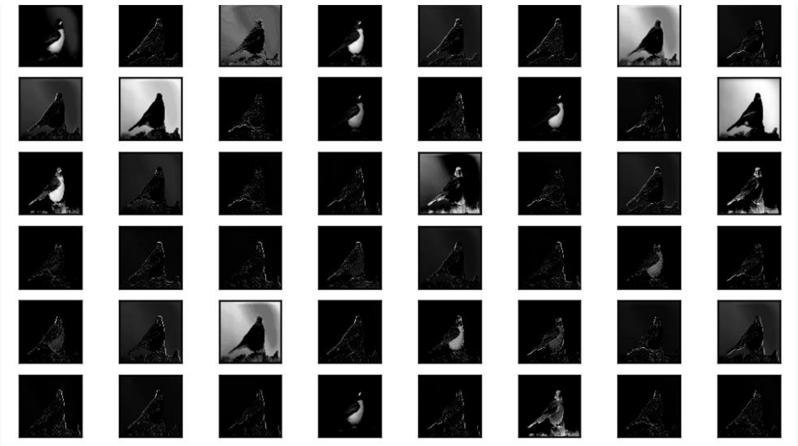
What to Visualize for CNNs?

The first things to try are:

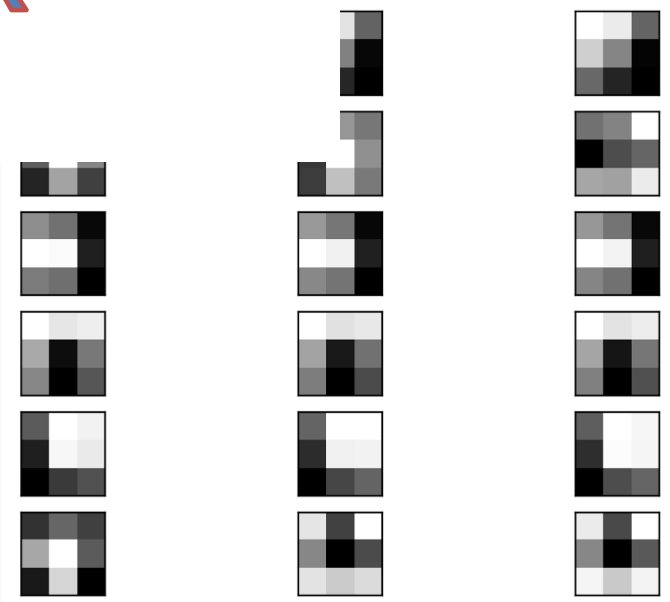
1. visualize the result of applying a learned filter to an image
2. visualize the filters themselves

HARD TO UNDERSTAND OR INTERPRETATE

Input image



Feature maps



Kernels

Occlusion methods

If we want to interpret what part of the image the network is paying more attention, these visualizations might not be the best solution.

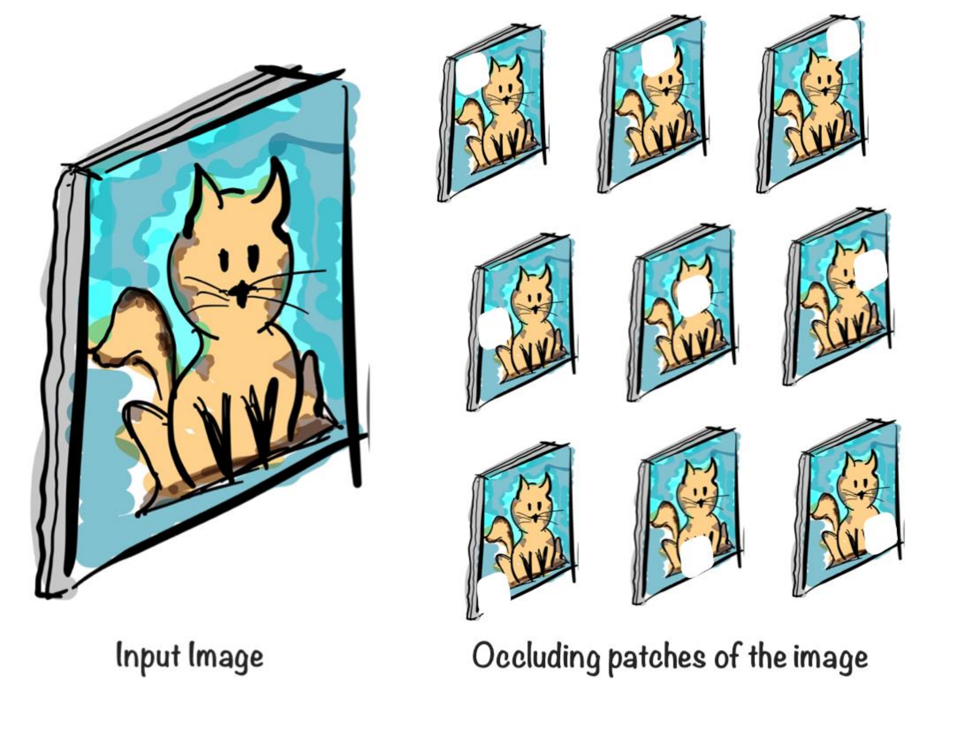
We have no guarantees that the feature maps will provide meaningful information. Their interpretation can be even more difficult than the original problem.

