Computer Science • Autumn 2024

COMP0188
Deep Representation and Learning

Joshua Spear
joshua.spear.21@ucl.ac.uk

Today

- Coursework 1
- Invariance and equivariance
- CNNs
- Inducing invariance and equivariance through data augmentations

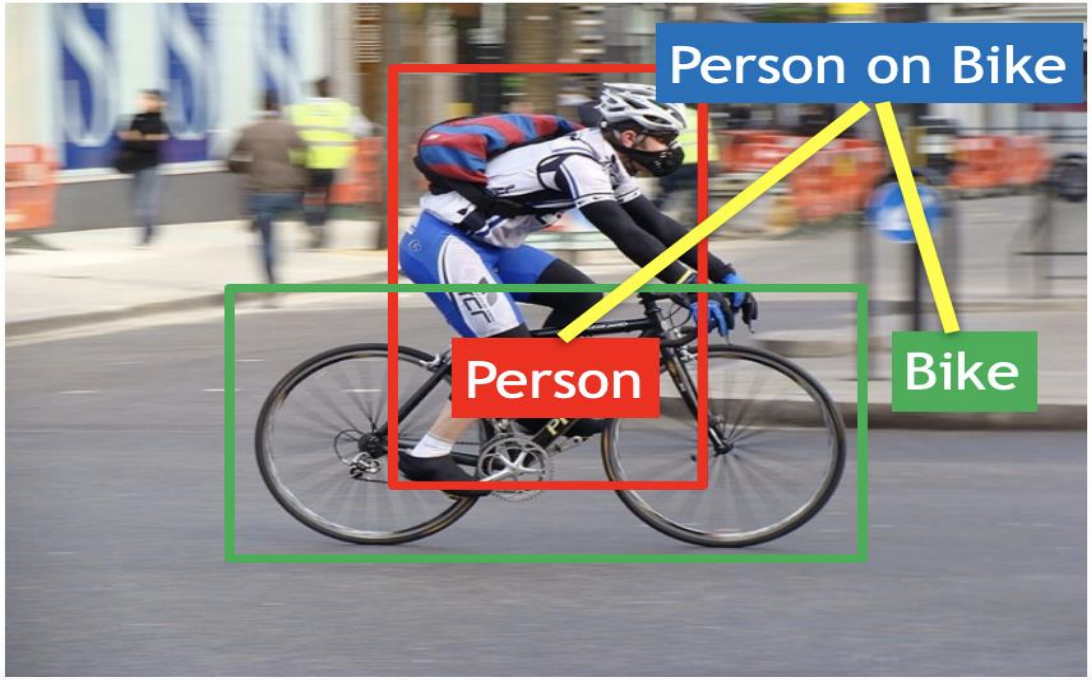
Announcement: Tomorrow’s office is hour is at 9am. Please let me know if you’d like to attend

Coursework 1

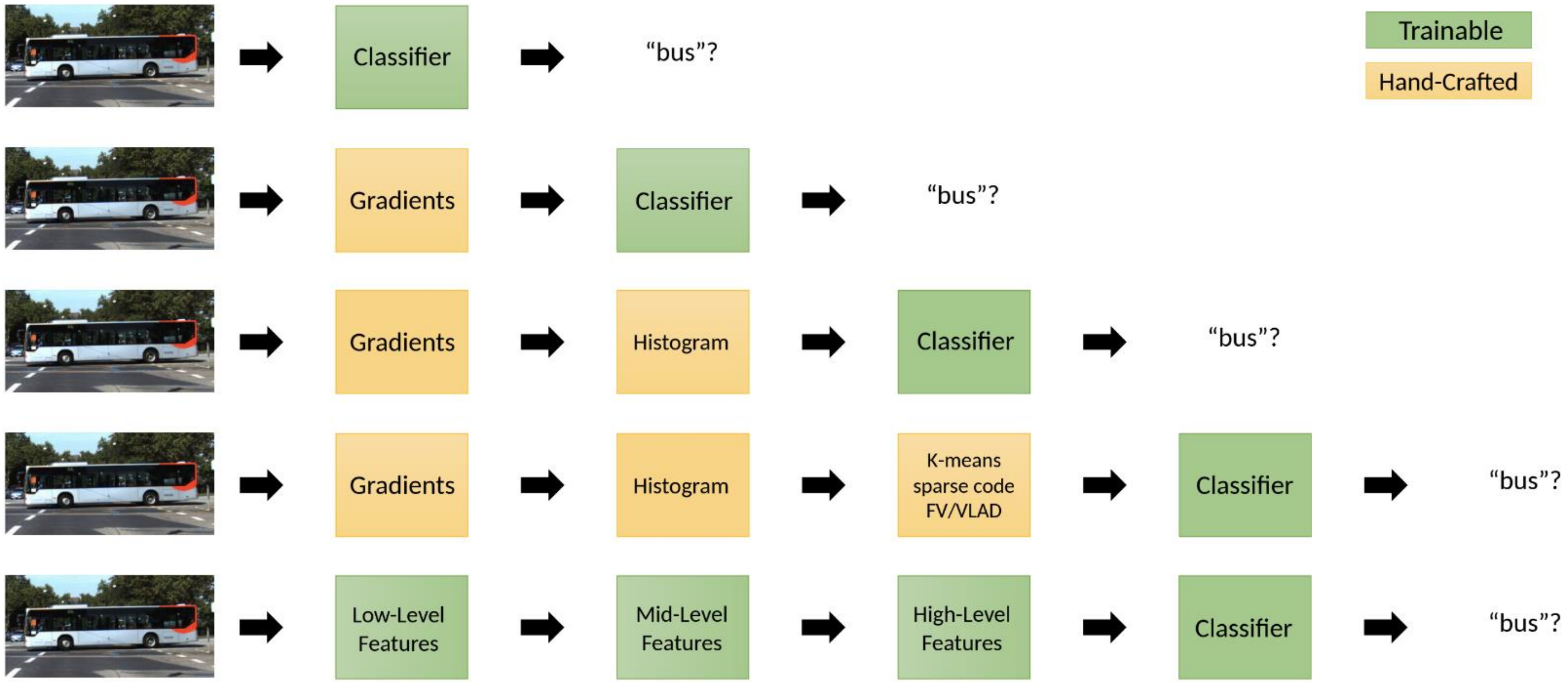
- First coursework (of two) released this Friday 18/10/2024
- Due 07/11/2024 at 16:00
- Accounts for 30% of the total mark for this module
- You will **only** be assessed on material covered in the **first 3 weeks** of lectures (i.e., including this lecture)
- Please refer to the UCL guidance on plagiarism and use of LLMs – these apply to this work!
- If you have any queries (at all!) please speak to me or a member of staff that you feel comfortable with

