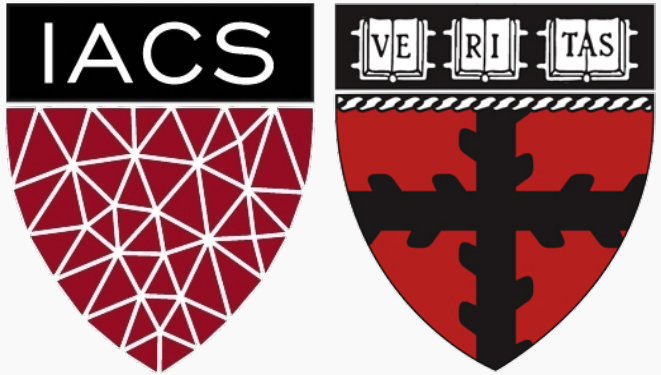
# Convolutional Neural Networks 3

## CS109B Data Science 2

Pavlos Protopapas, Mark Glickman, and Chris Tanner



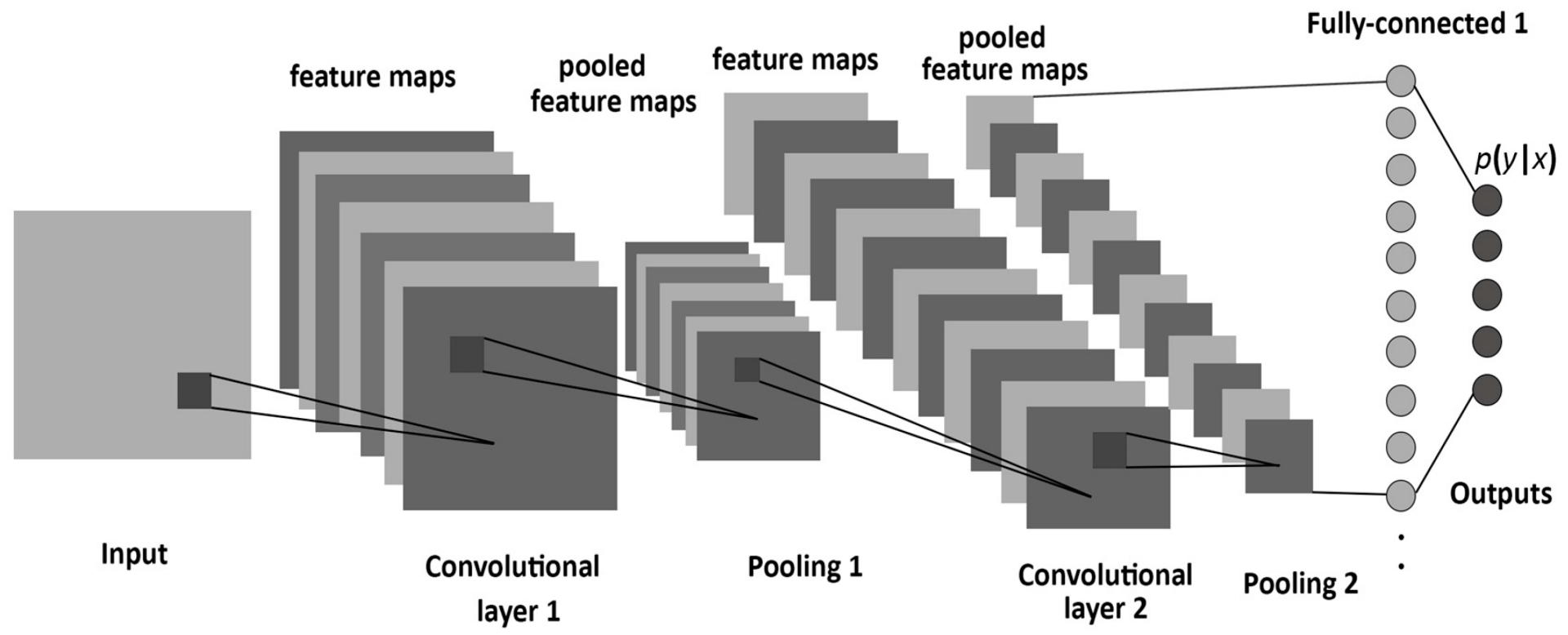
# Outline

1. Questions/Review
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. Weights and feature maps visualization

# Outline

1. **Questions/Review**
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. Weights and feature maps visualization

# From last lecture



# Outline

1. Review from last lecture
2. **Training CNNs**
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. Weights and feature maps visualization

# Training CNNs

Backprop works the same way.

Remember you can think of a kernel as a single neuron.

# 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 with the network's feature maps.

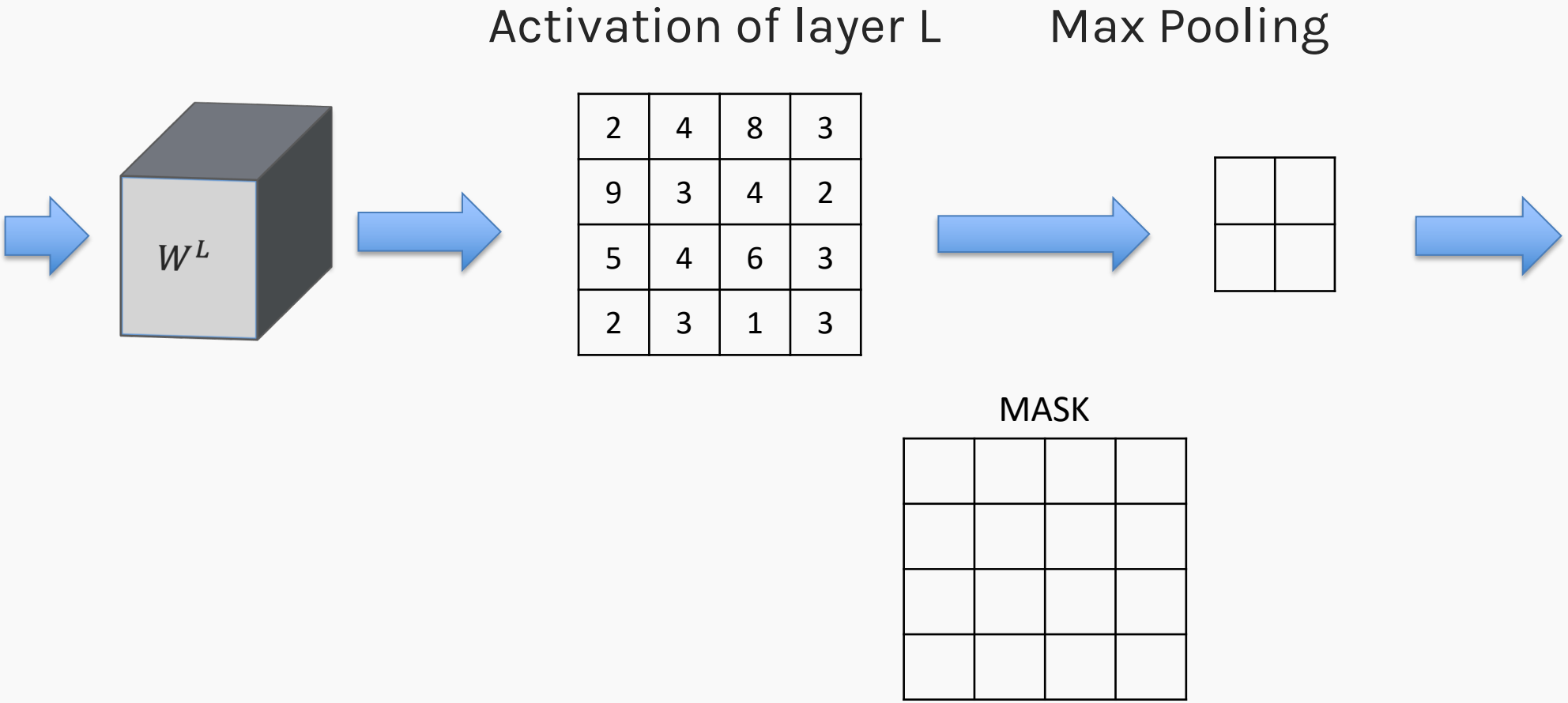
So, if you try adding dropout after a convolutional layer and get bad results, don't be disappointed! There doesn't appear that there is a good reason it *should* provide good results.

# Outline

1. Questions/Review
2. Training CNNs
3. **BackProp of MaxPooling layer**
4. Layers Receptive Field
5. Weights and feature maps visualization

