Lecture 6: Convolutional Neural Networks and Computer Vision

KI-Workshop
(HFT Stuttgart, 8-9 Nov 2023)

Michael Mommert
University of St. Gallen (soon-to-be HFT Stuttgart)

Today's lecture

Exam information!

Convolutions

Convolutional Neural Networks

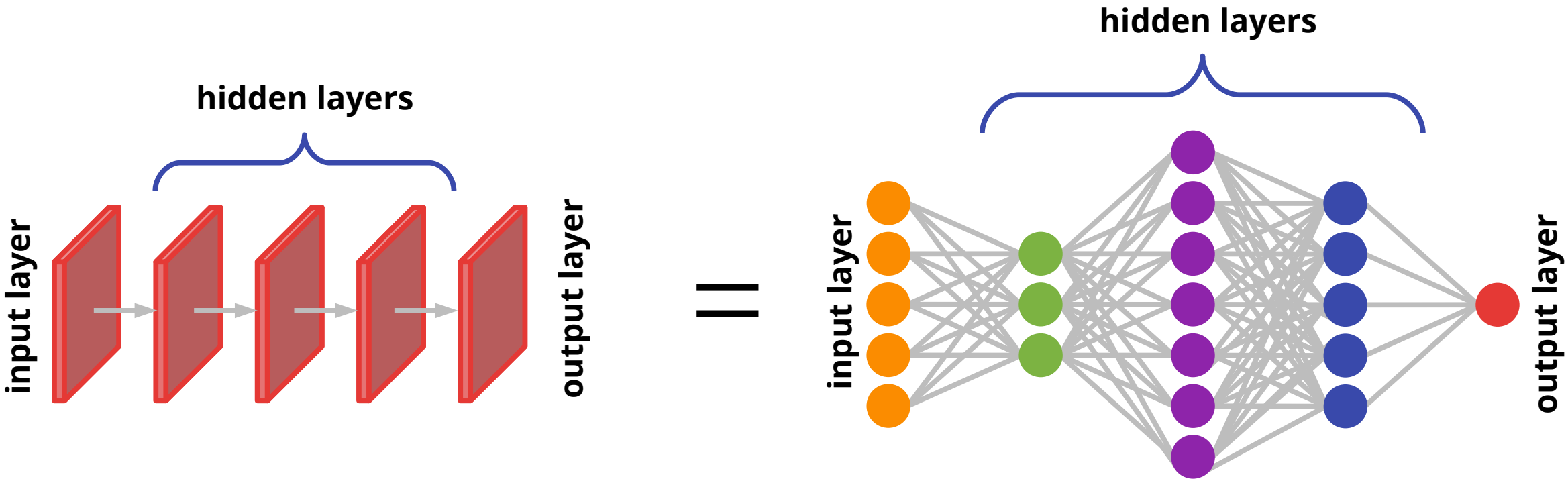
Semantic Segmentation

Object Detection

Exam information

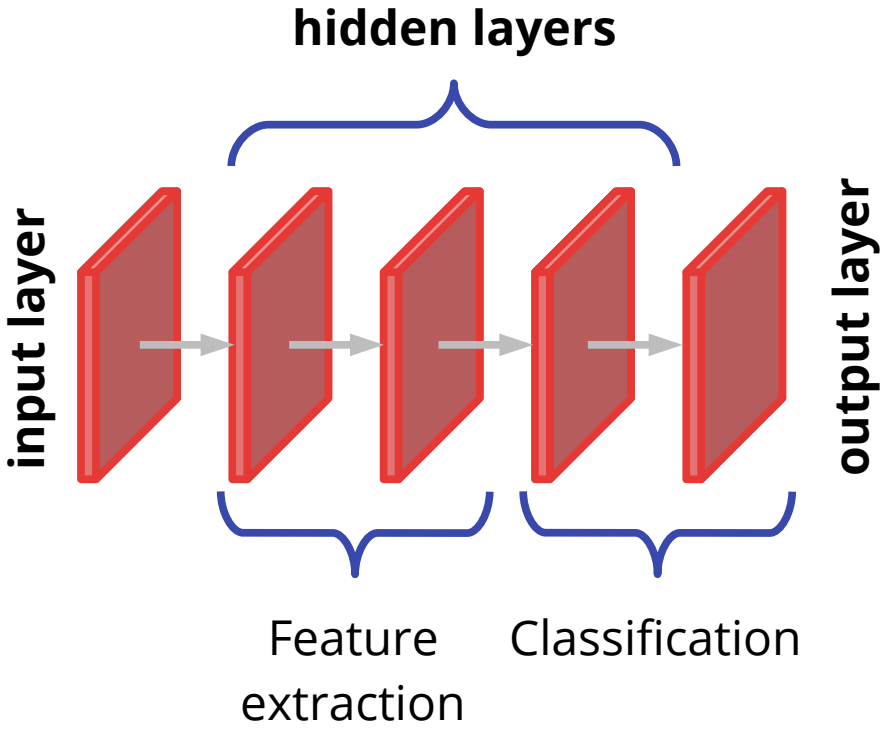
- Decentral exam for everyone!
- 19 December 2022, 14:15, in room 01-U123
- Written exam
- 120 min
- All course materials (lecture slides) are relevant for the exam (**except for the guest lecture!**)
- No additional reading is required for the exam
- Coding will not be part of the exam
- Understanding > memorizing
- Sample exam in the syllabus (canvas); Q&A session on 12 Dec (lecture 11) – byoq

Reminder: Fully connected neural networks

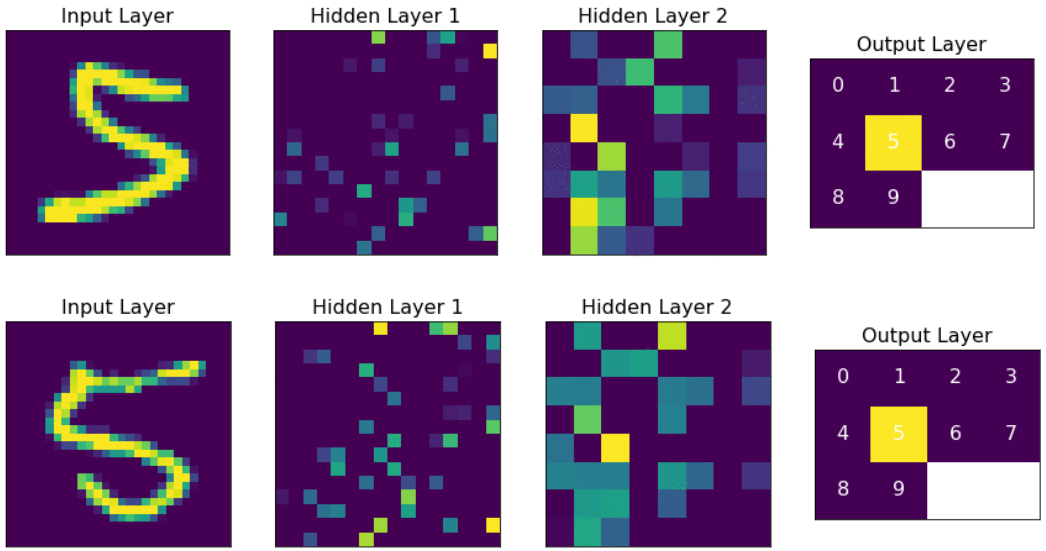


We saw in the previous lab course that fully connected networks are able to classify images.

Reminder: Fully connected neural networks

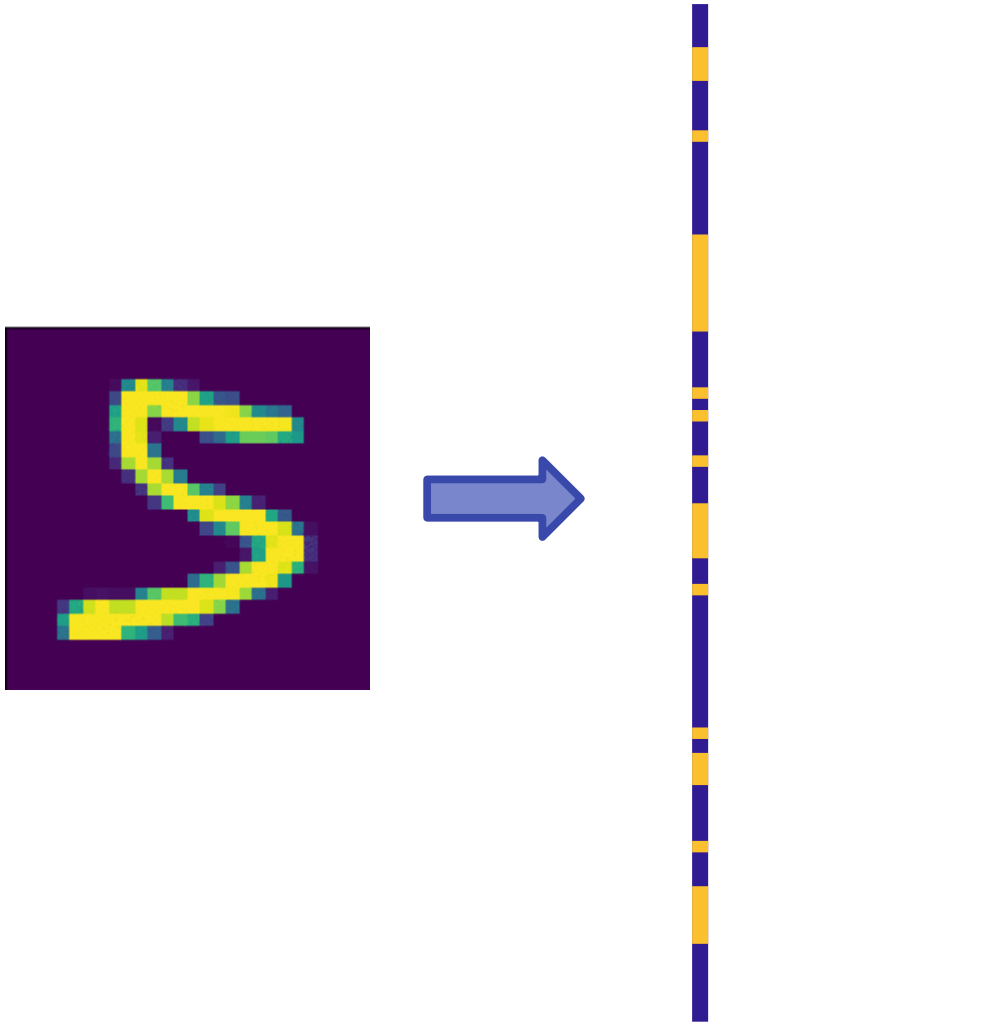


Learned patterns are stored in the weight parameters of the hidden layers' neurons.



We also learned that early hidden layers learn features while later layers act as the actual classifier that acts upon the learned features. This combination is often referred to as **end-to-end-learning**.

Fully connected neural networks are inefficient on image data



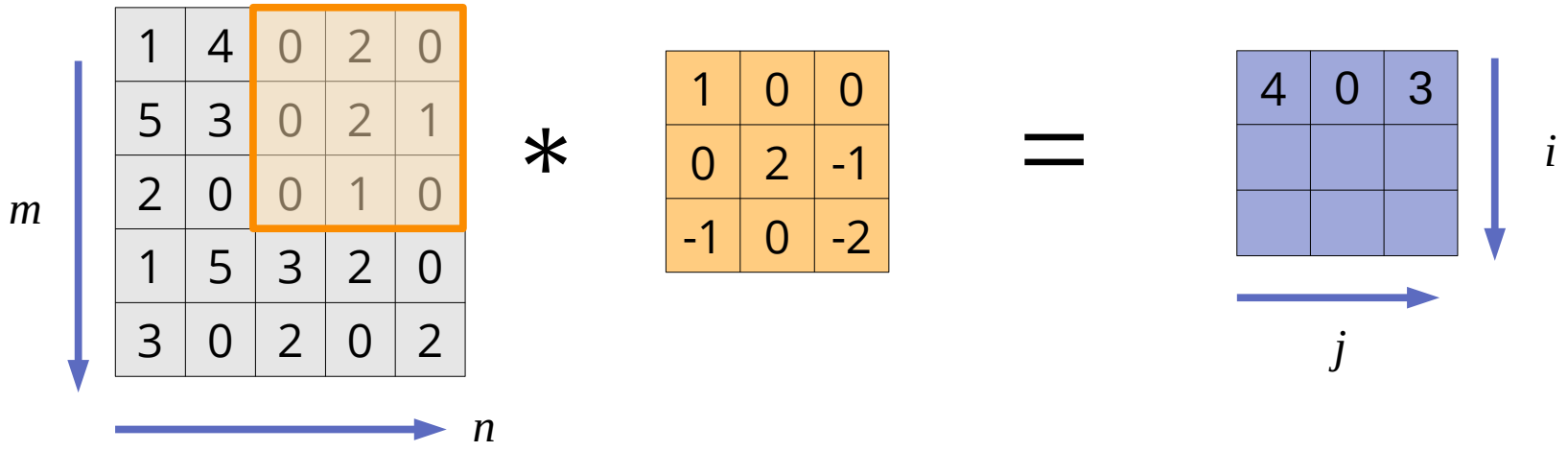
The input to our fully connected neural network is a linearized image:

The network is **unable to leverage spatial information** from the linearized image.

We will introduce a new mathematical operator and network layer to take advantage of the spatial information inherent to the image.

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Convolutions

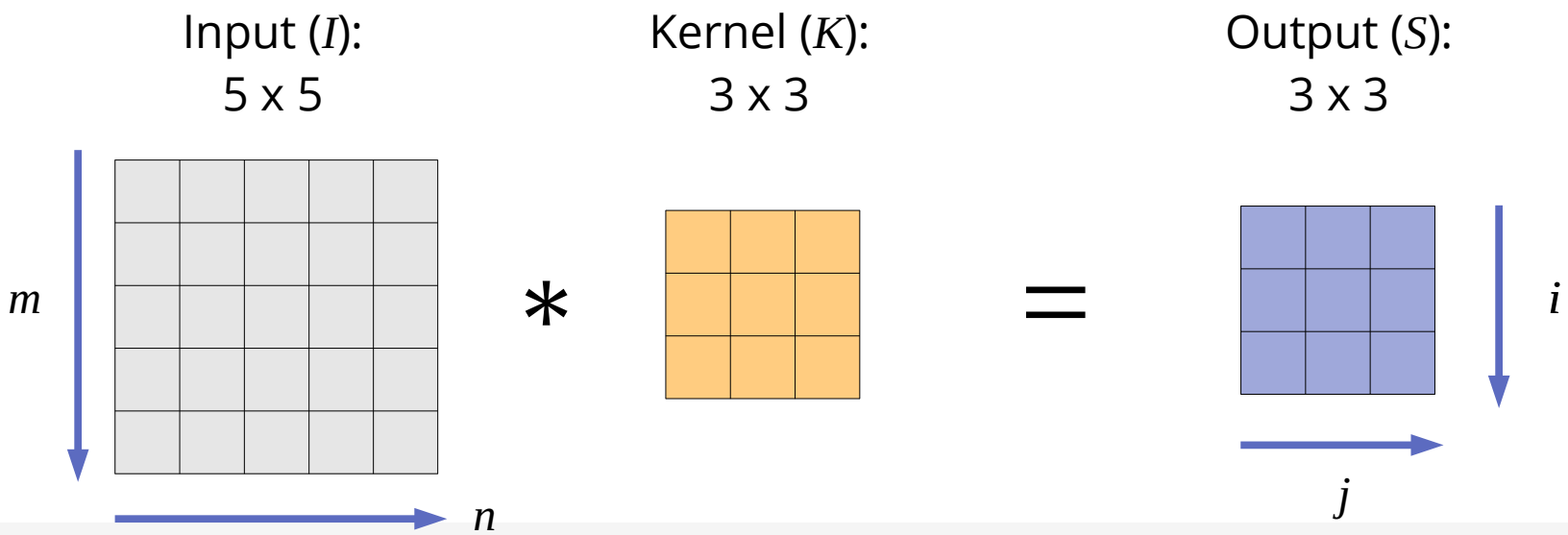


Convolutions

Convolutions are a mathematical operation that operate on dense data (such as images).