Occlusion methods

Occlusion methods attributes importance for the classification of the image. Occlusion involves running a patch over part of the image to see which pixels affect the classification the most.

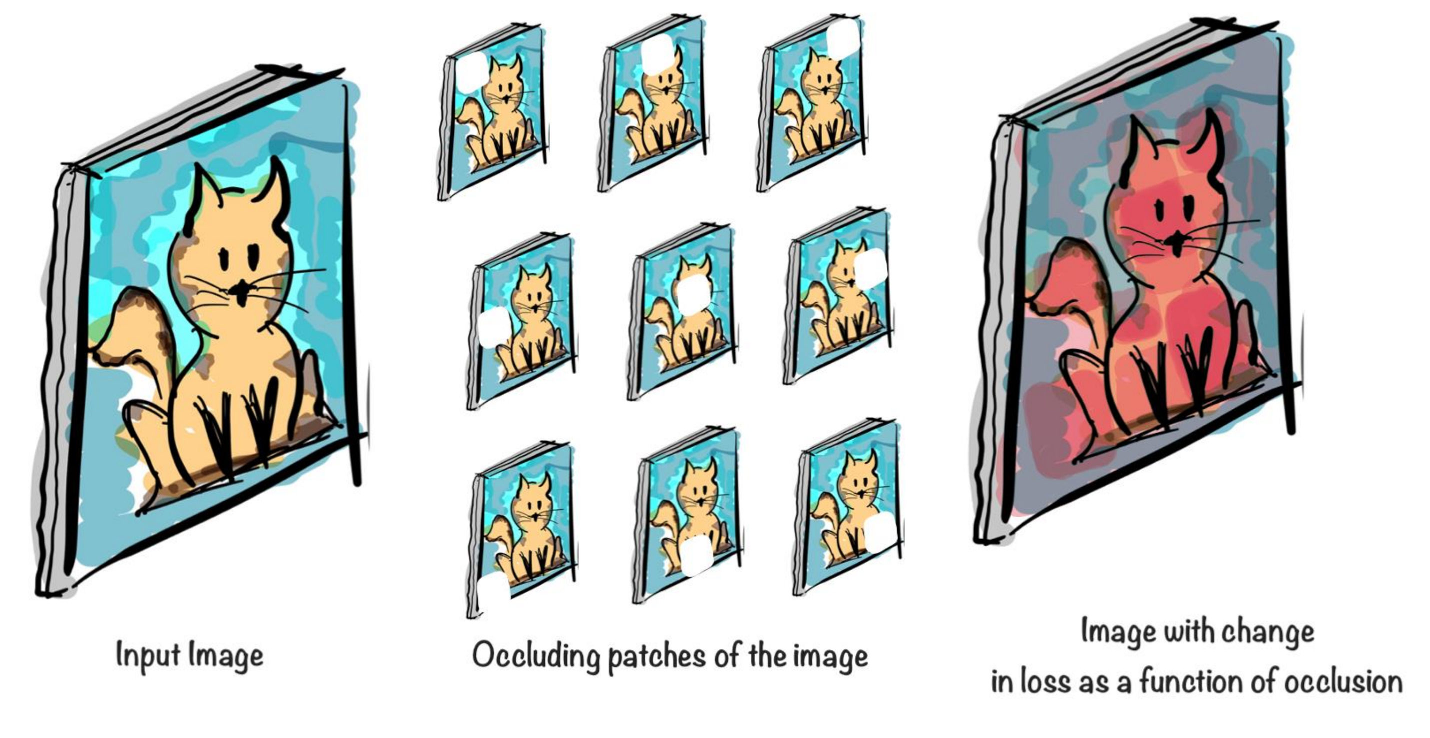
Occlusion methods

Occlusion methods attributes importance for the classification of the image. Occlusion involves running a patch over part of the image to see which pixels affect the classification the most.