Invariance and equivariance in computer vision

Computer vision

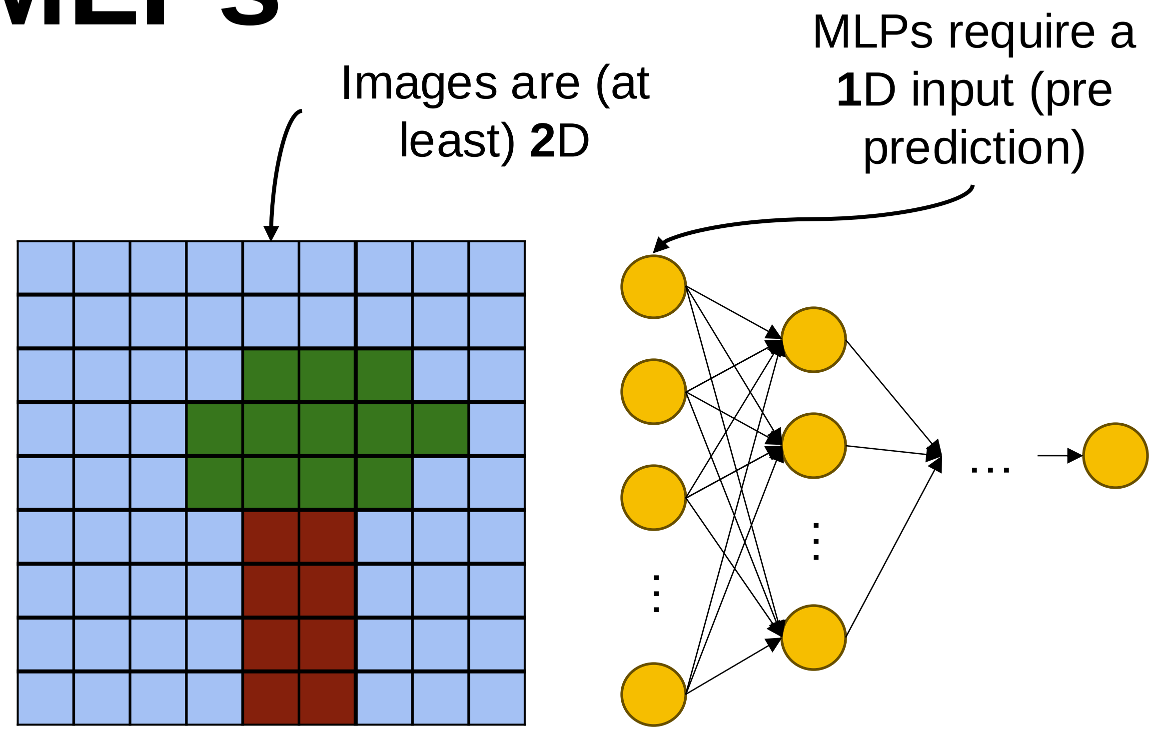


Computer vision: Representation learning

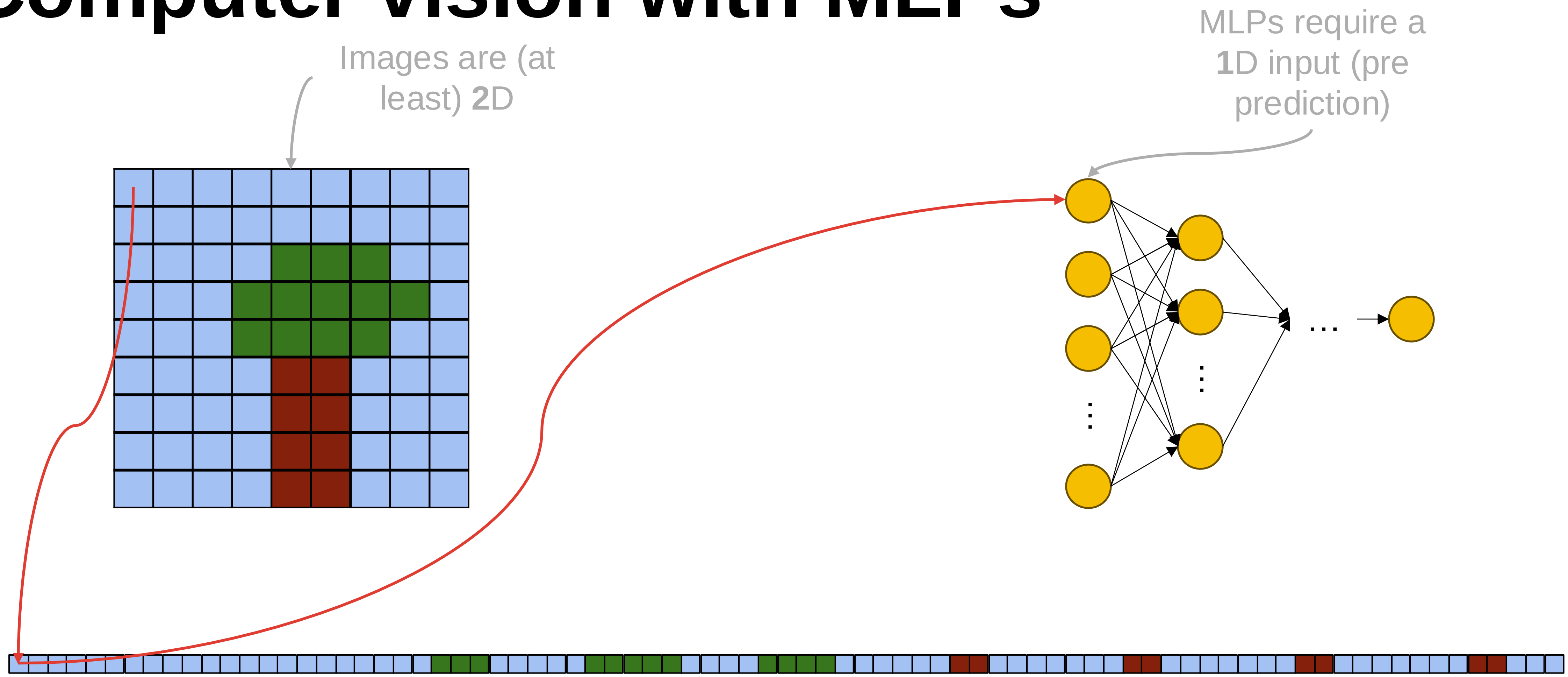


Computer vision with MLPs

- For a given prediction, vanilla MLPs assume a d-dimensional input i.e., d features
  - In the first lecture, we predicted the probability of you passing given 2 features
- Images are a $d \times d$-dimensional input