We consider an **image** I and a **"Kernel"** K. The "convolution" of I and K will result in an **output feature map** S.

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$



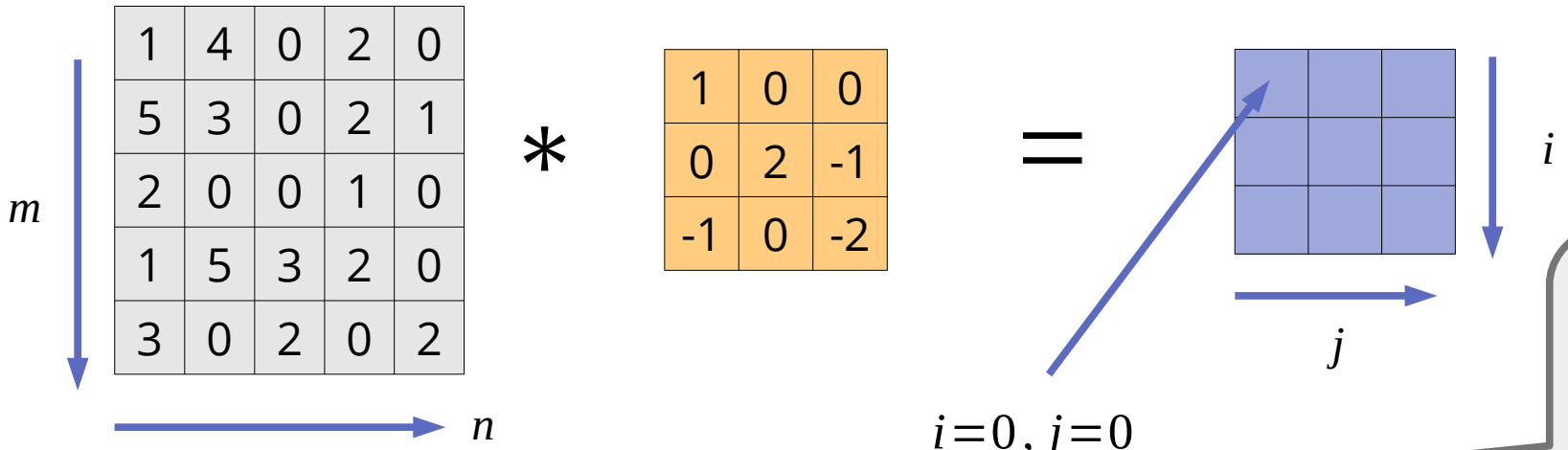
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



$i=0, j=0$

$$S(0,0)=(I*K)(0,0)=\sum_{m=0}^{2} \sum_{n=0}^{2} I(m,n)K(m,n)$$

Why no summation to $m=4$, $n=4$?

K is only defined for $m-i<3$ and $n-j<3$.

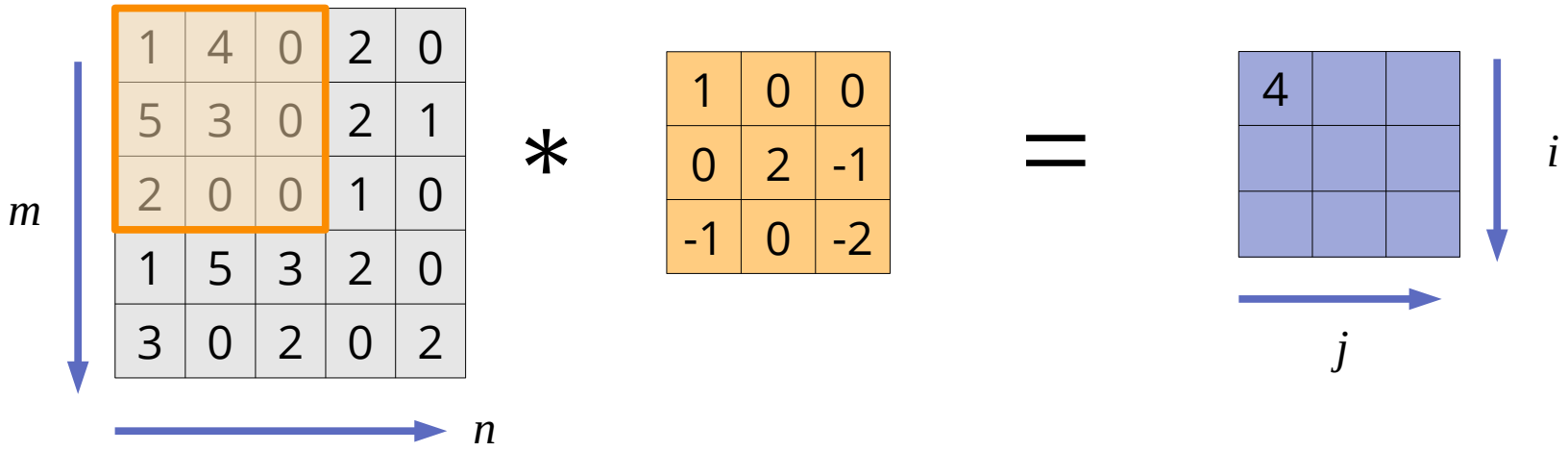
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



$$S(0,0)=(I*K)(0,0)=\sum_{m=0}^{2} \sum_{n=0}^{2} I(m,n)K(m,n)=1\cdot 1+4\cdot 0+0\cdot 0+5\cdot 0+3\cdot 2+0\cdot(-1)+2\cdot(-1)+0\cdot 0+0\cdot(-2)=4$$

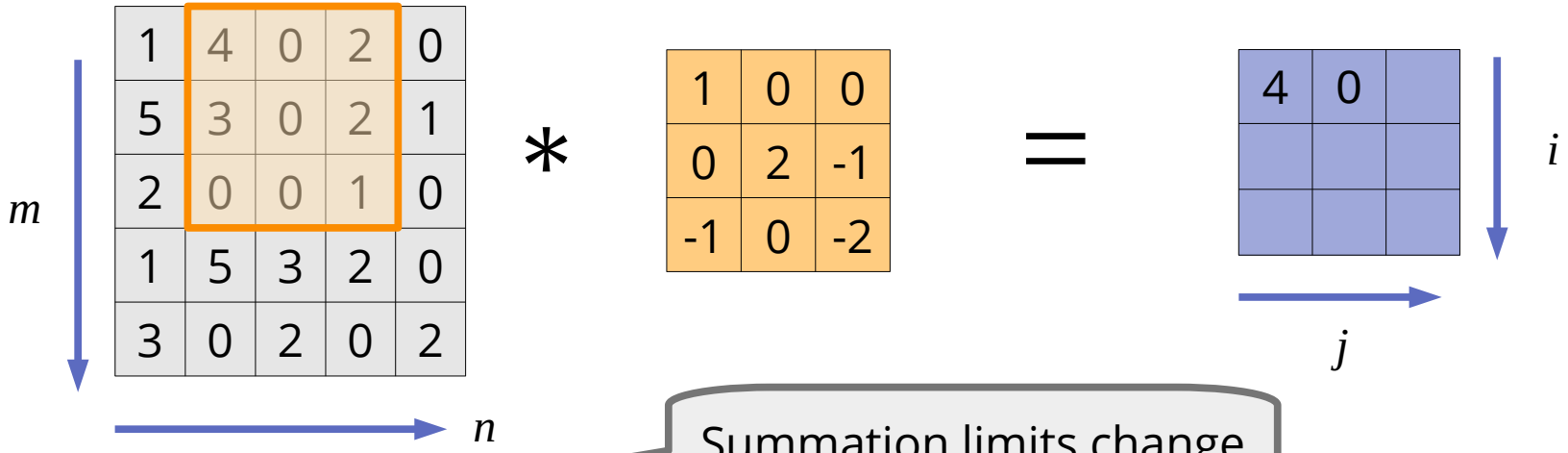
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



Summation limits change

$$S(0,1)=(I*K)(0,1)=\sum_{m=0}^{2} \sum_{n=1}^{3} I(m,n)K(m,n)=4\cdot 1+0\cdot 0+2\cdot 0+3\cdot 0+0\cdot 2+2\cdot(-1)+0\cdot(-1)+0\cdot 0+1\cdot(-2)=0$$

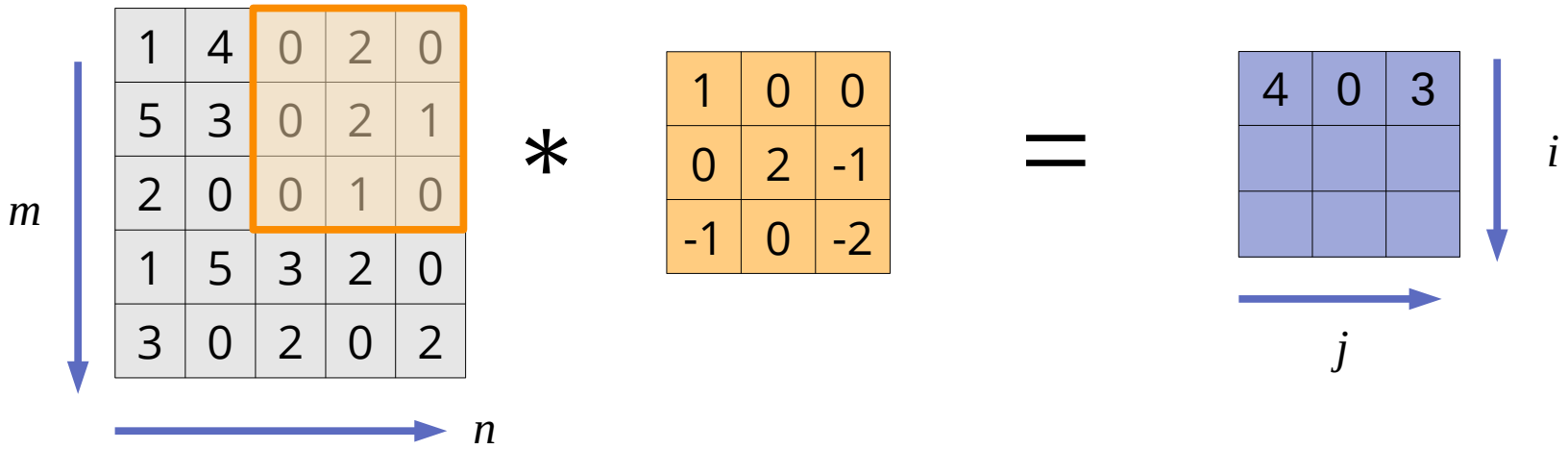
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



$$S(0,2)=(I*K)(0,2)=\sum_{m=0}^{2} \sum_{n=2}^{4} I(m,n)K(m,n)=0\cdot 1+2\cdot 0+0\cdot 0+0\cdot 0+2\cdot 2+1\cdot(-1)+0\cdot(-1)+1\cdot 0+0\cdot(-2)=3$$

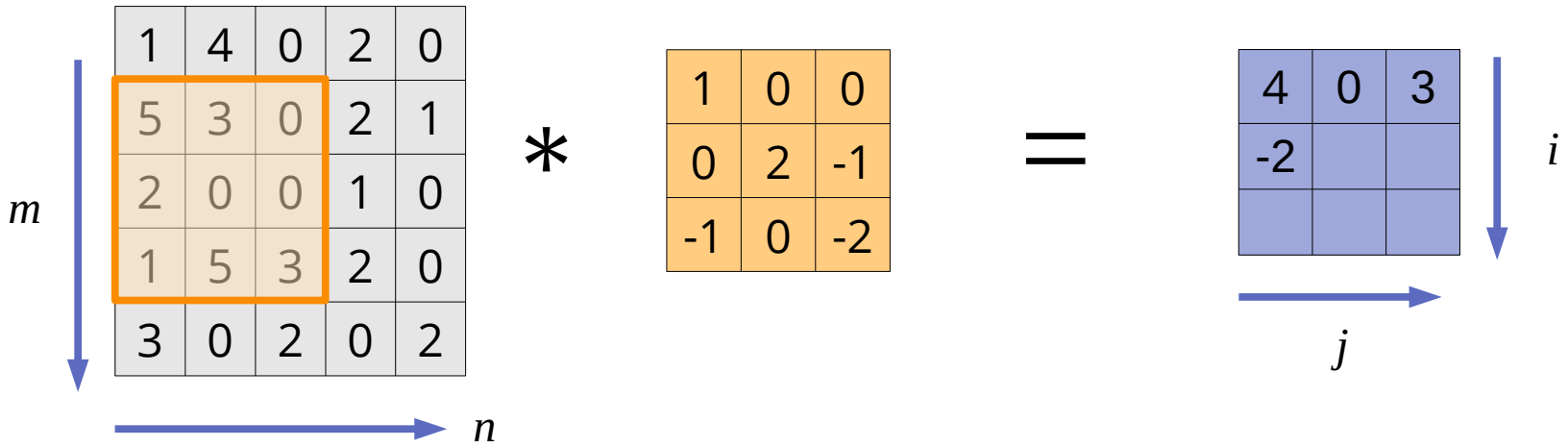
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



$$S(1,0)=(I*K)(1,0)=\sum_{m=1}^{3} \sum_{n=0}^{2} I(m,n)K(m,n)=5\cdot 1+3\cdot 0+0\cdot 0+2\cdot 0+0\cdot 2+0\cdot(-1)+1\cdot(-1)+5\cdot 0+3\cdot(-2)=-2$$

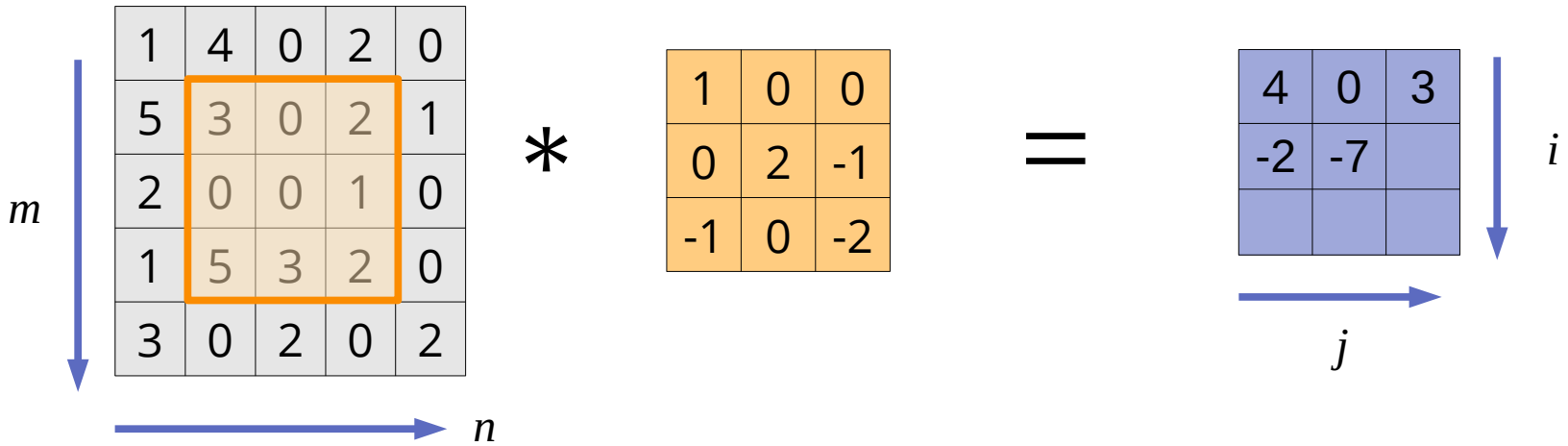
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



$$S(1,1)=(I*K)(1,1)=\sum_{m=1}^{3} \sum_{n=1}^{3} I(m,n)K(m,n)=3\cdot 1+0\cdot 0+2\cdot 0+0\cdot 0+0\cdot 2+1\cdot(-1)+5\cdot(-1)+3\cdot 0+2\cdot(-2)=-7$$