# Backward propagation of Maximum Pooling Layer

rest of the network

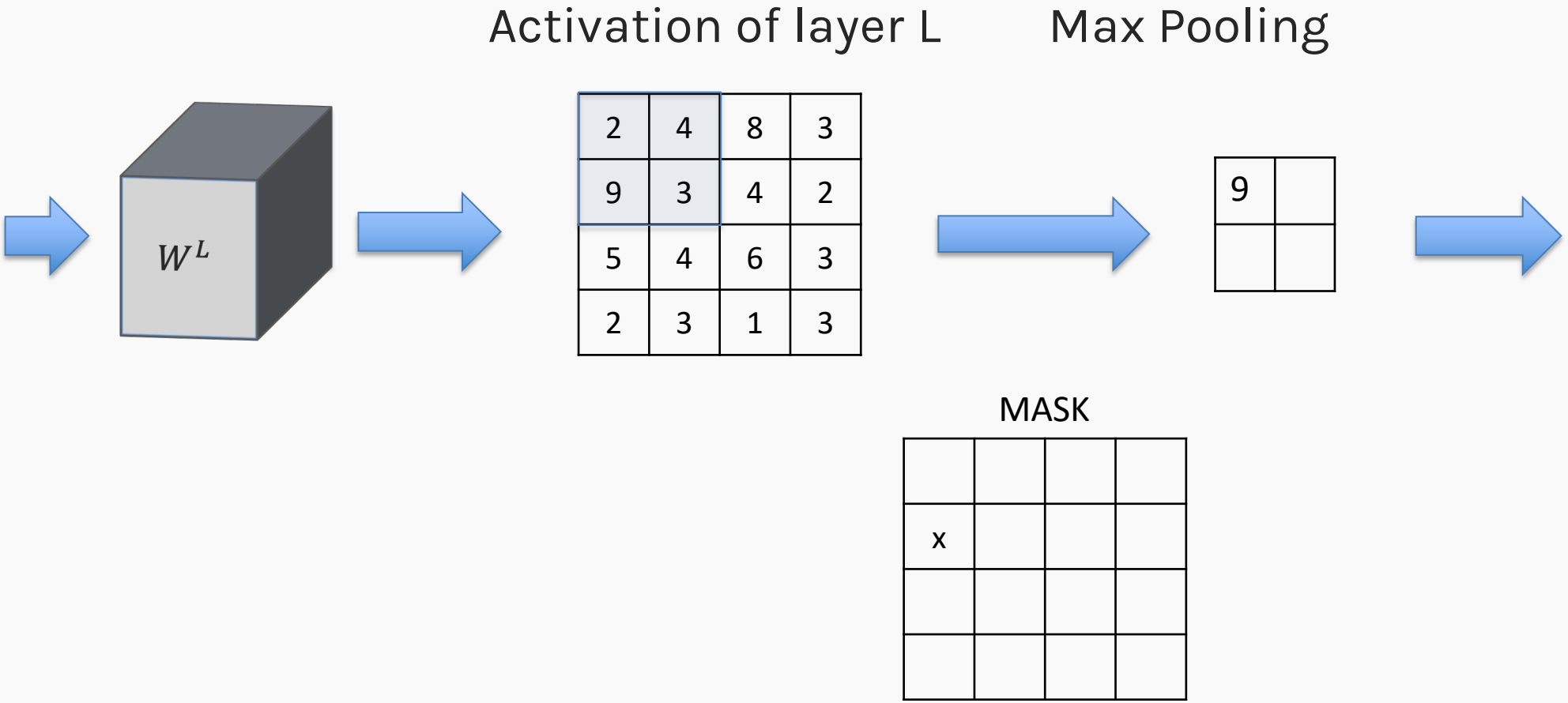



rest of the network

rward mode, 2x2 stride 2x2

# Backward propagation of Maximum Pooling Layer

rest of the network

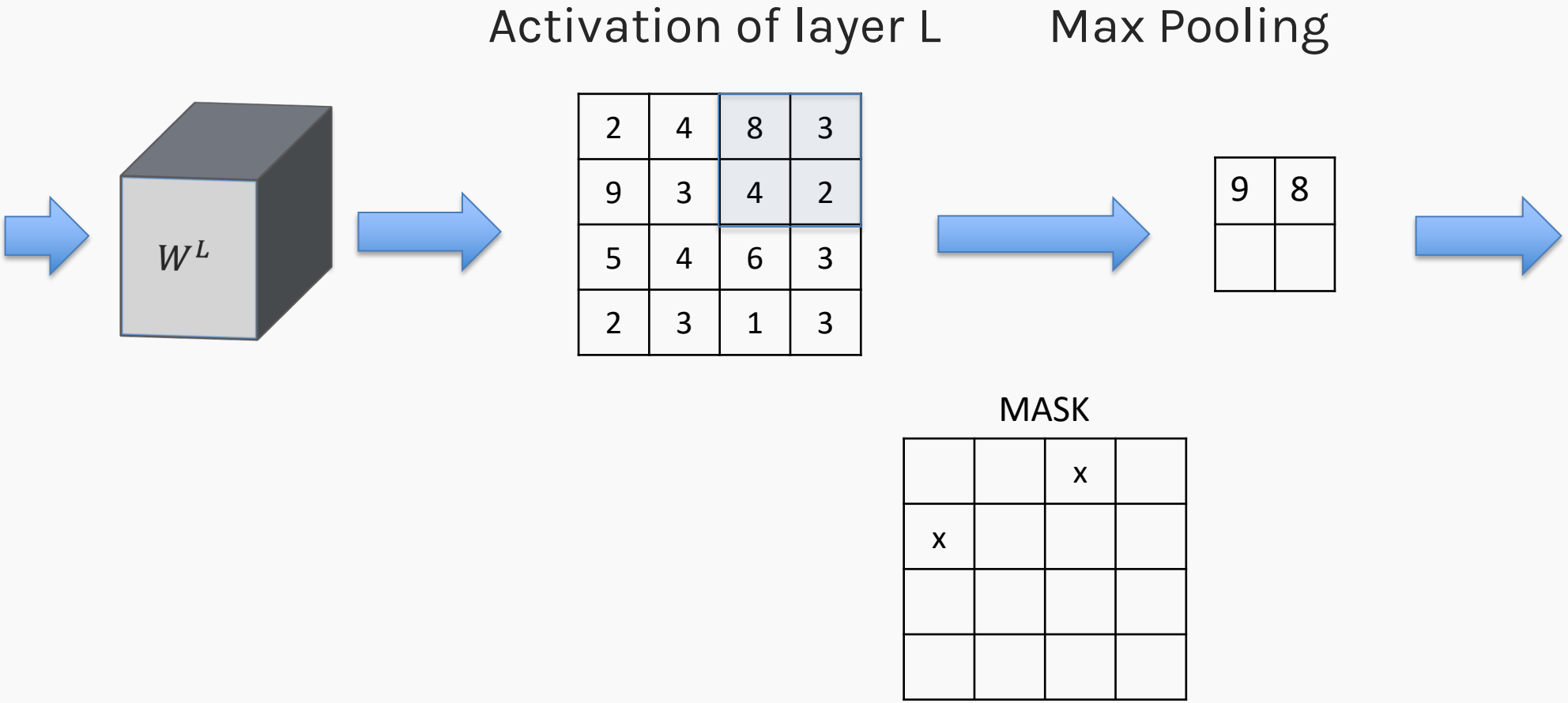



rest of the network

rward mode, 2x2 stride 2x2

# Backward propagation of Maximum Pooling Layer

rest of the network

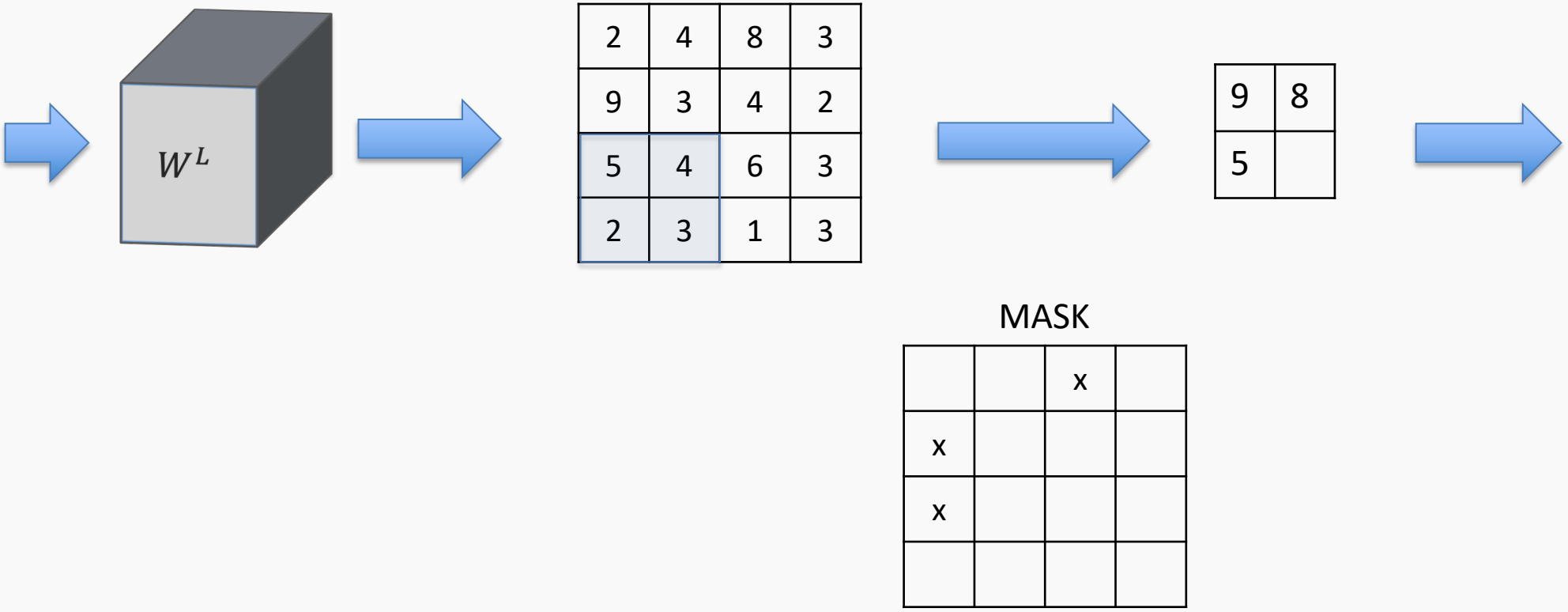



rest of the network

rward mode, 2x2 stride 2x2

# Backward propagation of Maximum Pooling Layer

rest of the network

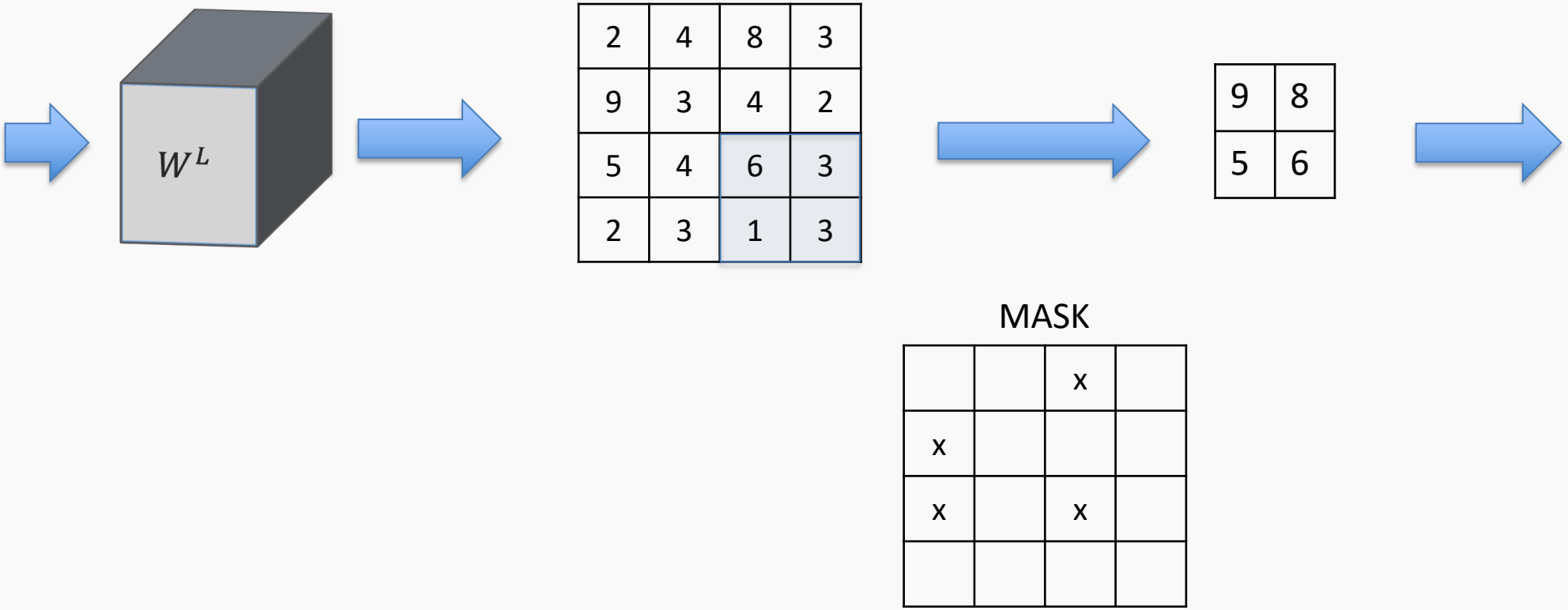



