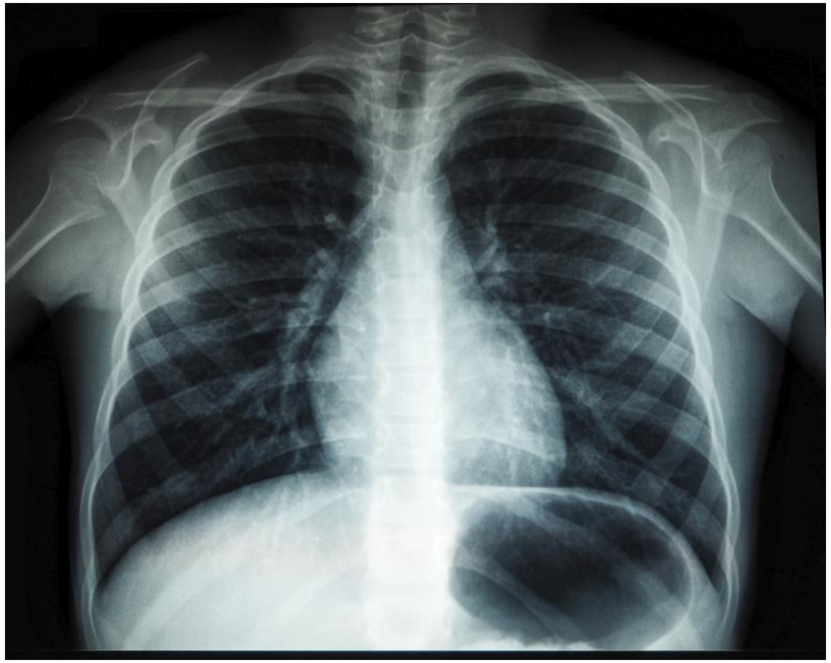
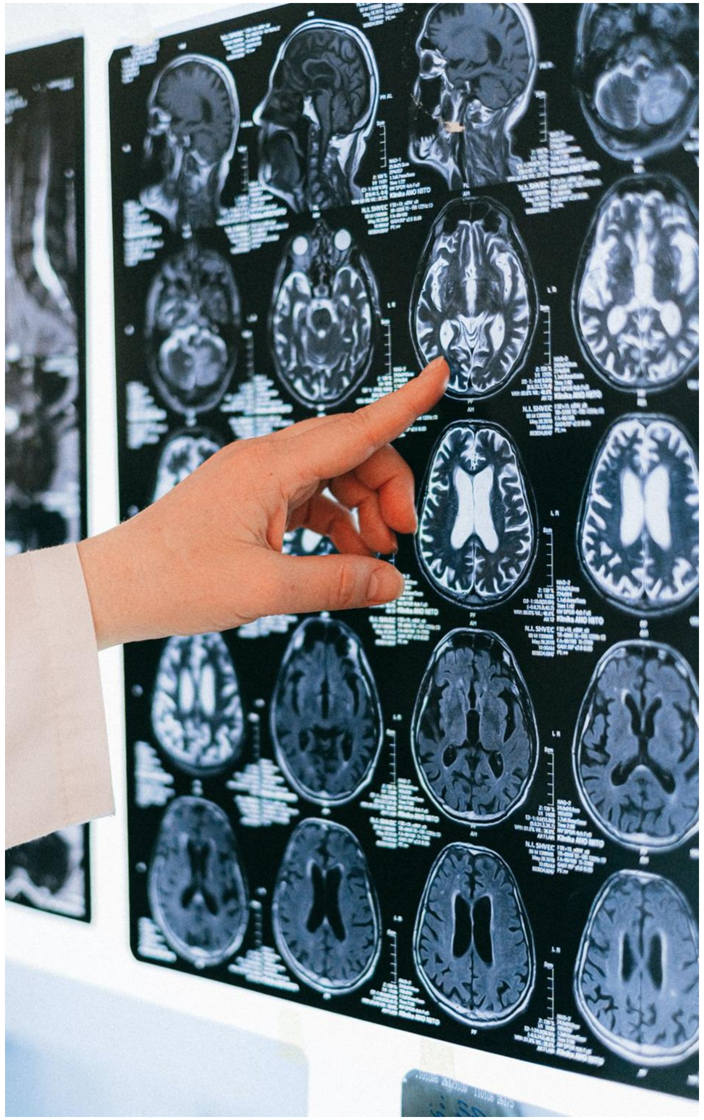
Machine Learning
Lecture 5 – Convolutional Neural Network (CNN)

A piece of advice

Do not feel overwhelmed by the mathematics – this is just to give you an overview of the knowledge of what is happening in the backend/background.

Computer Vision





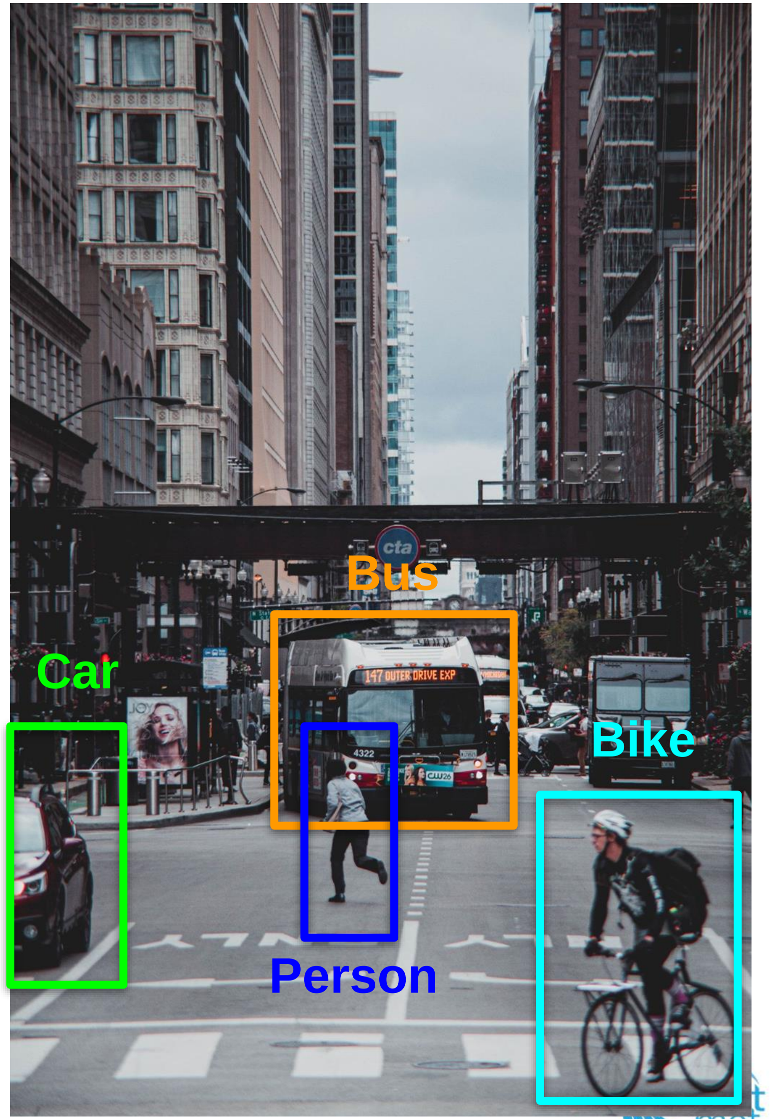


Image Representation

- An image consist of a grid of **pixels**, with each pixel having a **value** or a set of values
  - Greyscale: 1 value per pixel (0-255 / 0-1)
  - Colour images (RGB): 3 values per pixel

- Pixel values are considered to be the image **features**

- 2D grid is represented in code as a matrix

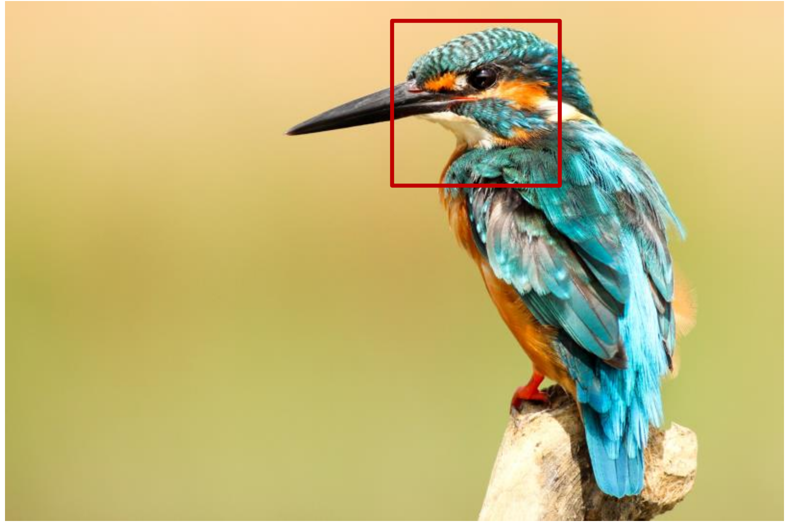
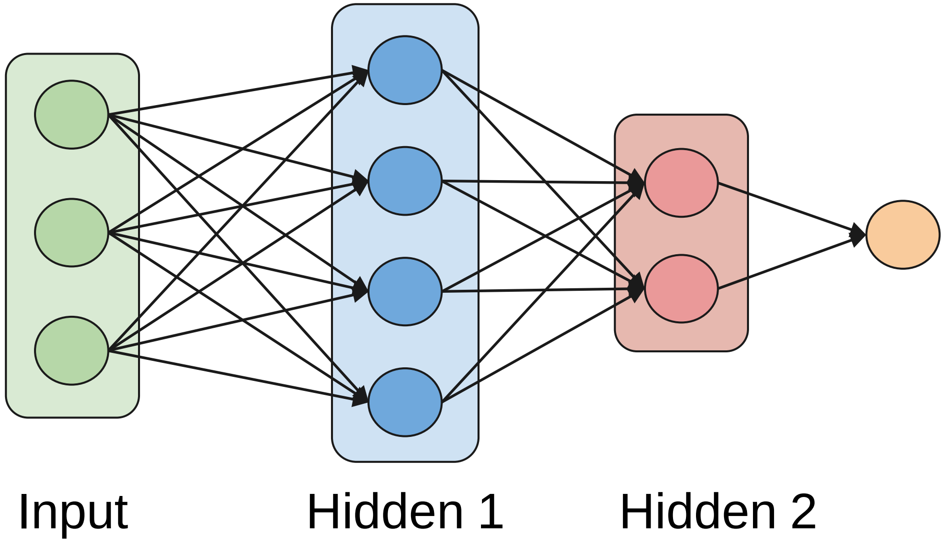


Image Features

- Using each image pixel as input value to a MLP results in extremely big networks
  - Image of 512 by 512 pixel: 262,144 input values

- If we look at images we do not observe individual pixel values but we look at the **whole context (**edges, areas with similar colour, connected lines)



Inspiration from Nature

- We have **individual neurons** in the visual cortex of the brain that respond to seeing specific patterns

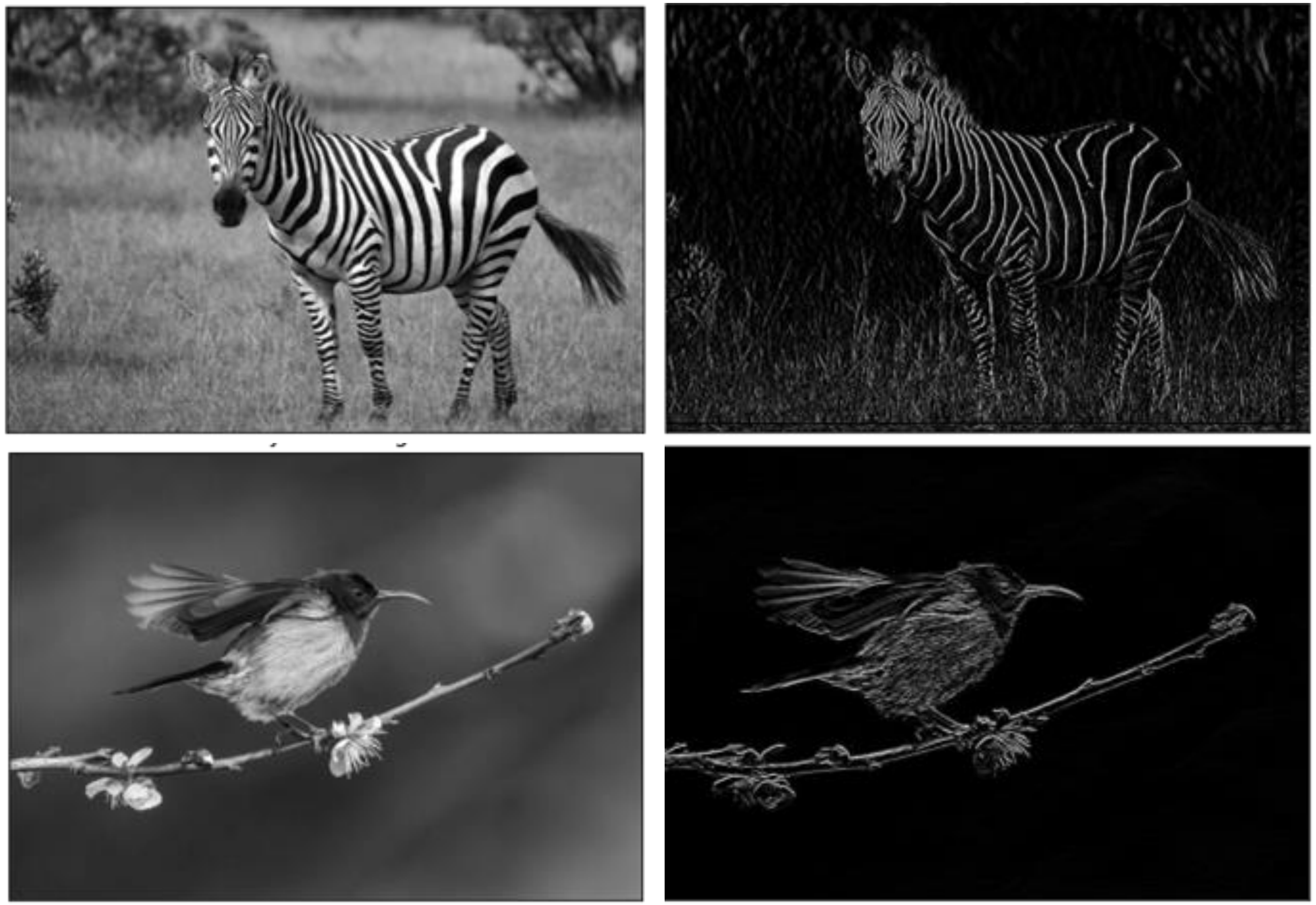
- Cats placed in a cylinder with only vertical lines directly after birth, could for the rest of their lives only see vertical lines*