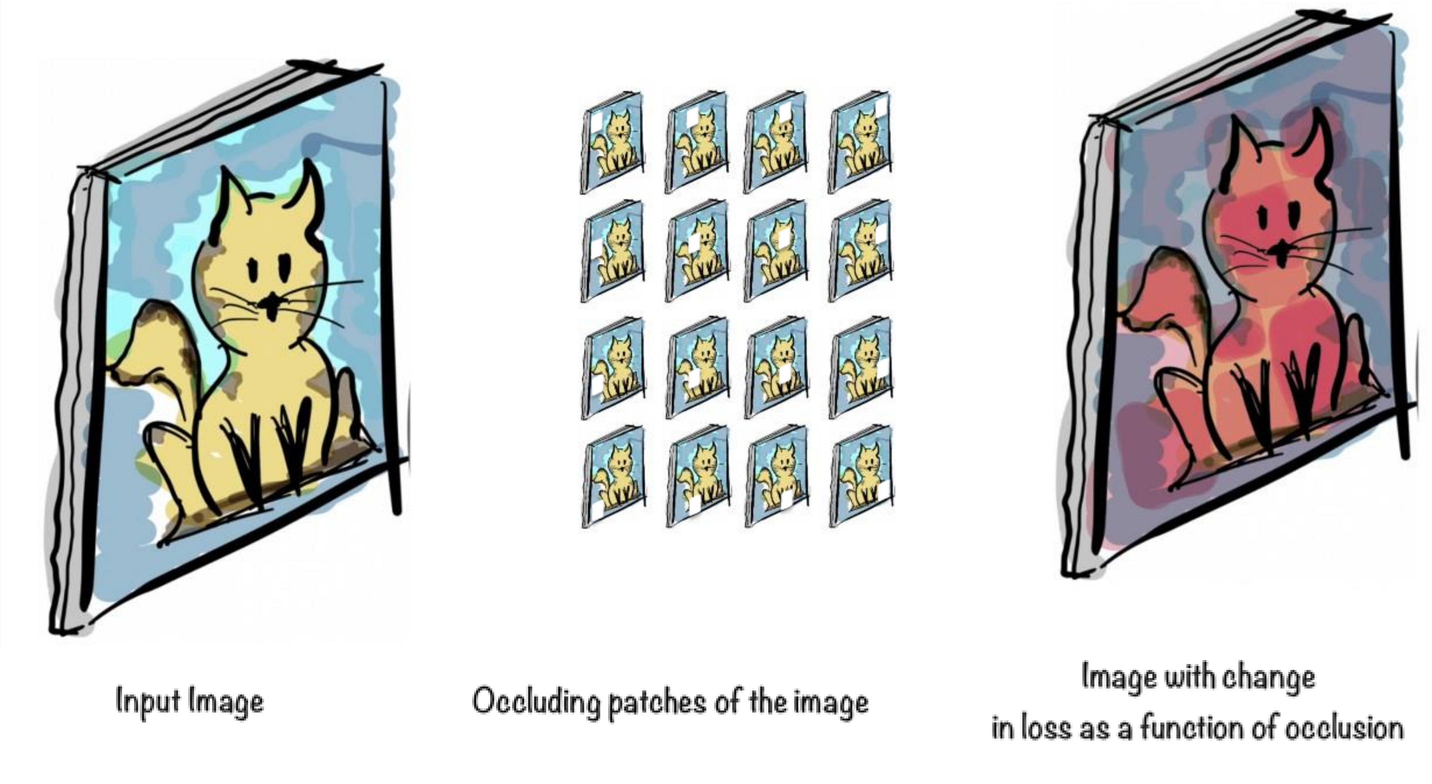
Occlusion methods

Occlusion methods attributes importance for the classification of the image. Occlusion involves running a patch over part of the image to see which pixels affect the classification the most.



Occlusion methods

However, to obtain fine details we need to use a small occlusion area, increasing the number of model evaluations. This can become impractical for a fine resolution and many test images.