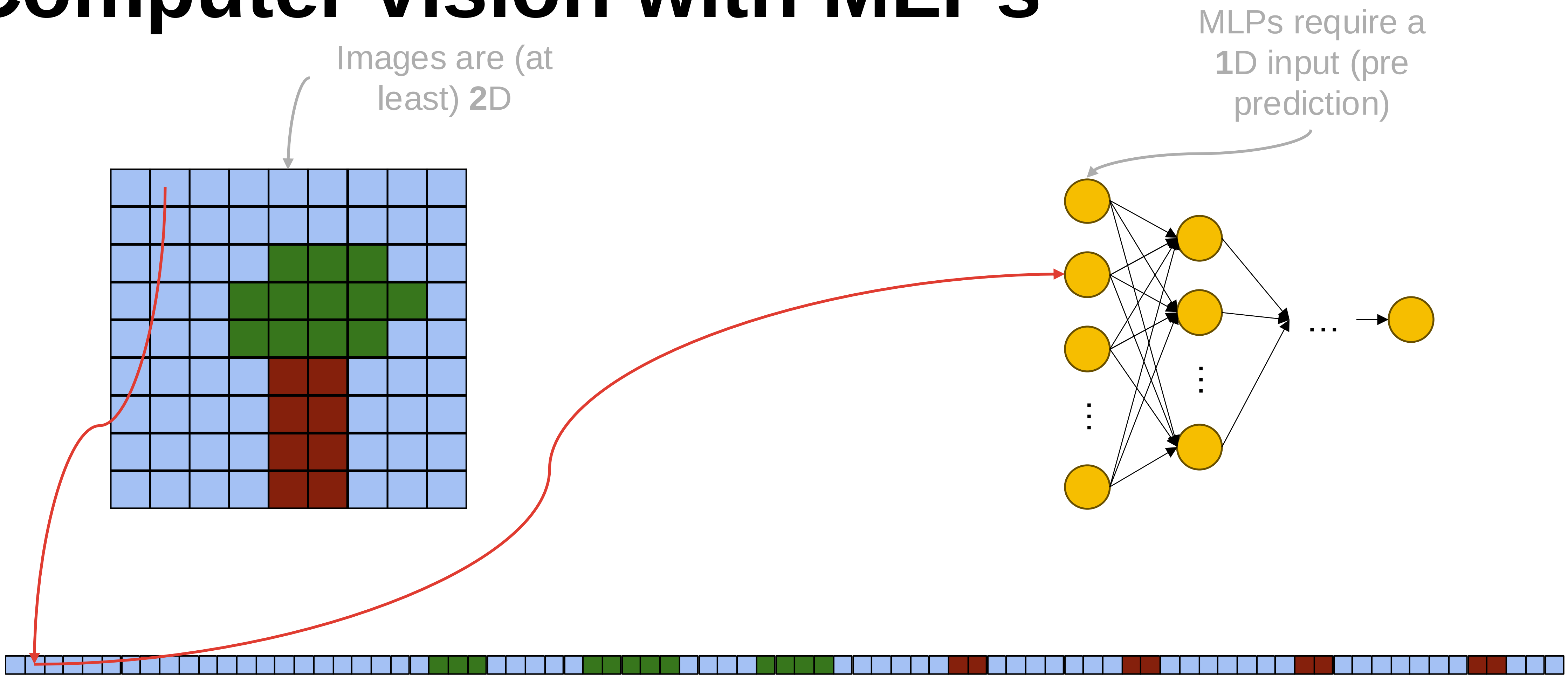


Computer vision with MLPs



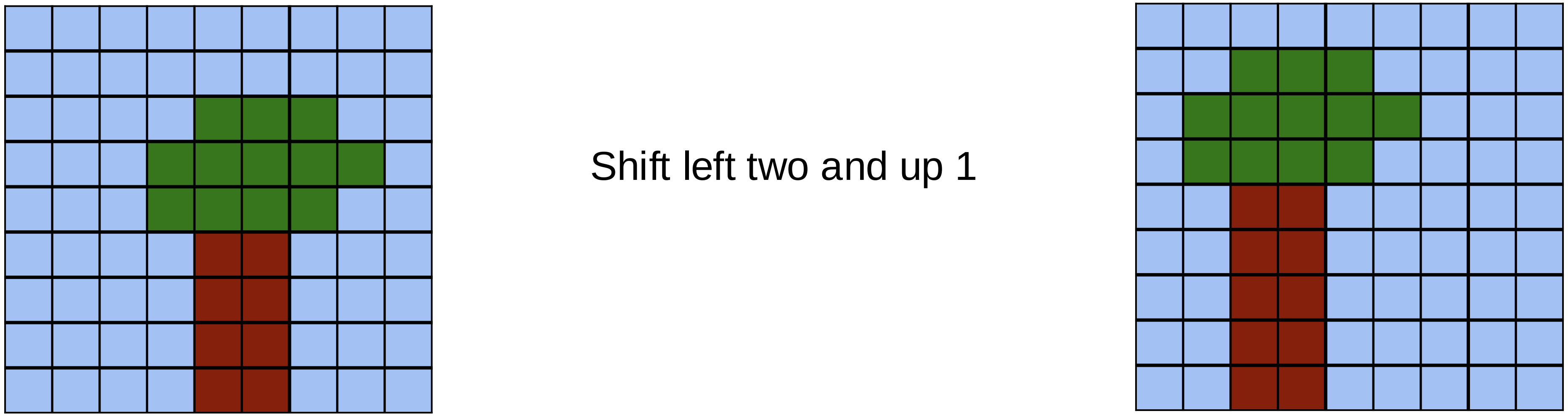
Flatten?

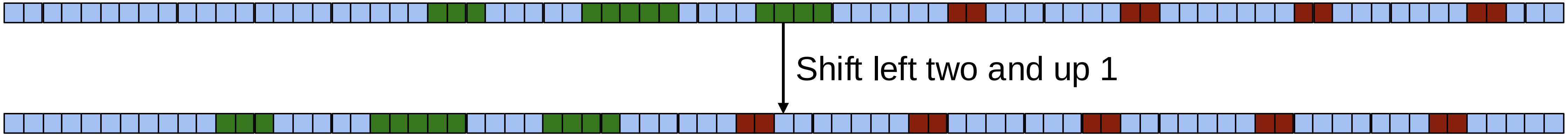
Computer vision with MLPs



Flatten?

Computer vision with MLPs



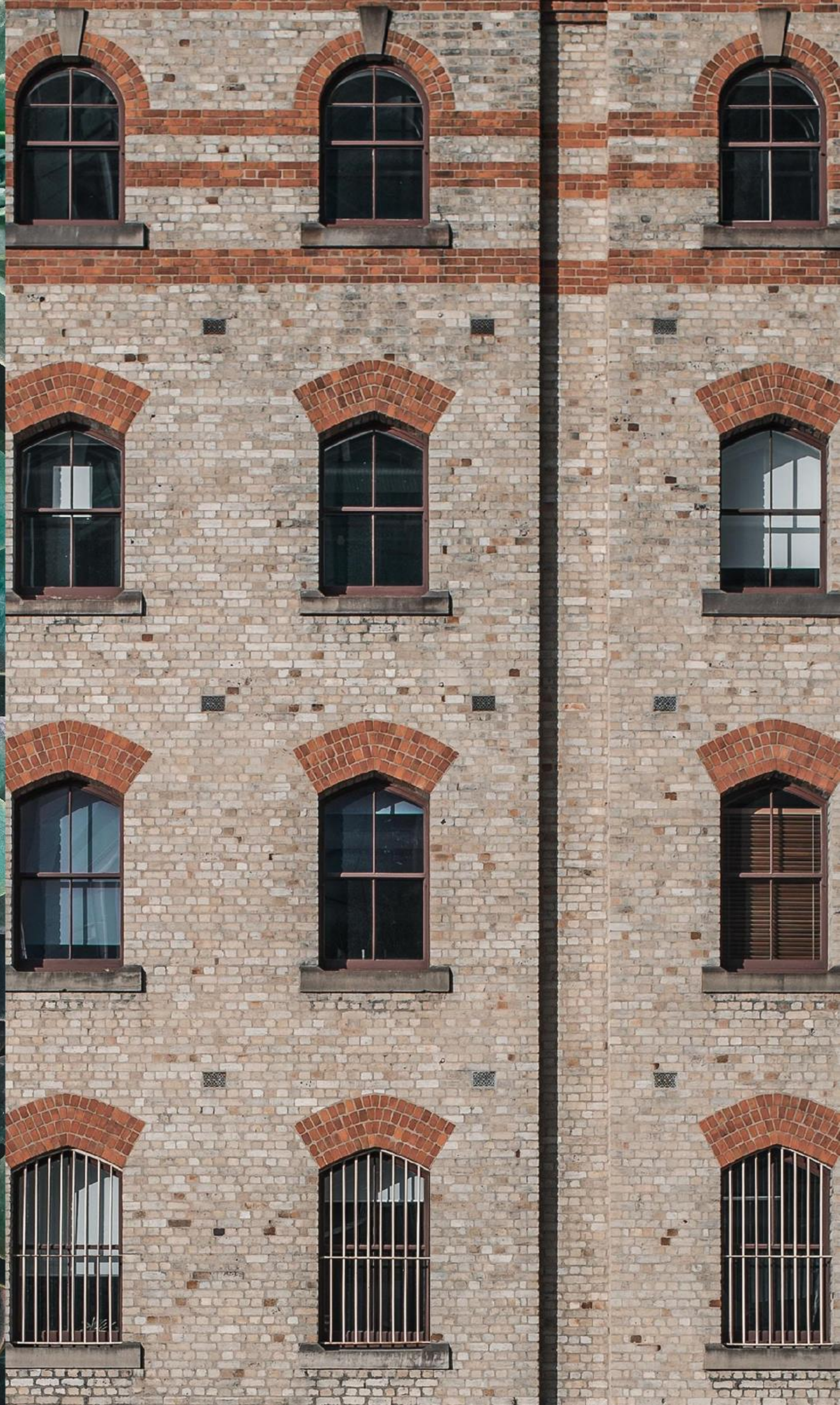


Problem: The model would need to see **every object** in **every position**

Inductive biases

- Elements of model development
  - Function $f: \mathcal{X} \rightarrow \mathcal{Y}$, where f $\in \mathcal{F}$
  - A (data)set of samples which have been taken from our input and output sets $d = \{(x, y)_i : x \in \mathcal{X}, y \in \mathcal{Y}, i \in 1, \ldots, n, n \in \mathbb{N}\}$
    - $a: \mathcal{X} \times \mathcal{Y} \times l \rightarrow \mathcal{F}$
    - $l: \hat{\mathcal{Y}} \times \mathcal{Y} \rightarrow \mathbb{R}$
- Each design choice imposes an **inductive bias** on the learning process
- Informally, inductive biases **impose a preference over $\mathcal{F}$/directly alter the shape of $\mathcal{F}$**
- When the inductive bias **prefers a subset of $\mathcal{F}$** which contains a function $\hat{f}$ that **obtains minimal generalisation error**, the inductive bias is **useful** and **improves learning**





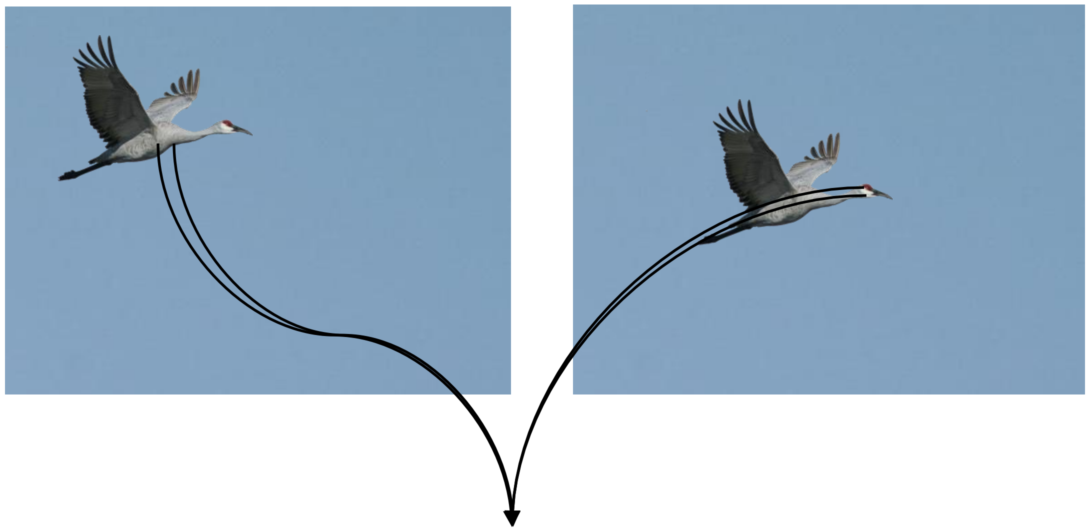
Utilising topological structure: Locality and translational equivariance