rest of the network

rward mode, 2x2 stride 2x2

# Backward propagation of Maximum Pooling Layer

rest of the network

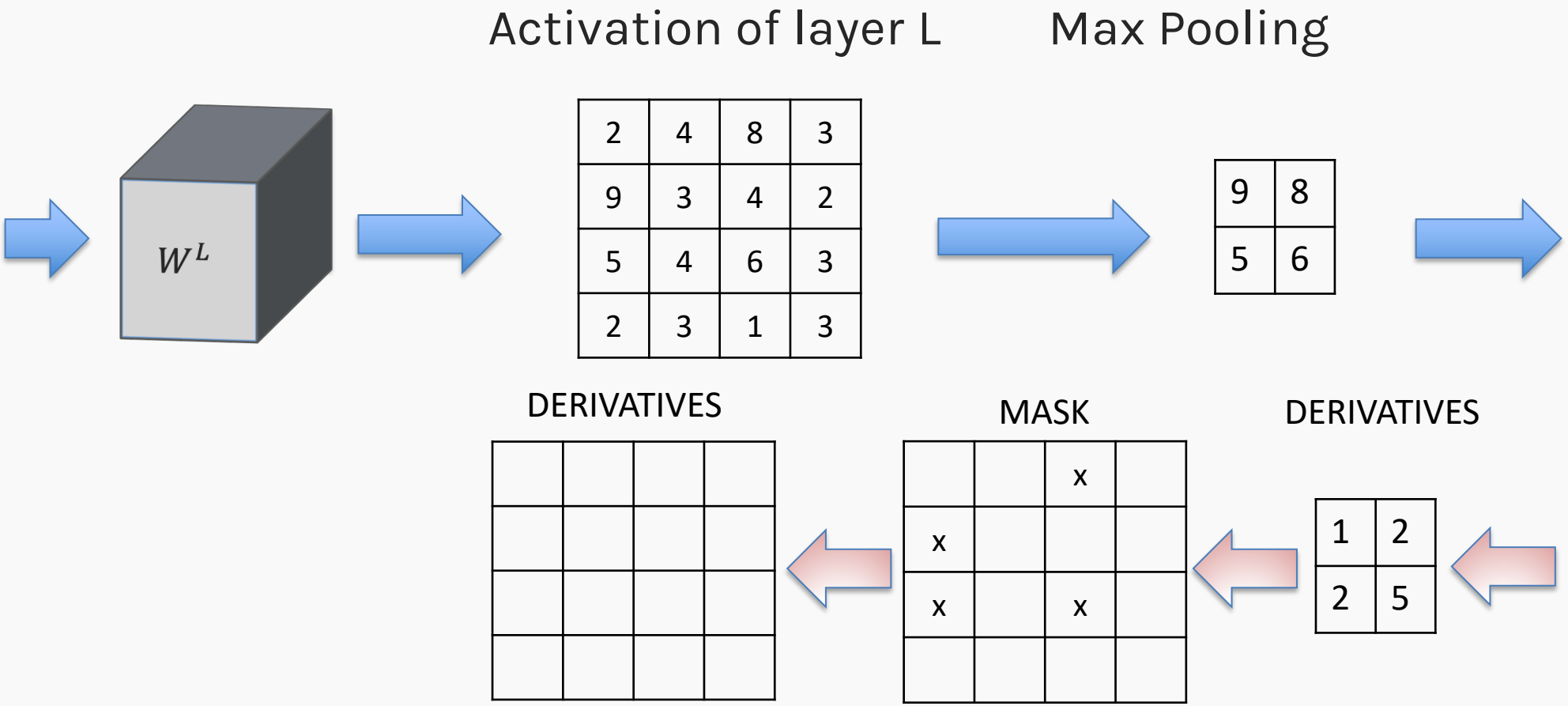



rest of the network

rward mode, 2x2 stride 2x2

# Backward propagation of Maximum Pooling Layer

rest of the network

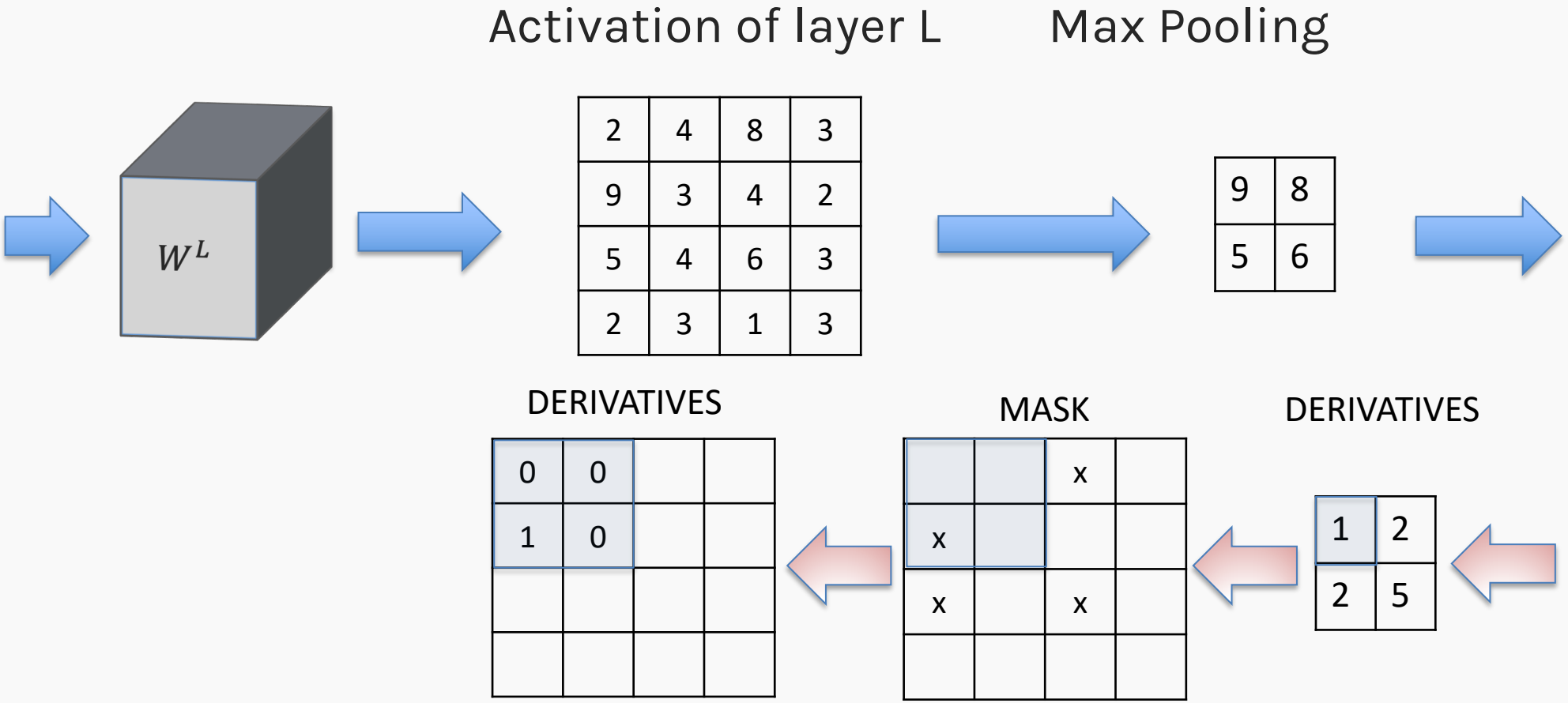



rest of the network

Reverse mode

# Backward propagation of Maximum Pooling Layer

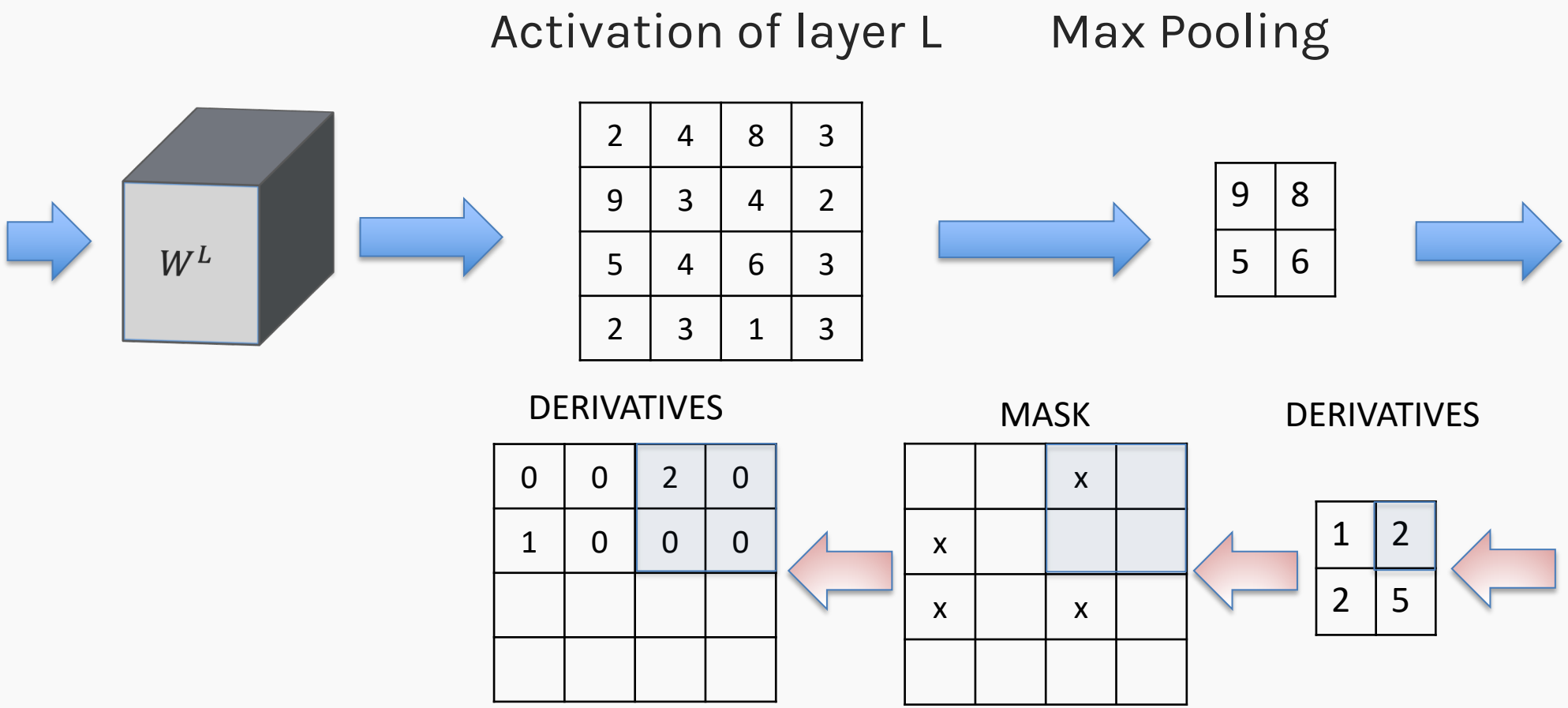
rest of the network




rest of the network

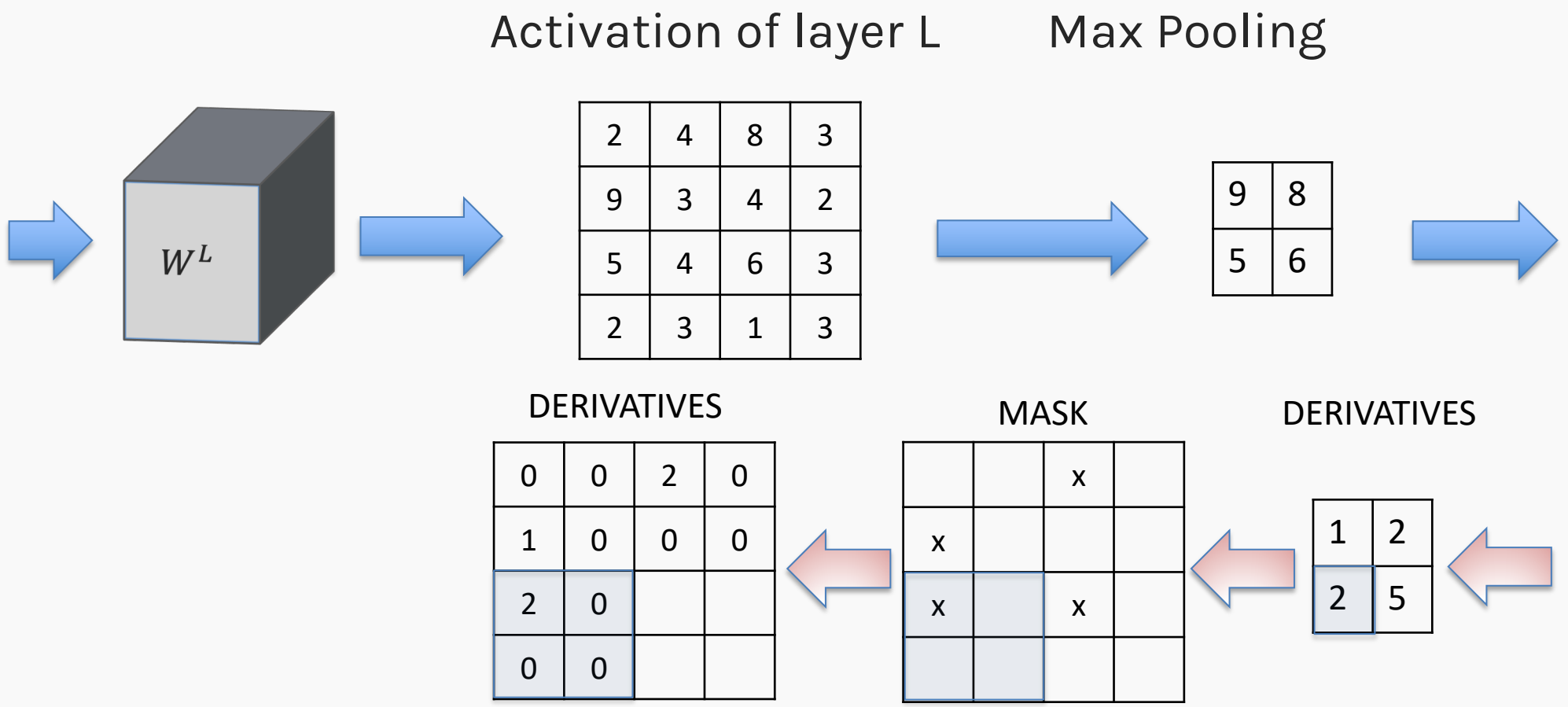
Reverse mode