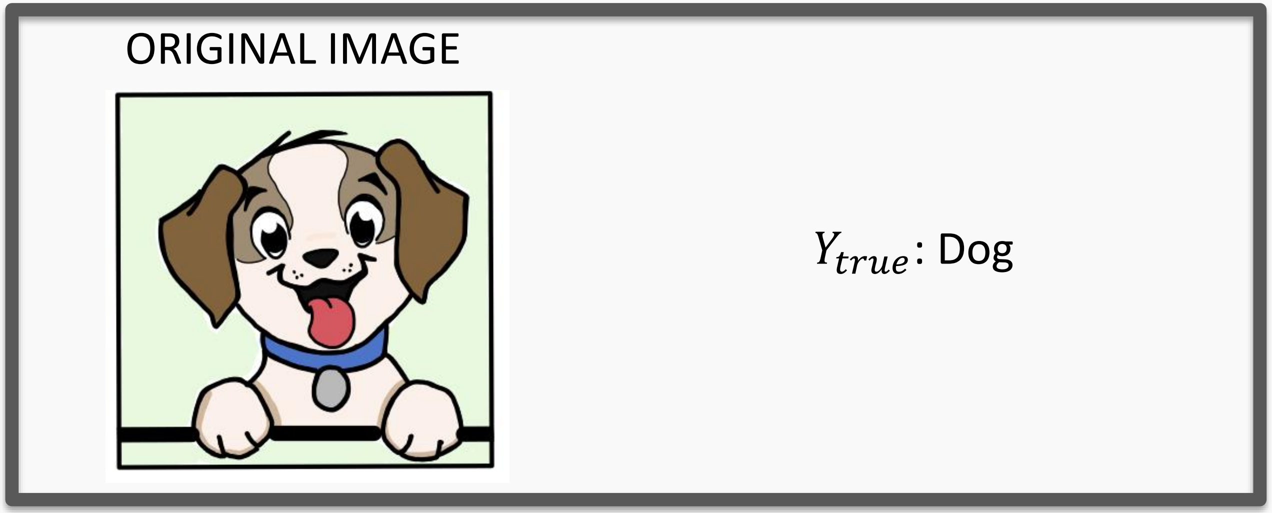


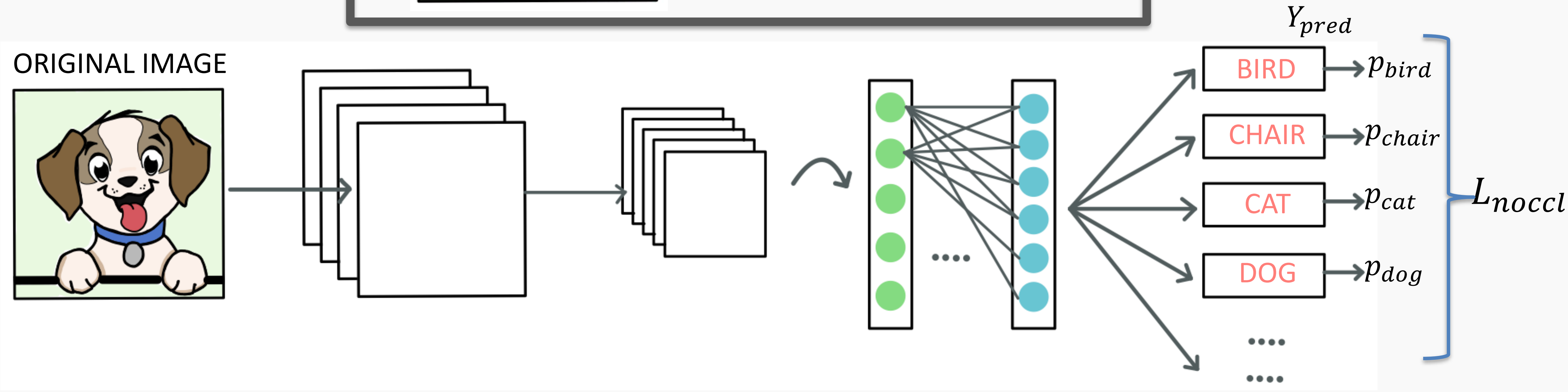
Occlusion process

ORIGINAL IMAGE

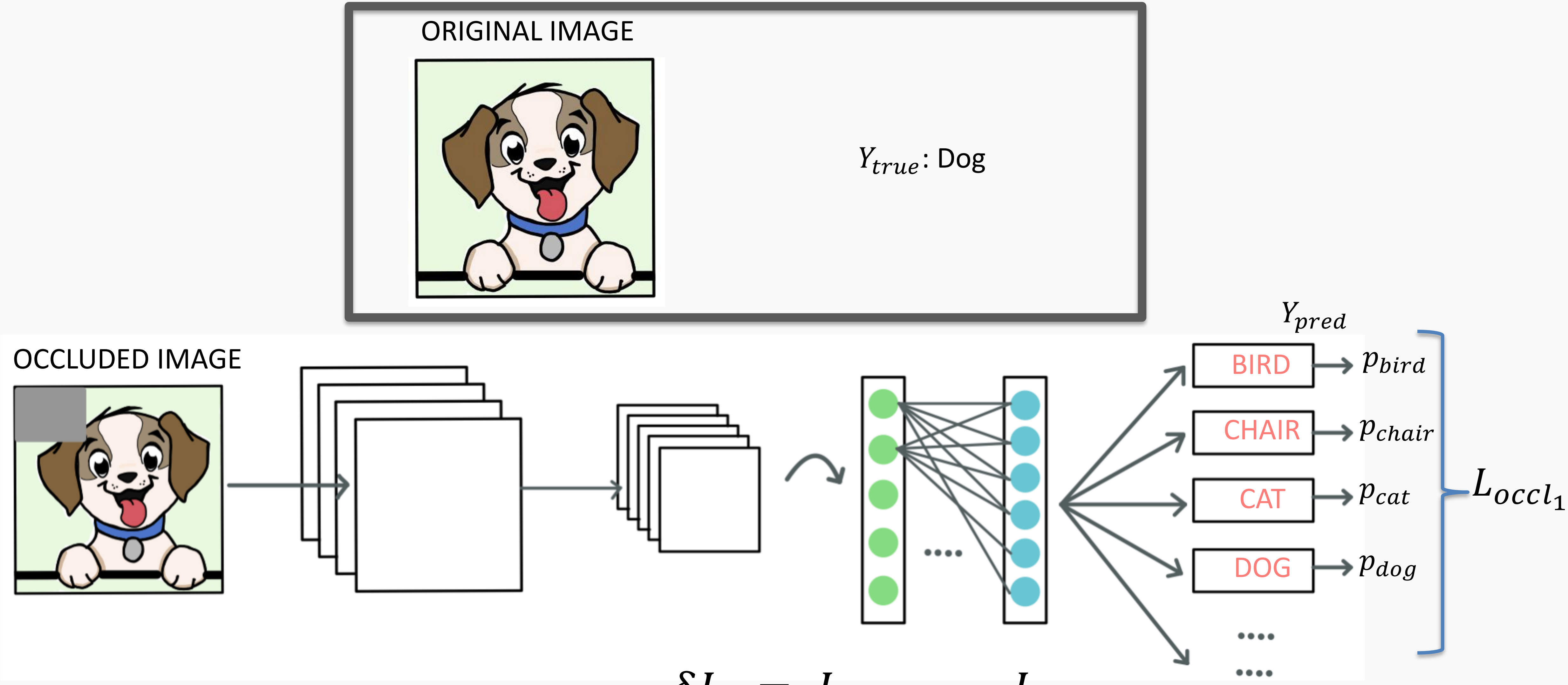
Y_{true}: Dog

Occlusion process



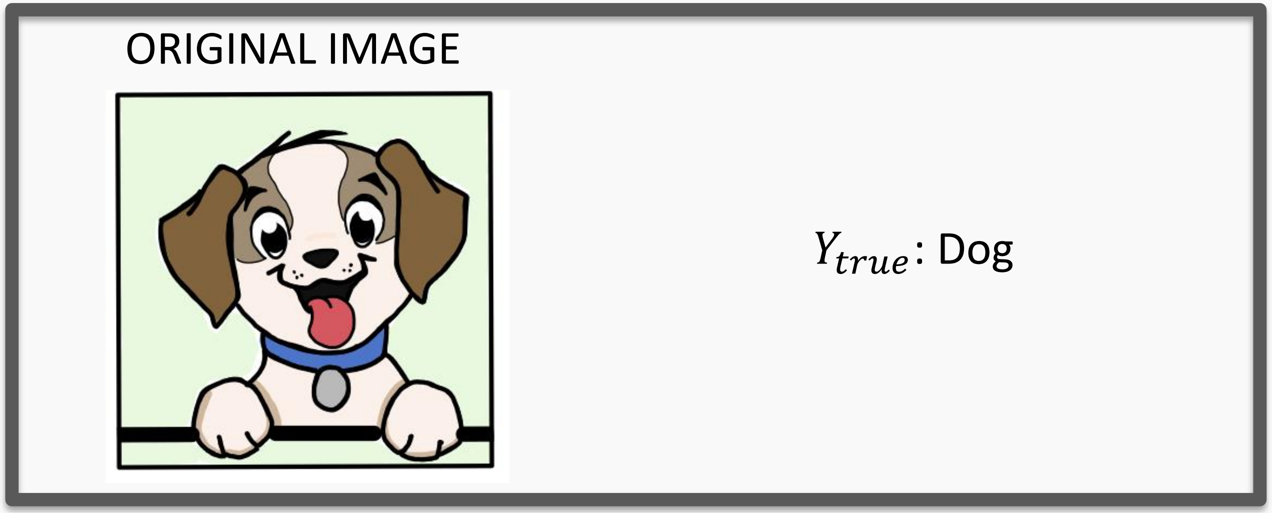


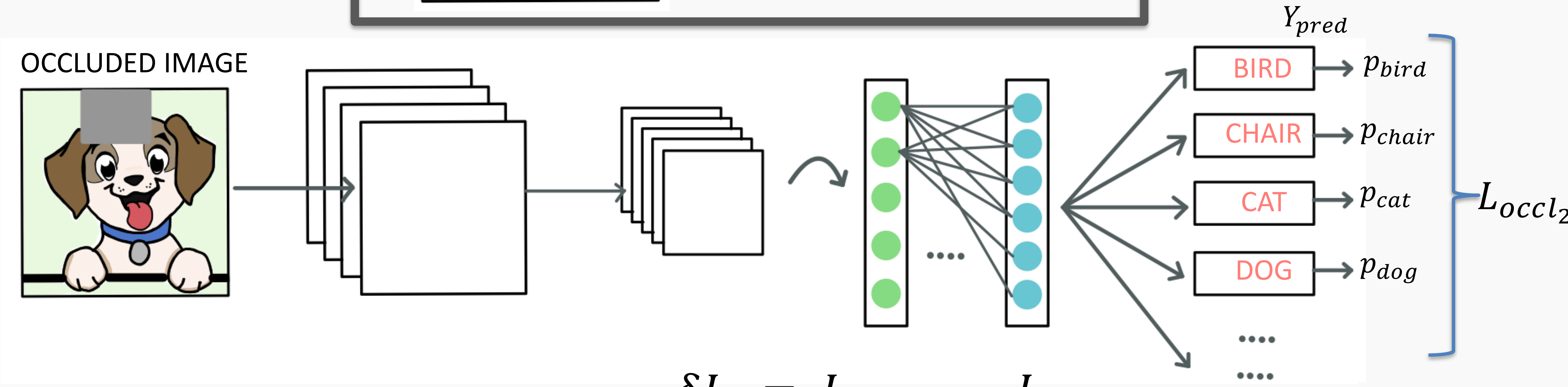
Occlusion process



$$\delta L_1 = L_{noccl} - L_{occl_1}$$

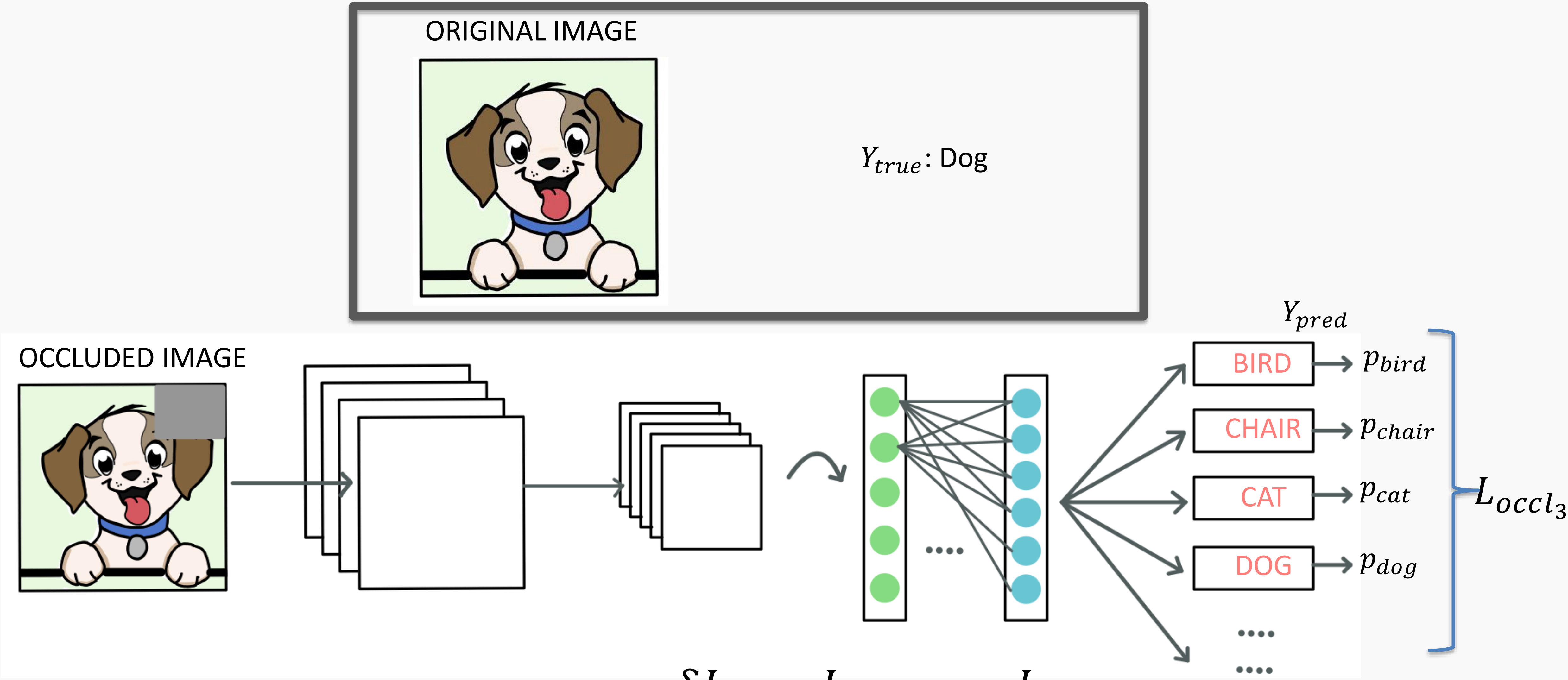
Occlusion process