- This shows that vision is **learned**, and that we have different visual neurons in the brain for **specific patterns** (e.g. horizontal lines, vertical lines)

* Blakemore and Cooper 1970: https://www.youtube.com/watch?v=RSNofraG8ZE

Feature extraction

- Can we use the idea of detecting specific patterns individually in computer vision?
- Obtain more meaningful image features than the pixel values
    - Horizontal edge detection
    - Vertical edge detection
    - Noise removal



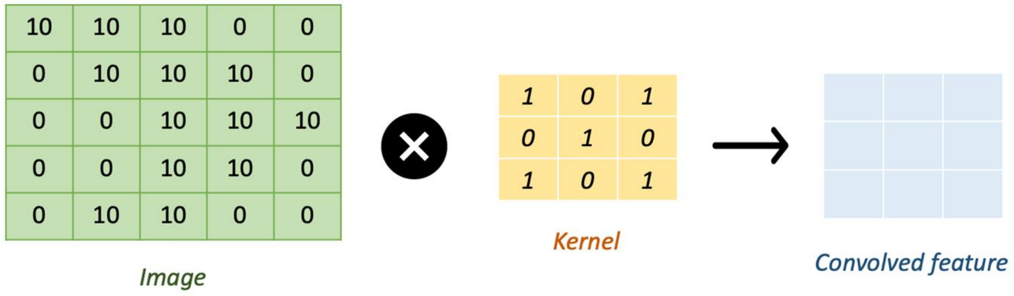
Convolutions

- In **a convolution,** a matrix with certain weights is moved over the whole image to obtain a filtered image

- This (usually small) matrix is called the **kernel** or **filter**

- Convolution of image $f(x, y)$ with kernel w is written as:

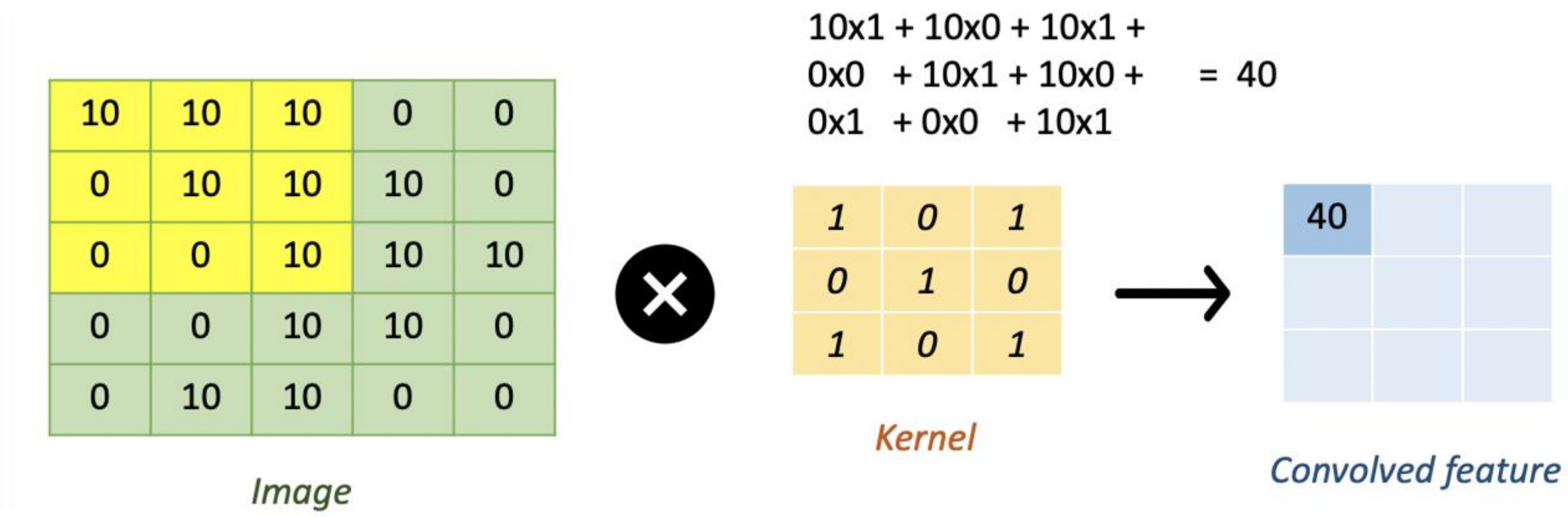
$$g(x, y) = w * f(x, y)$$



Kernel Design

- New pixel values are determined by multiplying the kernel values with the pixel values and summing these together for each position

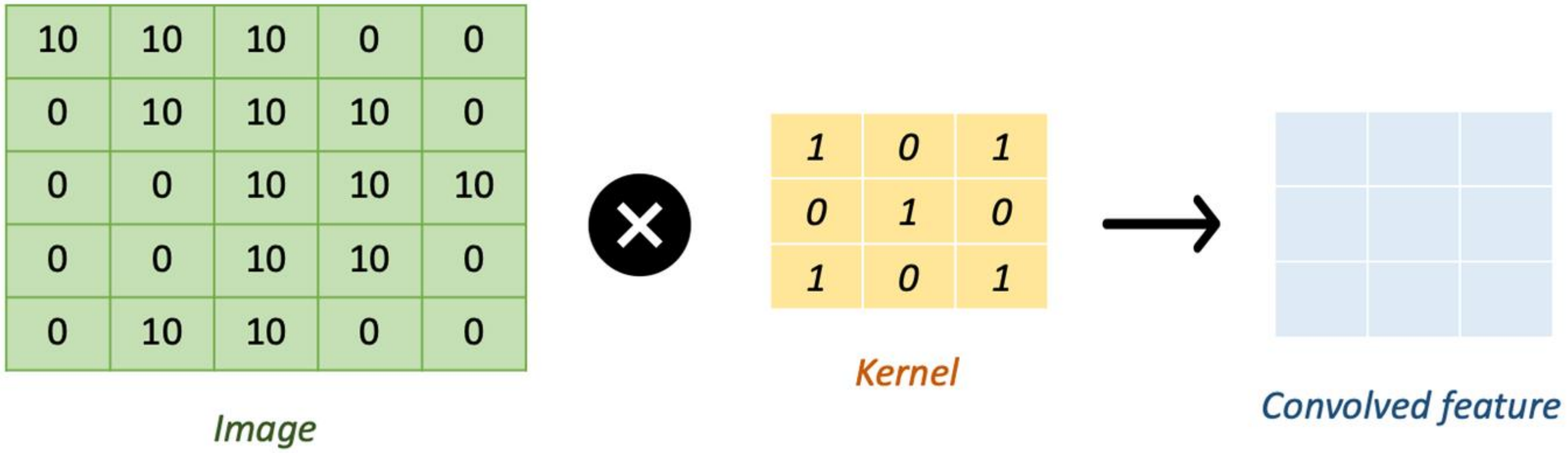
- The kernel can have a variable size (3x3 in the example)



Kernel Design

- New pixel values are determined by multiplying the kernel values with the pixel values and summing these together for each position

- The kernel can have a variable size (3x3 in the example)



Vertical Edge Detector

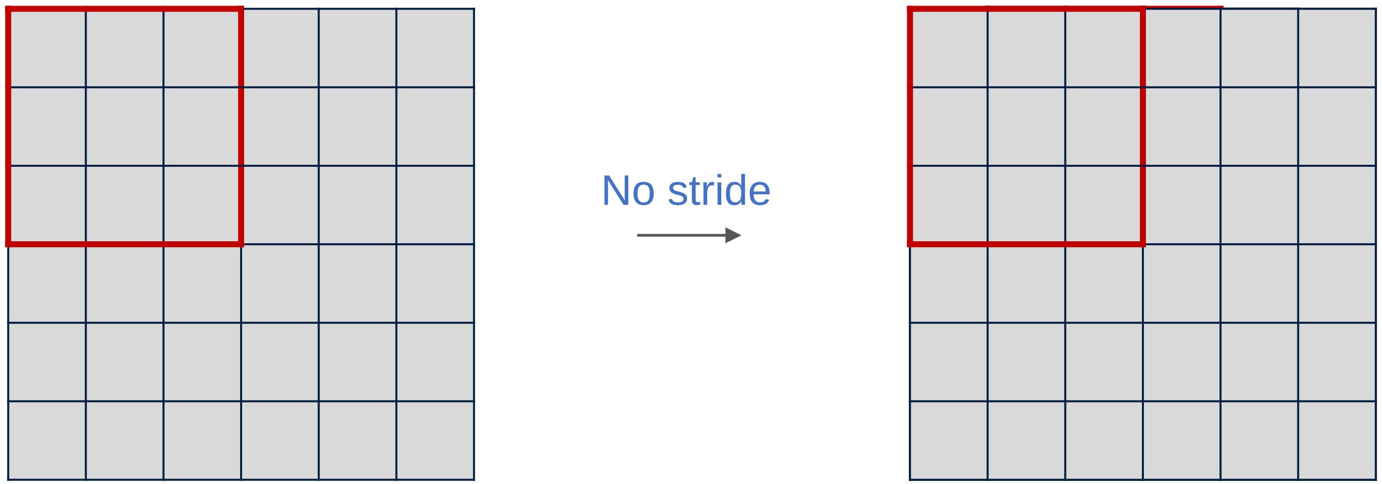
$*$

1	0	-1
1	0	-1
1	0	-1

$=$

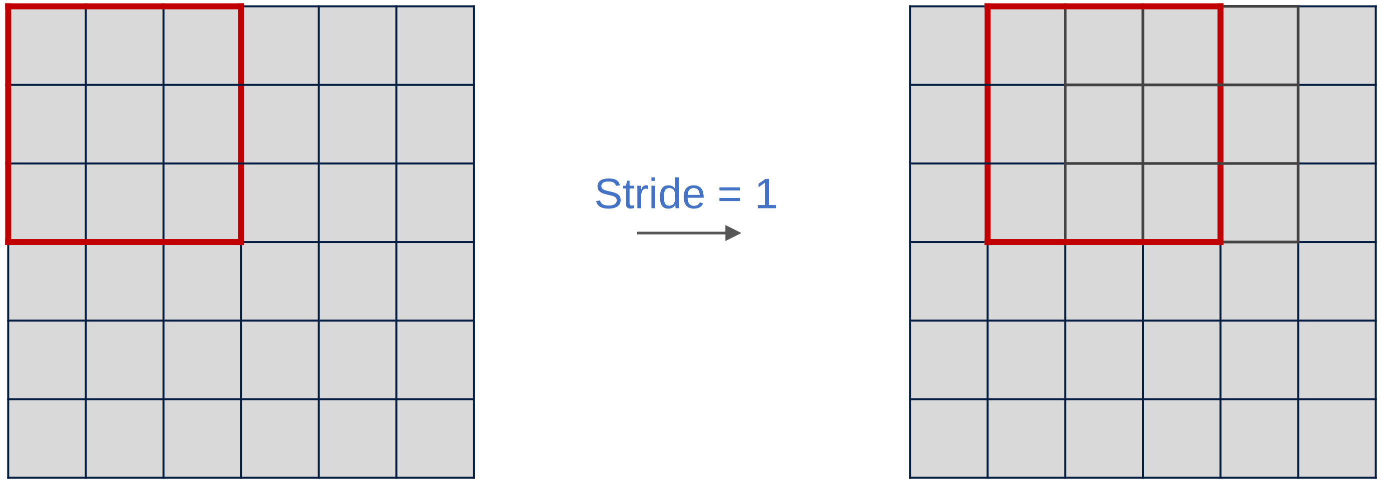
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1