# Backward propagation of Maximum Pooling Layer



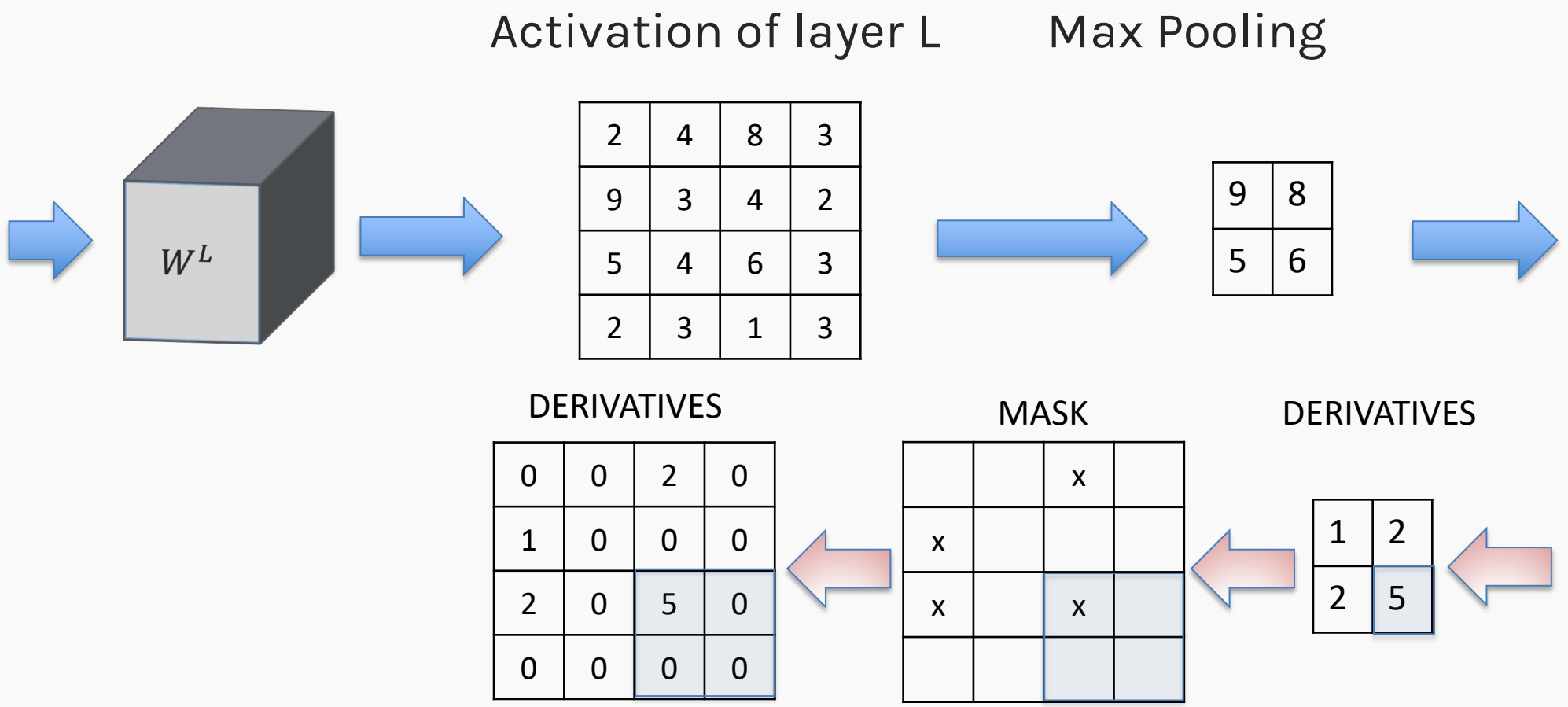

Reverse mode

# Backward propagation of Maximum Pooling Layer



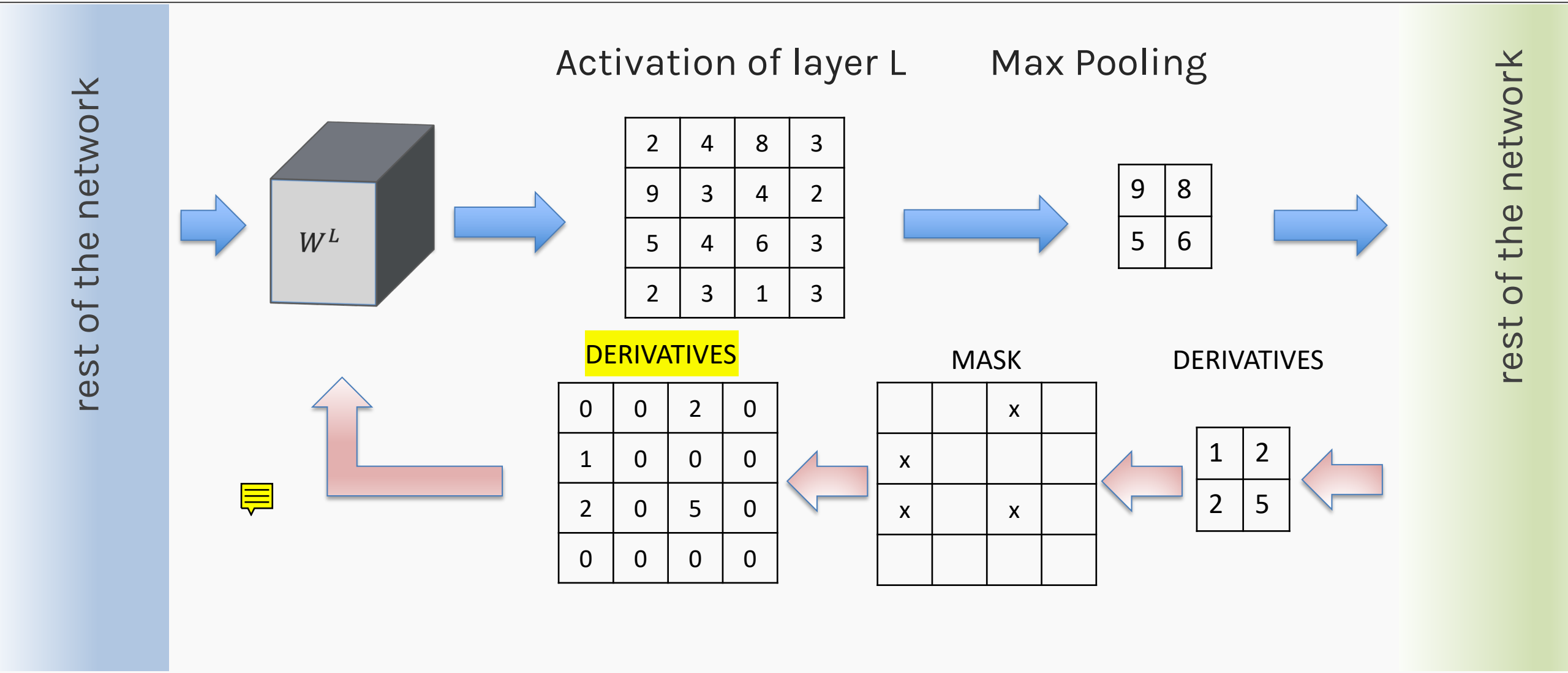

Reverse mode

# Backward propagation of Maximum Pooling Layer




Reverse mode

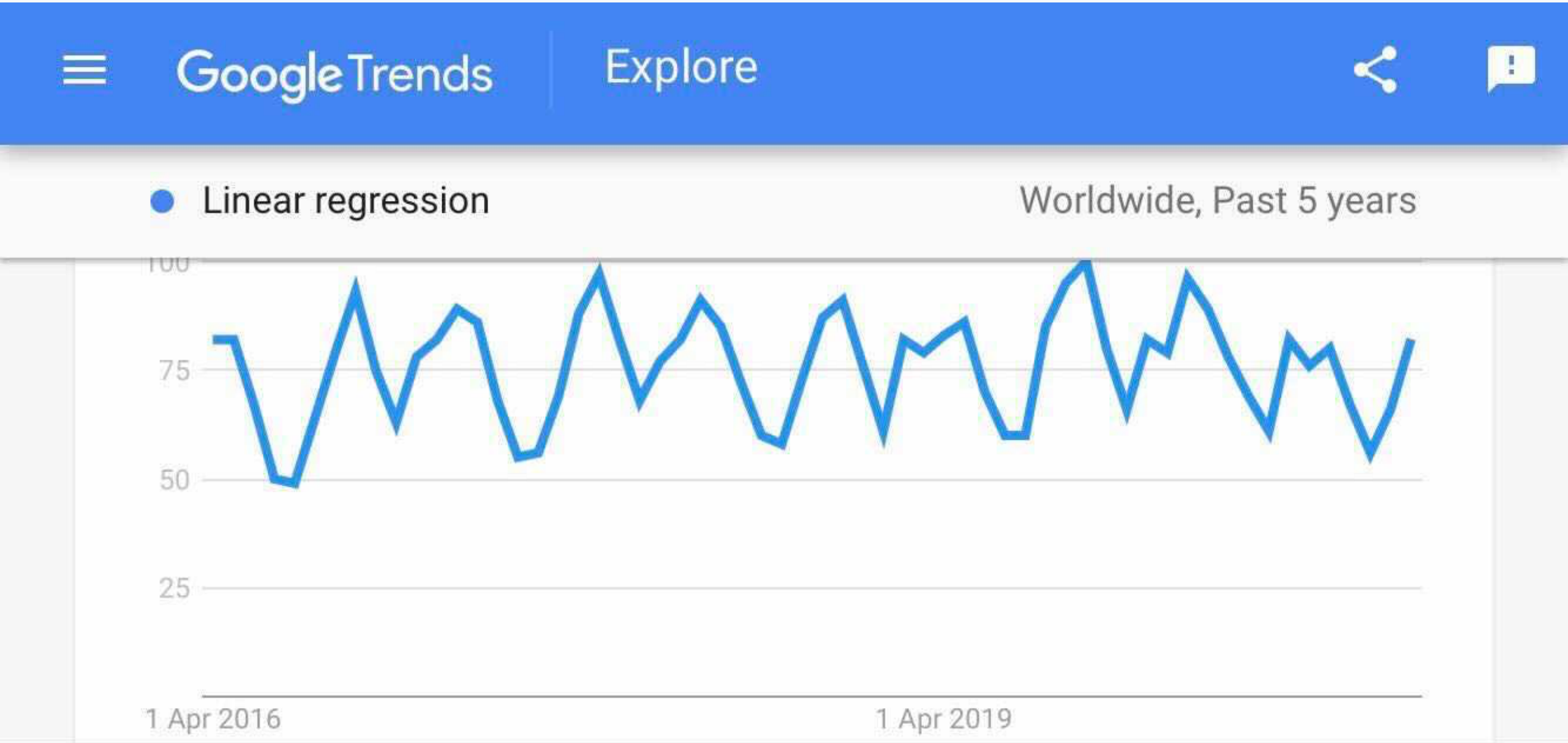
# Backward propagation of Maximum Pooling Layer




Reverse mode

# Outline

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. **Layers Receptive Field, dilated CNNs**
5. Weights and feature maps visualization


Google Trends
Explore
Linear regression
Worldwide, Past 5 years
75
50
25
1 Apr 2016
1 Apr 2019