$$\delta L_2 = L_{noccl} - L_{occl_2}$$

Occlusion process



$$\delta L_3 = L_{noccl} - L_{occl_3}$$

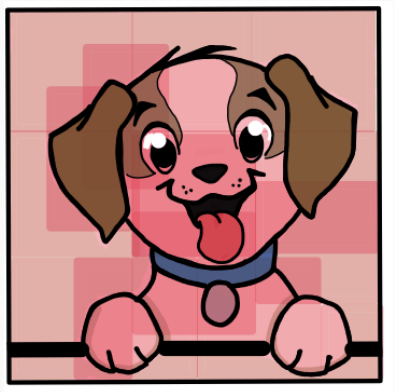
Occlusion process

ORIGINAL IMAGE

Occlusion

$$\xrightarrow{\text{Occlusion}} \begin{bmatrix} \delta L_1 & \delta L_2 & \delta L_3 & \delta L_4 \\ \delta L_5 & \delta L_6 & \delta L_7 & \delta L_8 \\ .. & .. & .. & .. \\ \delta L_{i-3} & \delta L_{i-2} & \delta L_{i-1} & \delta L_i \end{bmatrix} \longrightarrow$$

OCCLUSION LOSS MAP



A smaller value of δL_i implies a larger degradation of the classification loss. Meaning a higher importance for that patch.

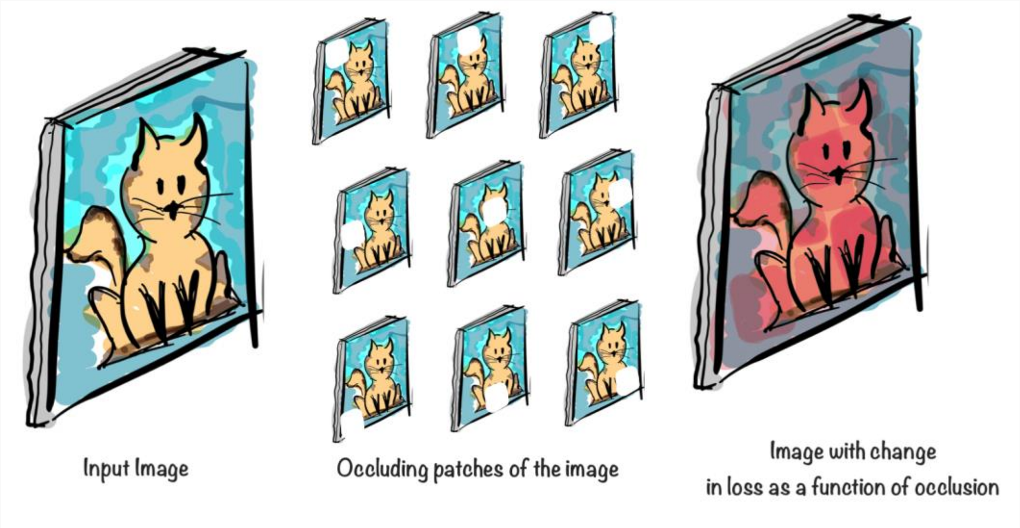
Occlusion process

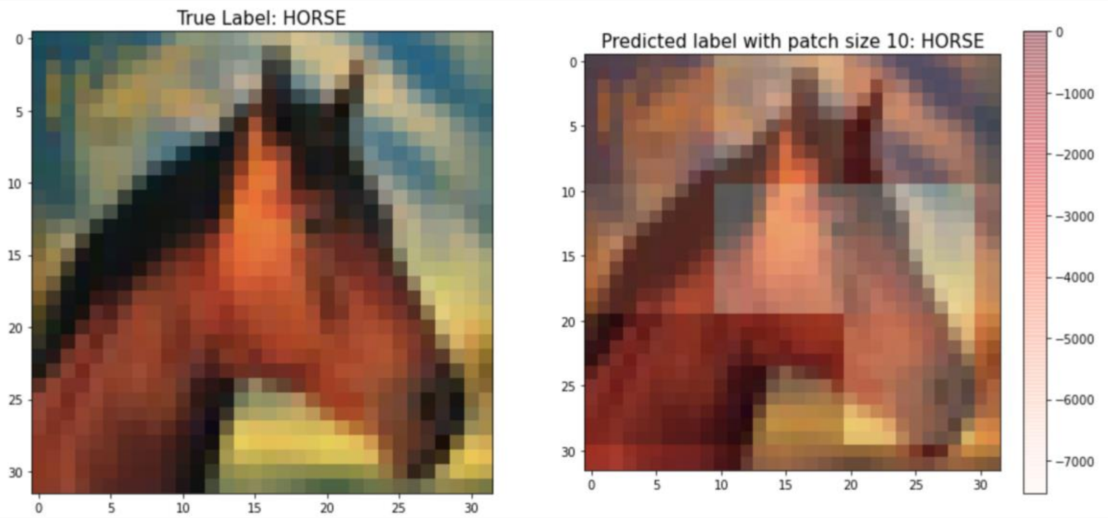
- Input image to a trained network
- Take note of the true label, **K**
- Get the prediction of the true image, Q
- Compute the loss, $L_{noccl} = -\log \mathrm{P}(\mathrm{y} = \mathrm{Q})$
- Occlude patches of the image with gray blocks starting at the top left
  - Get the prediction of this occluded version of the image
  - Compute the loss, L_{ocl_i} = $-\log \mathrm{P}_{occ}(\mathrm{y} = \mathrm{Q})$
  - Compute the difference of the losses, $L_{noccl} - L_{ocl_i}$