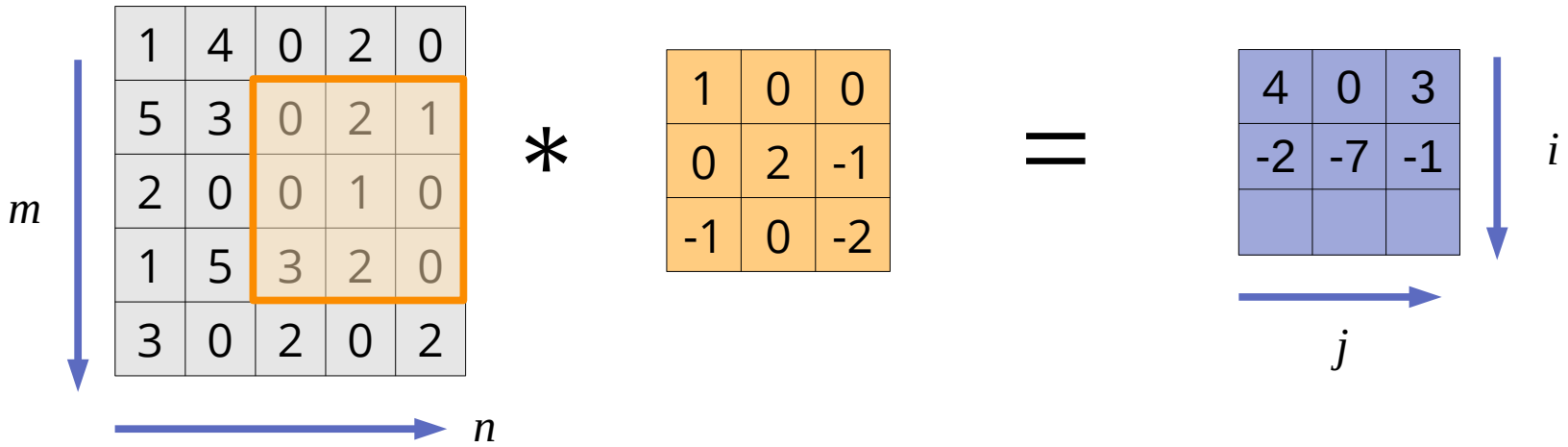
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



$$S(1,2)=(I*K)(1,2)=\sum_{m=1}^{3} \sum_{n=2}^{4} I(m,n)K(m,n)=0\cdot 1+2\cdot 0+1\cdot 0+0\cdot 0+1\cdot 2+0\cdot(-1)+3\cdot(-1)+2\cdot 0+0\cdot(-2)=-1$$

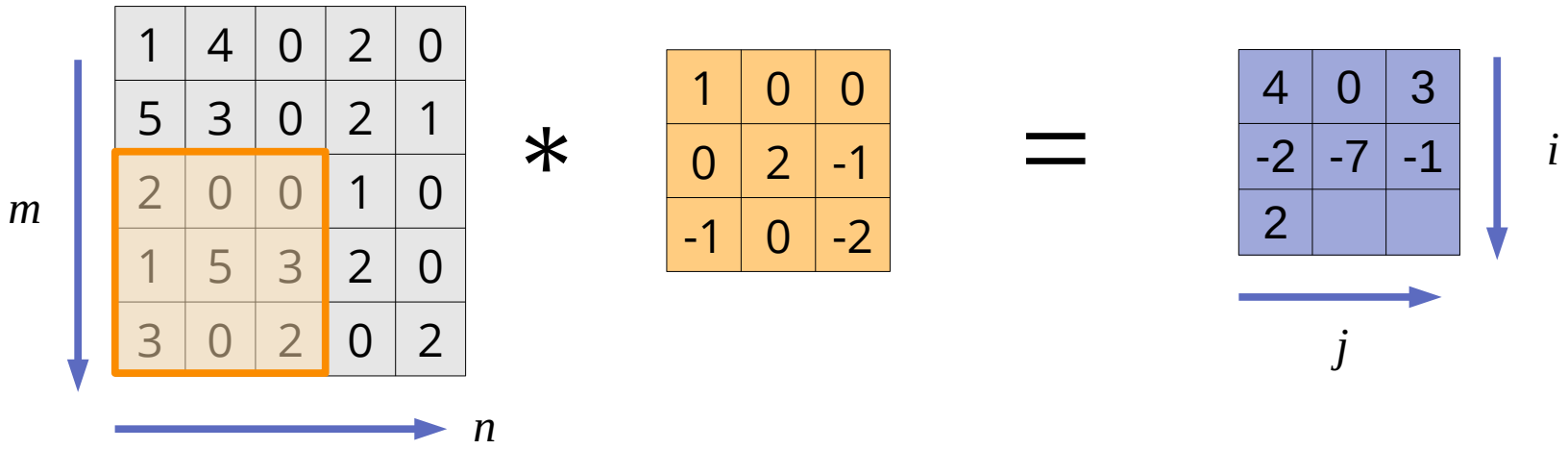
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



$$S(2,0)=(I*K)(2,0)=\sum_{m=2}^{4} \sum_{n=0}^{2} I(m,n)K(m,n)=2\cdot 1+0\cdot 0+0\cdot 0+1\cdot 0+5\cdot 2+3\cdot(-1)+3\cdot(-1)+0\cdot 0+2\cdot(-2)=2$$

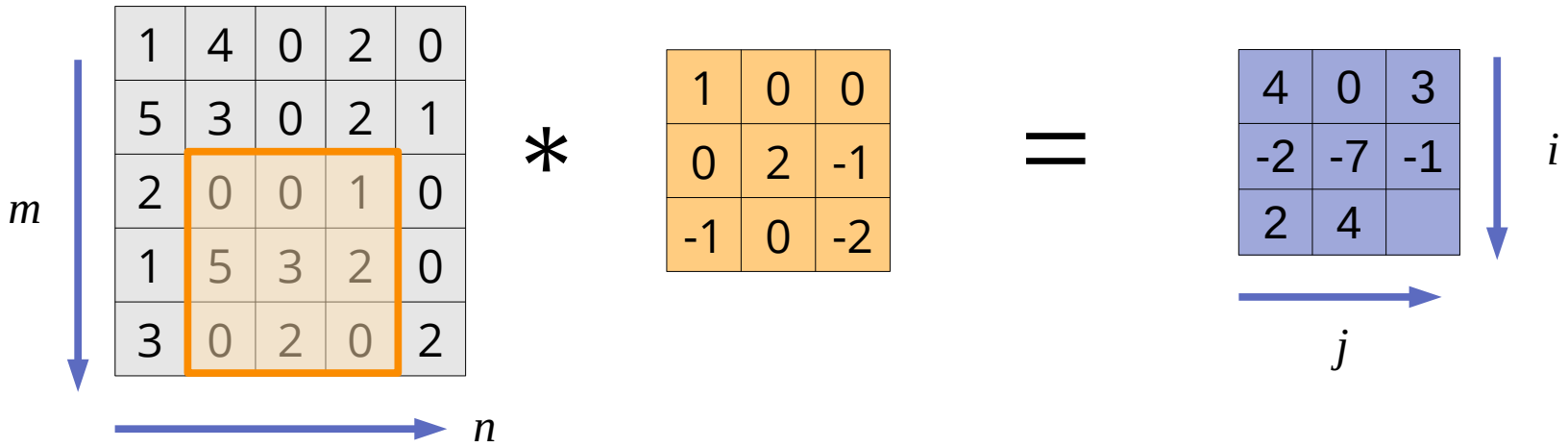
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

Kernel (K):
3 x 3

Output (S):
3 x 3



$$S(2,1)=(I*K)(2,1)=\sum_{m=2}^{4} \sum_{n=1}^{3} I(m,n)K(m,n)=0\cdot 1+0\cdot 0+1\cdot 0+5\cdot 0+3\cdot 2+2\cdot(-1)+0\cdot(-1)+2\cdot 0+0\cdot(-2)=4$$

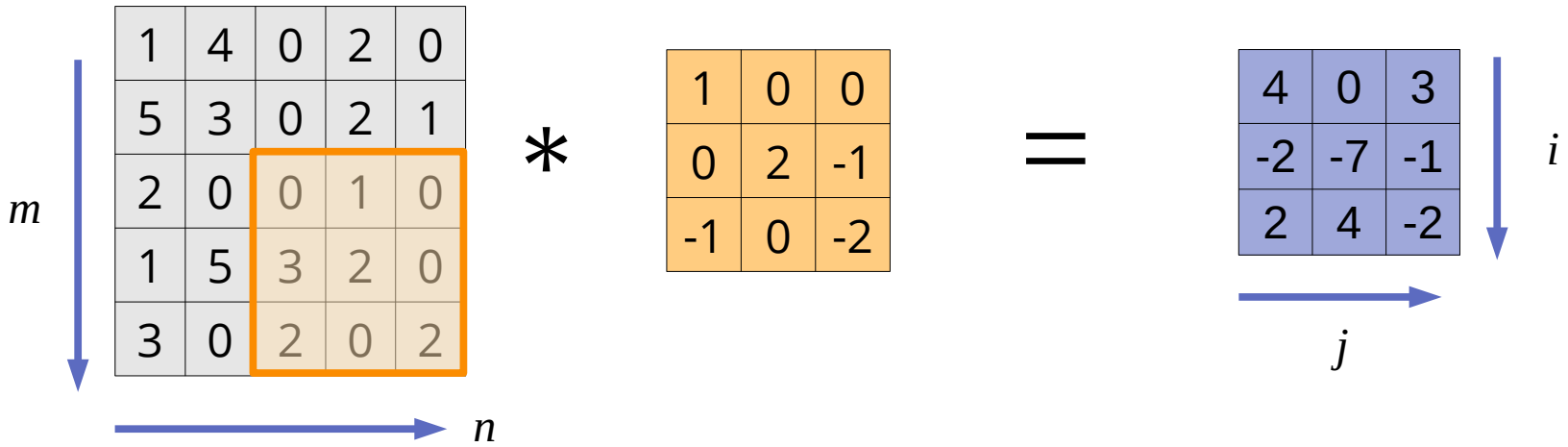
Convolutions: an example

$$S(i,j)=(I*K)(i,j)=\sum_m \sum_n I(m,n)K(m-i,n-j)$$

Input (I):
5 x 5

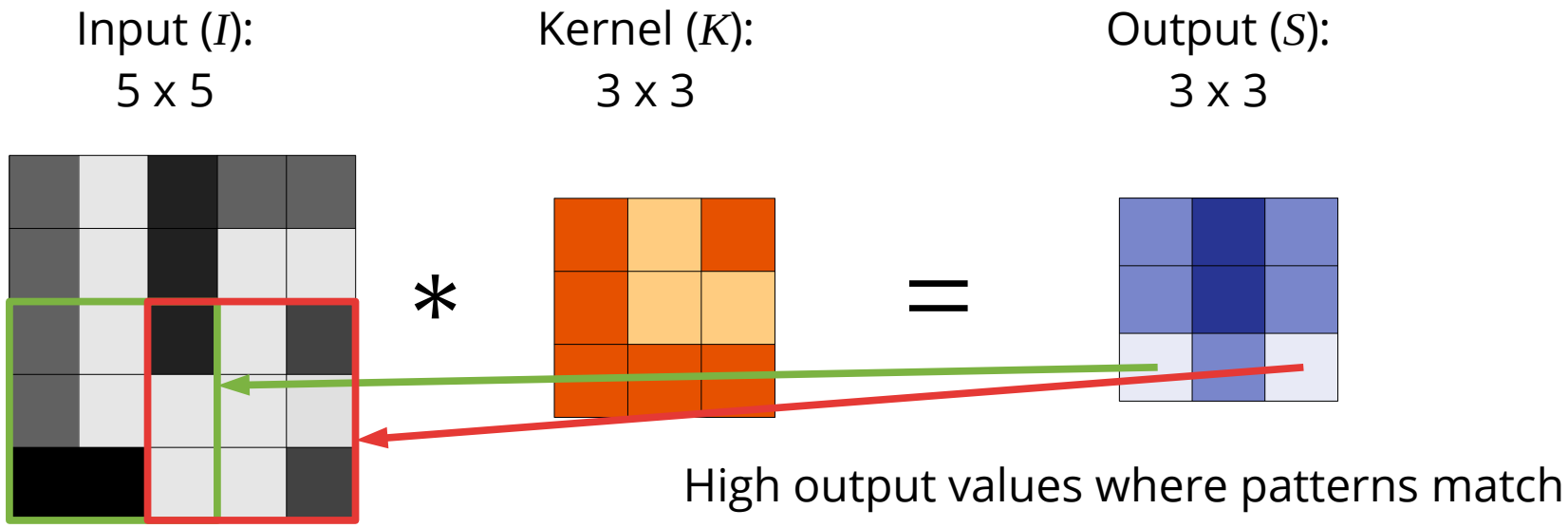
Kernel (K):
3 x 3

Output (S):
3 x 3



$$S(2,2)=(I*K)(2,2)=\sum_{m=2}^{4} \sum_{n=2}^{4} I(m,n)K(m,n)=0\cdot 1+1\cdot 0+0\cdot 0+3\cdot 0+2\cdot 2+0\cdot(-1)+2\cdot(-1)+0\cdot 0+2\cdot(-2)=-2$$

Convolutions: a visualization

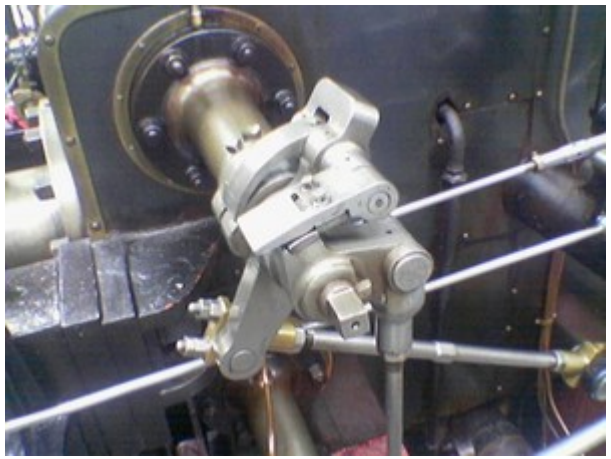


Convolutions provide high output signals where the projection of the kernel resembles the input pattern.

→ Convolutional kernels acts as feature filters and are sometimes called **filters** for this exact reason.