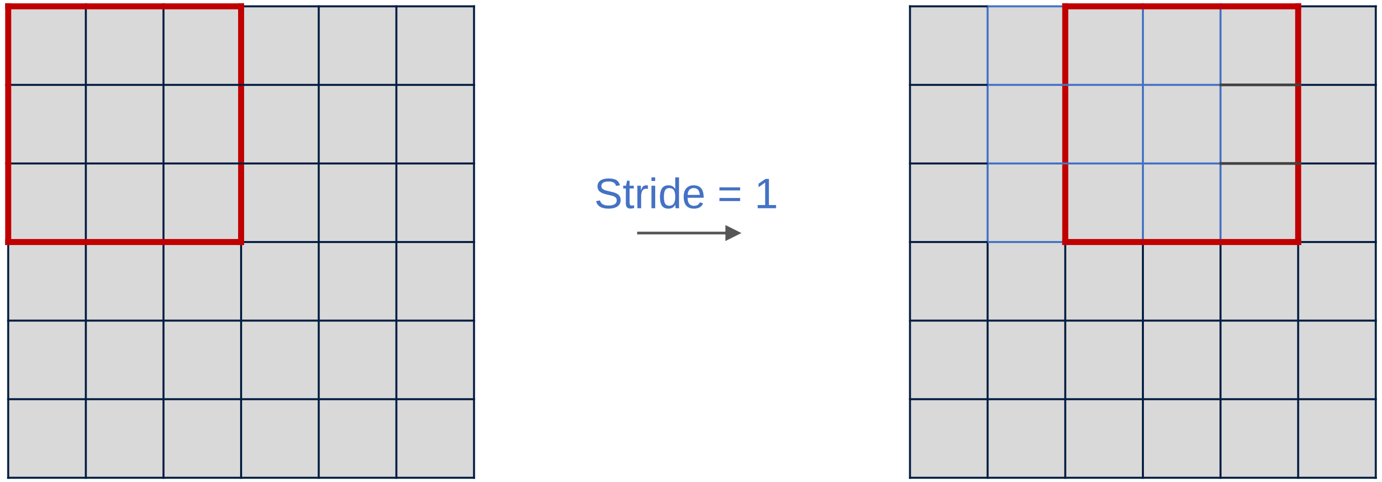
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1



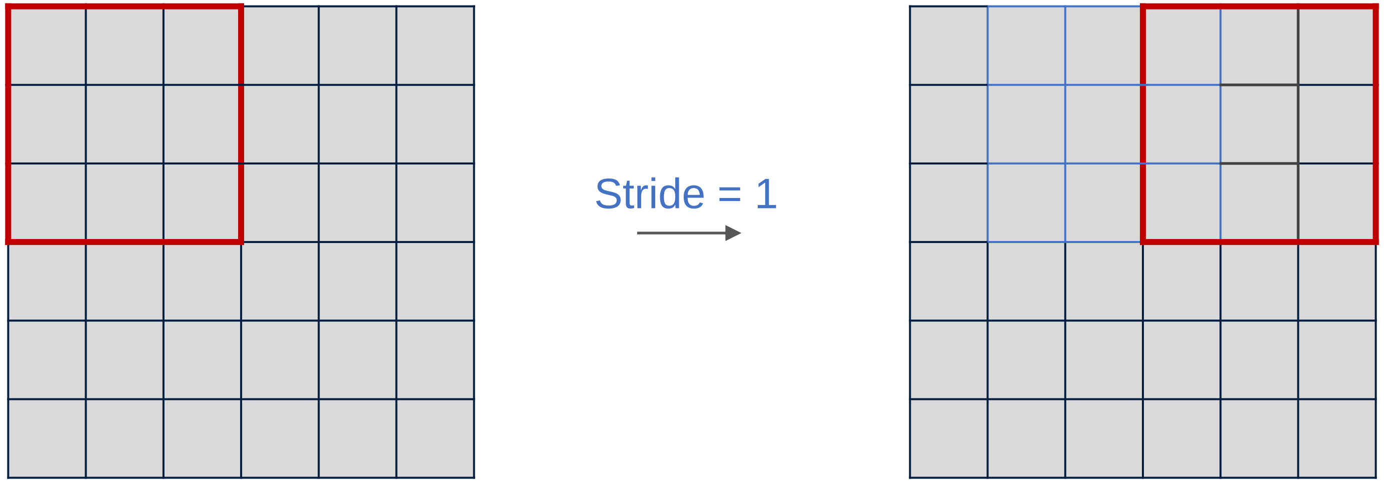
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1



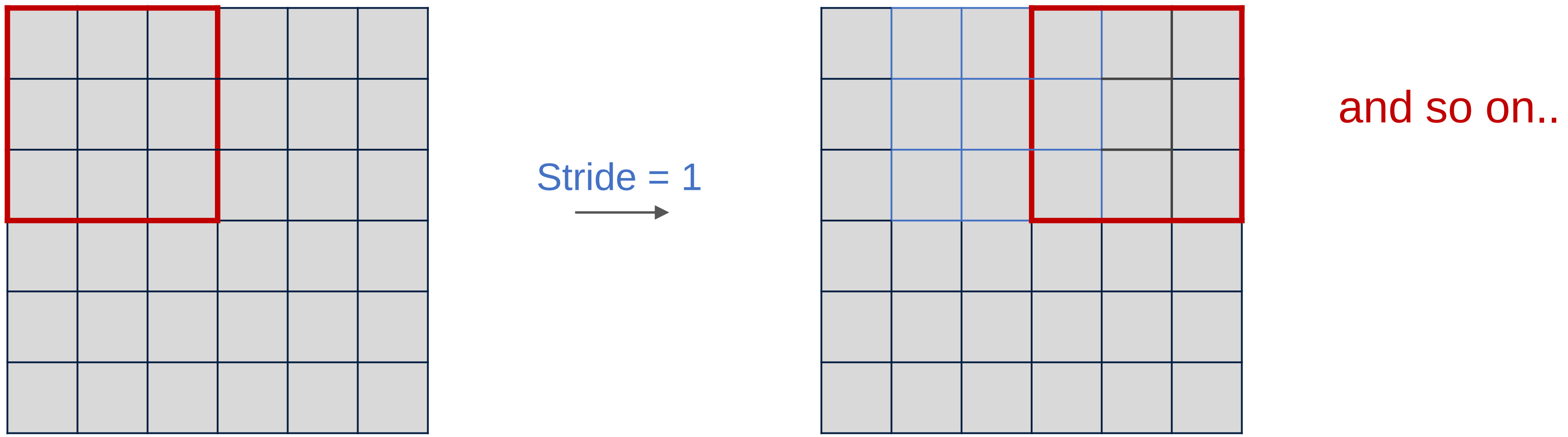
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1



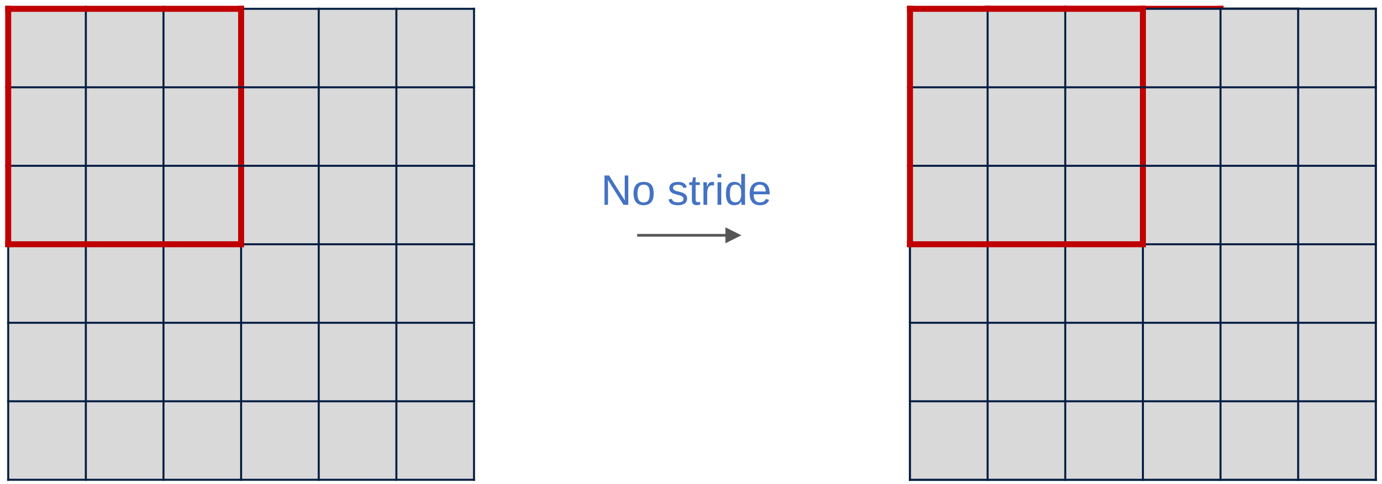
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1



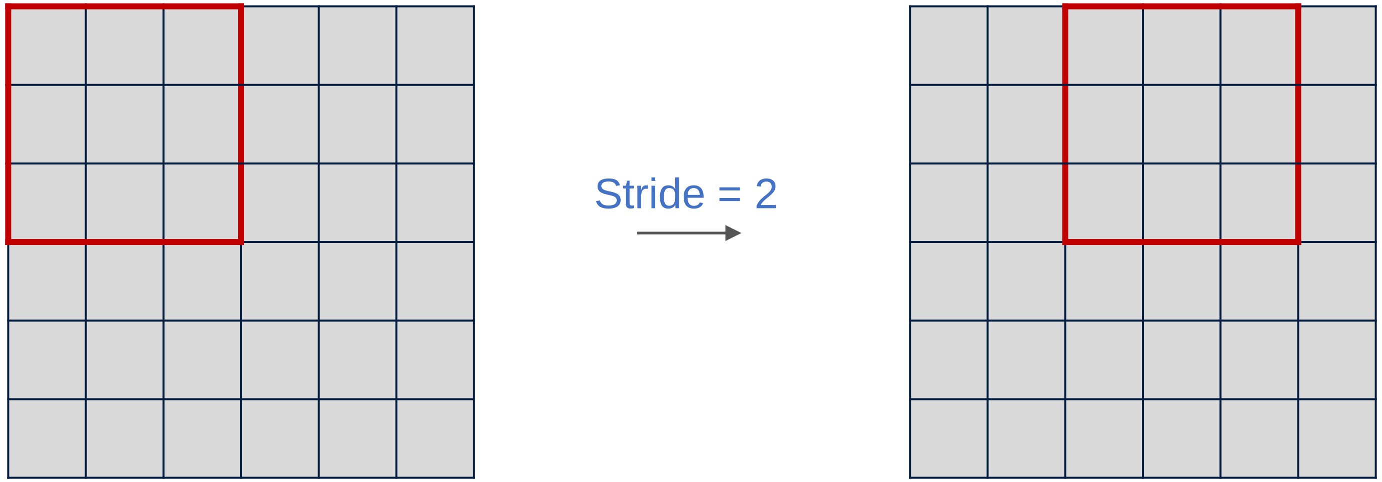
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1
  - Sometimes stride of 2



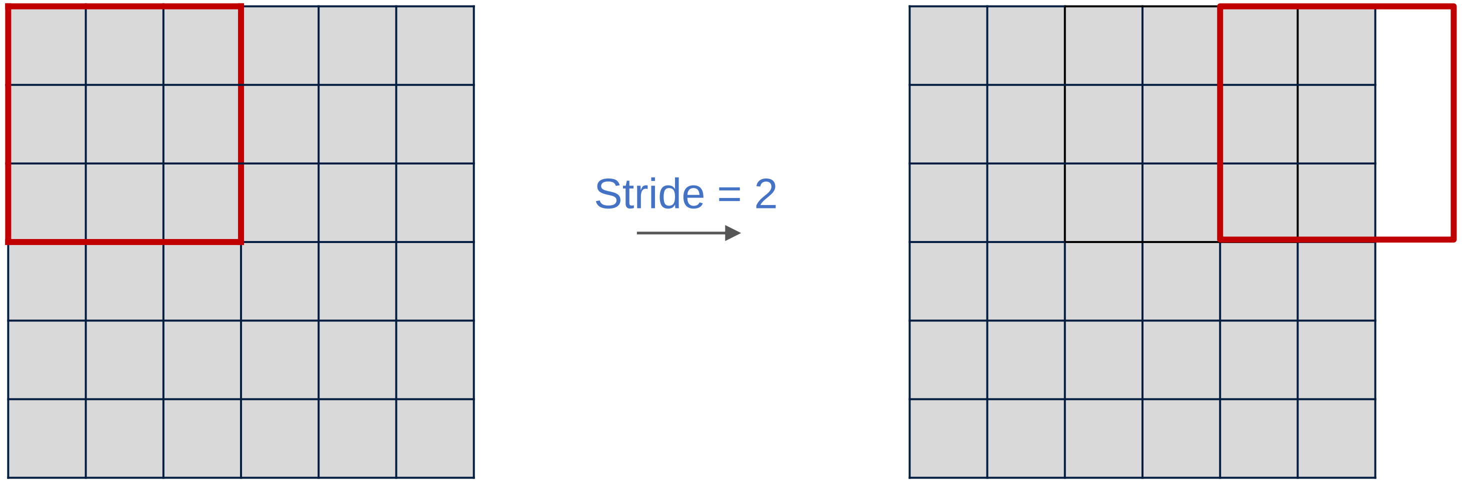
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1
  - Sometimes stride of 2