Locality: Nearby pixels are strongly correlated

Translational equivariance: Similar patterns can occur anywhere in the image

Locality

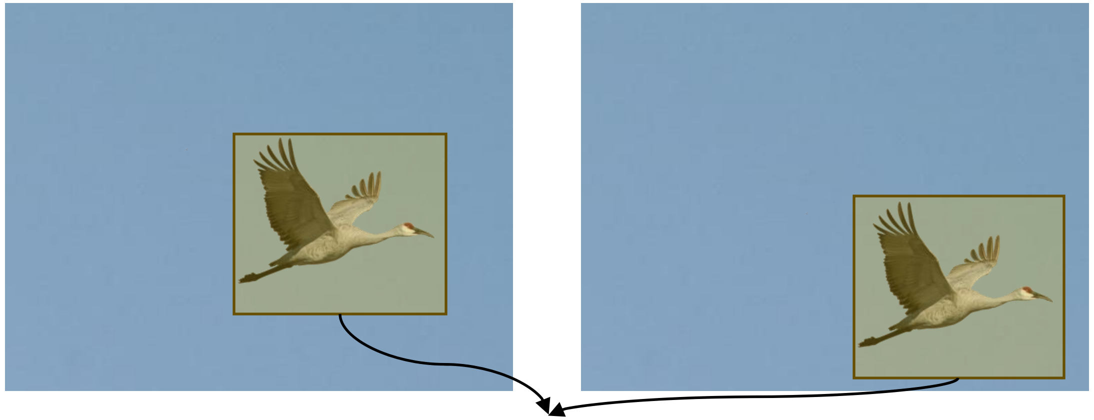
- A condition being local loosely means that its effect is restricted to objects in a neighbourhood
  - The importance of locality in image recognition is loosely that: we care about objects **within the image** not the entire image
- Formally defined in: Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges, 2021



Locality: Nearby pixels are strongly correlated

Translational equivariance

- **Invariance**:the output of function f is **unaffected** by a “transformation” of the input
  - $f(\rho x) = f(x)$
- **Equivariance**: The input and output are **affected in the same way** by a transformation on the input
  - $f(\rho x) = \rho f(x)$
- References:
  - Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gauges, 2021
  - Understanding deep learning, 2024

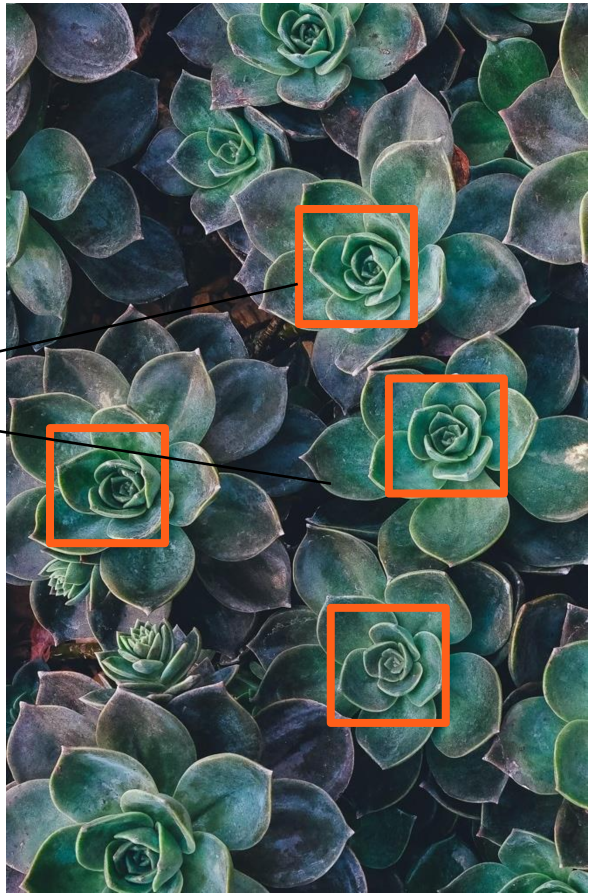


(Local) translational invariance/equivariance: Similar patterns can occur anywhere in the image

- Identifying whether an image contains a bird: require translational **invariance**
- Identify bounding box of bird: require translational **equivariance**

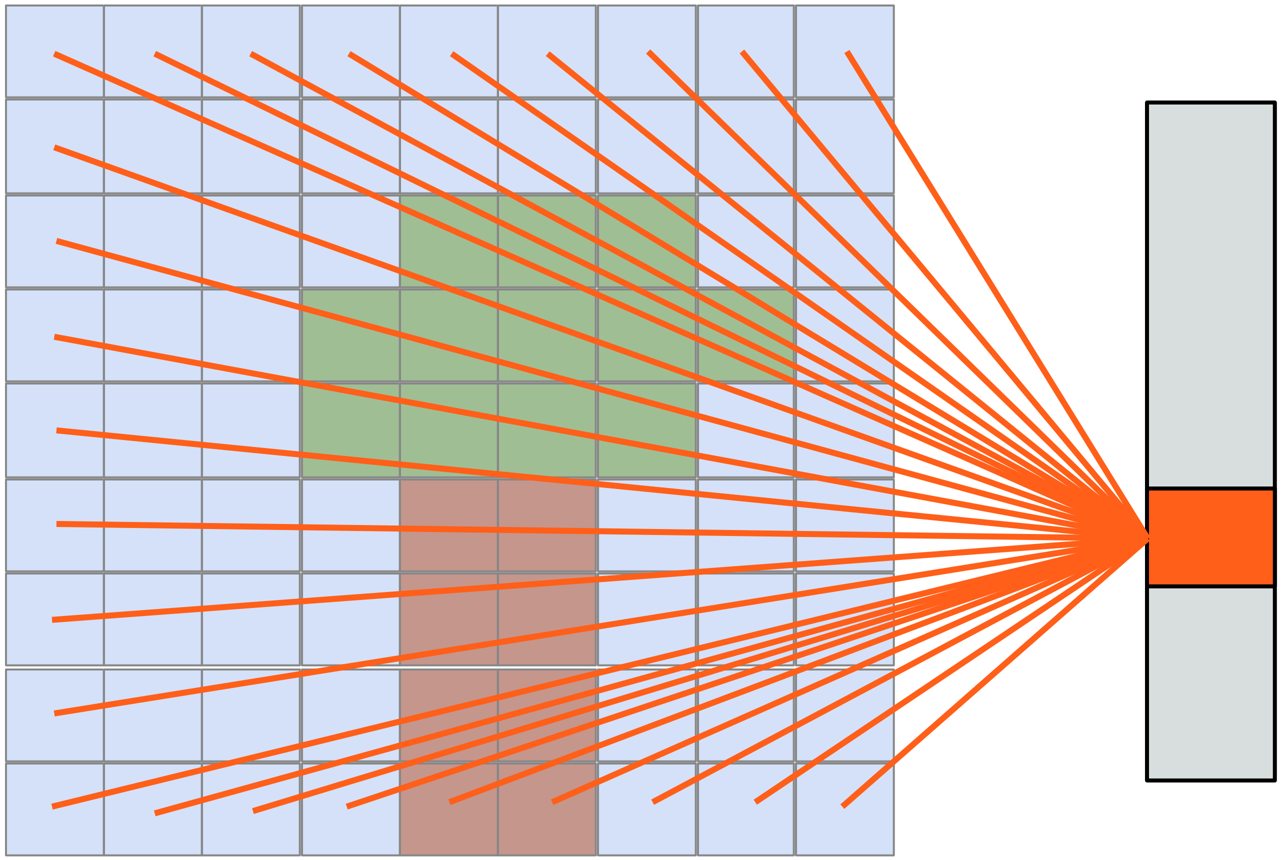
Implementing (local) translational equivariance

- **Weight sharing**
  - Neural network weights are **activated** in the same way **if the same input is provided**
- Applying the **same weights** to these blocks of input will result in the **same values for the activations**