Convolutions: use in computer vision

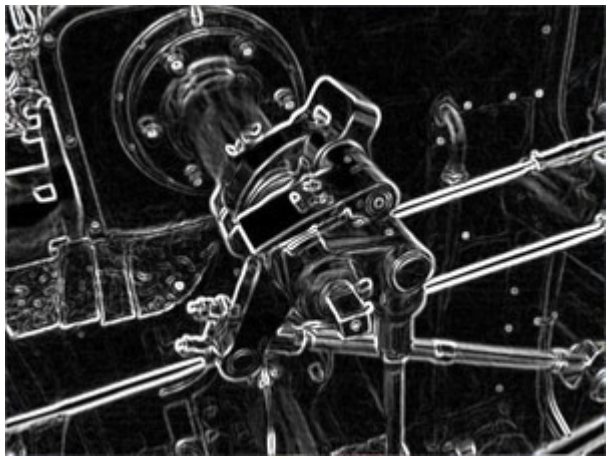
Filters extract information from images, e.g., the Sobel filter is able to extract edges:



wikipedia

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1



wikipedia

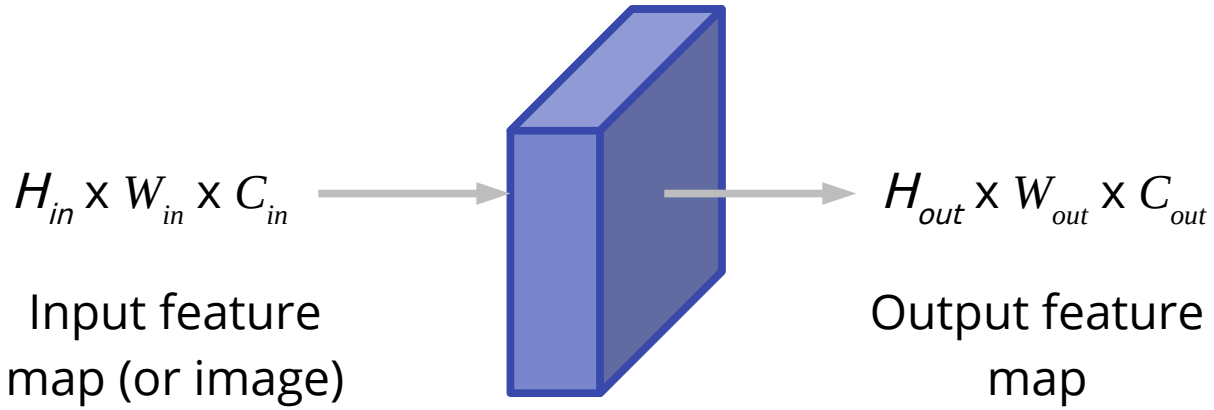
The Sobel filter has been manually engineered. We will see later that neural networks can **learn** filters.

Convolutional layers

Convolutional layers implement the concept of convolutions for neural networks. These layers are defined based on the following parameters:

- Number of **input channels**
- Number of **output channels**
- **Kernel size**
- **Stride**
- **Padding**

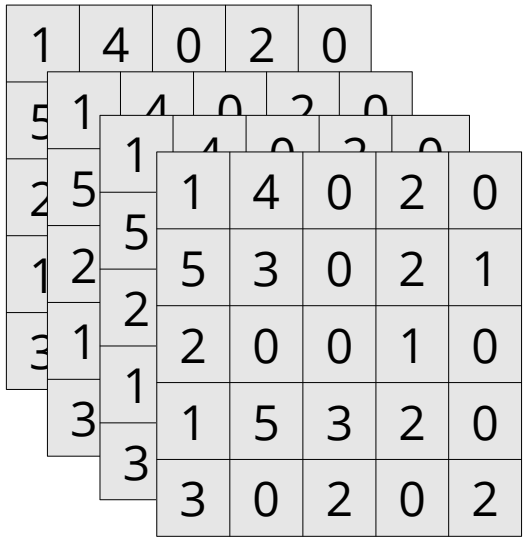
} Affect output feature map size



Channels

A channel is simply a two-dimensional feature map.

Network layers can take multi-channel data as input. Those are simply stacks of feature maps:



4-channel (5x5) feature map

Convolutions: multi-channel input

We saw in one of our first lectures that color images consist of three channels (R, G and B). How to process multi-channel data?

1	0	0
0	2	-1
-1	0	-2

3	-2	1
1	1	0
5	4	0

2	3	1
-2	0	-2
1	4	-3

pixnio via scipy

We simply use a separate kernel for each input channel: RGB input (3 channels) = 3 different kernels

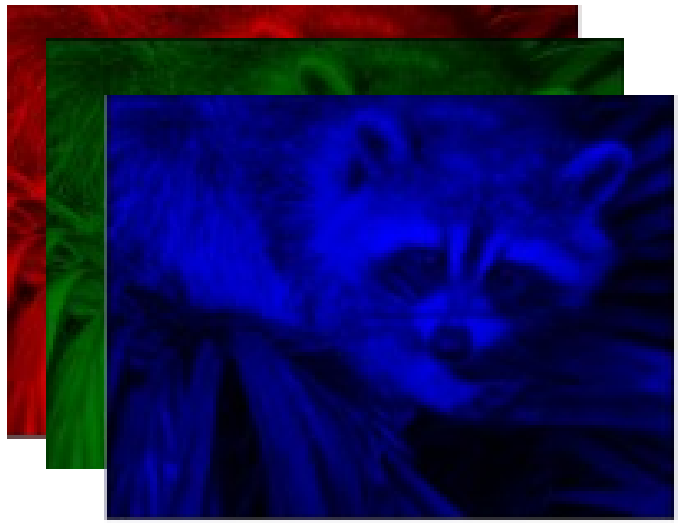
$$\sum$$

Output

By default, the resulting feature map has only a **single output channel**; the output contains the sum of all convolutions of the input channels.

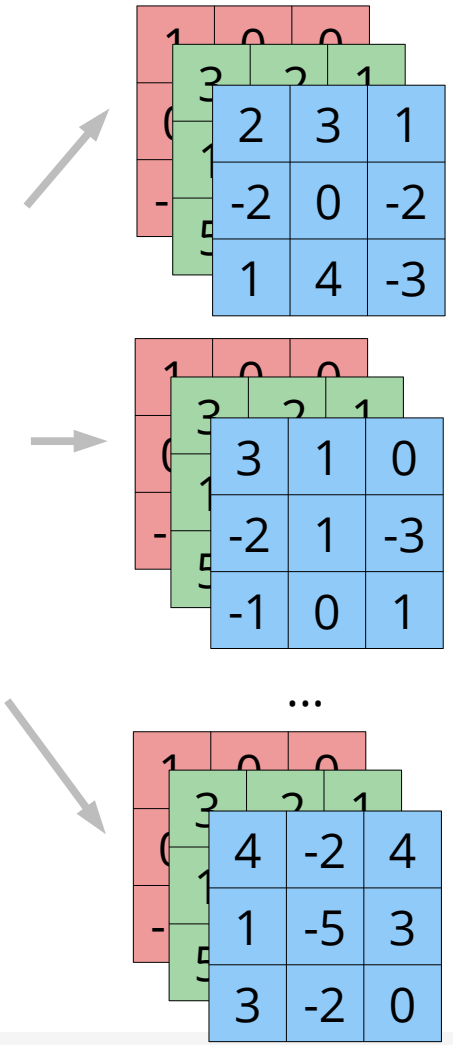
Convolutions: multi-channel output

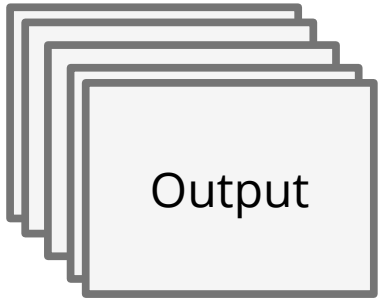
It may be beneficial to have multiple output channels as they can learn different from the input data. How to implement this?



pixnio via scipy

H_{in} x W_{in} x (C_{in}=3)





H_{out} x W_{out} x (C_{out}=n)

Convolutions: kernel size

The kernel size defines the **size of the kernel operator** and therefore affects the size of the output of the convolution. For the sake of simplicity, we assume kernels to be quadratic and kernels of the convolutional layer to have the same size.

1	2
0	0

2x2 kernel

1	0	-1
2	0	-2
1	0	-1

3x3 kernel

1	0	-1	-3
2	0	-2	0
1	0	-1	1
4	1	-2	1

4x4 kernel

How does the kernel size affect the size of the output feature map?

$$O = I - K + 1$$

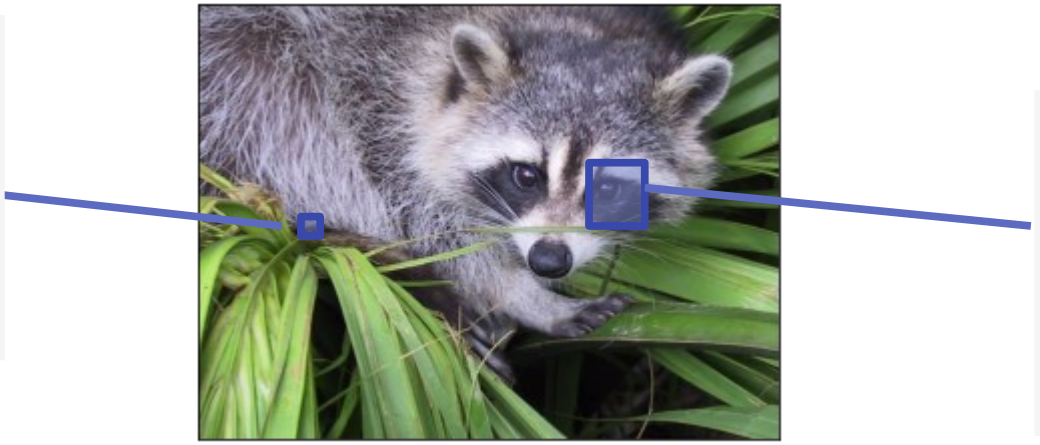
K: kernel size
I: input feature map size
O: output feature map size

Convolutions: kernel size

Kernels of different sizes “see” features of different sizes: this is called their **receptive field** (we will discuss this in detail later).

Consider the following example (this is what the first convolutional layer in a CNN sees):

Small kernels can only “see” their immediate neighborhood. They filter simple features such as edges.



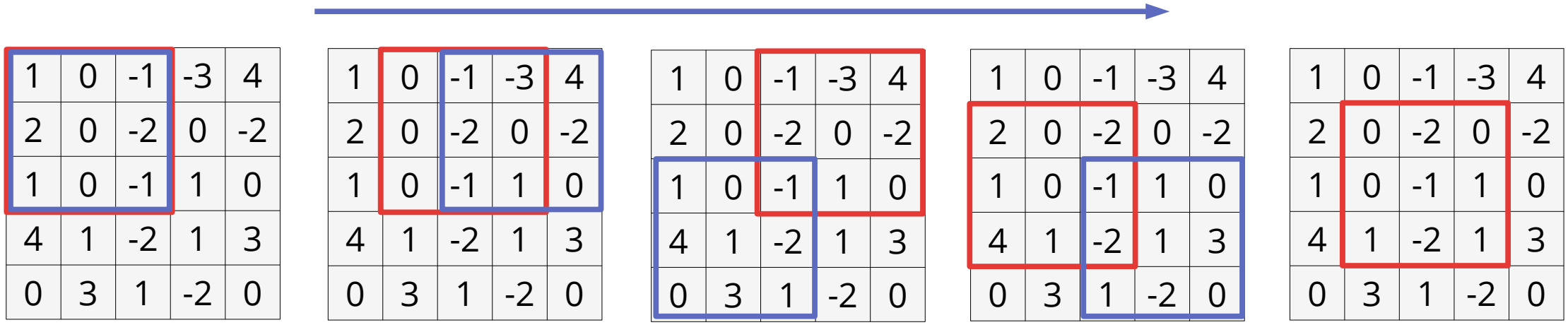
Large kernels are able to identify larger and more complex features such as an eye: they have more “**context**”.

The choice of the right kernel size is a trade off between receptive field and model size.

Convolutions: stride

"Stride" defines the **step size** of the kernel on the input feature map. The default is a stride of 1, meaning that the kernel is moved one cell at a time as it runs over the input map.

Choosing a stride larger than 1 enables the kernel to skip over cells, thereby reducing the size of the output feature map.



Stride = 1

Stride = 2

Output size: $O = \frac{I - K}{S} + 1$ S: Stride

Convolutions: padding

Convolutions shrink feature maps. This can be counter-acted by padding the input map with rows and columns of zeros:

1	0	-1	-3	4
2	0	-2	0	-2
1	0	-1	1	0
4	1	-2	1	3
0	3	1	-2	0

Input size: 5
Kernel size: 3
Stride: 1
Padding: 0
→ Output: 3x3

0	0	0	0	0	0	0
0	1	0	-1	-3	4	0
0	2	0	-2	0	-2	0
0	1	0	-1	1	0	0
0	4	1	-2	1	3	0
0	0	3	1	-2	0	0
0	0	0	0	0	0	0

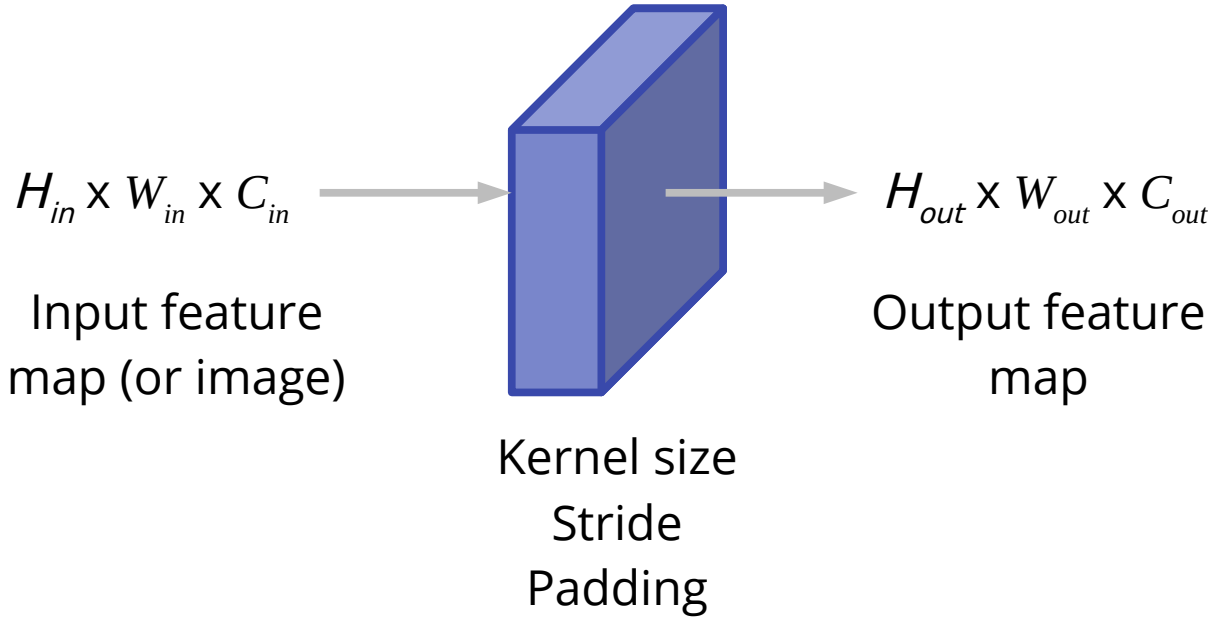
Input size: 5
Kernel size: 3
Stride: 1
Padding: 1
→ Output: 5x5

With proper padding we can preserve the size of the input feature map:

Output size: $$O = \frac{I - K + 2P}{S} + 1$$

I: input feature map size
O: output feature map size
K: kernel size
S: stride
P: padding

Convolutional layers



In a convolutional layer, we learn the kernel parameters in the training process in the same way that we learned weights in linear layers.