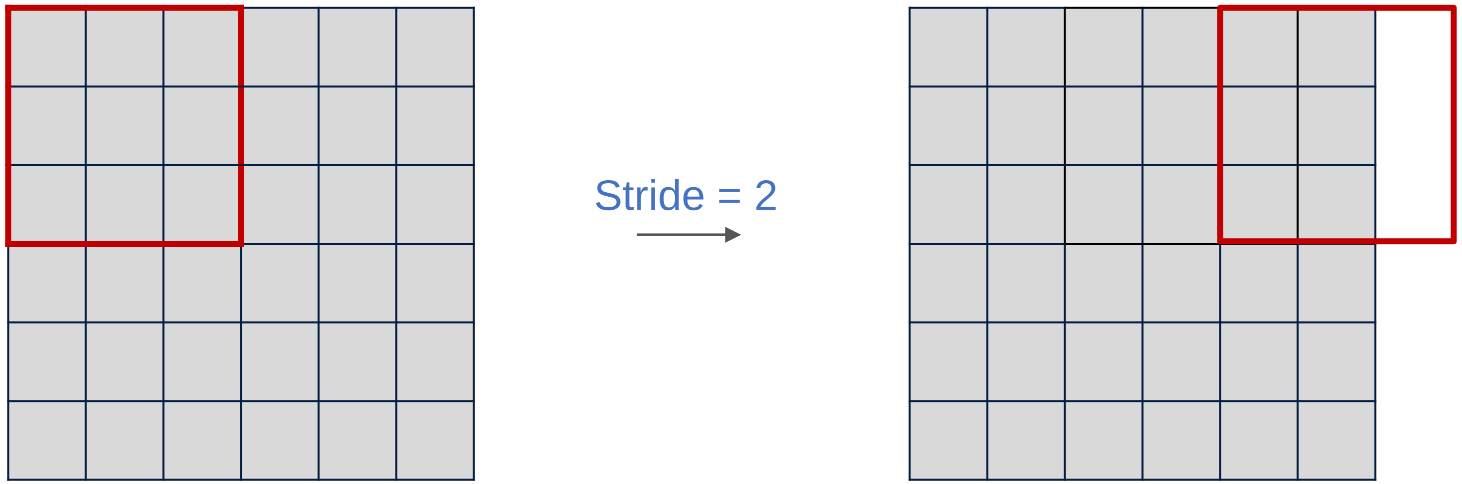
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1
  - Sometimes stride of 2



Kernel movement

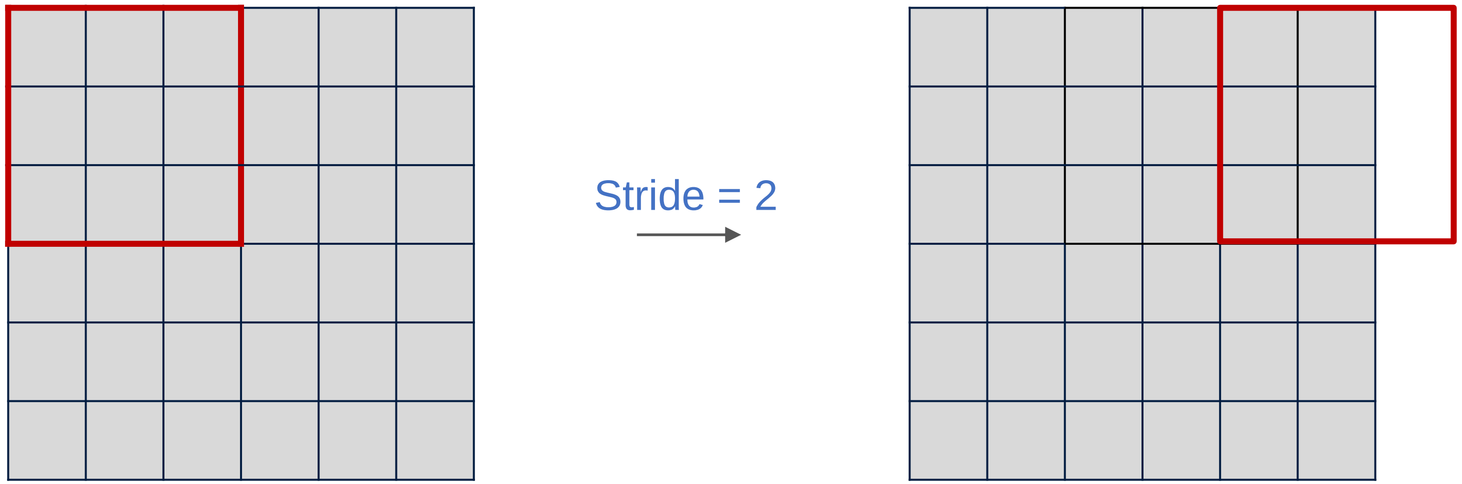
- Kernel stride: shift of kernel
    - Usually stride of 1
    - Sometimes stride of 2



Out of image bounds - Problem?

Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1
  - Sometimes stride of 2
- Kernel padding:
  - Usually zero padding

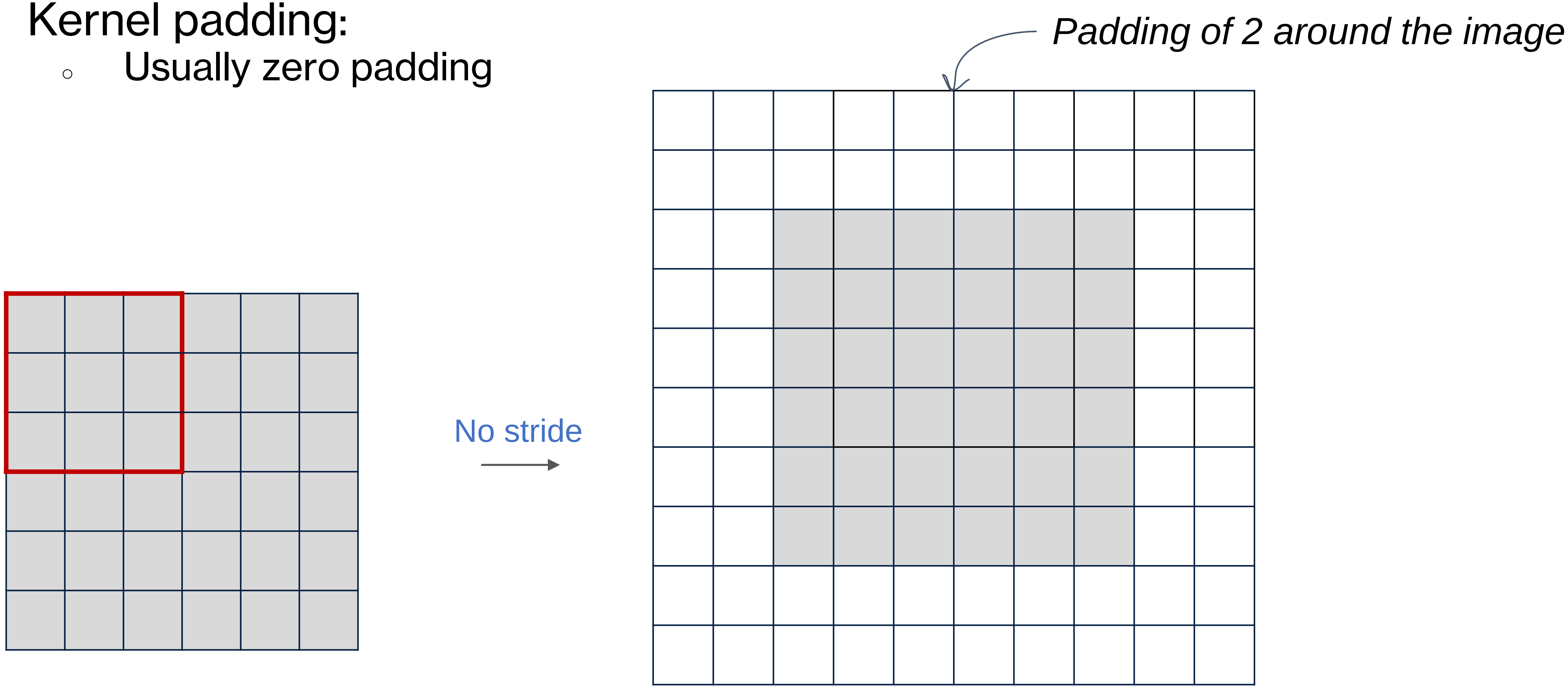


Out of image bounds - Problem?

Solved by padding

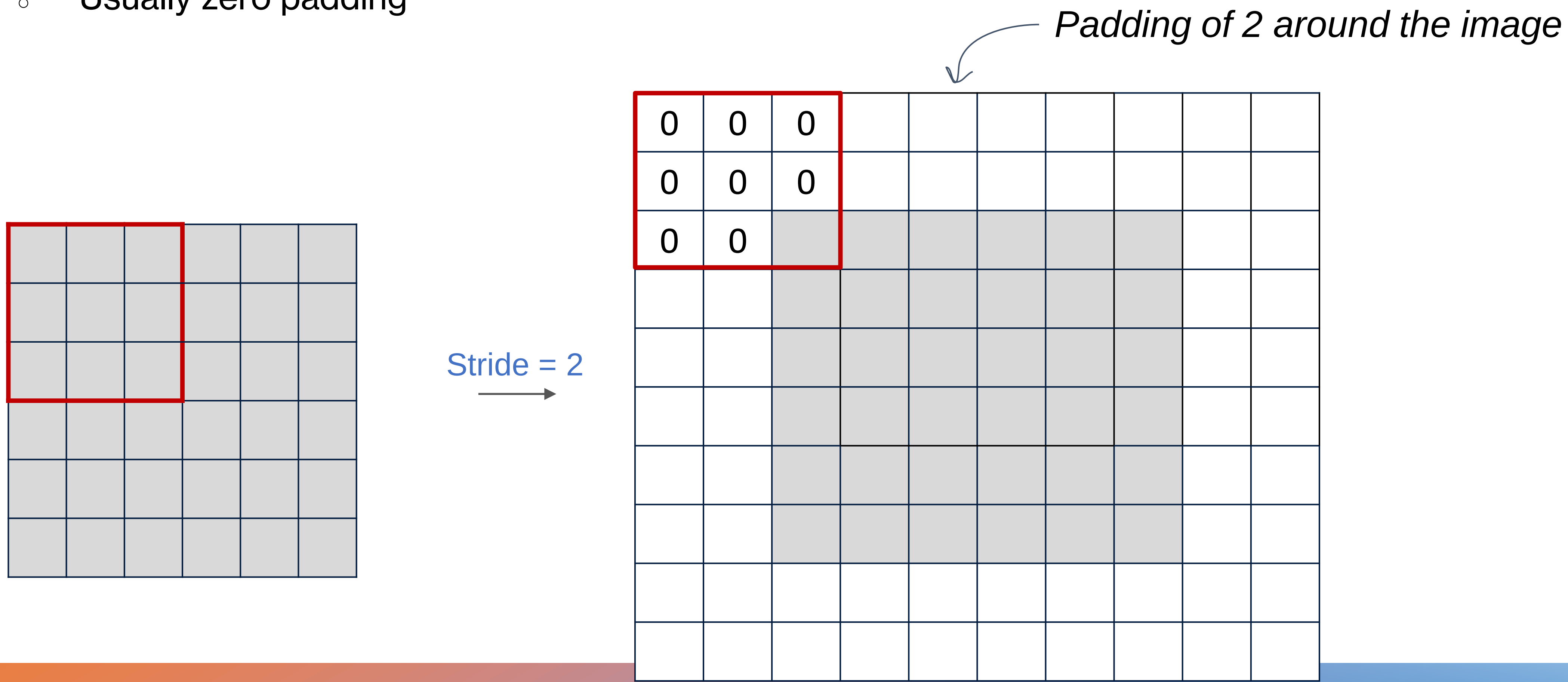
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1
  - Sometimes stride of 2
- Kernel padding:
  - Usually zero padding



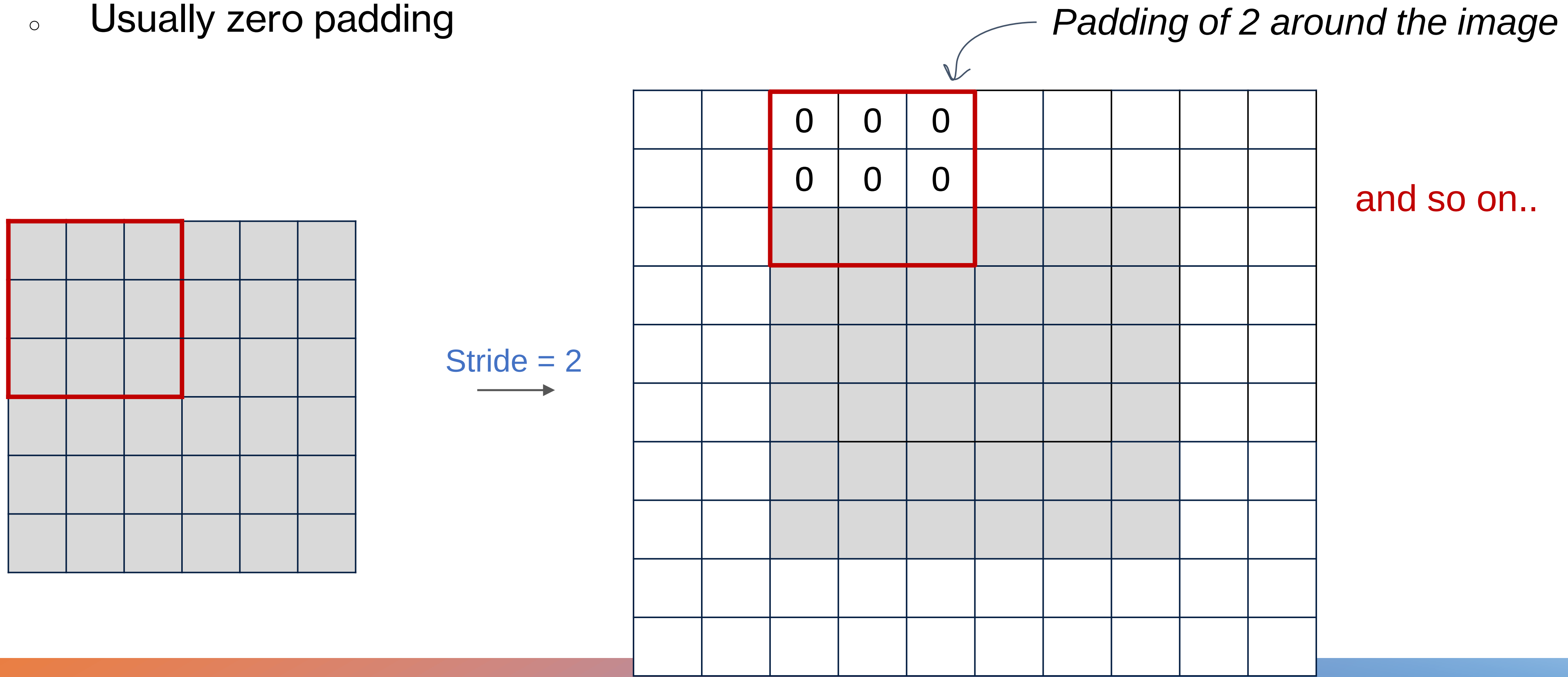
Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1
  - Sometimes stride of 2
- Kernel padding:
  - Usually zero padding