# Layers Receptive Field

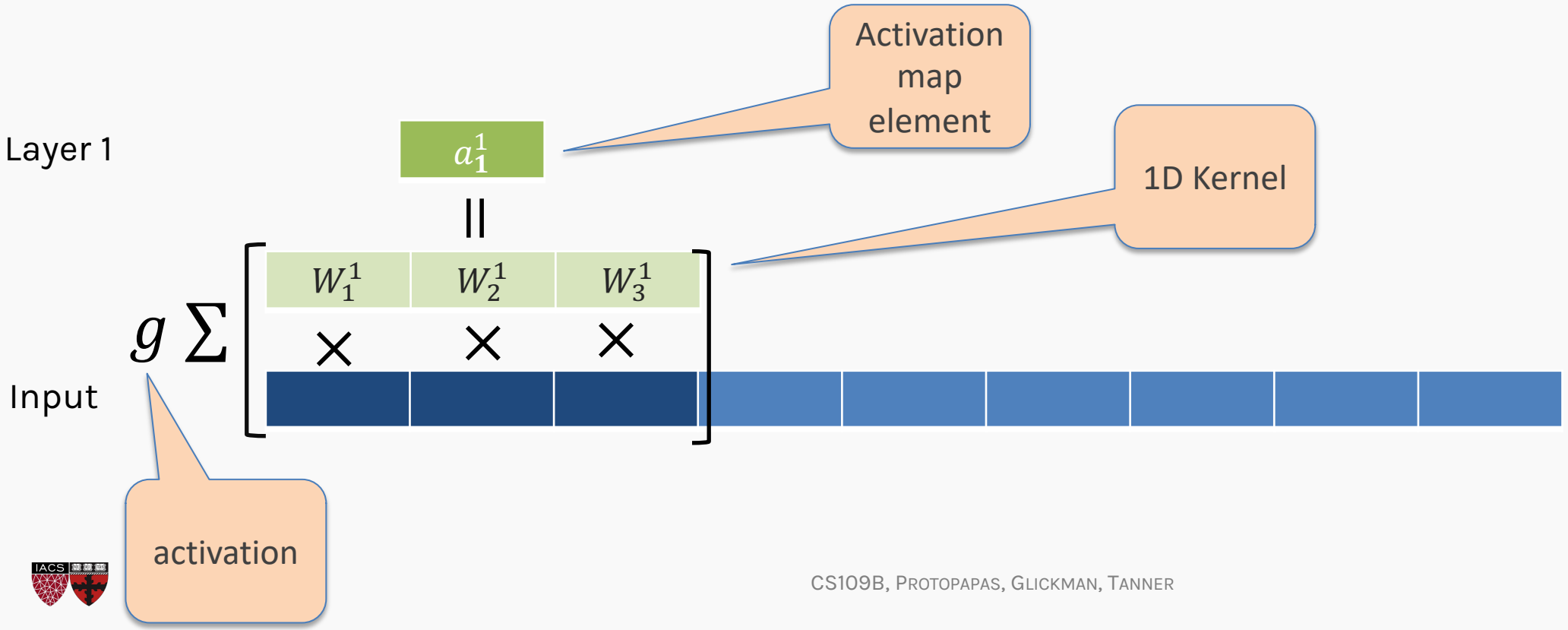
The **receptive field** is defined as the region in the input space that a particular CNN's feature is looking at (i.e., be affected by).

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).

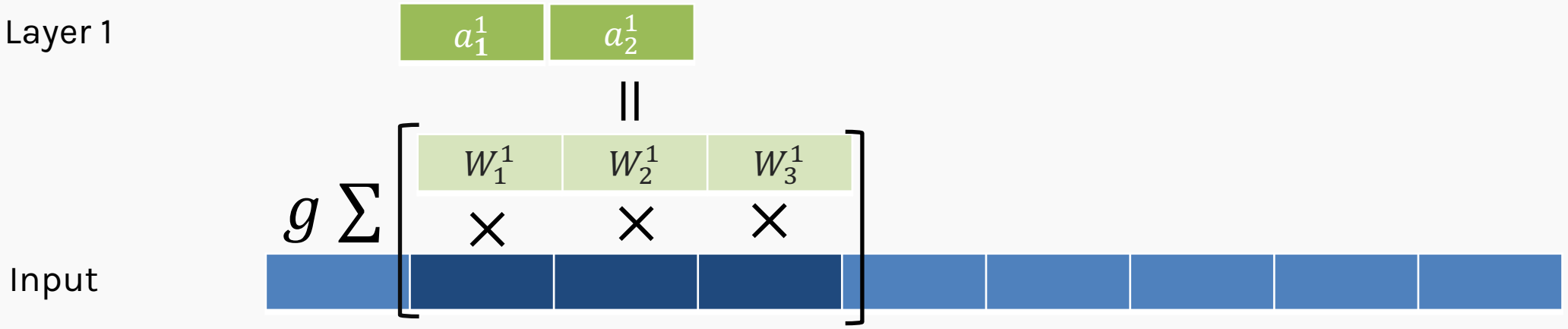
Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



# Layers Receptive Field

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

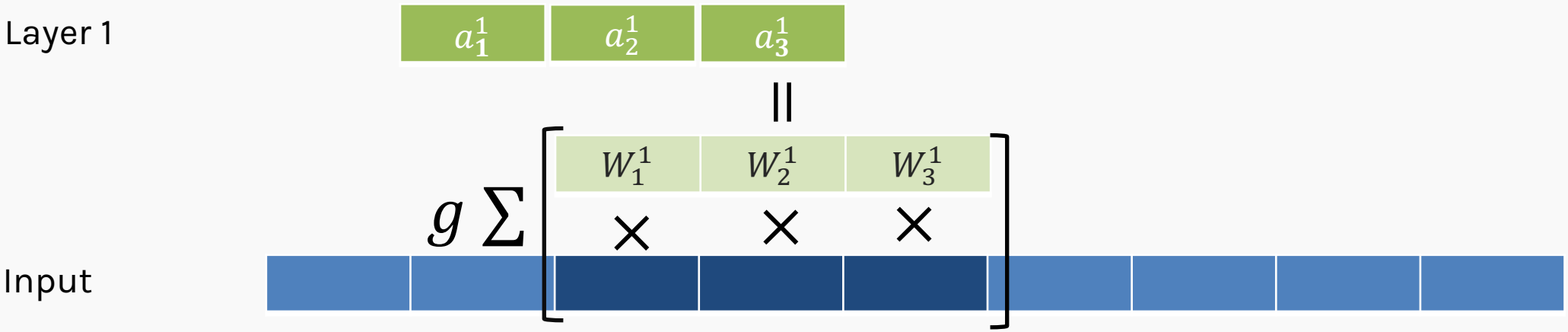
Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



# Layers Receptive Field

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

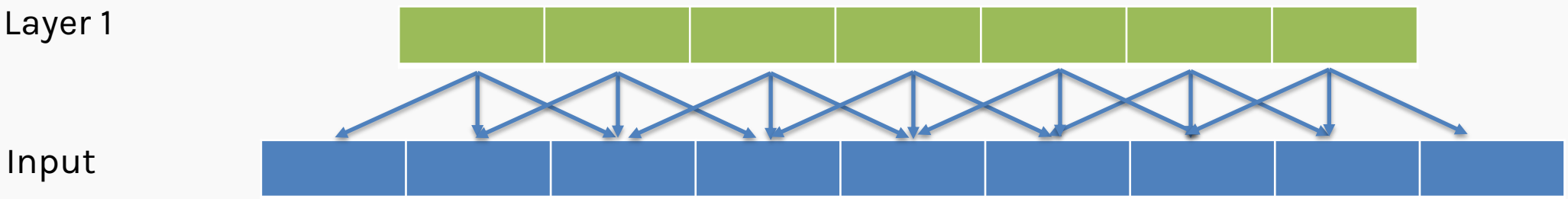
Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



# Layers Receptive Field

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

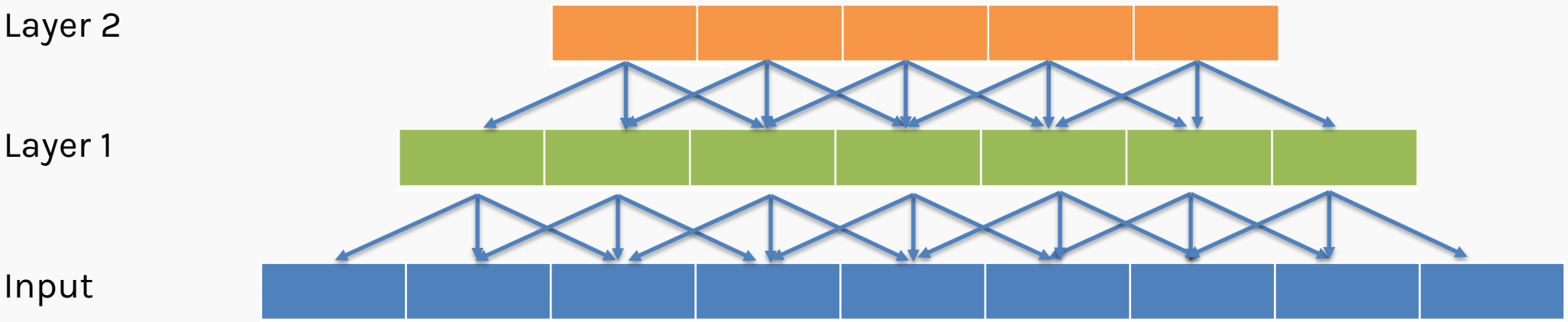
Let's look at the receptive field in 1D, no padding, stride 1 and kernel 3x1



# Layers Receptive Field

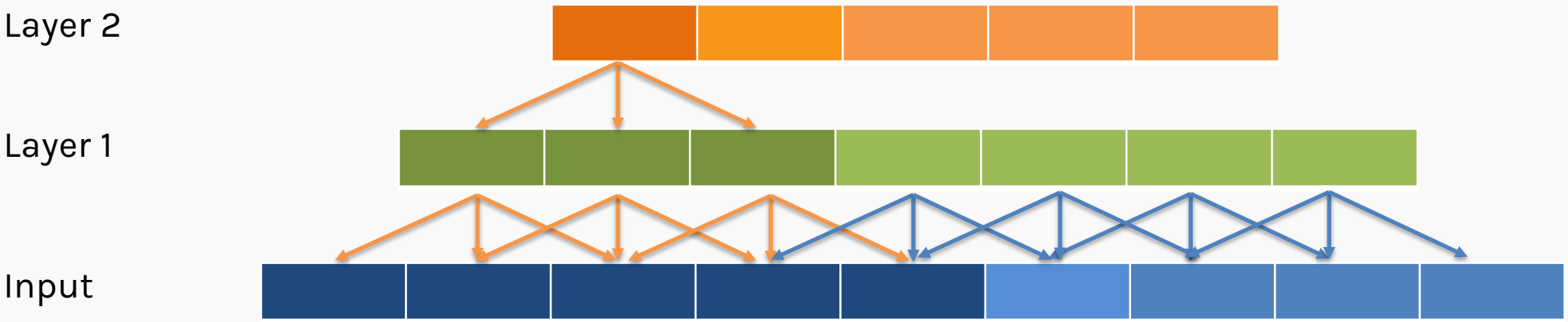
The **receptive field** is defined as the region in the input space that a particular CNN's feature is looking at (i.e. be affected by).