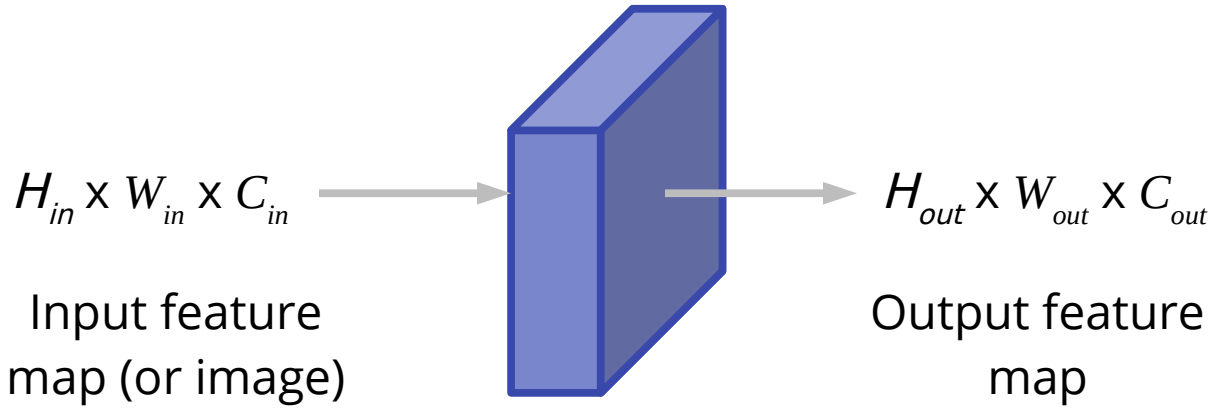
The huge advantage of convolutional layers is that their **parameters are shared** over the entire input feature map, vastly reducing the number of parameters of the network!

Number of parameters

The values stored in the kernels are the parameters of the convolutional layer (disregarding bias values).

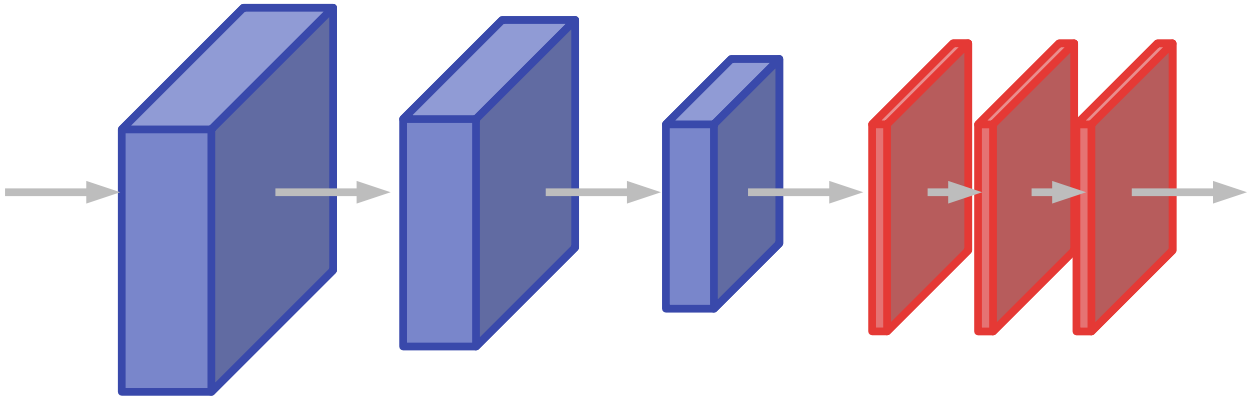
How many parameters are there?

$$K^2 \times C_{in} \times C_{out}$$

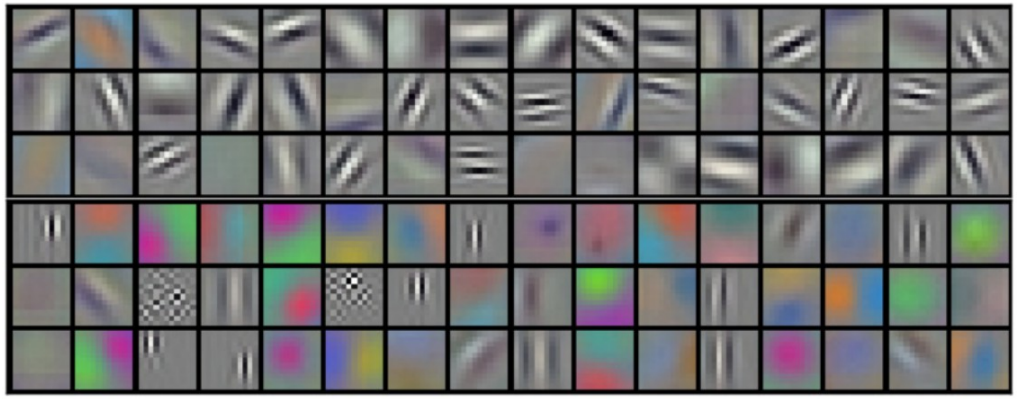


Example:
3 input channels, 10 output channels, 3x3 kernel
→ 270 parameters

Note that the number of parameters is **independent** of the input feature map size as opposed to linear layers.

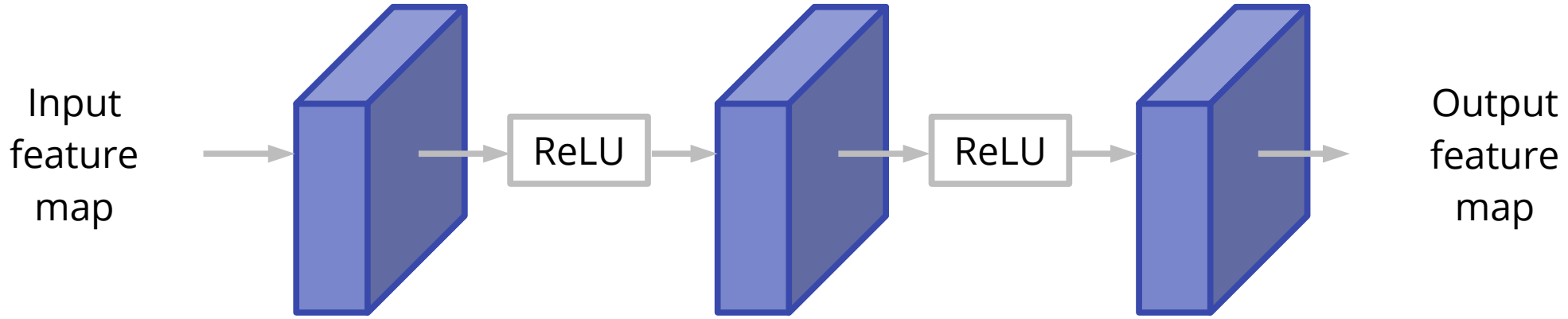


Convolutional Neural Networks



Stacking convolutional layers

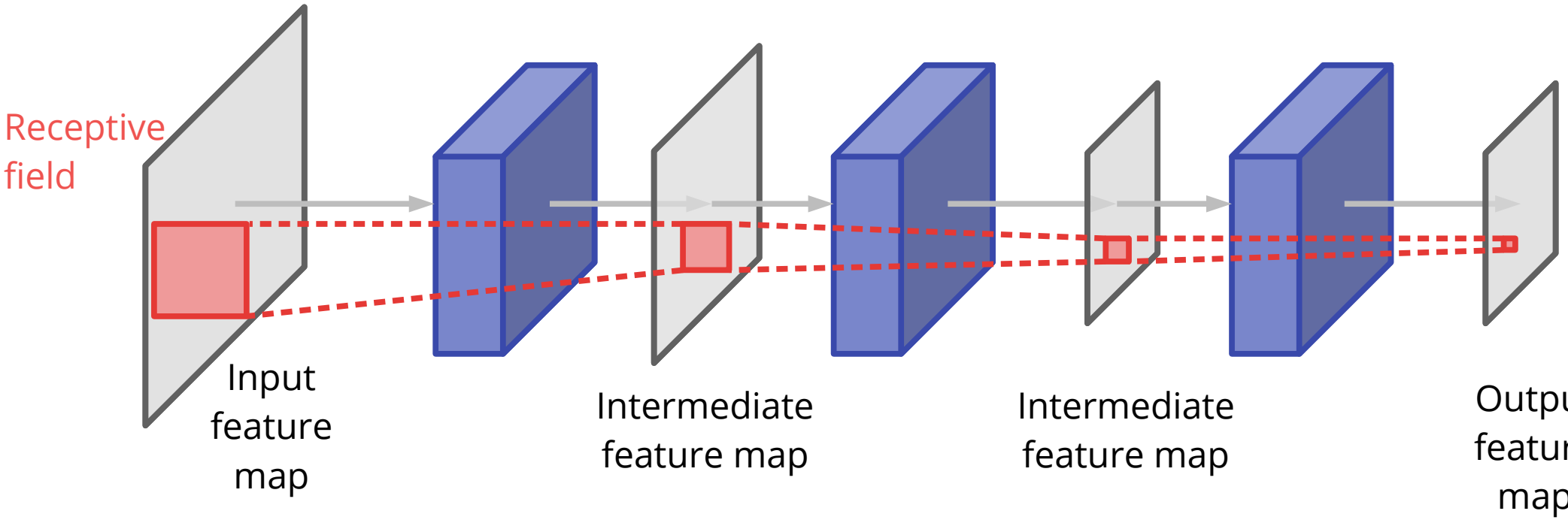
We can stack a number of convolutional layers and combine them with non-linear activation functions:



The size and number of channels in the output feature map depends on the parameters of all convolutional layers inbetween.

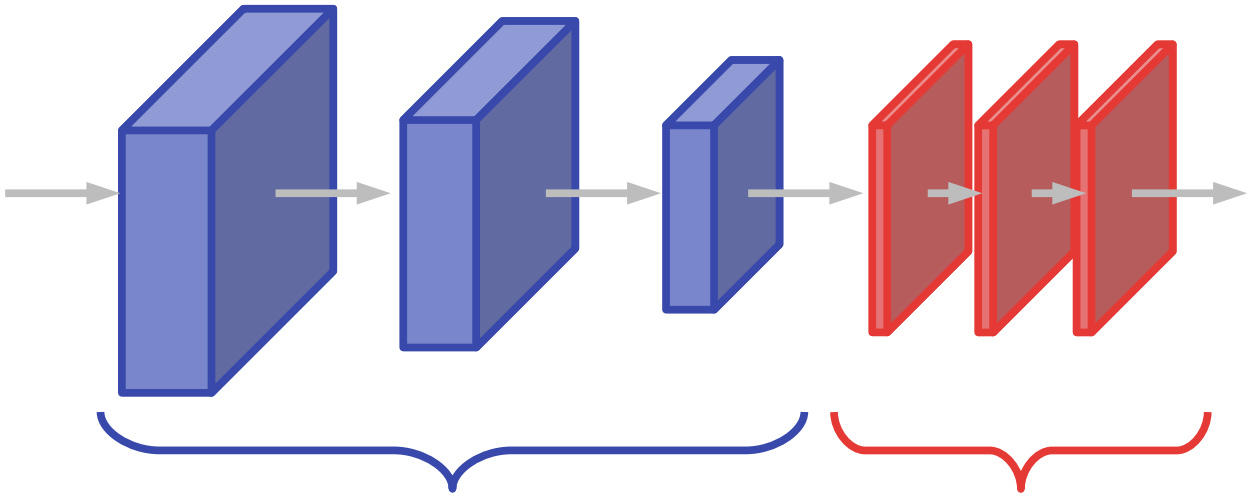
Receptive field

Let's see what happens here (disregarding the non-linearities for a moment and assuming that all convolutional layers have the same kernel size):



The combination of convolutional layers leads to a large **receptive field**: the information that is seen by a single element of the output feature map covers a much larger area in previous feature maps.