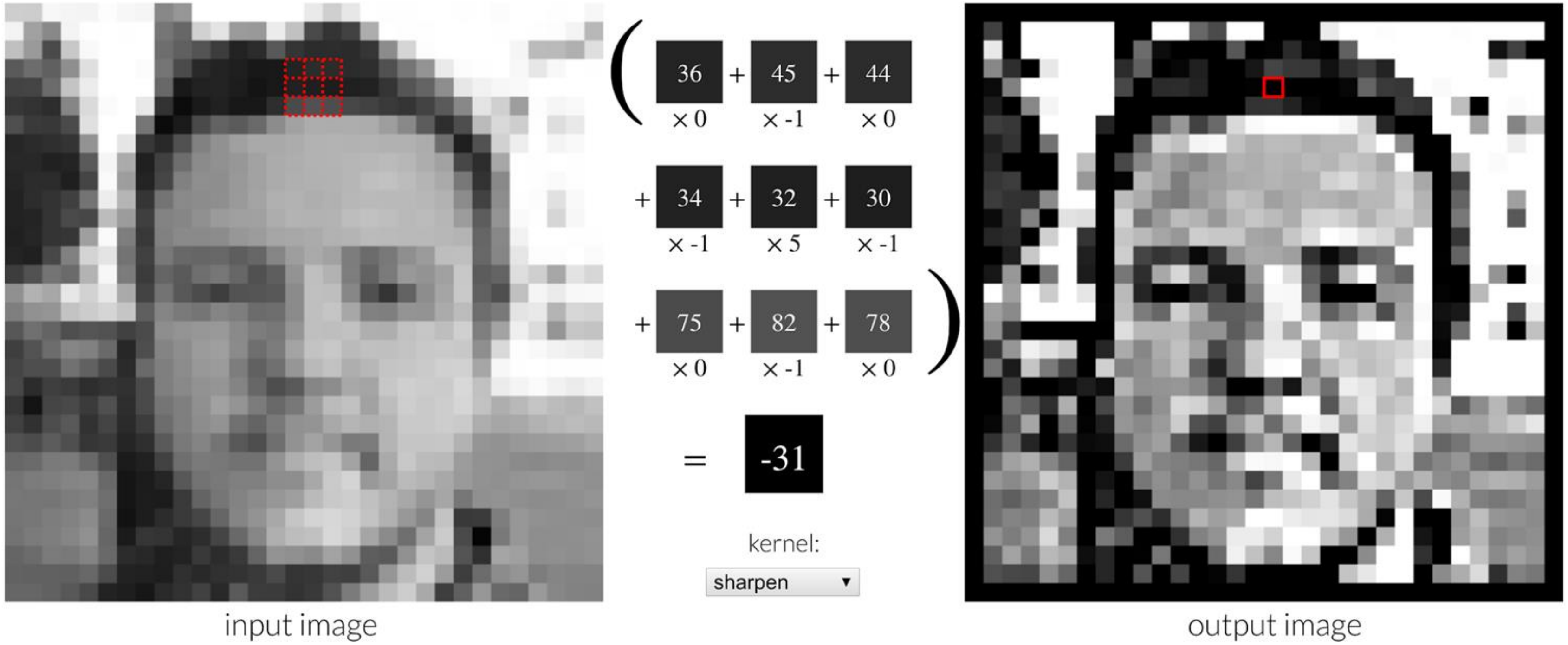


Kernel movement

- Kernel stride: shift of kernel
  - Usually stride of 1
  - Sometimes stride of 2
- Kernel padding:
  - Usually zero padding



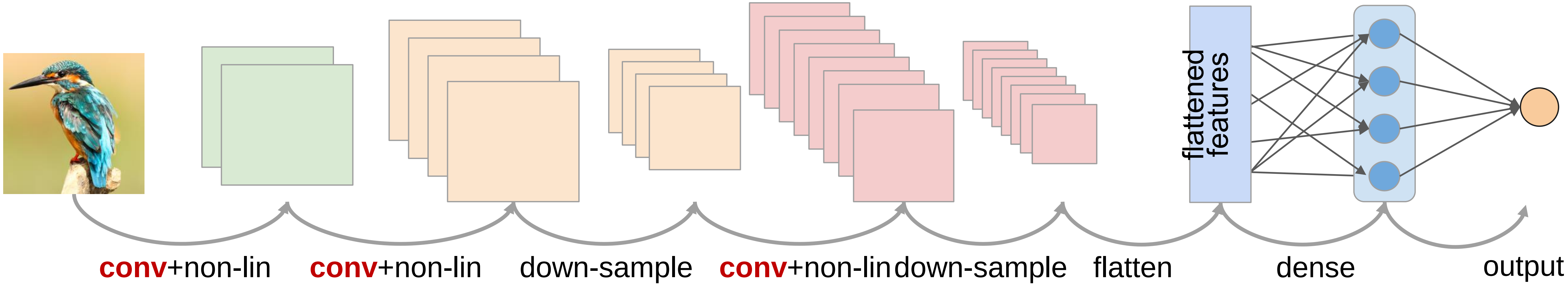
Convolution demo



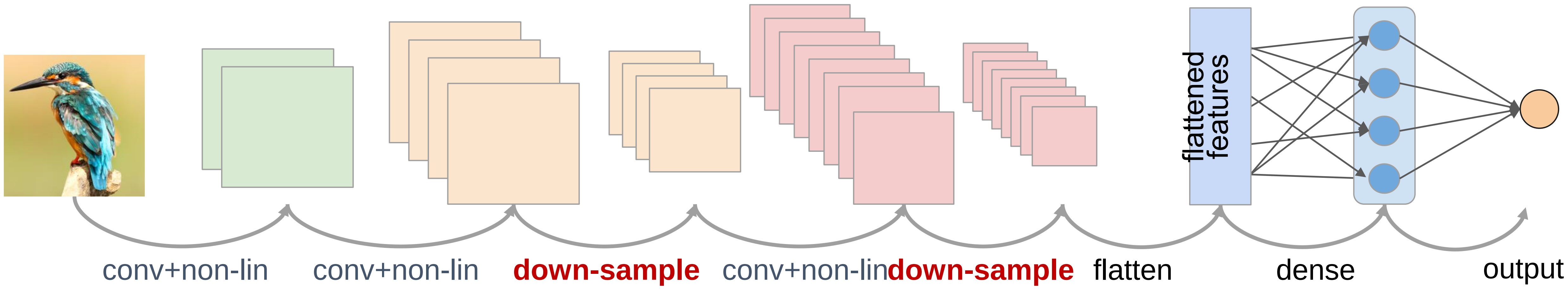
Live demo: http://setosa.io/ev/image-kernels/

Convolutional Neural Network

- Combine multiple convolutions into a network
    - In each layer, a number of kernels is applied
    - Each kernel results in a **feature map** (i.e. edge map)
- Values of the kernels are the weights of the network
    - Learned by training



Down-Sampling



Down-Sampling

- Feature maps resulting from convolutional layers need to be eventually **aggregated** into a single prediction

- This can be done by **down-sampling** between conv layers to gradually reduce dimensionality

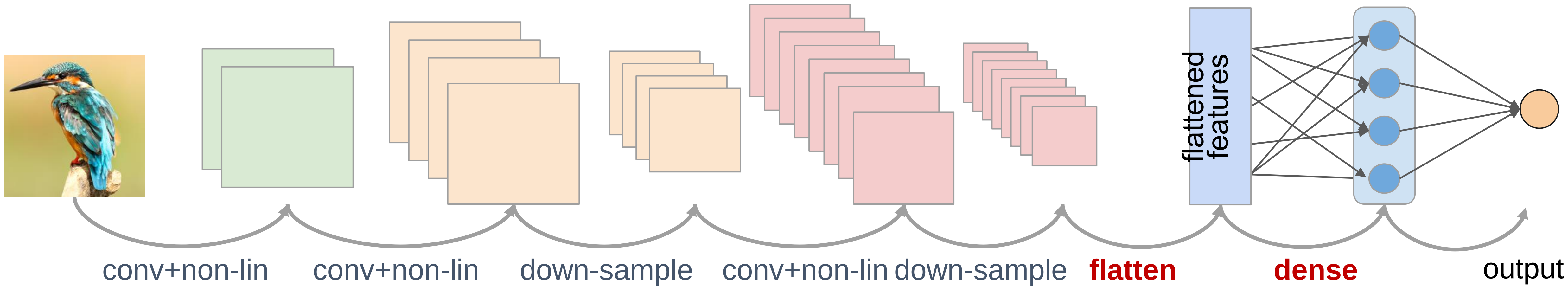
- Most commonly used down-sampling operation: **max-pool**
  - Move kernel over image and keep maximum value
  - Usually **kernel size** of 2x2 and **stride** of 2

4	1	2	8
9	2	7	1
2	5	3	1
0	3	1	2

$$\xrightarrow{\text{Max-pool}}$$

9	8
5	3

Dense Layers

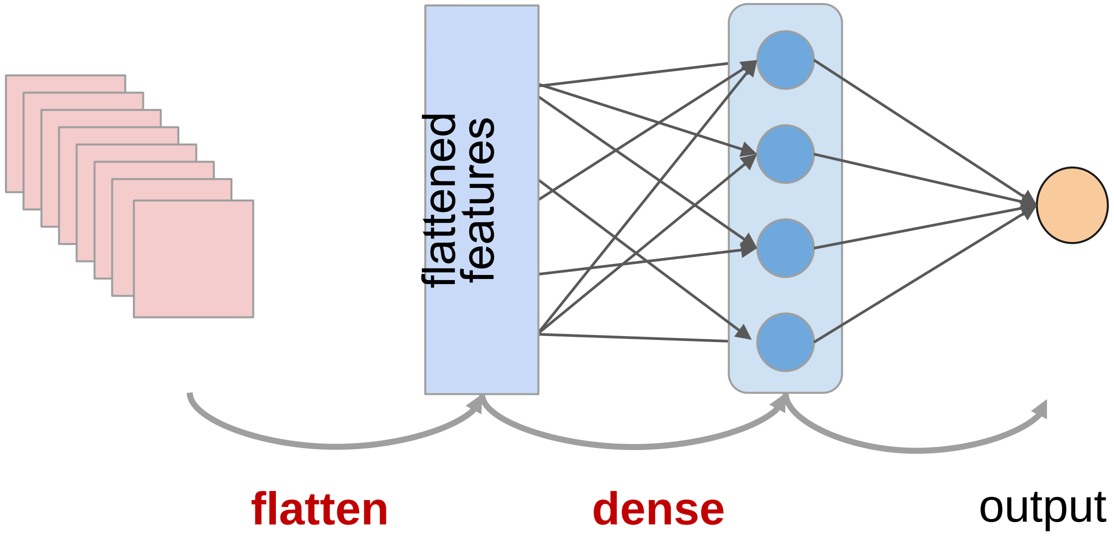


Dense Layers

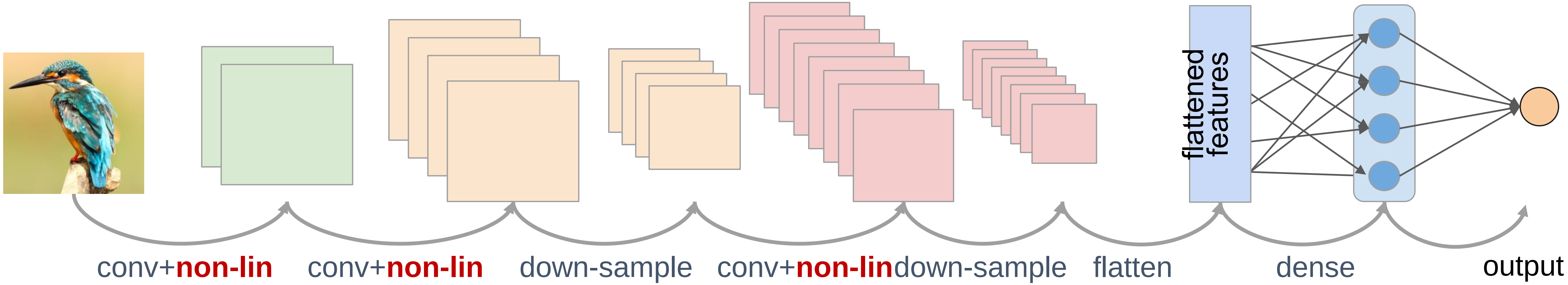
- To aggregate information into prediction after the final convolutional layer **dense** layers can be used

- Dense layer connect all input to all output (= standard layer in MLP), can also be called **fully-connected** layer