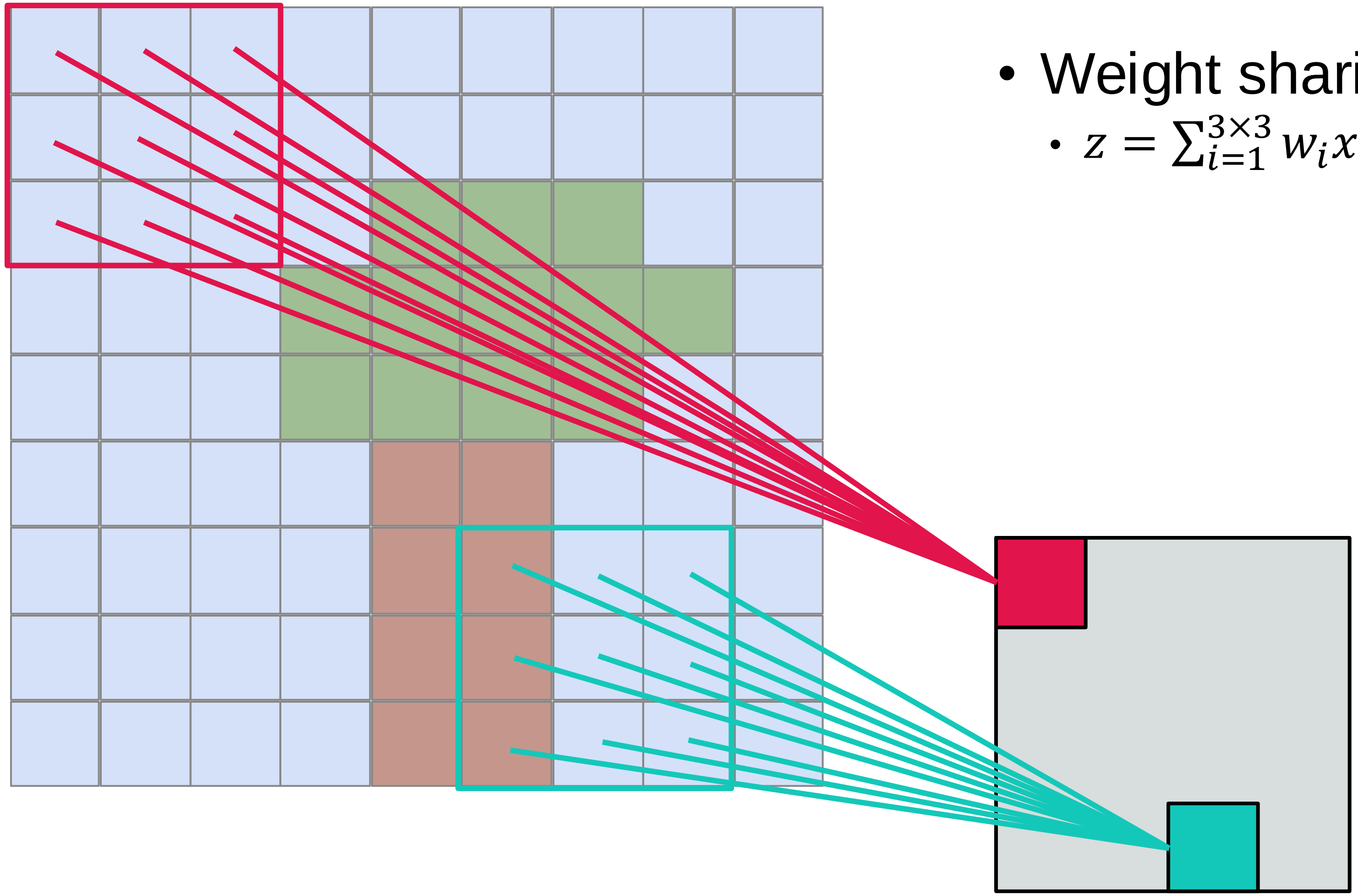
CNNs

From fully connected to locally connected: weight sharing



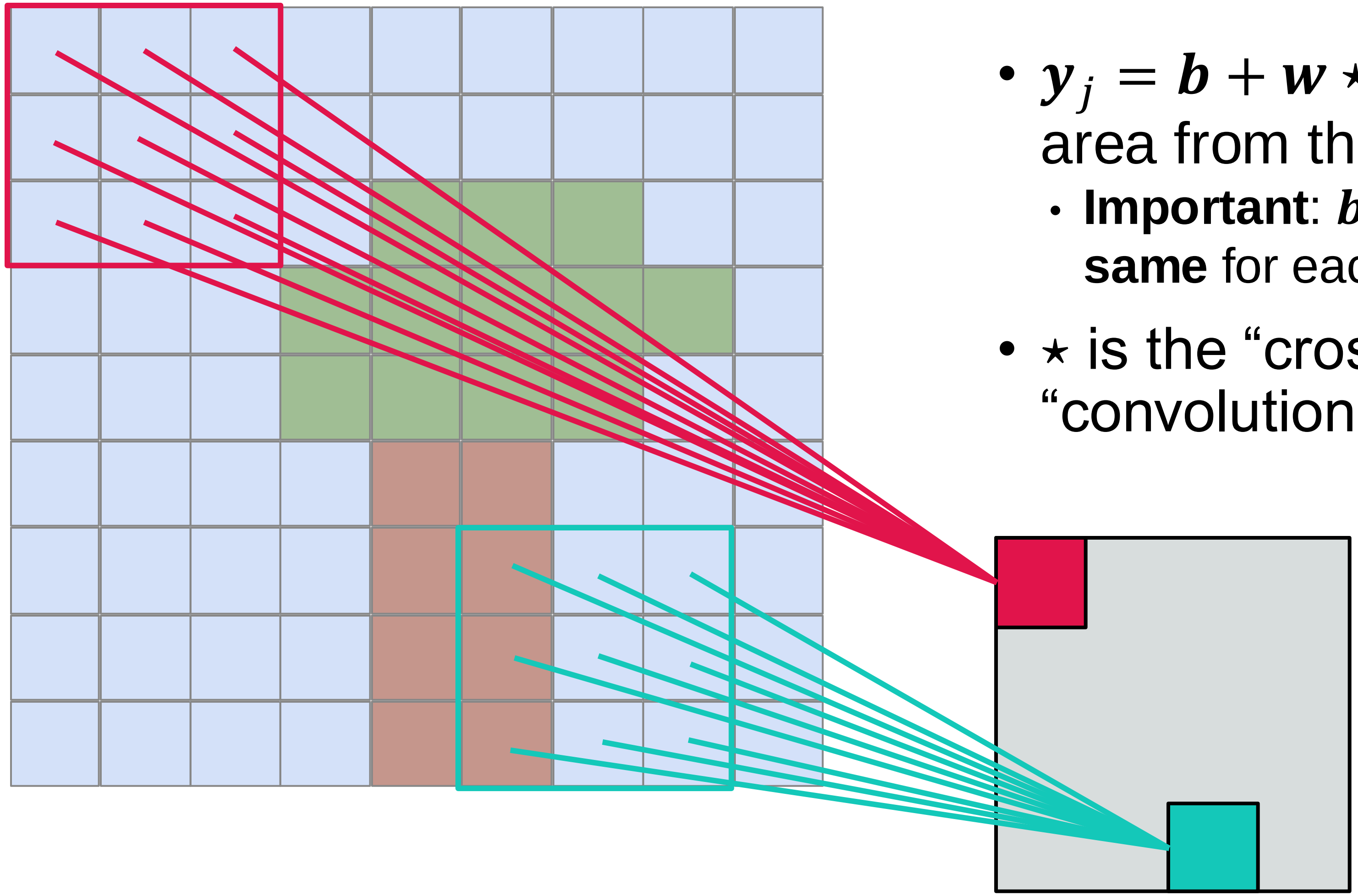
- Fully connected unit
  - $z = \sum_{i=1}^{|d|\times|d|} w_i x_i + b$