A Convolutional Neural Network (CNN) for image classification



Dog
Cat
Bird
Trash Panda
Fish
Walrus

Convolutional layers

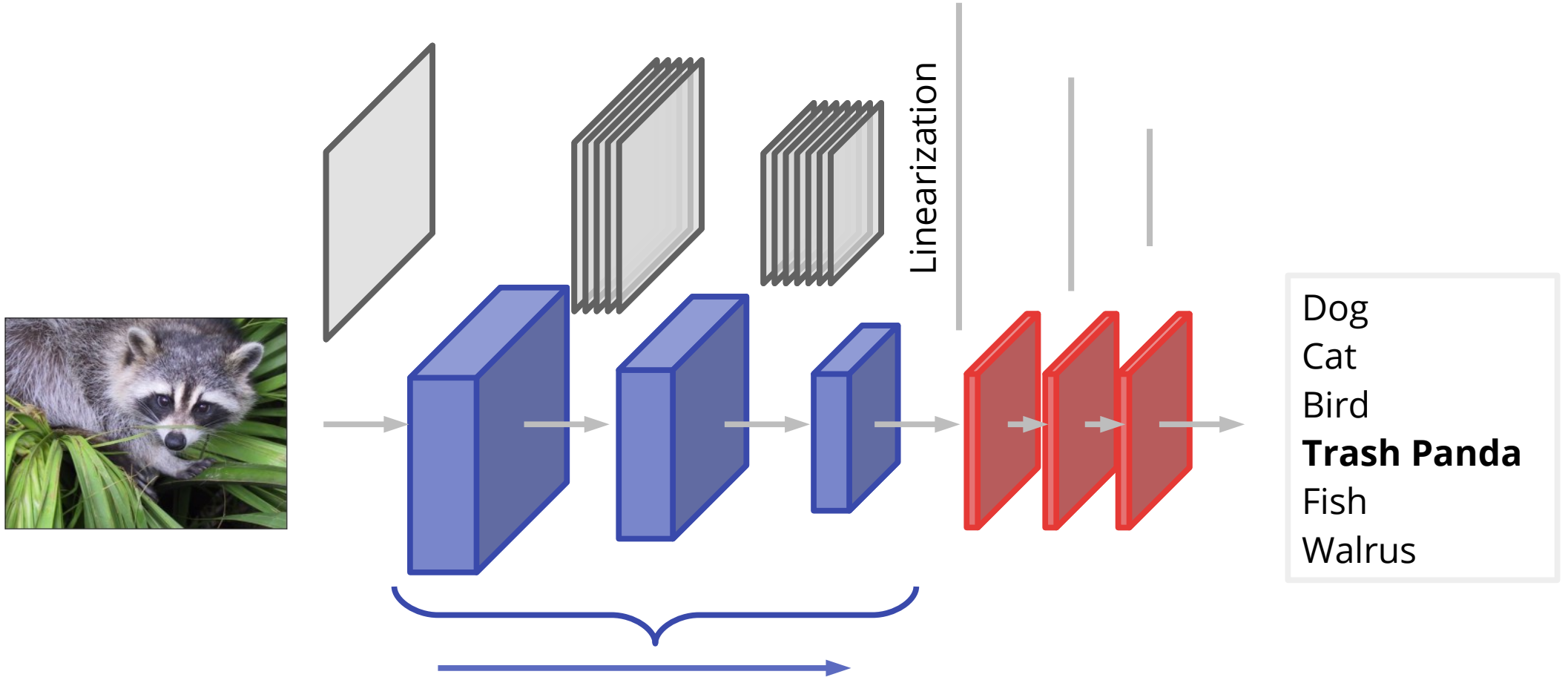
Linear layers

Feature Extraction

Classifier

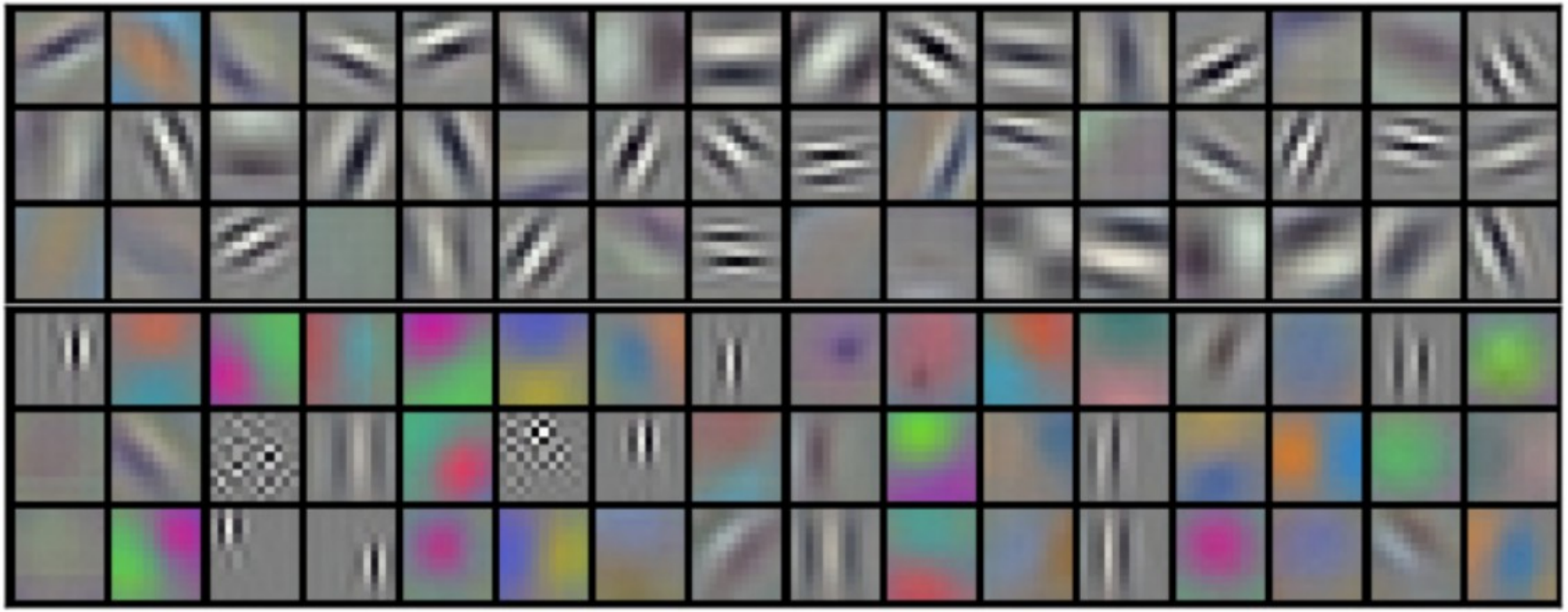
(Non-linearities after each layer not shown here)

A Convolutional Neural Network (CNN) for image classification



Decreasing feature map size: **decreasing resolution**
increasing number of channels: **increasing semantic information**

What do the kernels learn?



Learned kernel weights in a convolutional neural network (Krizhevsky et al. 2012).

Image segmentation

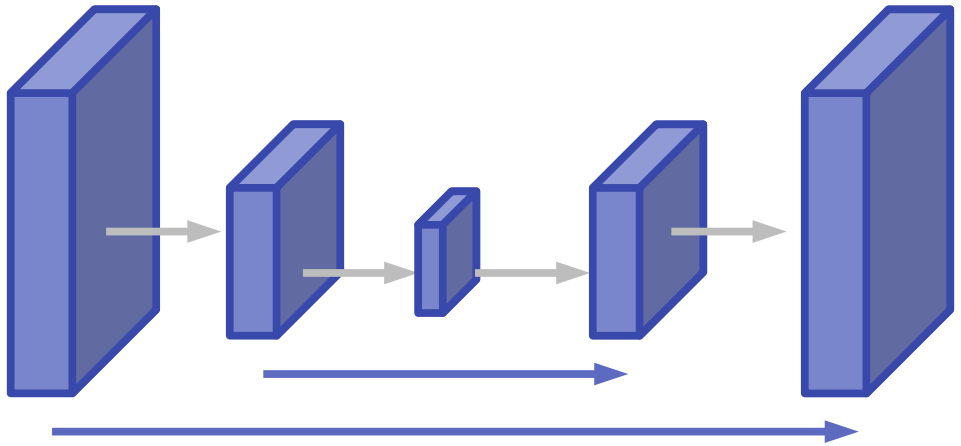


Image segmentation

publicdomainpictures.net

Input image

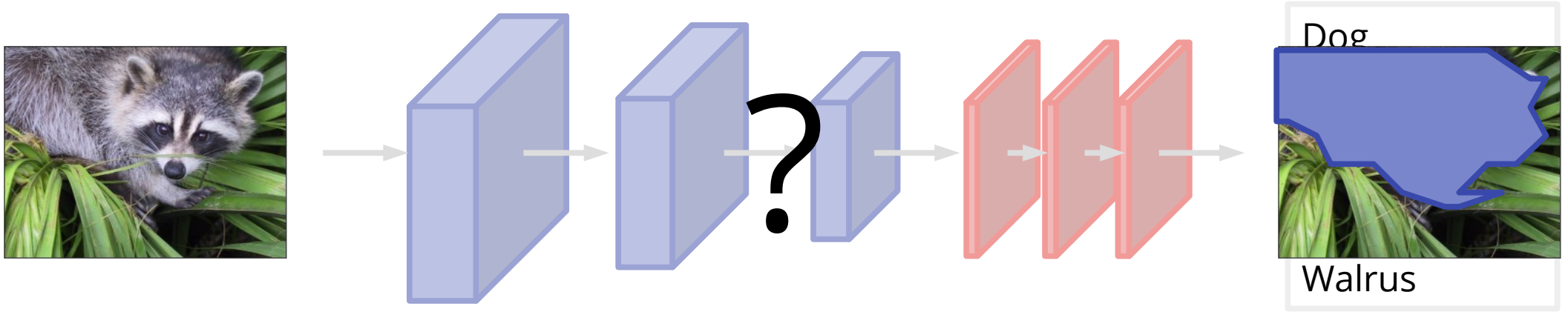
Semantic Segmentation

Create a map of all pixels that belong to an animal.

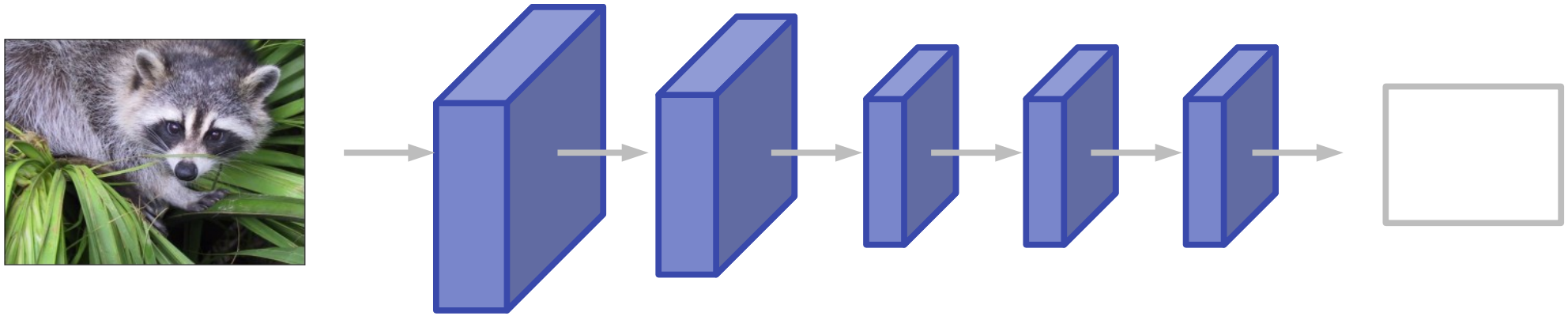
Instance Segmentation

Create a map of all pixels that belong to an animal and assign them to different instances.

How to implement a segmentation model?



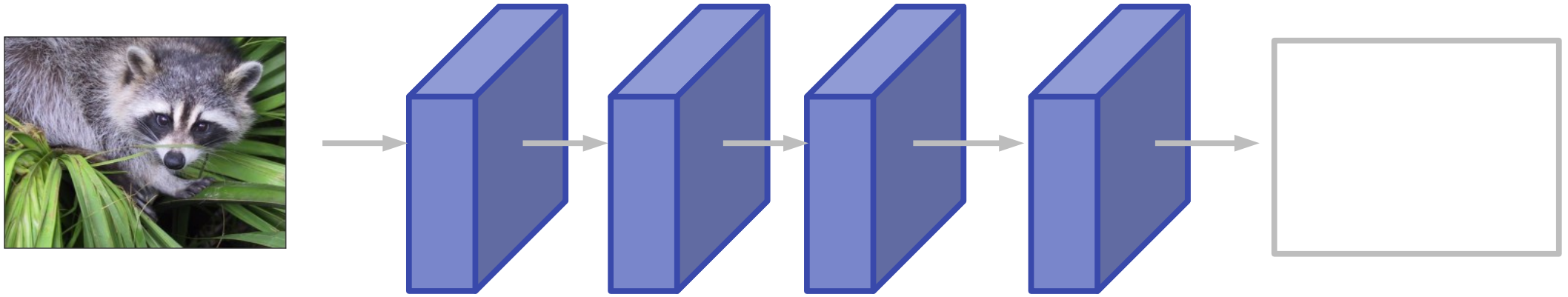
How to implement a segmentation model?



ToDo:

1) Replace linear layers with convolutional layers (output must be feature map)

How to implement a segmentation model?

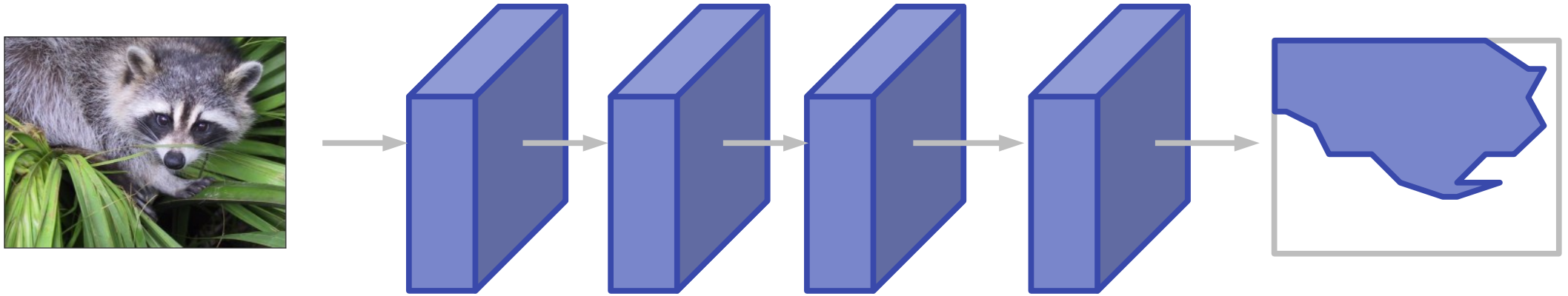


ToDo:

1) Replace linear layers with convolutional layers (output must be feature map)
2) Adjust convolutions so that final output has the same size as input image

How to implement a segmentation model?

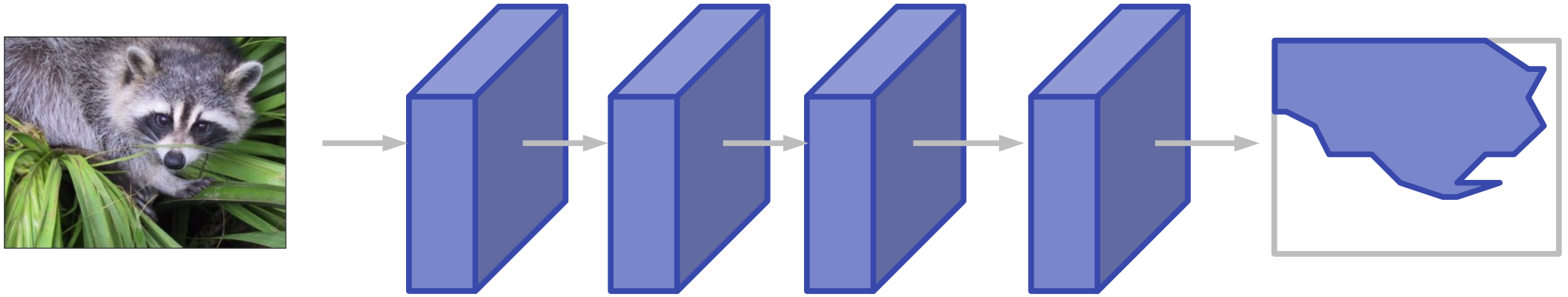
Fully Convolutional Network



ToDo:

1) Replace linear layers with convolutional layers (output must be feature map)
2) Adjust convolutions so that final output has the same size as input image
3) Apply Softmax after final layer to get class probability for each pixel

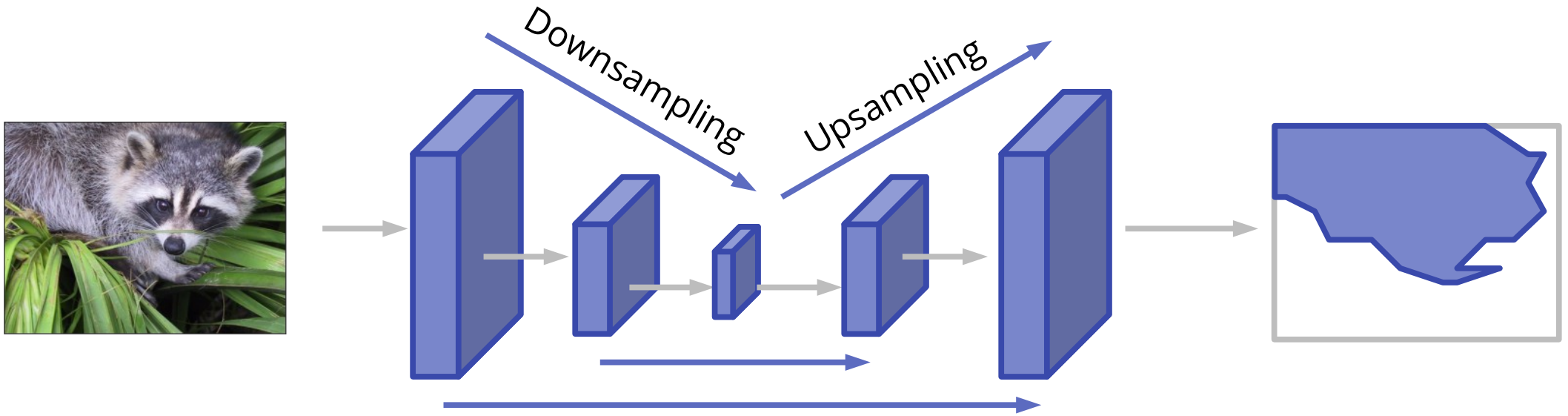
Fully Convolutional Network



This works in principle, but it is very expensive due to the large size of the feature map after each layer.