- Before a dense layer, the feature maps are converted into a 1D array, which is sometimes referred to as **flattening**



Activation Function

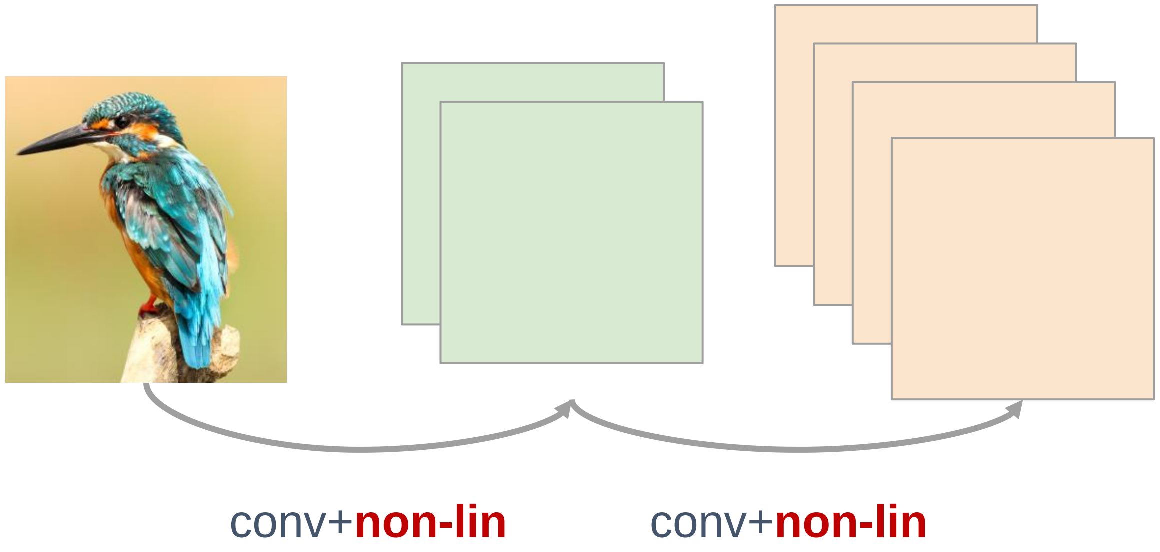


Activation Function

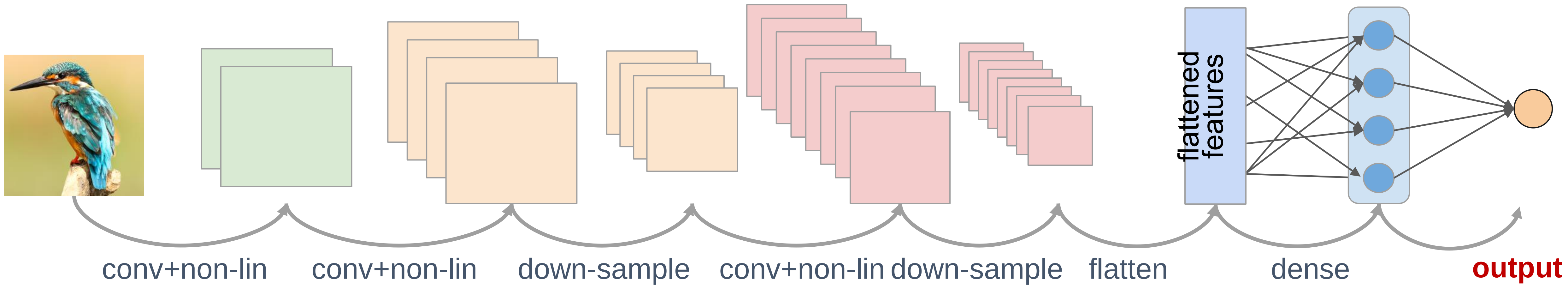
- As for standard MLP networks, every convolution layer has an **activation** function
  - Introduces non-linearity in the network

- **Rectified Linear Unit** (ReLU) is most commonly used, is defined as:

$$R(z) = max(0, z)$$



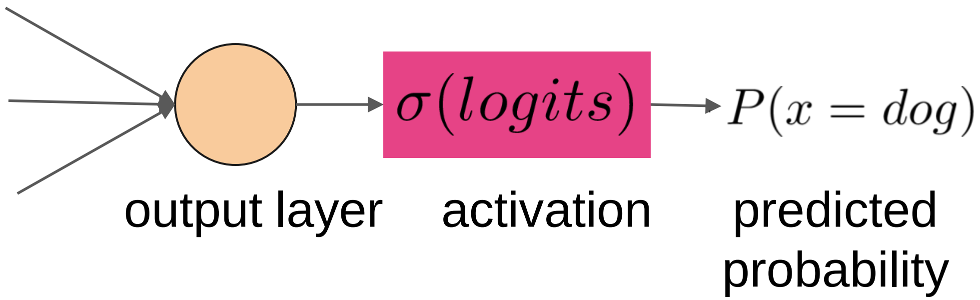
Output Layer

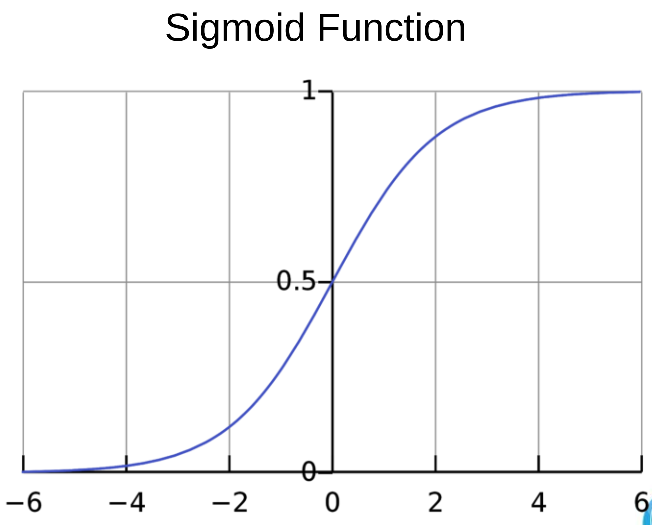


Output layer

- Output layer size and activation function depend on **task**
- Raw output of last layer (before any activation function takes place) are called **logits**
- Binary classification
  - Classifying an image **x** as either ‘dog’ or ‘cat’
  - We need **1 output neuron**, as:
  - Output probability should be between 0 and 1:
    - **Sigmoid** function (*squeezes* output between 0 and 1)

$$P(x = cat) = 1 - P(x = dog)$$

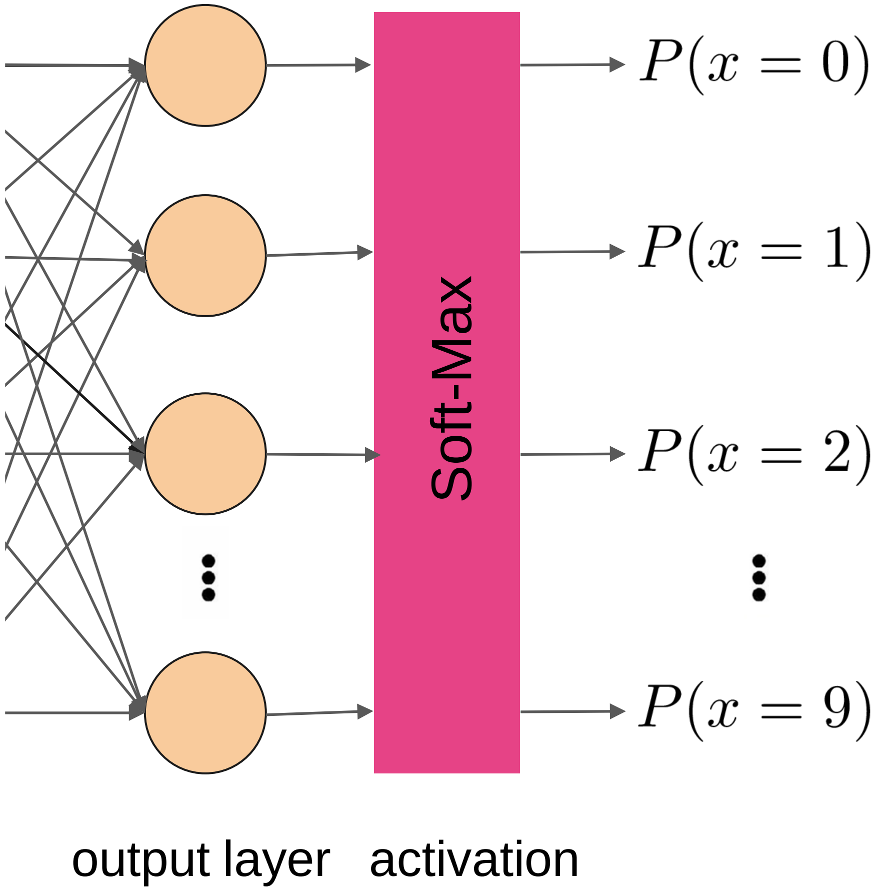




Output layer

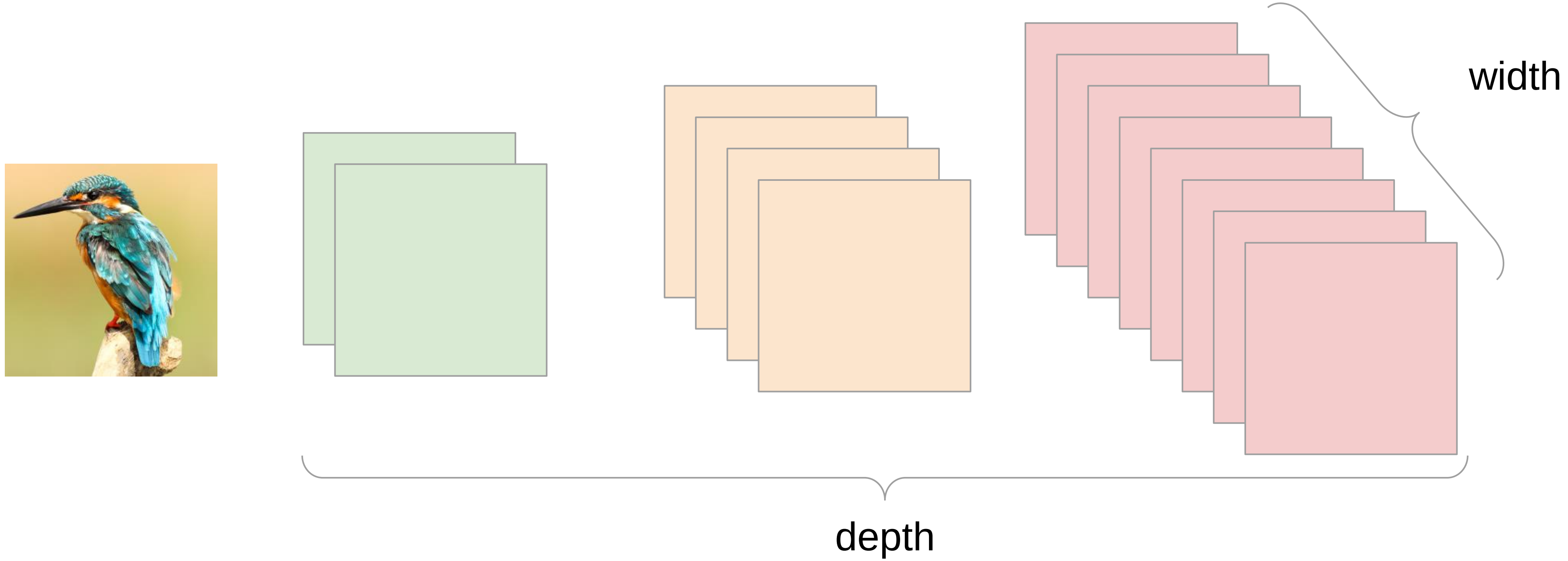
- Output layer size and activation function depend on **task**
- Raw output of last layer (before any activation function takes place) are called **logits**
- Multi-class classification
  - Classifying an image **x** as digit 0-9
  - Number of output neurons is equal to number of classes
  - Probabilities of individual output neurons should **sum up to 1**
    - **Soft-Max** function, given by:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{z_j}}$$



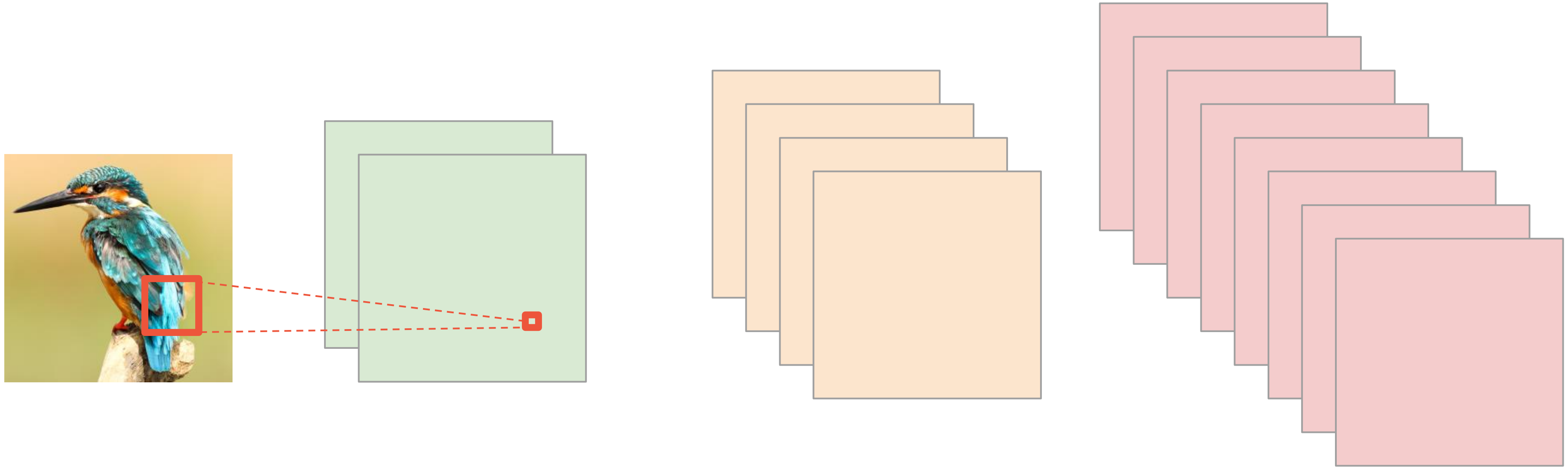
CNN architecture

- The **depth** of a network refers to the number of layers
- The **width** of a conv layer refers to the **number of feature maps** in the layer. This corresponds to the number of learned filters in the layer.