From fully connected to locally connected: weight sharing



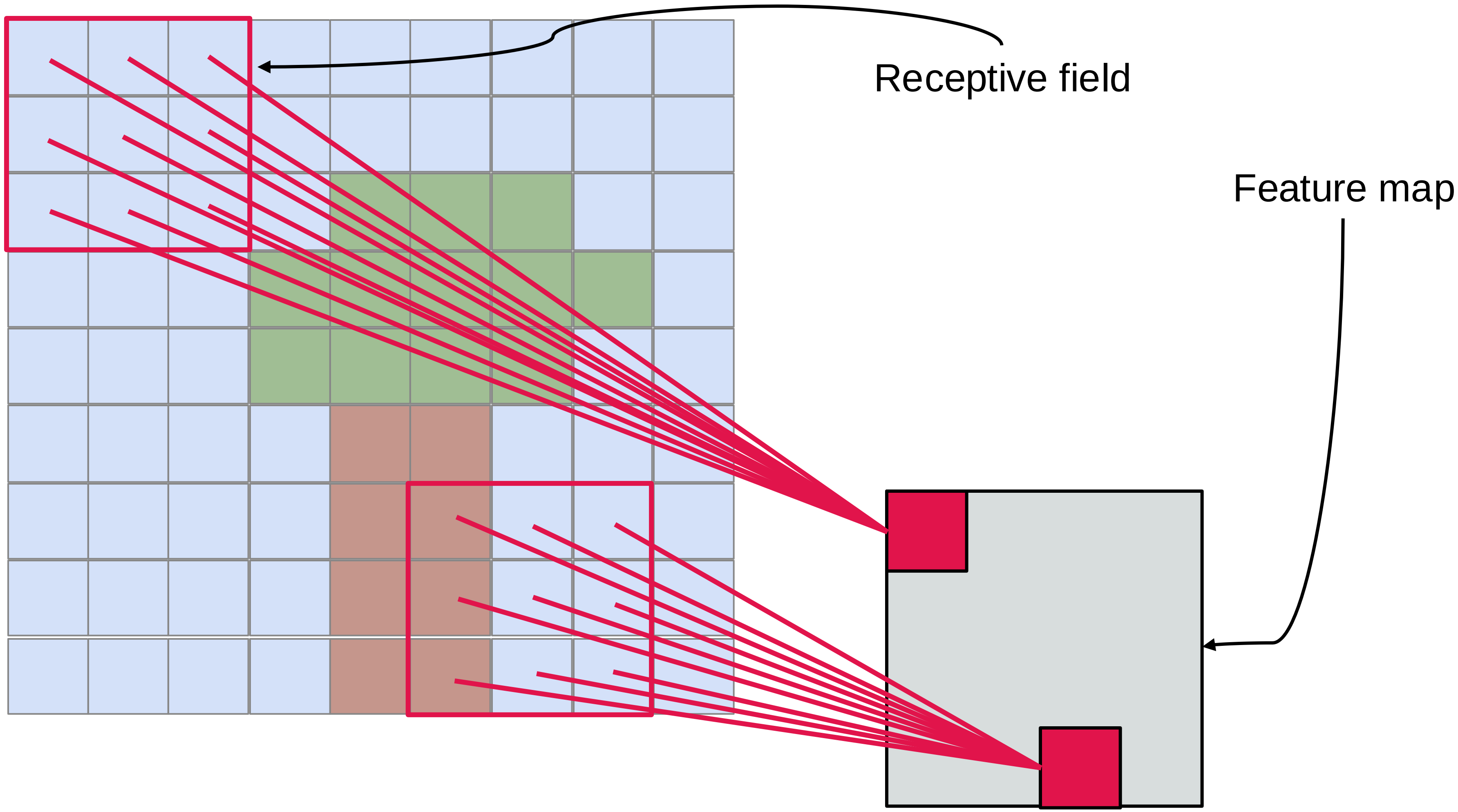
- Weight sharing over 3×3 filter
  - $z = \sum_{i=1}^{3\times3} w_i x_i + b$

Convolution

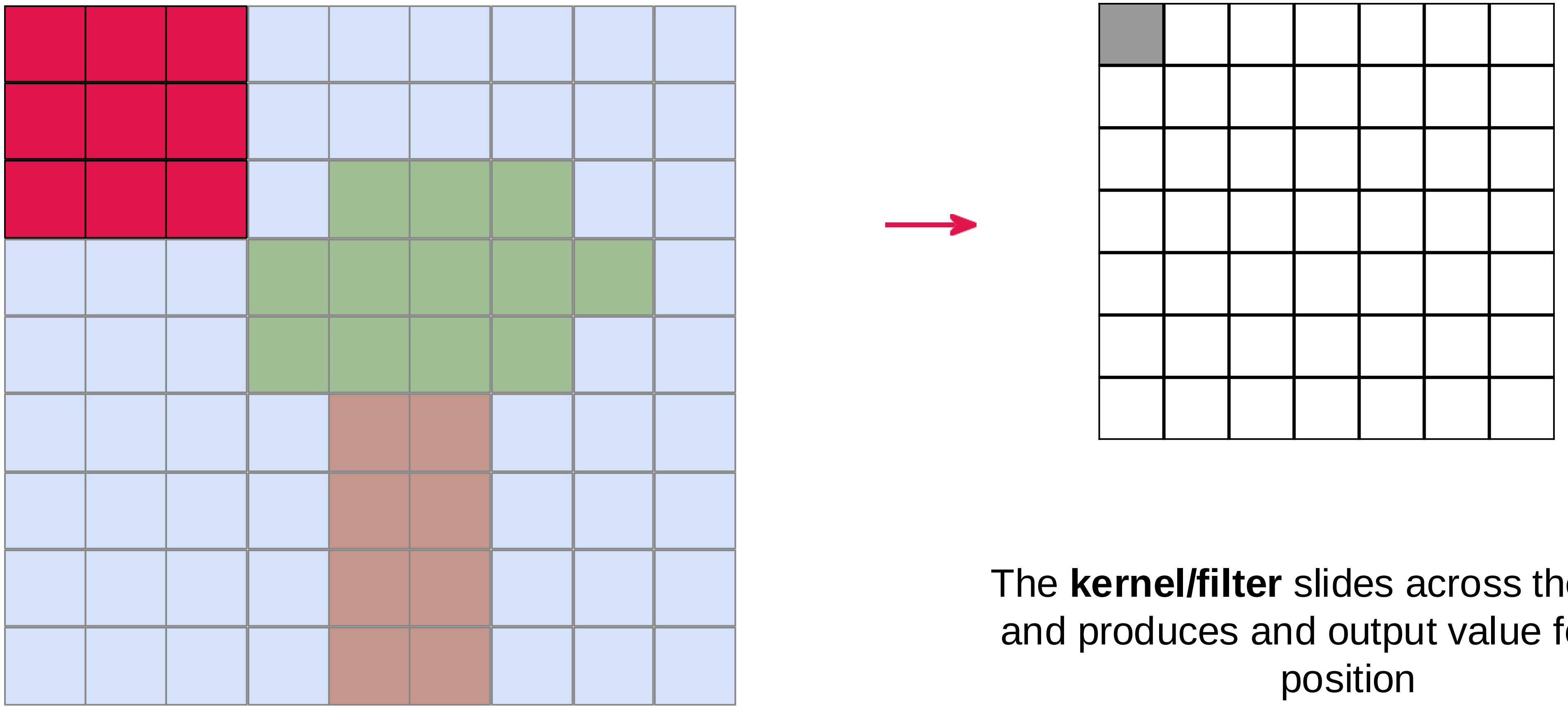


- $\boldsymbol{y}_j = \boldsymbol{b} + \boldsymbol{w} \star \boldsymbol{x}_j$ where $\boldsymbol{x}_j$ is a local 3×3 area from the input image
  - **Important**: $\boldsymbol{b}$ and $\boldsymbol{w}$ (the weights) are the **same** for each 3×3 input area
- $\star$ is the "cross-correlation" but called "convolution" in ML
- Extended exercise:
  - Global cross-correlation is defined as $(f \star g)(\tau) = \int_{-\infty}^{\infty} f(t) g(t + \tau) dt$
  - Obtain the discrete local version from the previous slide