We have to down-sample the image...

U-Net

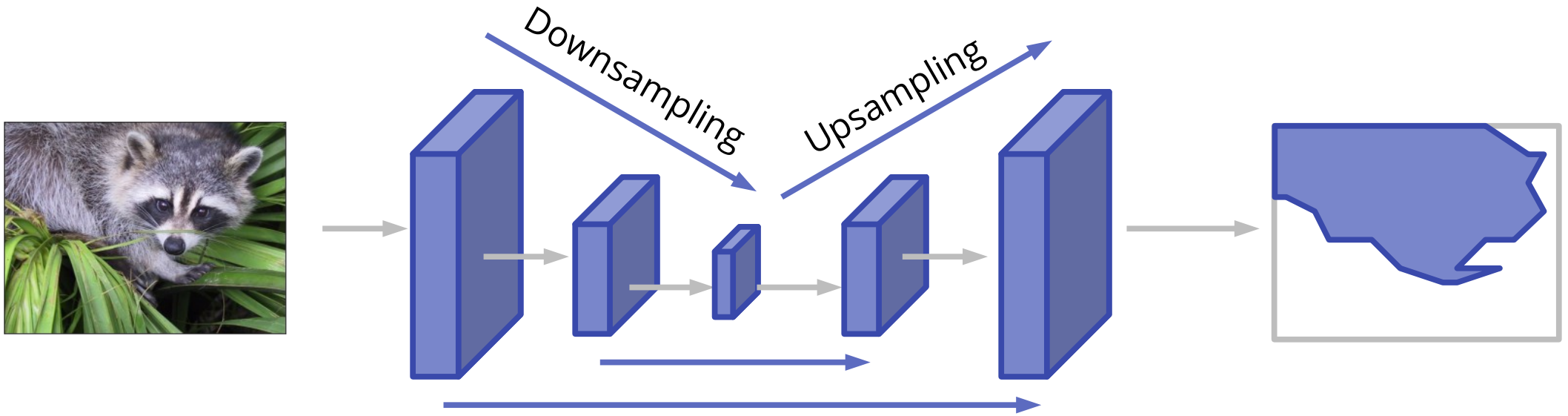


Downsampling of feature map greatly reduces the feature map extents (and improves ability to generalize).

Much better performance with **skip connections**: we supplement the semantically rich information from the layers near the bottleneck with high-resolution feature maps from the downsampling branch.

Ronneberger et al. (2015)

U-Net

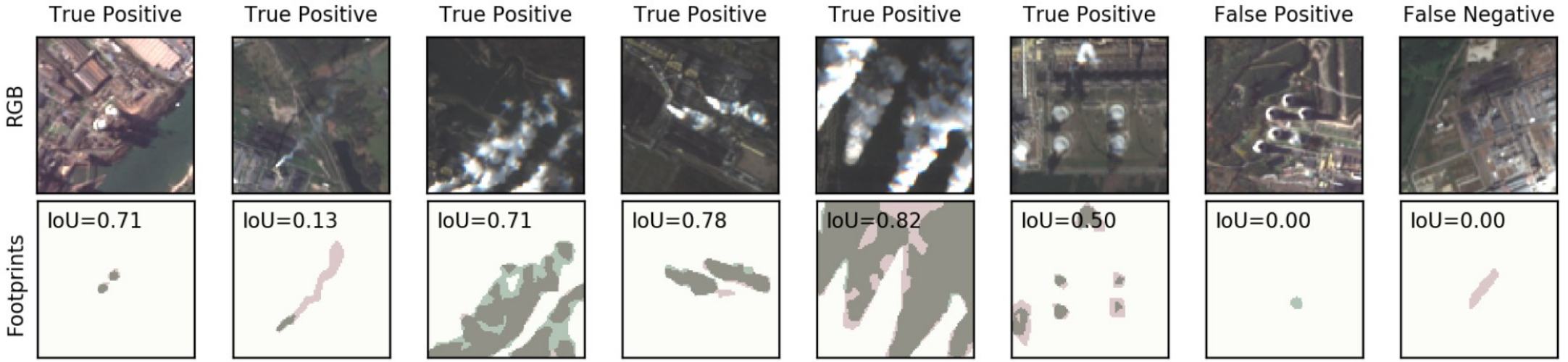


How to do the upsampling?

Transposed convolutions: by using a stride < 1, we can use convolutions as learnable upscaling operations.

Ronneberger et al. (2015)

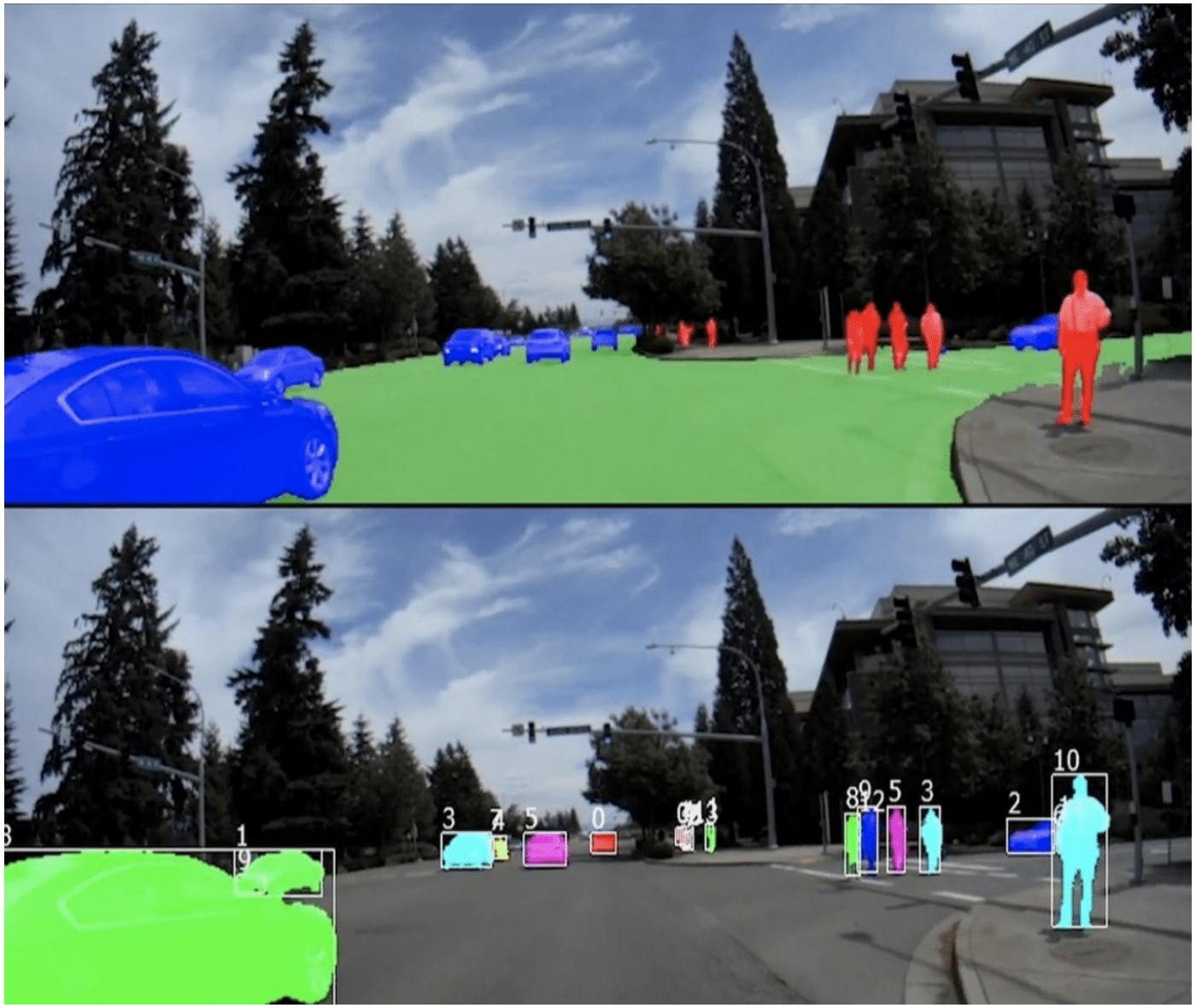
Semantic segmentation: a real world example



Ground Truth / Prediction

Mommert et al. (2020)

Panoptic segmentation



=

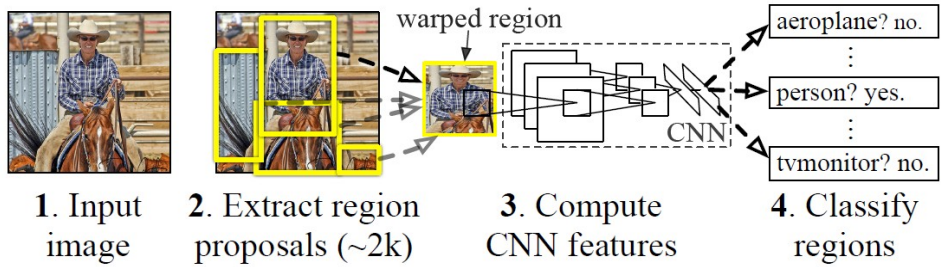
Semantic segmentation

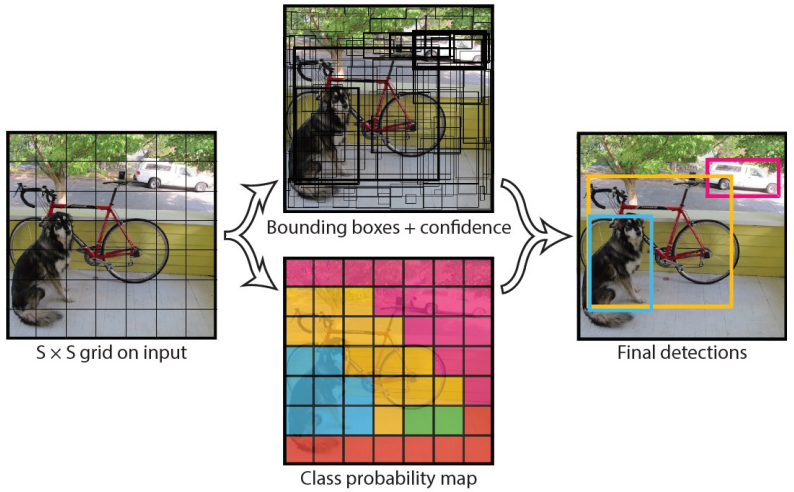
+

Instance segmentation

NVIDIA (2019)

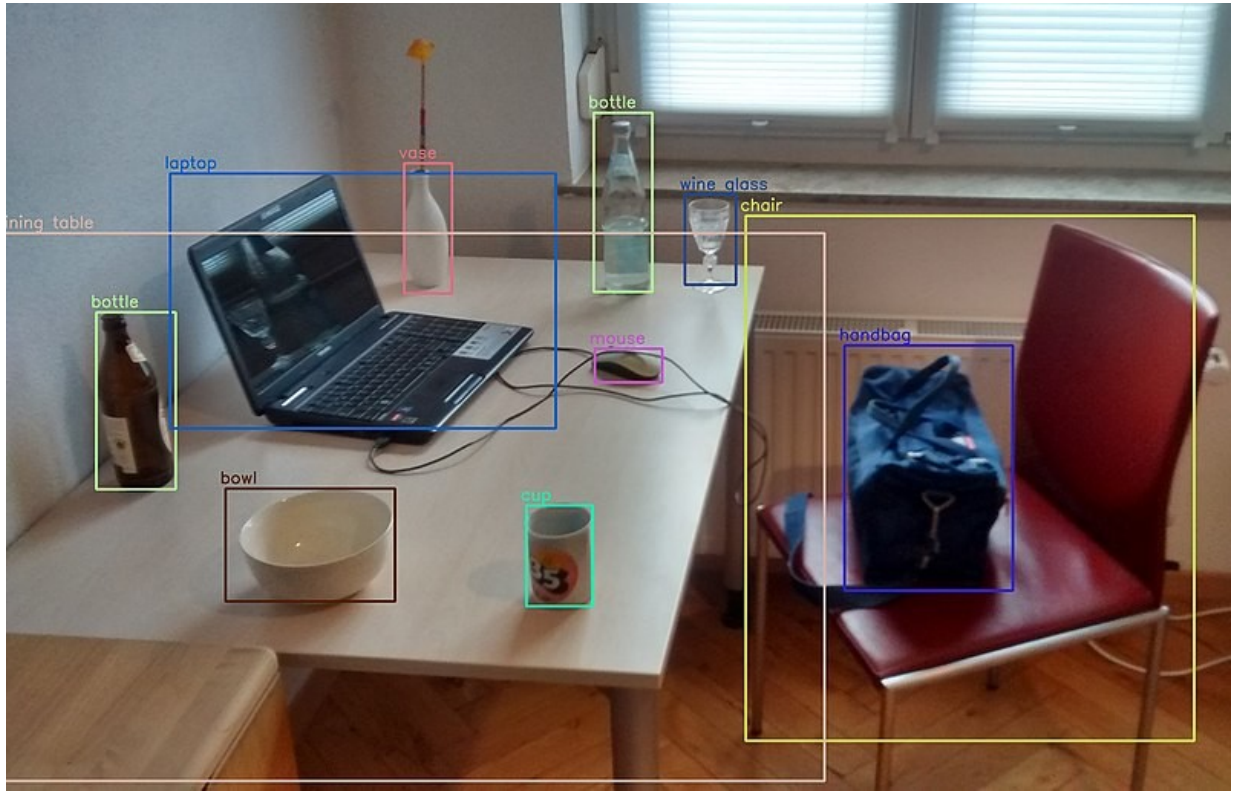
Object detection





Object detection as a task

Goal: Find instance of a given class in an image.
→ provide class, approximate location and extent (bounding box)



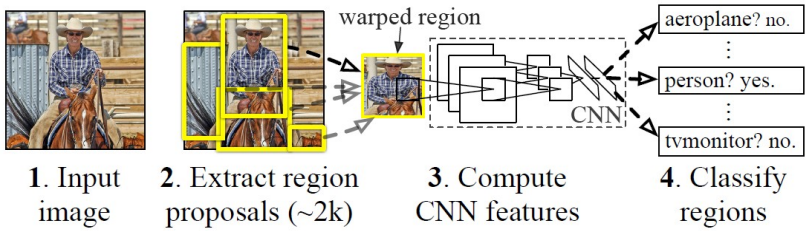
MTheiler

Complications

- Scale variability: a given object does always appear in the same size
- Objects might be occulted
- Lighting and shadows might affect object images
- Perspective affects the appearance of an object
- How do we define the location of the object?