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



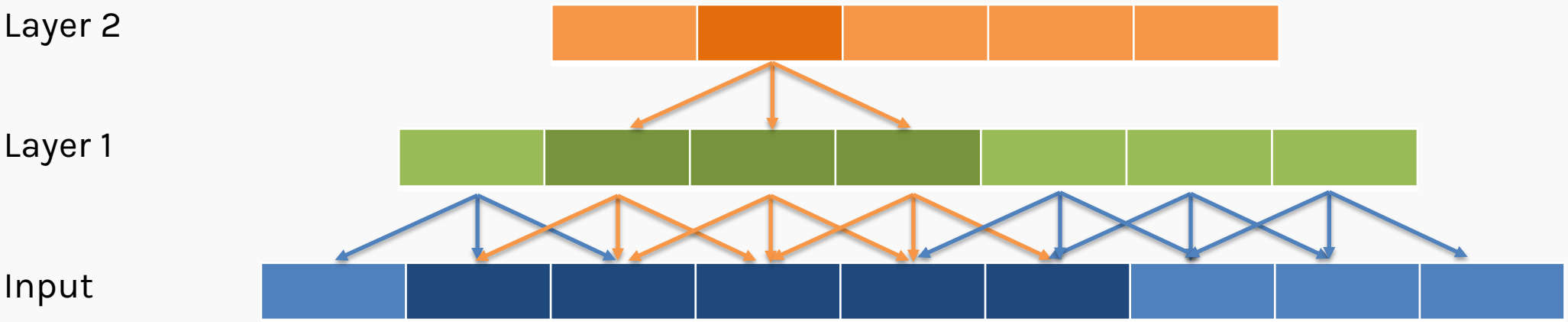
# Layers Receptive Field

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



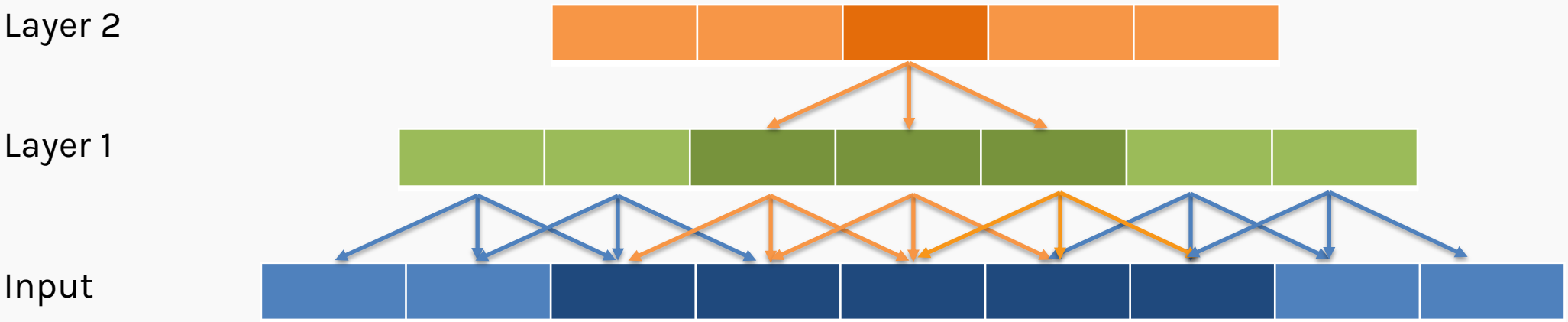
# Layers Receptive Field

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 layer's 2 is shown below.

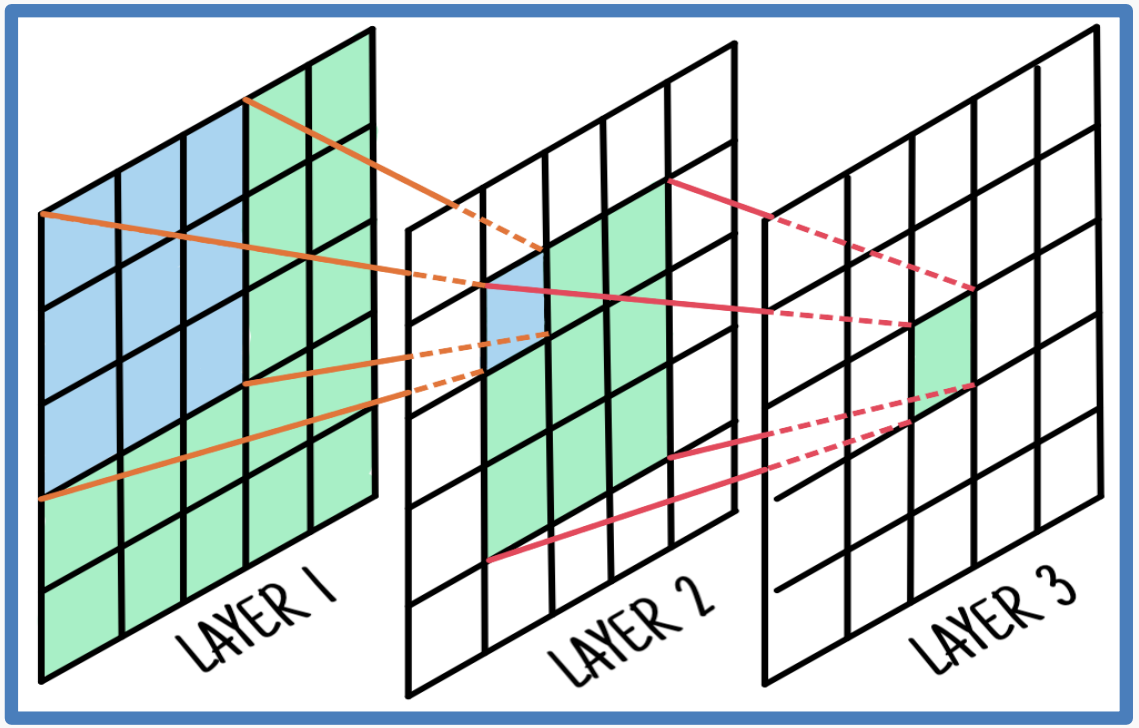
# Layers Receptive Field

In 2D, it works the same way.

The receptive field for layer $L, r_0$, can be calculated using the recursive formula:

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

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




**Note:** For every max-pooling layer, we multiply the receptive field by the window size (assuming stride is the same as the window size)

# Deep vs shallow network

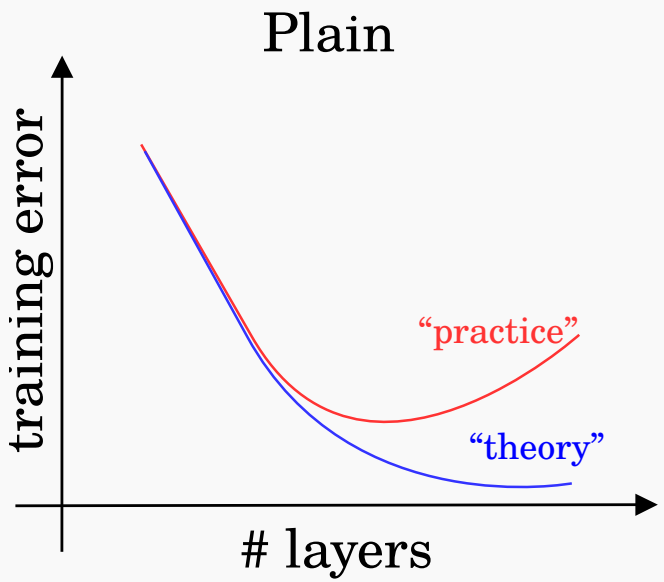
- We want a **large receptive** field before the dense layer.

# Deep vs shallow network

- We want a **large receptive** field before the dense layer.
- Going deeper resolves this issue, and we have seen that it does not increase the number of parameters, especially if we use max-pooling or stride>1.

# Deep vs shallow network

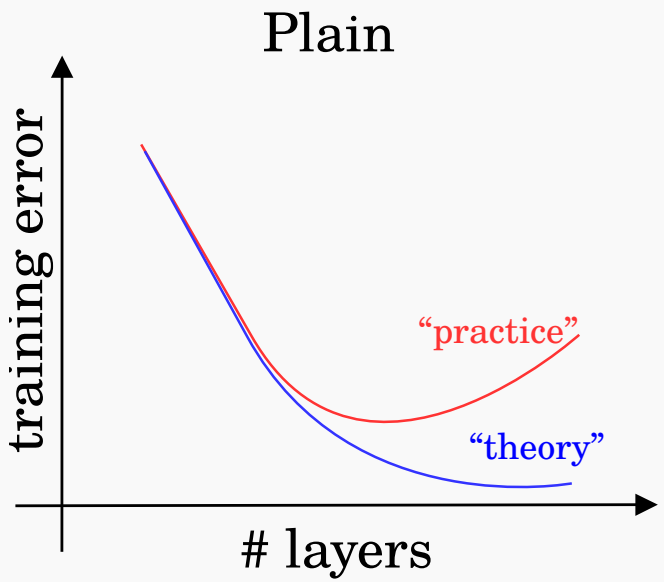
- We want a **large receptive** field before the dense layer.
- Going deeper resolves this issue, and we have seen that it does not increase the number of parameters, especially if we use max-pooling or stride>1.
- However, going deeper can lead to slow learning due to the vanishing gradient problem.

# Deep vs shallow network

- We want a **large receptive** field before the dense layer.
- Going deeper resolves this issue, and we have seen that it does not increase the number of parameters, especially if we use max-pooling or stride>1.
- However, going deeper can lead to slow learning due to the vanishing gradient problem.

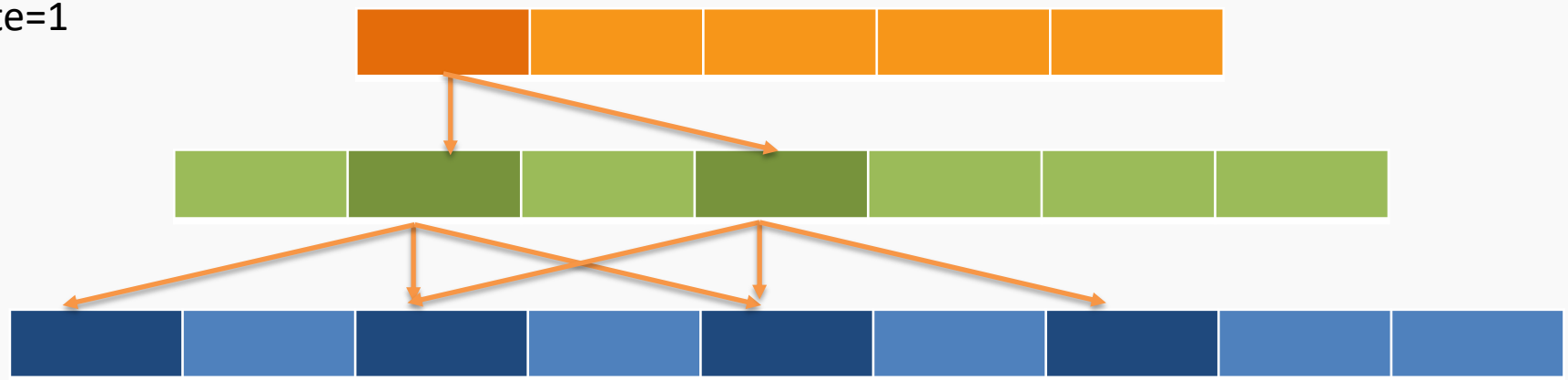
**Alternatively ...**

# 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

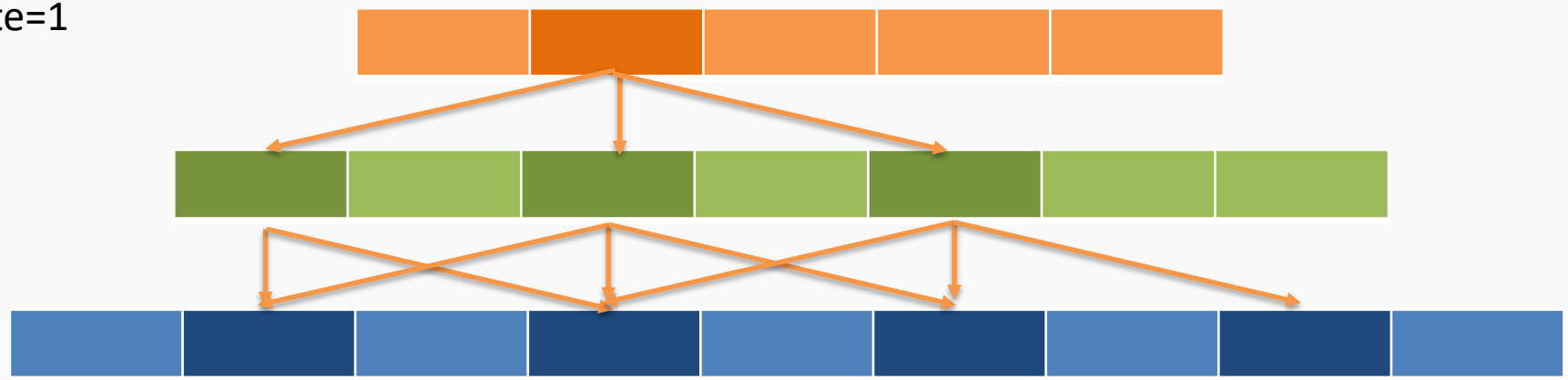


*Original Idea: Algorithme a trous*, an algorithm for wavelet decomposition (Holschneider et al., **1987**; Shensa, 1992)