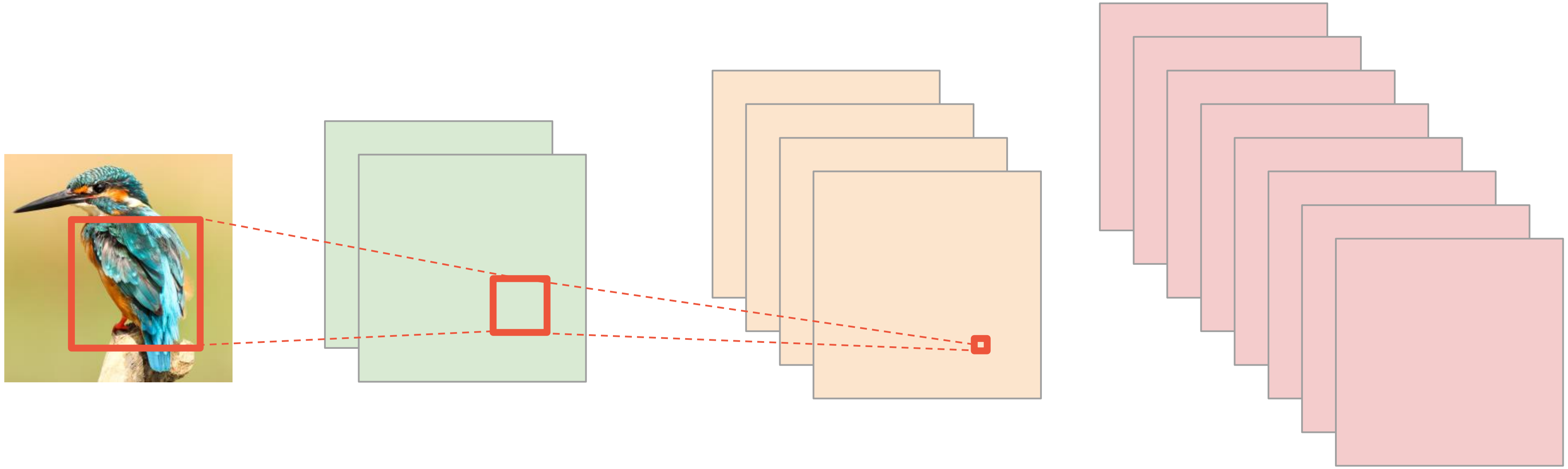
CNN architecture

- The **receptive field** shows which input pixels have influence over a given neuron in the feature map. As we go deeper in the neural network the receptive field grows larger and larger



CNN architecture

- The **receptive field** shows which input pixels have influence over a given neuron in the feature map. As we go deeper in the neural network the receptive field grows larger and larger



CNN architecture

- The **receptive field** shows which input pixels have influence over a given neuron in the feature map. As we go deeper in the neural network the receptive field grows larger and larger

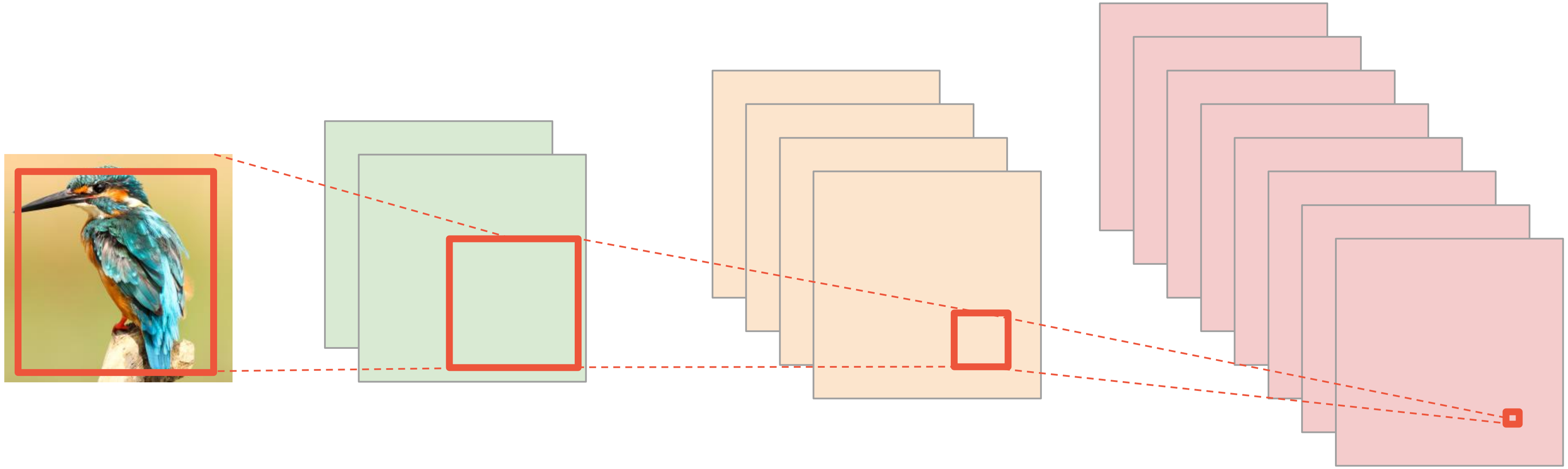


Image Dimensions

The dimensions of an image are: **[C, H, W]**

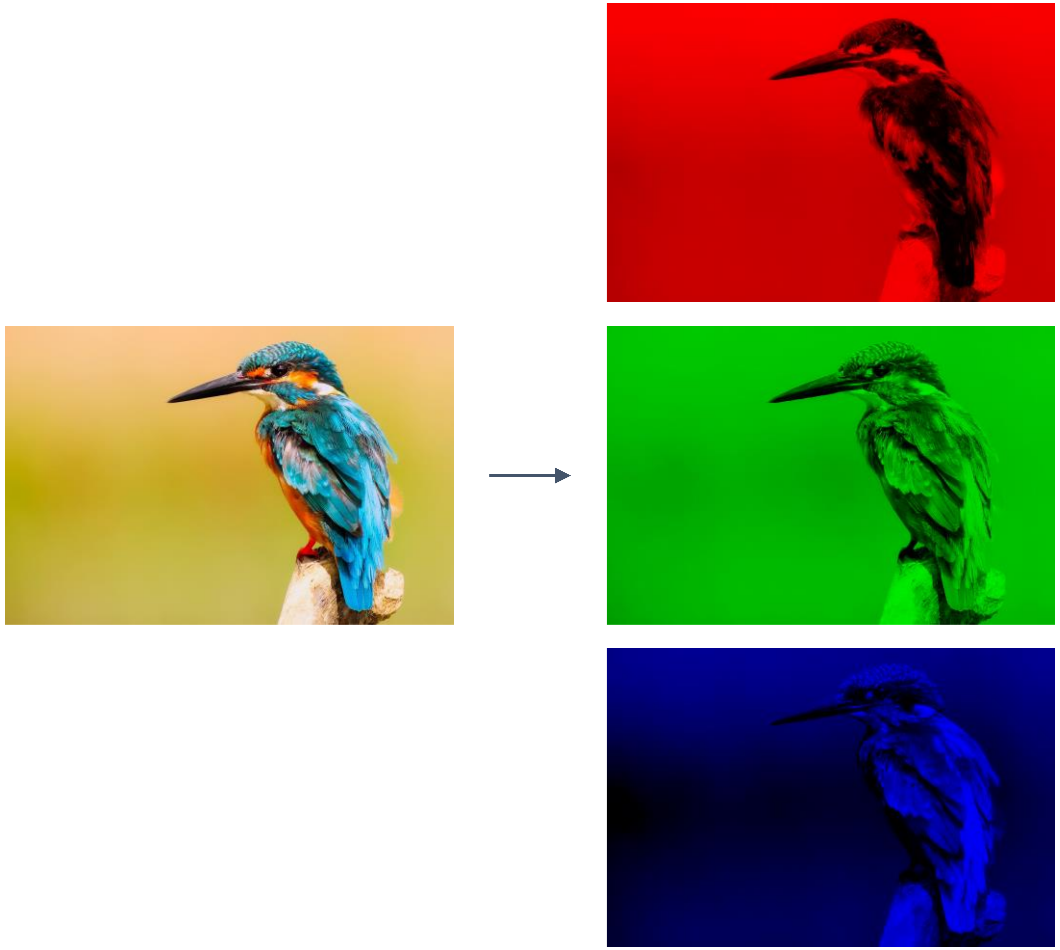
- **C** is the channel dimension
- **H** is the height of the image
- **W** is the width of the image

For greyscale images the **C** is 1, because we have 1 value per location: **[1, H, W]**

- This is equal to a dimension of [H,W] but for Pytorch computation we need the channel dimension

For color images the C can vary:

- For RGB the C is 3, as we have a Red, Green and Blue component: **[3, H, W]**



k = 1 .. C

Convolution Dimensions single-channel

Input:
[1, H, W]

*

0	-1	0
-1	**4**	**-1**
0	**-1**	**0**

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

Weight:
[2, 1, 3, 3]

=

Output:
[2, H, W]

Convolution Dimensions multi-channel

-1	0	1
-1	0	1
-1	0	1

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

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

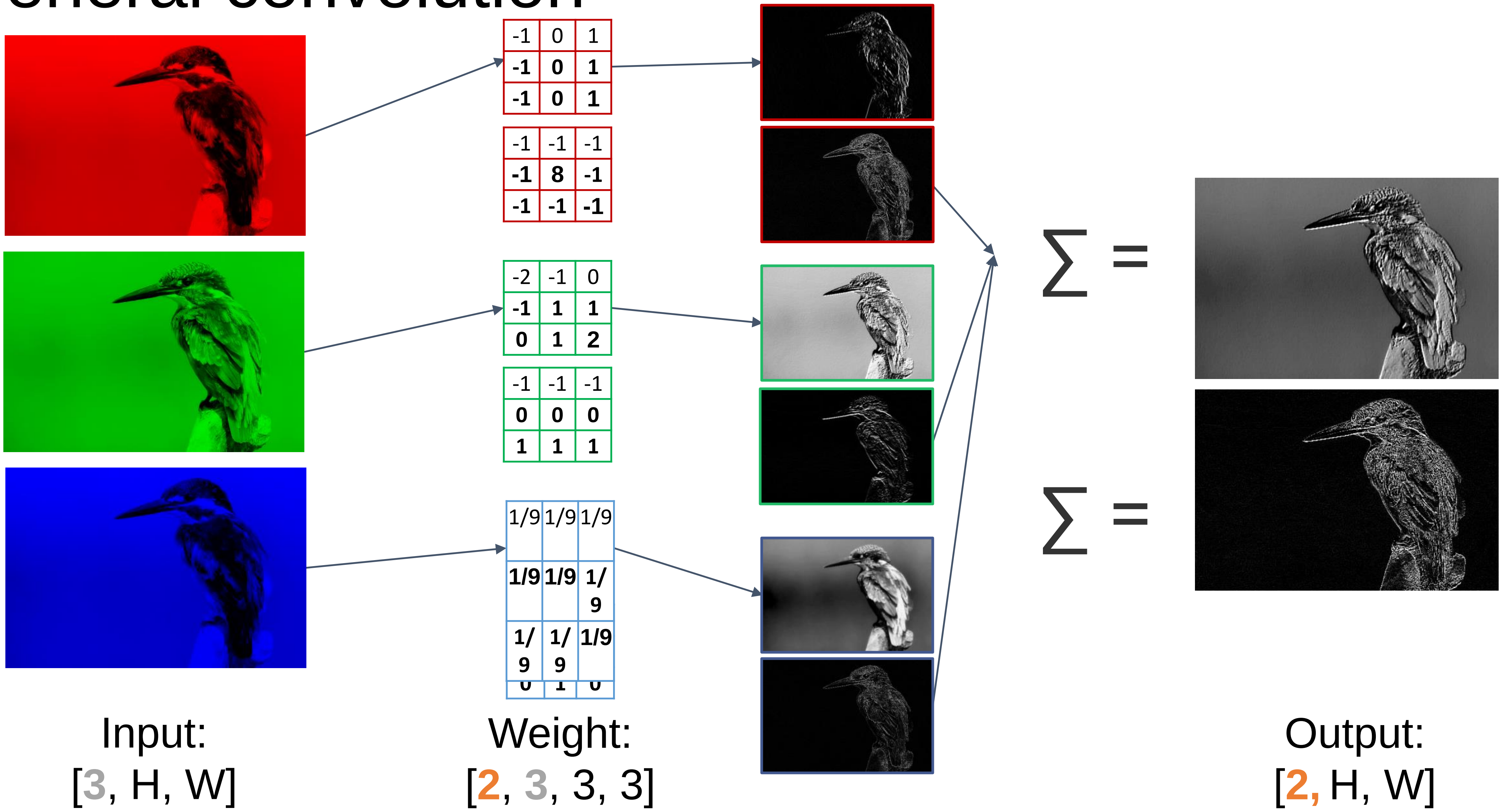
$\sum =$

Input:
[3, H, W]