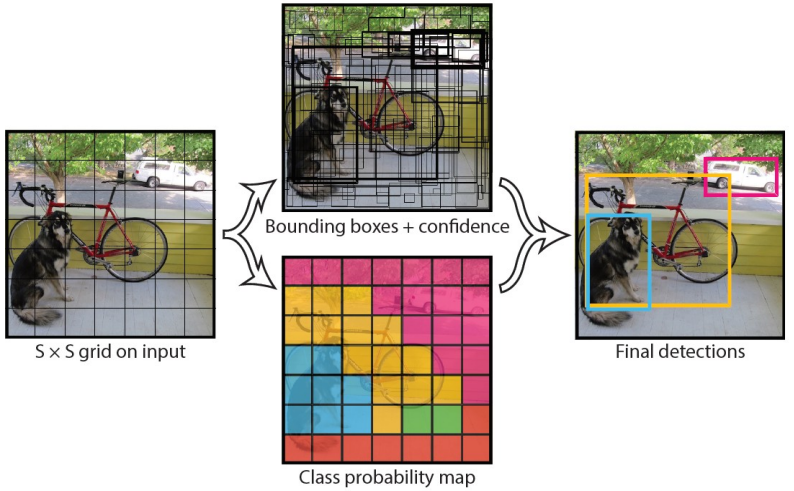
Object detection methods

Multi-stage object detection



Region-based CNNs (R-CNN)
Fast R-CNN
Faster R-CNN

Single-stage object detection

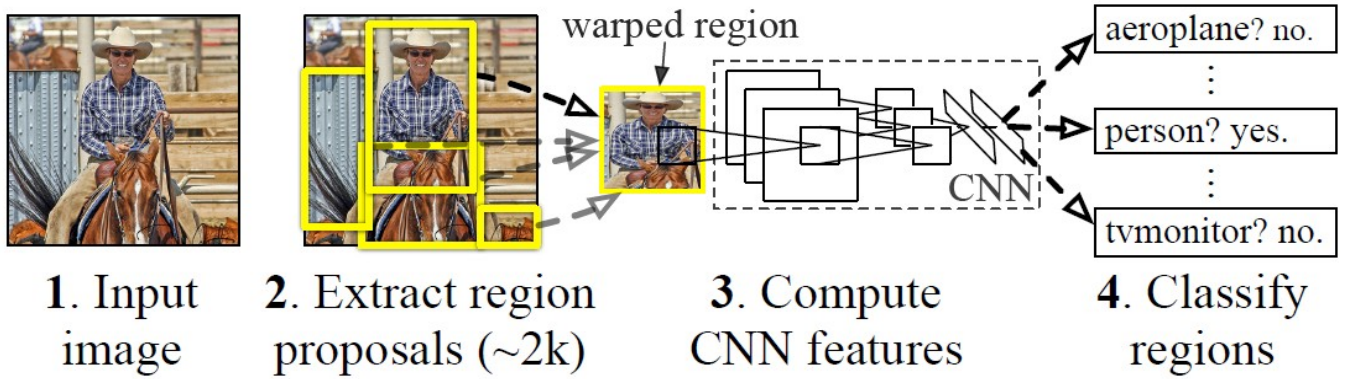


You only look once (YOLO)

Multi-stage object detection

The object detection process is split into different steps that require to process the image data multiple times.

Region-based CNNs (**R-CNN**), Girshick et al. 2014

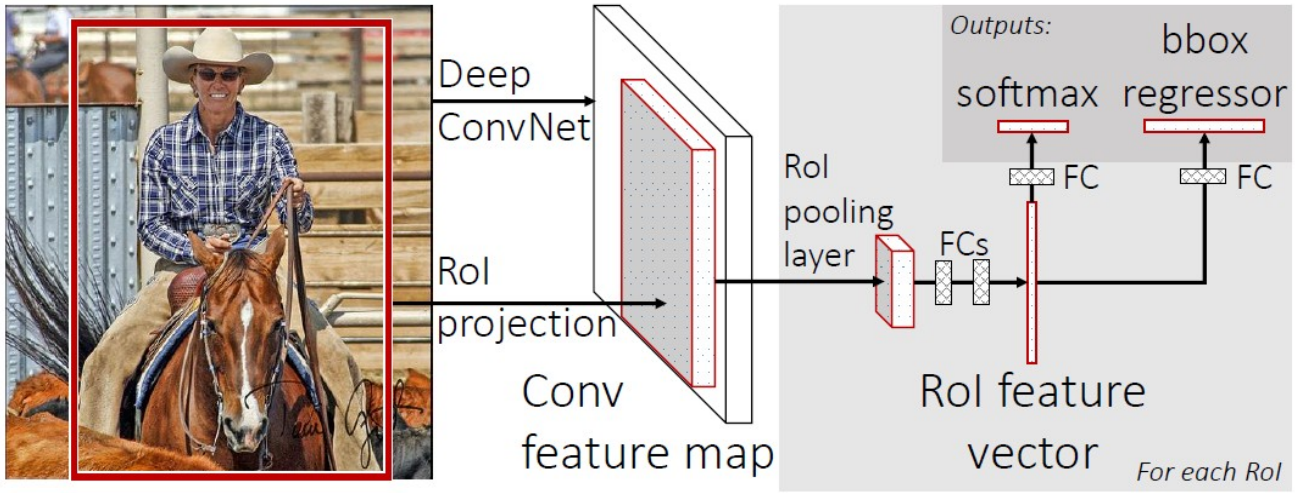


TL;DR: extract region proposals based on image features, run each proposal region through a CNN and finally run each proposal presentation through a support vector machine for classification

Multi-stage object detection

The object detection process is split into different steps that require to process the image data multiple times.

Fast R-CNN, Girshick et al. 2015

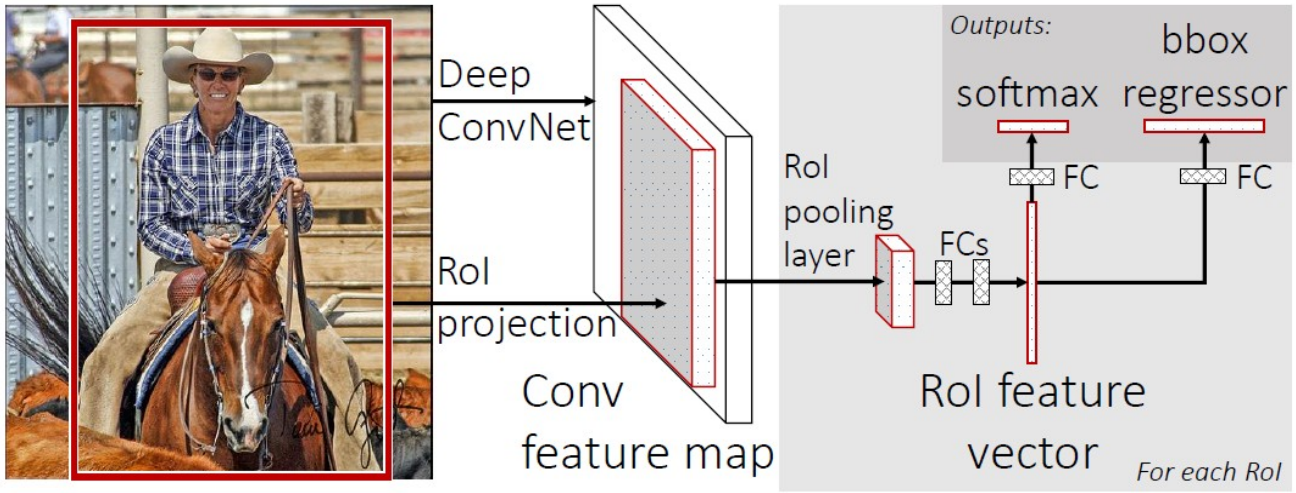


TL;DR: extract region proposals based on image features, run entire image through a CNN and extract presentations of proposal regions; each presentation then runs through a bounding box regressor and a classification model.

Multi-stage object detection

The object detection process is split into different steps that require to process the image data multiple times.

Faster R-CNN, Ren et al. 2015

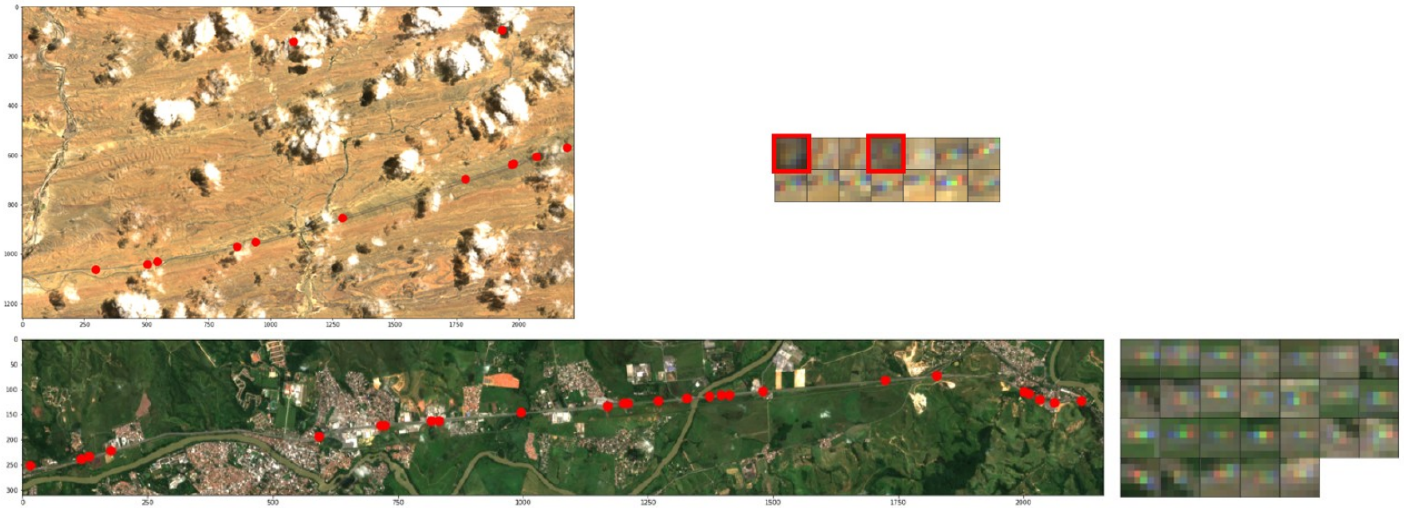


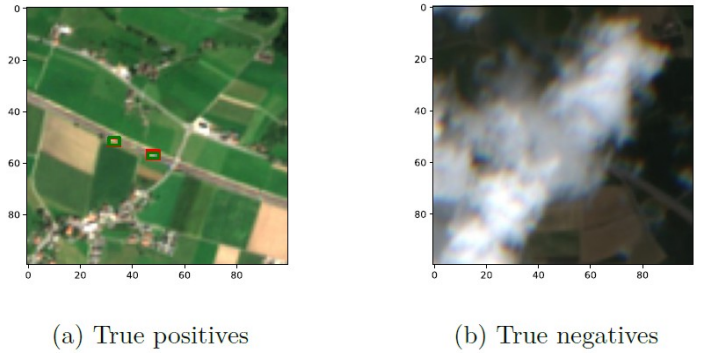
TL;DR: same as Fast R-CNN except that proposal regions are extracted from CNN feature map, which makes it much more efficient.

Faster R-CNN example

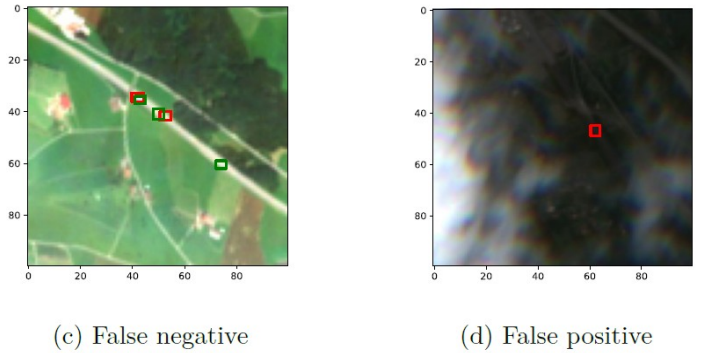
Extract trucks from Sentinel-2 image data with Faster R-CNN

Moritz Blattner, Master Thesis
Tackling Climate Change with ML Workshop @ ICML 2021





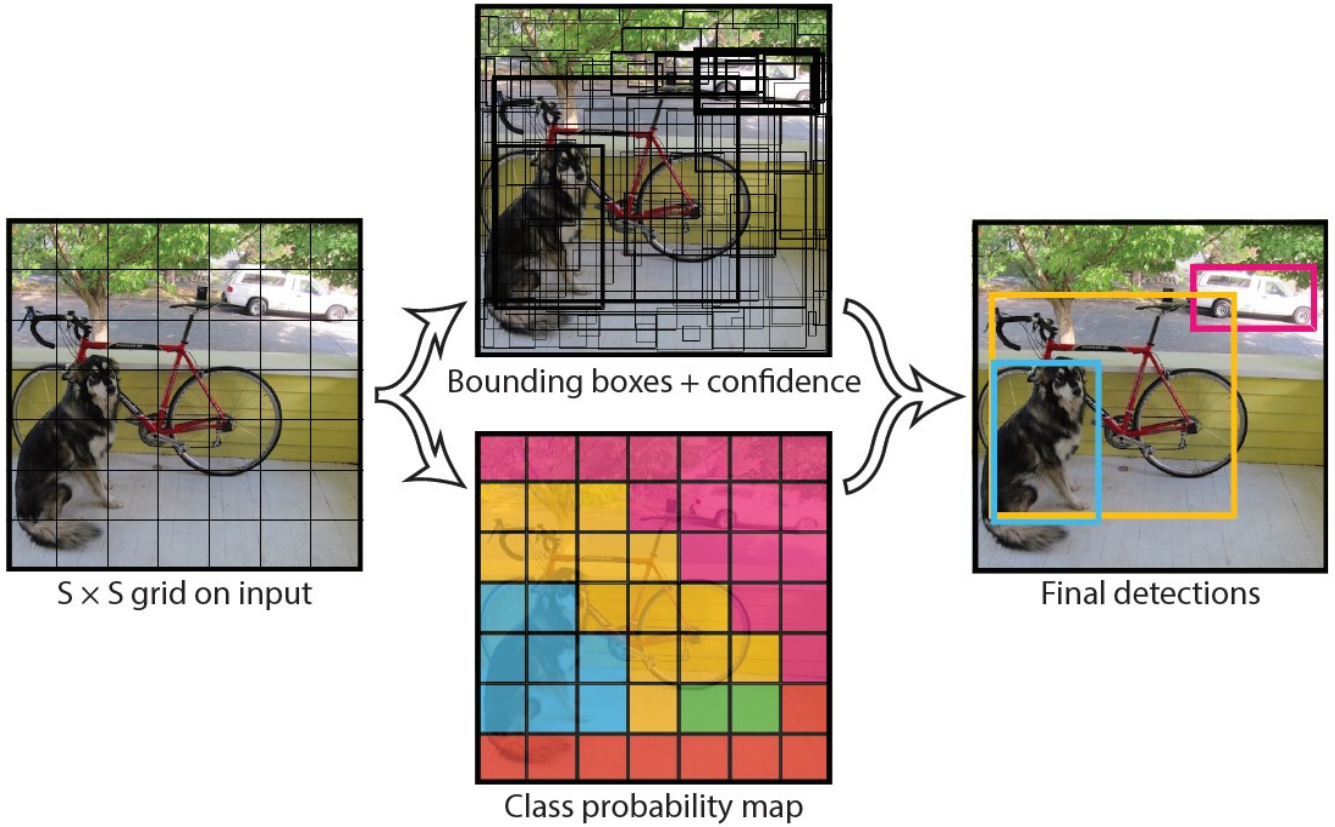
(a) True positives (b) True negatives



(c) False negative (d) False positive

Single-stage object detection: YOLO

Redmon et al. (2016)



TL;DR: predict bounding boxes and corresponding confidences, as well as class probabilities over a grid in one go; combination of both enable robust object detection

Example video: YOLOv3

That’s all folks!