# 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.

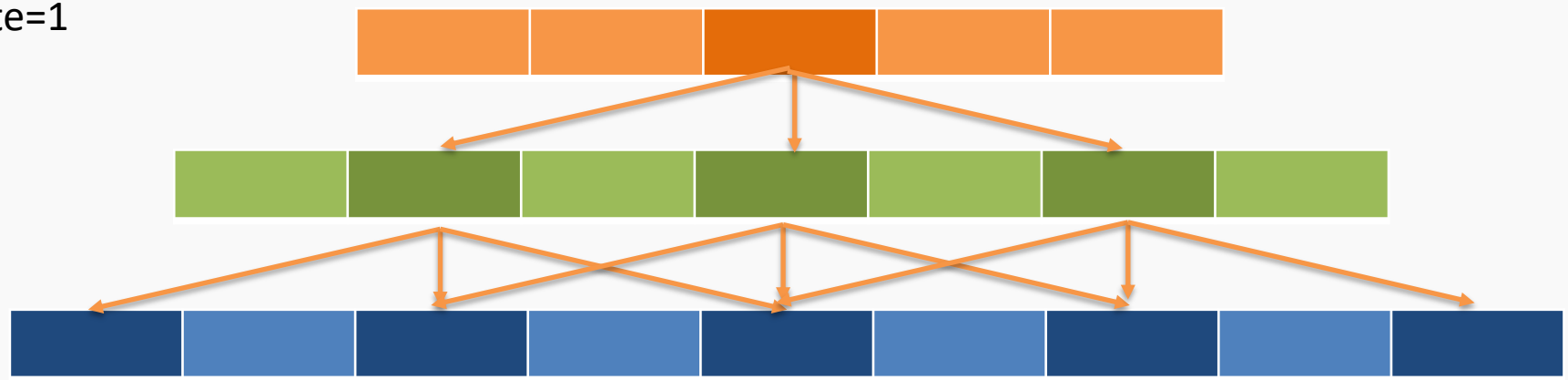
Dilate rate=1



# 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.

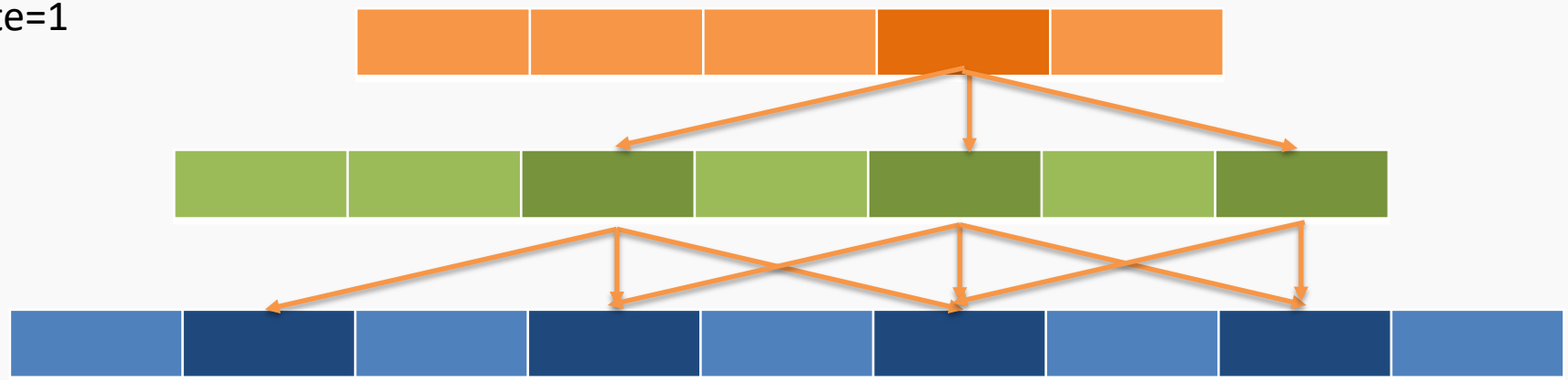
Dilate rate=1



# Dilated CNNs

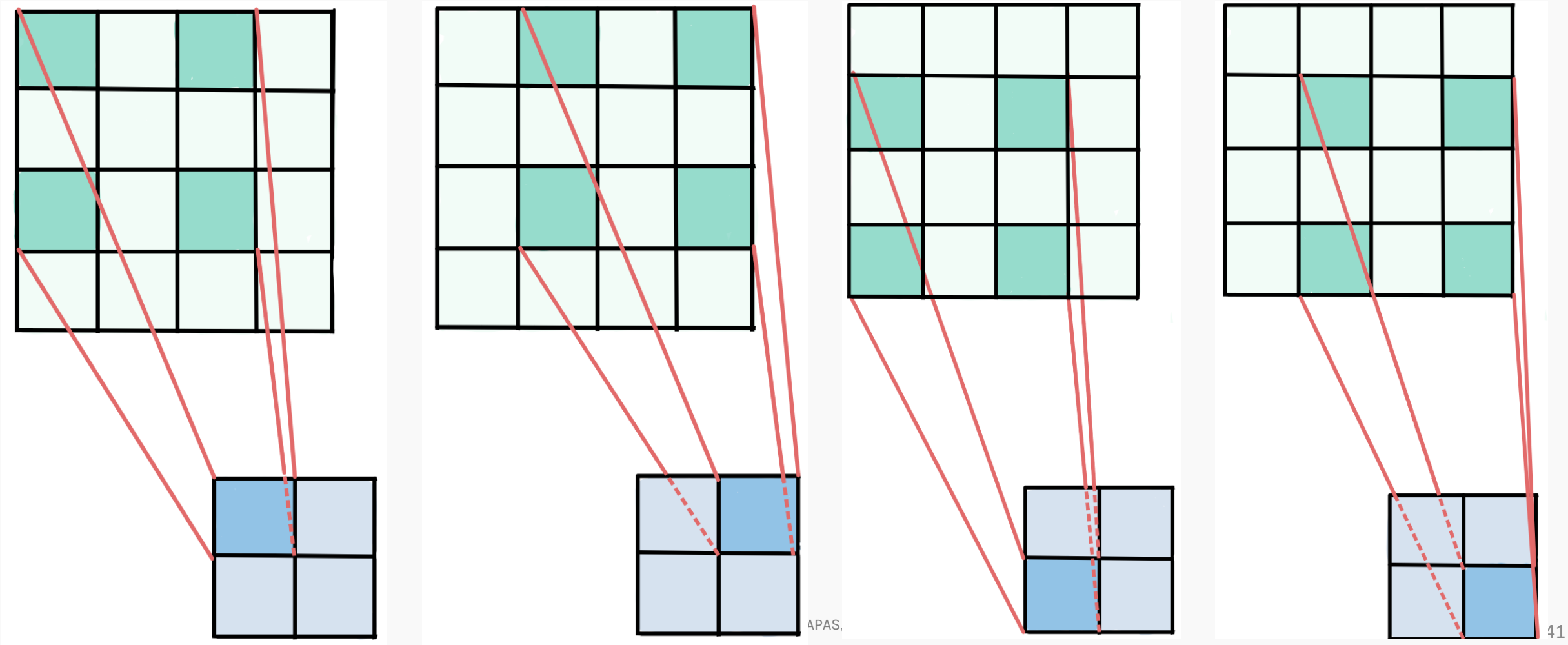
- We can "inflate" the receptive field by inserting holes between the kernel elements.
- These are called **Dilated Convolutions.**
- Dilation rate indicates how much the kernel is widened.

Dilate rate=1



# Dilated CNNs

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



Author: Our model is simple and easy to implement

Their implementation:

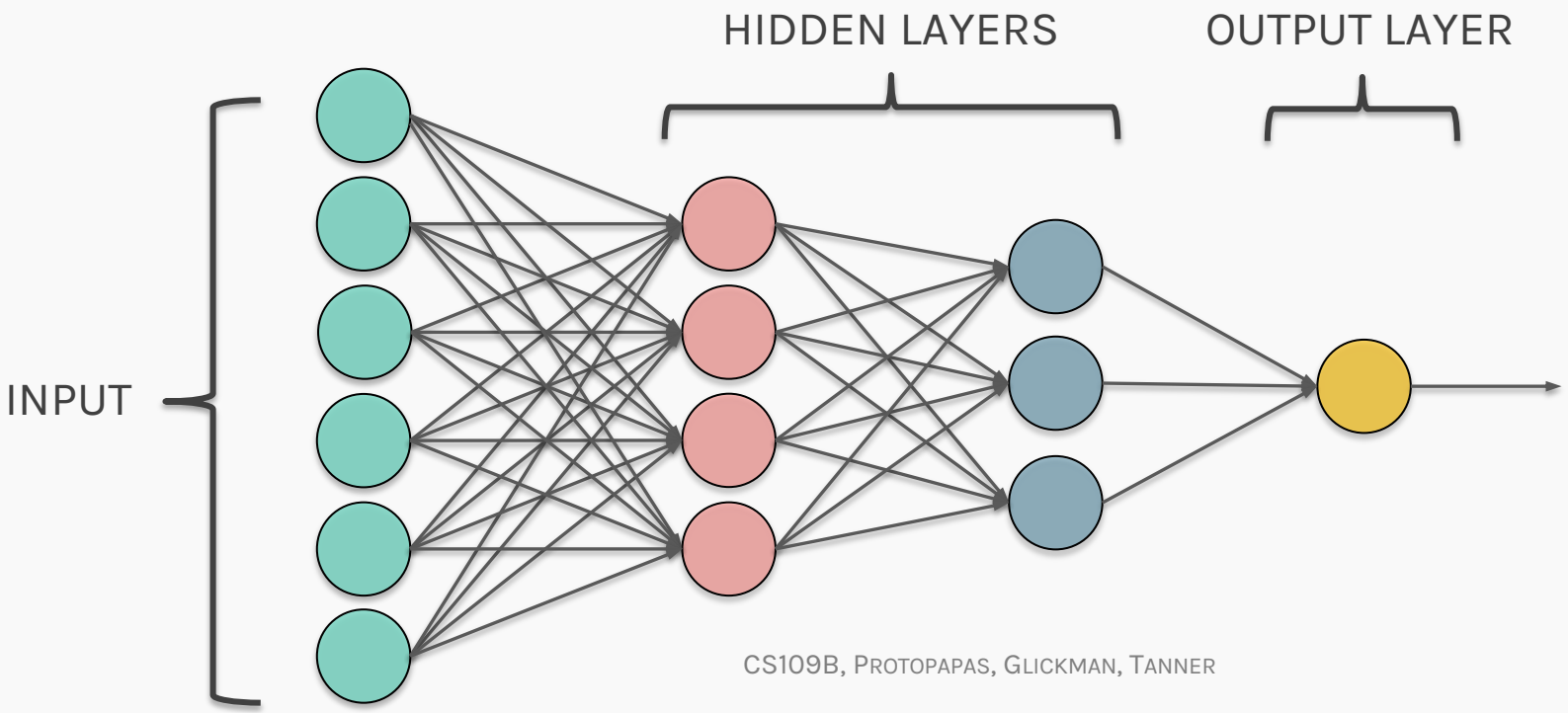
# Outline

1. Review from last lecture
2. Training CNNs
3. BackProp of MaxPooling layer
4. Layers Receptive Field
5. **Weights and feature maps visualization**

# What to Visualize for Neural Network Models?

For logistic regression, $p(y = 1 \mid \mathbf{w}, \mathbf{x}) = \text{sigmoid}\left(\mathbf{w}^\top \mathbf{x}\right)$ we can interrogate the model by printing out the weights of the model.

For a neural network classifier, $p(y = 1 \mid \mathbf{w}, \mathbf{x}) = \text{sigmoid}\left(\hat{g}_{\mathbf{W}}(\mathbf{x})\right)$ 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 $\hat{g}$ 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).

# Lessons for Visualization

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

***Why and for whom* to visualize**: for example

- are we visualizing to diagnose problems with our models?
- are we visualizing to interpret our models' meaningfulness?
- are we visualizing to teach deep learning concepts?