If $K = Q$, we answer the what parts of the image have contributed to correctly predict. If $K \neq Q$, we answer what parts of the image contributed to predict the incorrect class.

Exercise: Image Occlusion

The aim of this exercise is to understand occlusion. Occlusion involves running a patch over the entire image to see which pixels affect the classification the most.





Taylor series expansion

Any differentiable function $f(x)$ can be approximated as a series around x_0 as:

$$f(x) = f(x_0) + \frac{(x - x_0)^1}{1!} \frac{\partial f}{\partial x}\bigg|_{x_o} + \frac{(x - x_0)^2}{2!} \frac{\partial^2 f}{\partial x^2}\bigg|_{x_o} + \cdots$$

This function can be the logistic regression or even a complex neural network.

Note: Including more terms will improve the approximation.

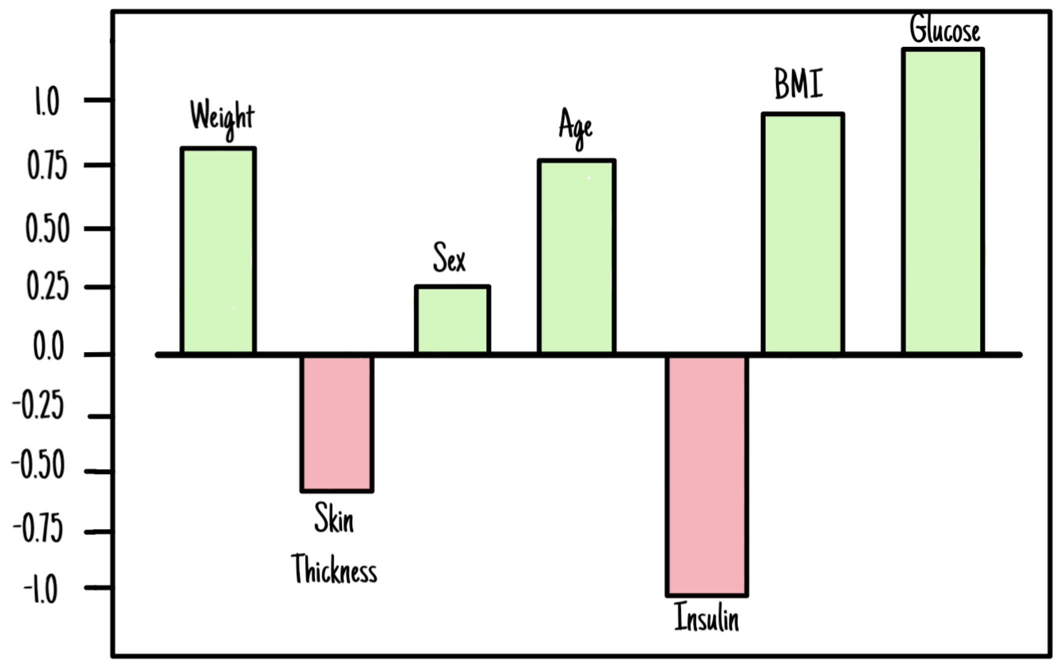
Visualizing Top Predictors by Input Gradient

Since the input gradient of an objective function for a trained model indicates which input dimensions has the greatest effect on the model decision at an input $\mathbf{x}$, we can visualize the "top predictors" of outcome for a particular input $\mathbf{x}$.

We can think of this as approximating our neural network model with a linear model locally at an input $\mathbf{x}$ and then interpreting the weights of this linear approximation.

$$\boxed{\begin{aligned} NN(\boldsymbol{x}) &\approx NN(\boldsymbol{x}_0) + w^T(\boldsymbol{x} - \boldsymbol{x}_0) \\ &\approx NN(0) + w^T(\boldsymbol{x}) \\ &\approx w^T\boldsymbol{x} + b \end{aligned}}$$

$$\boxed{w = \left.\frac{\partial NN}{\partial \boldsymbol{x}}\right|_{x_o}}$$



Thank you