Convolution

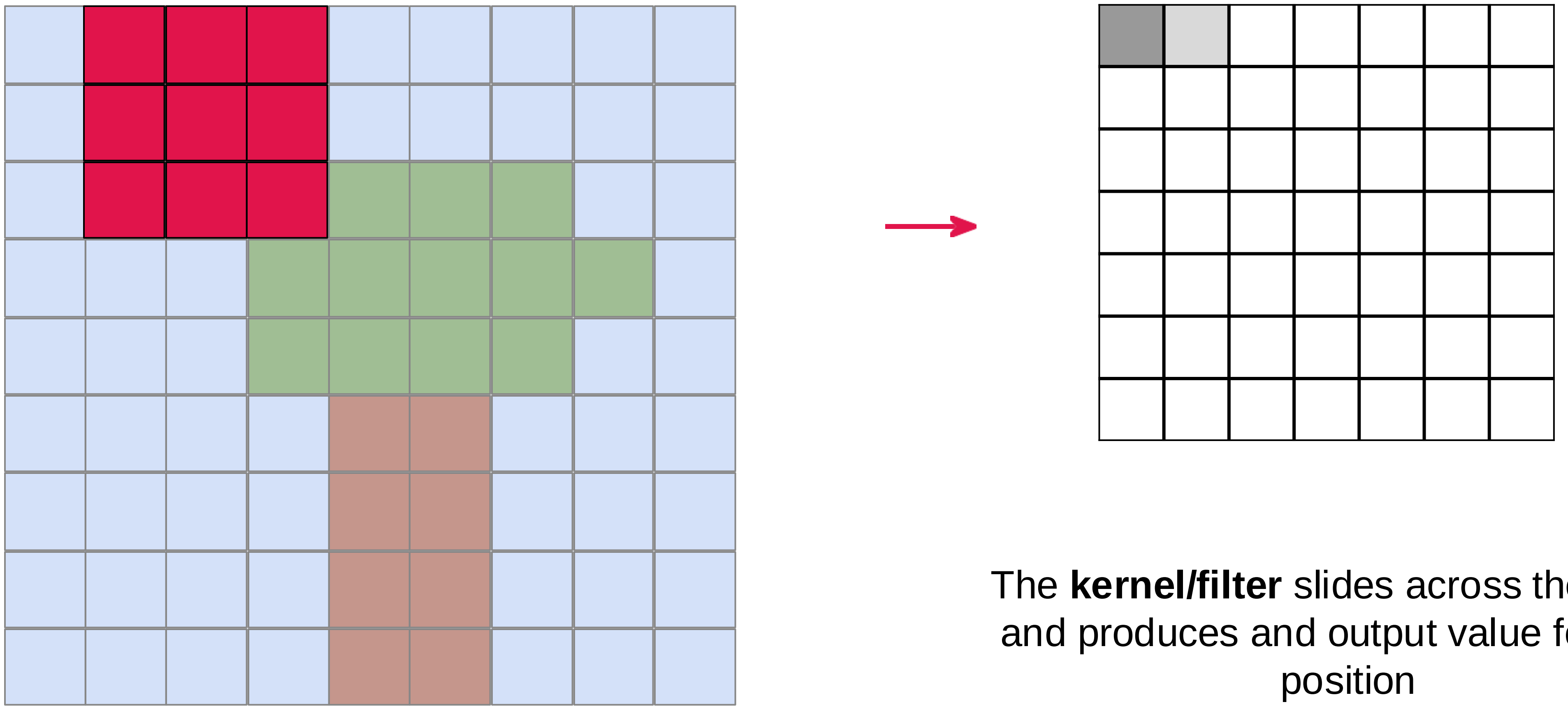


Implementing the convolution



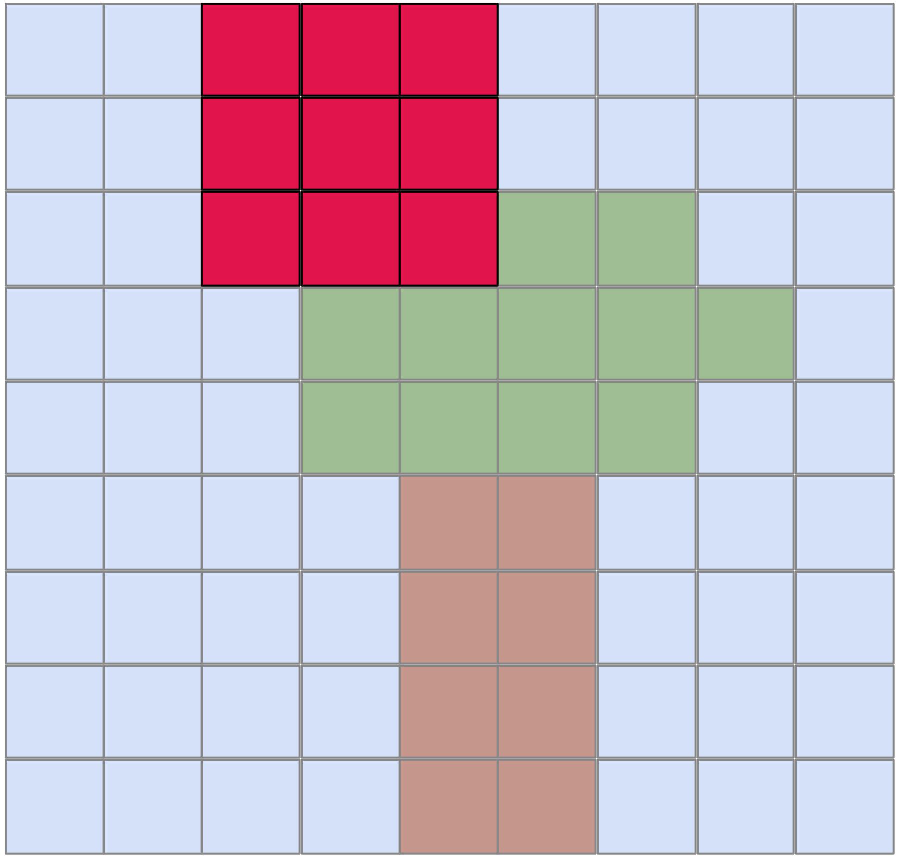
The **kernel/filter** slides across the image and produces and output value for each position

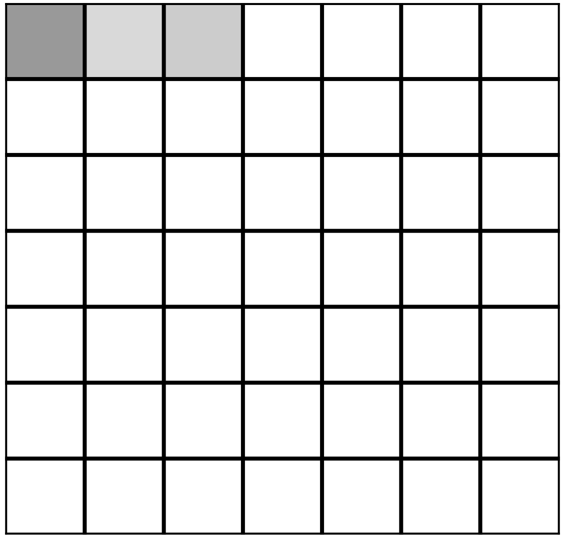
Implementing the convolution



The **kernel/filter** slides across the image and produces and output value for each position