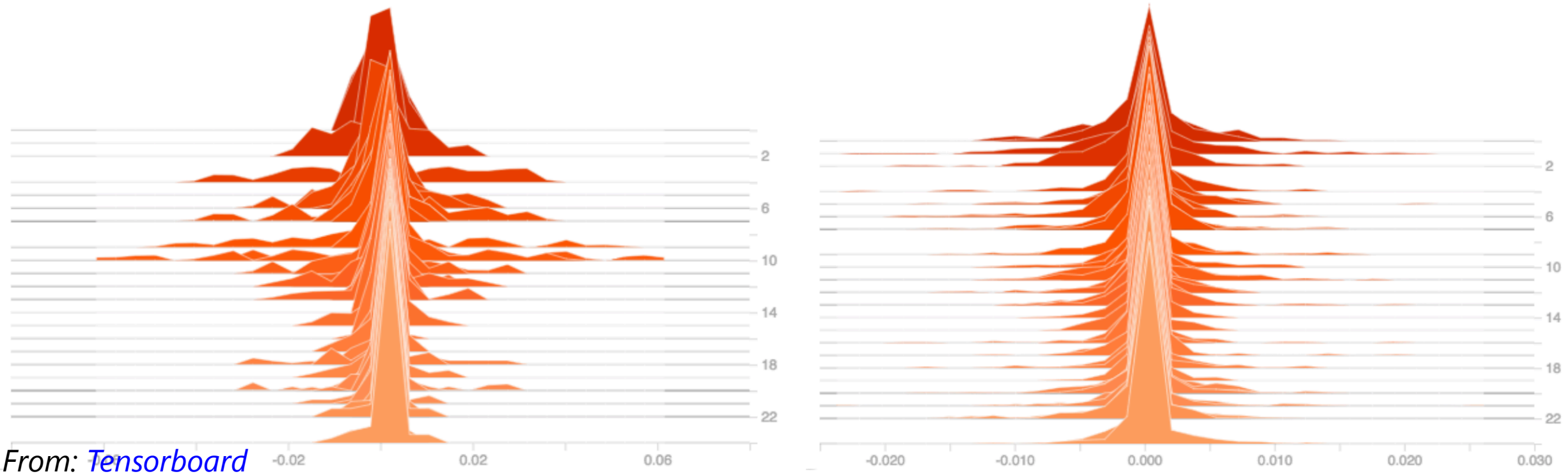
***What and how* to visualize**: for example

- do we visualize decision boundaries, weights of our model, activations, gradients, performance metrics?

# Activations and Weights

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 activations in two hidden layers over the course of training. What problems do we see?

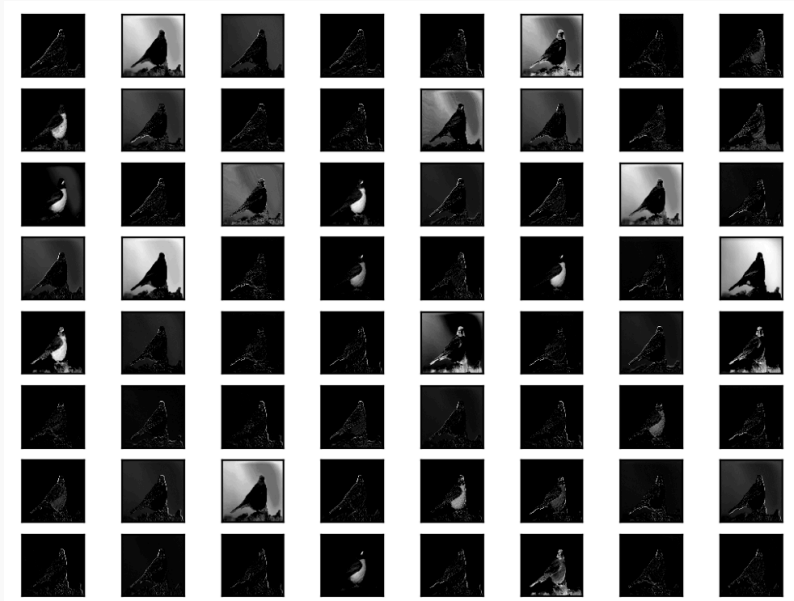


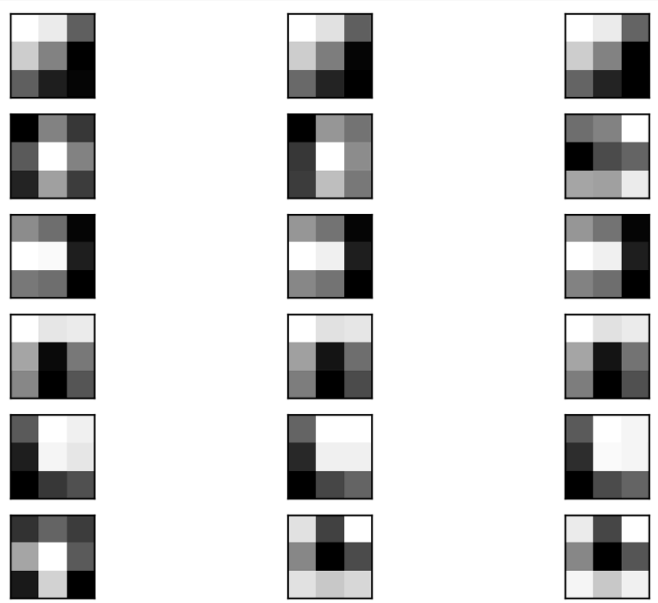

*From: Tensorboard*

# What to Visualize for CNNs?

The first things to try are:

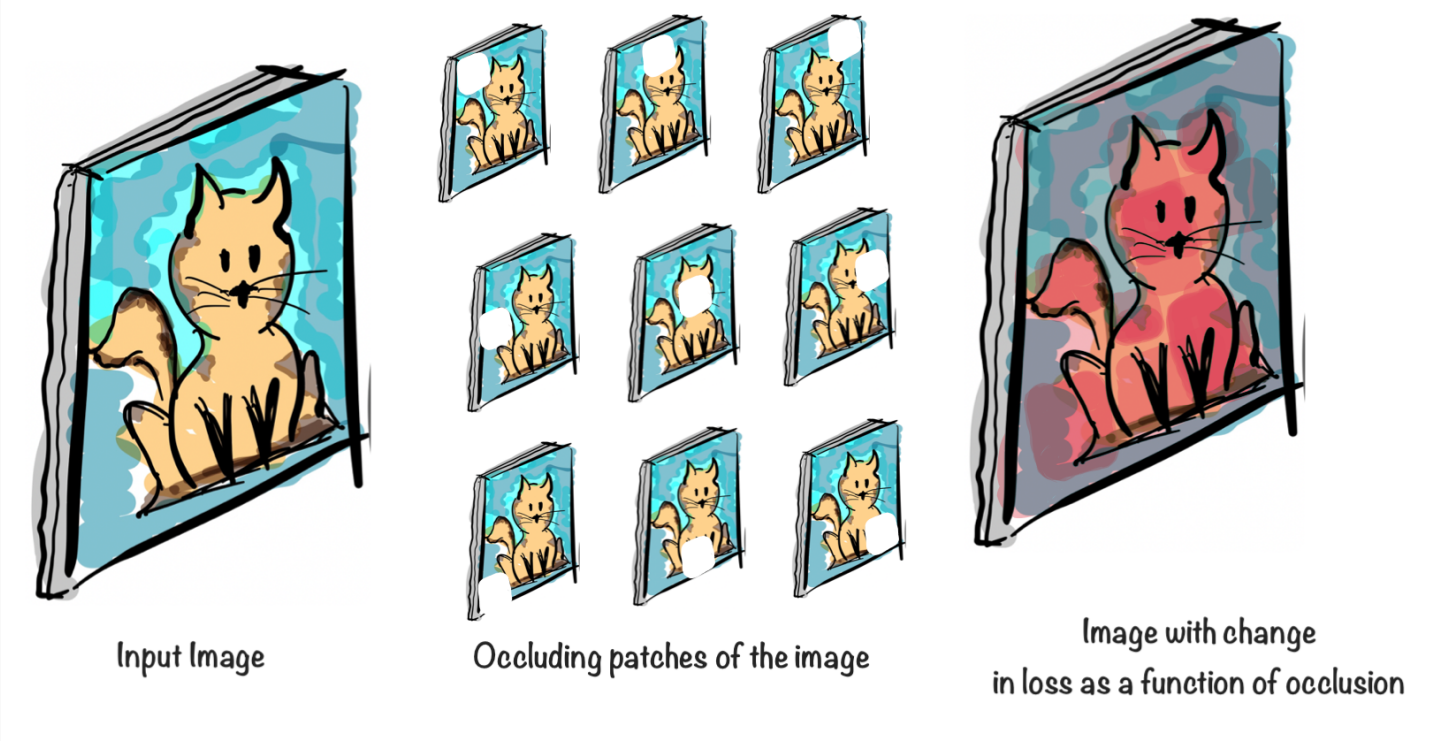
1. visualize the result of applying a learned filter to an image
2. visualize the filters themselves:



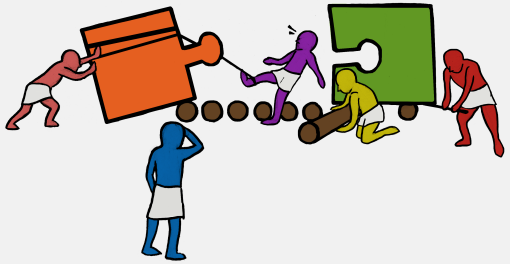


# Occlusion methods

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

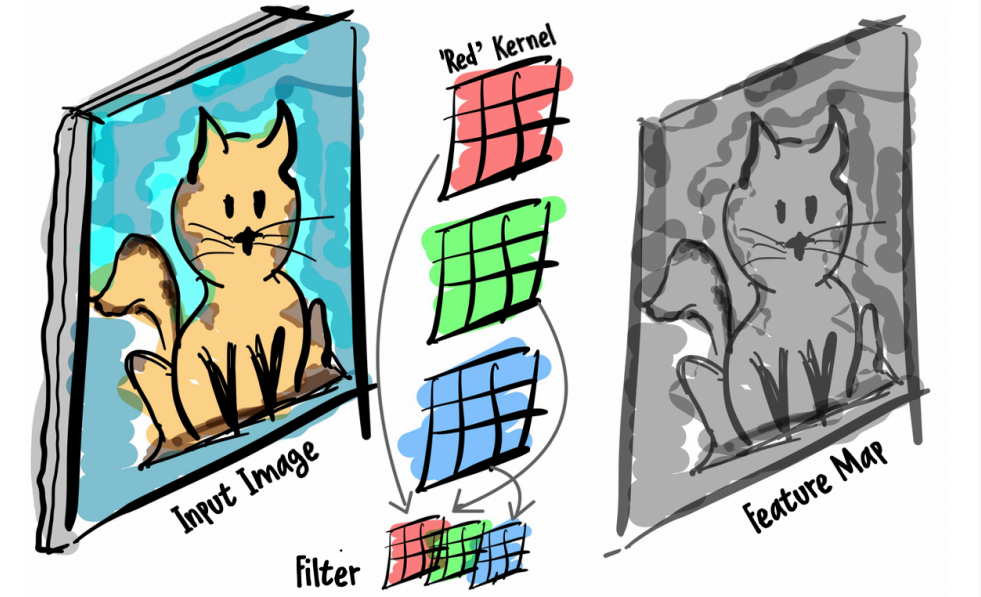


# Exercise: Investigating CNNs

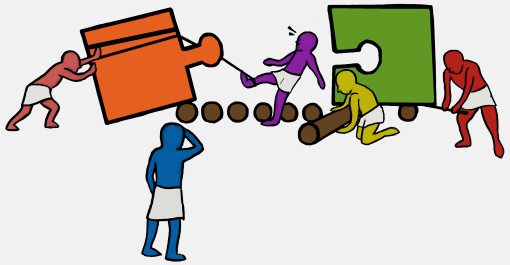


The goal of the exercise is to investigate the building blocks of a CNN, such as kernels, filters, and feature maps using a CNN model trained on the CIFAR-10 dataset.

- Use a pre-trained model trained on the CIFAR-10 dataset
- Investigate the kernels & filters
- Investigate the feature maps & activation maps



# 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.