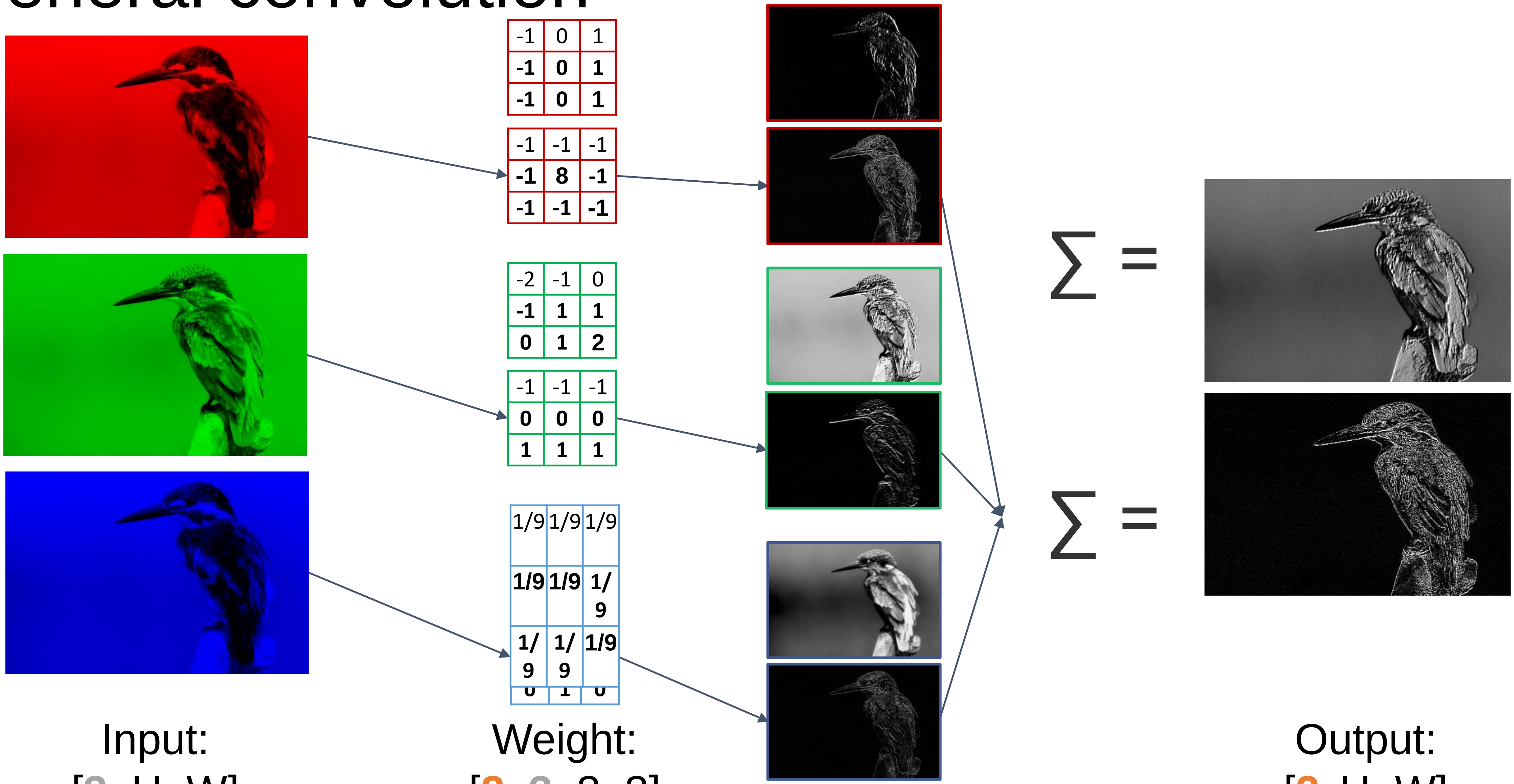
Weight:
[1, 3, 3, 3]

Output:
[1, H, W]

General convolution



General convolution

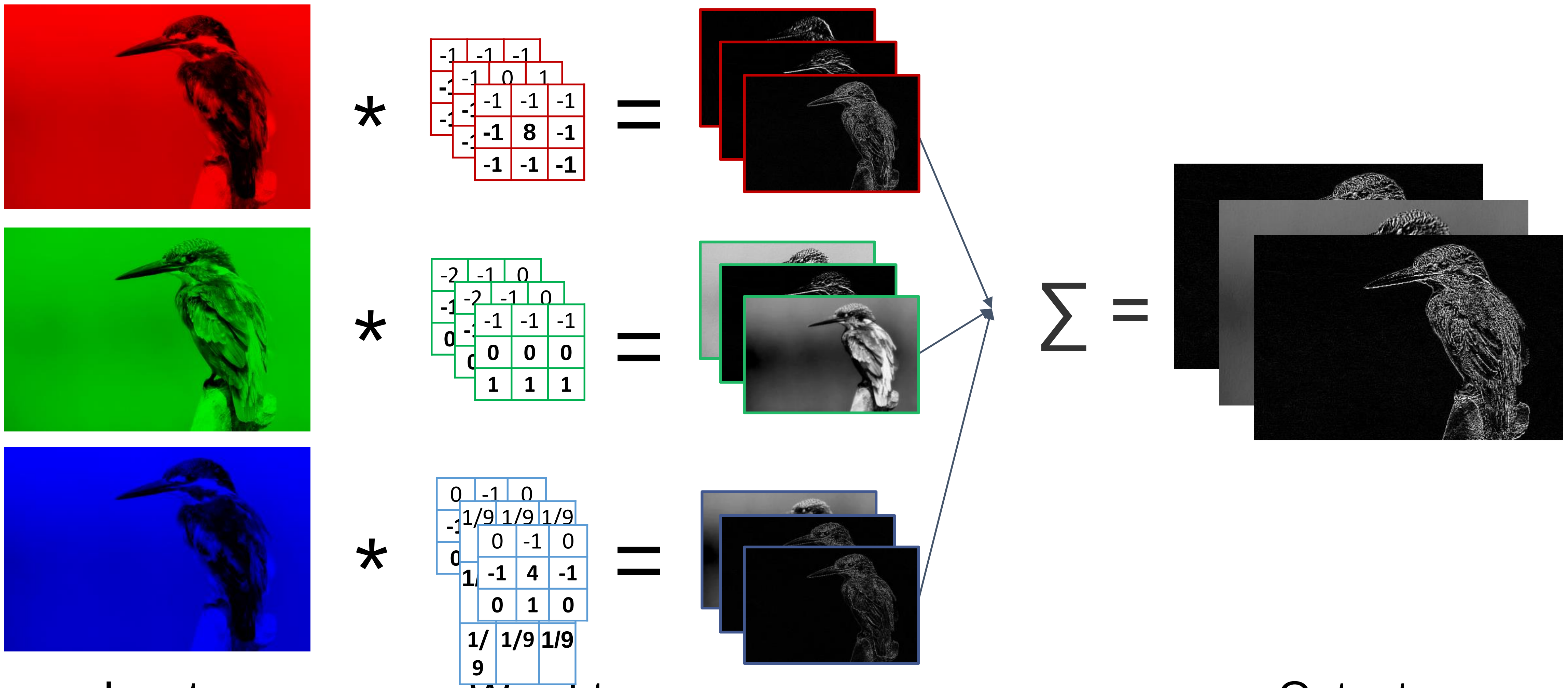


Input:
[3, H, W]

Weight:
[2, 3, 3, 3]

Output:
[2, H, W]

General convolution



Input: $[C_{in}, H, W]$

Weight: $[C_{out}, C_{in}, 3, 3]$

Output: $[C_{out}, H, W]$

Downsampling layers

- Downsampling layers only affect the **image dimensions**

max - pool

Input:
[C_{in},**H, W**]

Output:
[C_{in}, **H/2, W/2**]

CNN dimensions

- In Pytorch we need to specify the number of input and output channels for each convolutional layer (C_{in} , $\mathbf{C_{out}}$).
- It is important to match the number of input channels with the number of output channels of the previous layer

- First layer:
    - Set C_{in} equal to the number of channels of your input images (1 for greyscale, 3 for RGB)
    - Choose $\mathbf{C_{out}}$ based on the number of feature maps you want in the first layer

- For subsequent layers:
    - Set the C_{in} to the $\mathbf{C_{out}}$ of the previous layer
    - Choose $\mathbf{C_{out}}$ based on the number of feature maps you want

Training a CNN

- Passing all our data through the network is computationally very expensive and not memory efficient (datasets can contain millions of images)
- Instead we divide our data in multiple subsets, and pass these subsets after each other through our network. One subset of data is called **a batch**.
- For each batch we perform a forward and backward pass
- Passing all batches of the dataset through the network once is called **an epoch**

The Training Loop

- Define the network
- Load all training data into a PyTorch Tensor

- For number of epochs:
  - For number of batches:
    - Give the data to the network to obtain a prediction
    - Compute the loss value

      Forward Pass

    - Perform backpropagation to obtain the gradients of the loss
    - Update the network parameters (the weights) based on the gradient

      Backward Pass

belfast met

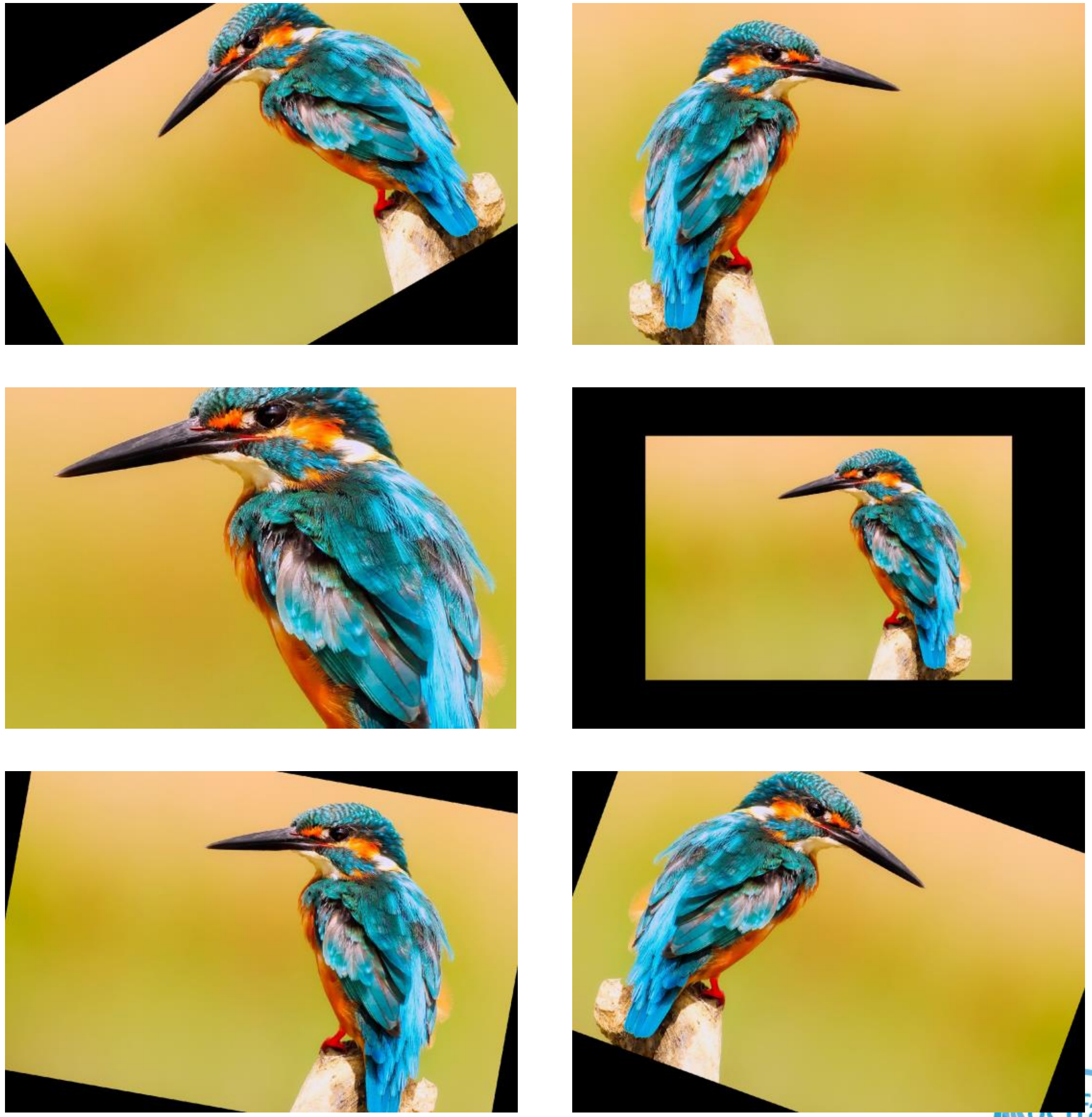
Data augmentation

We need a lot of images to get good predictions

If we have too little images, our networks will “remember” the training images and not generalize well to a validation set **overfitting**)

We can enlarge our dataset by applying **data augmentation** to the training images.

- Scaling, Rotation, Flipping, etc.



Questions?

See you soon

Thank You