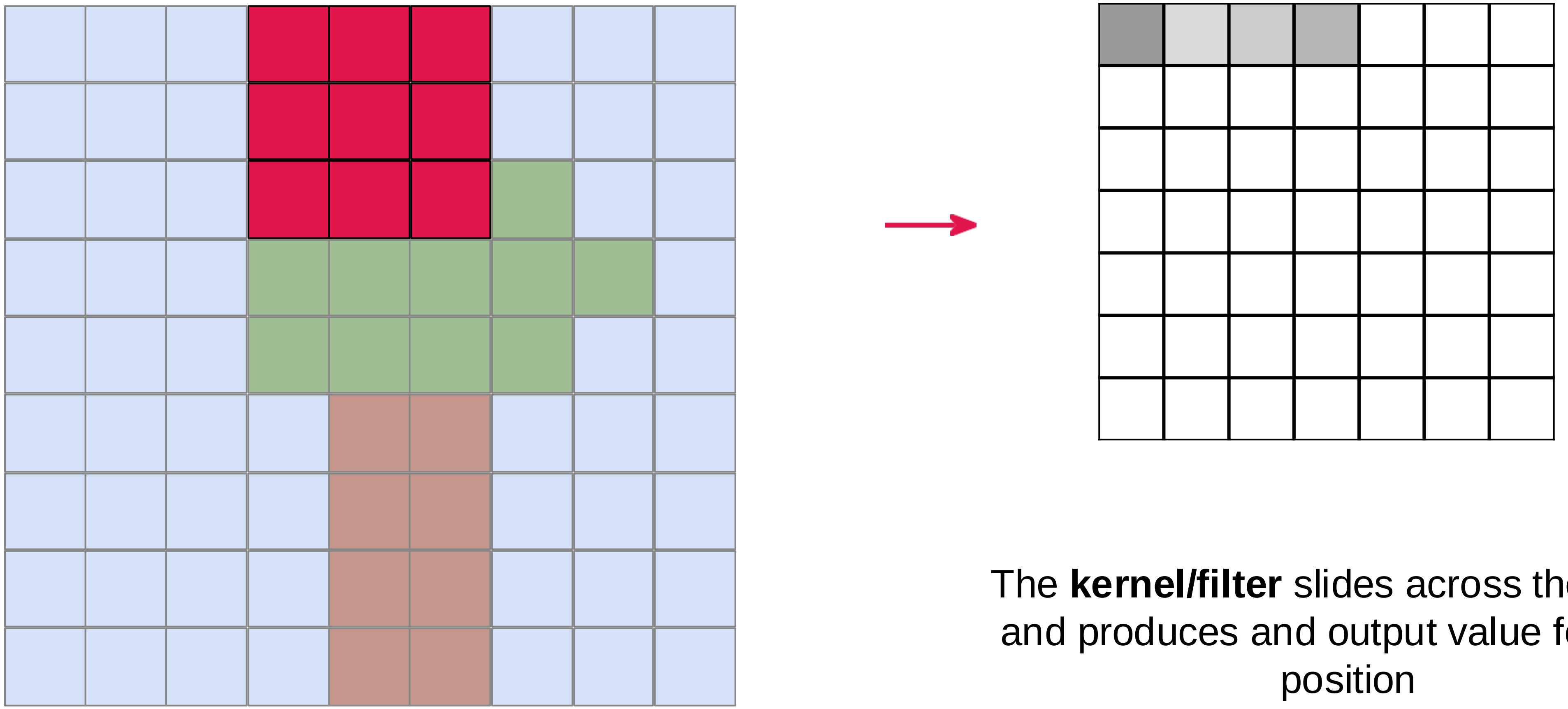
Implementing the convolution





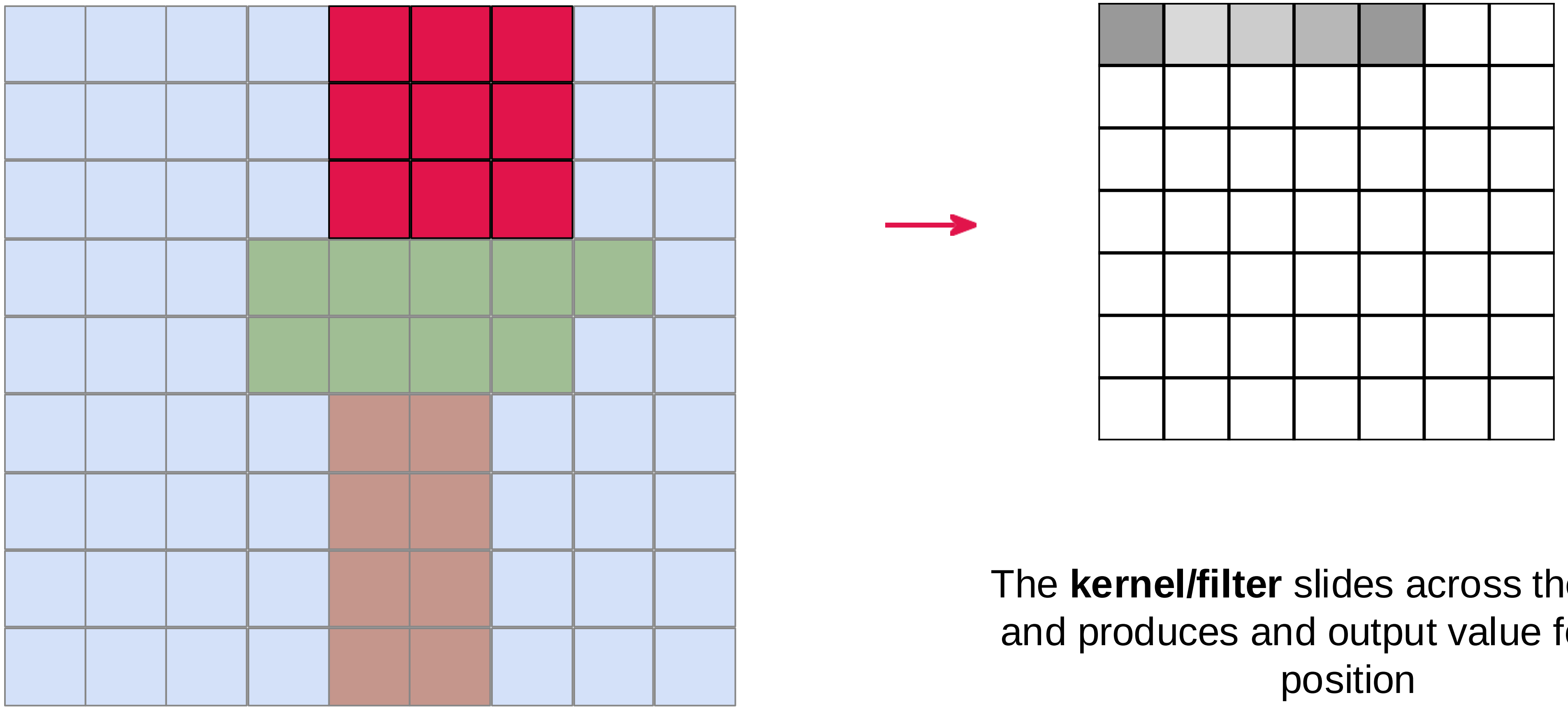
The **kernel/filter** slides across the image and produces and output value for each position

Implementing the convolution



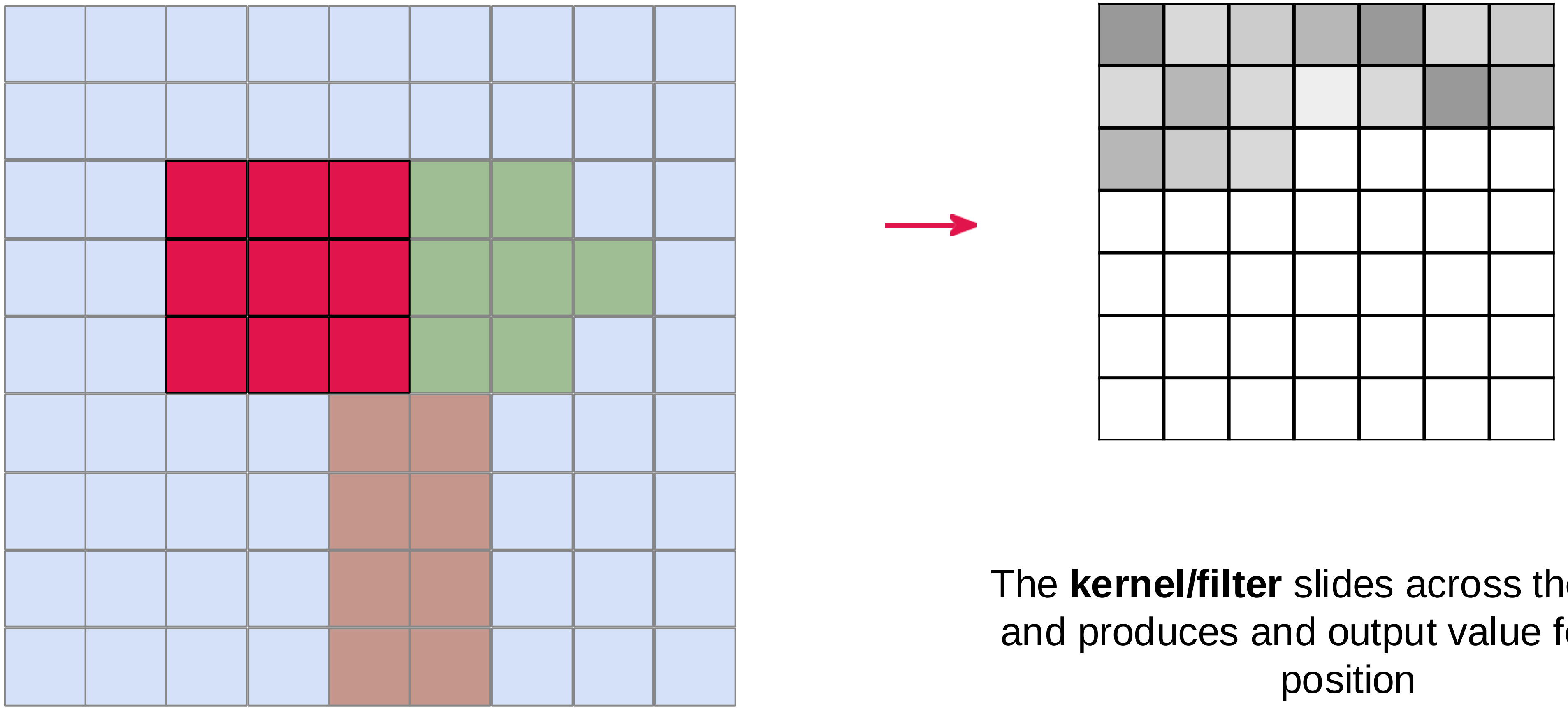
The **kernel/filter** slides across the image and produces and output value for each position

Implementing the convolution



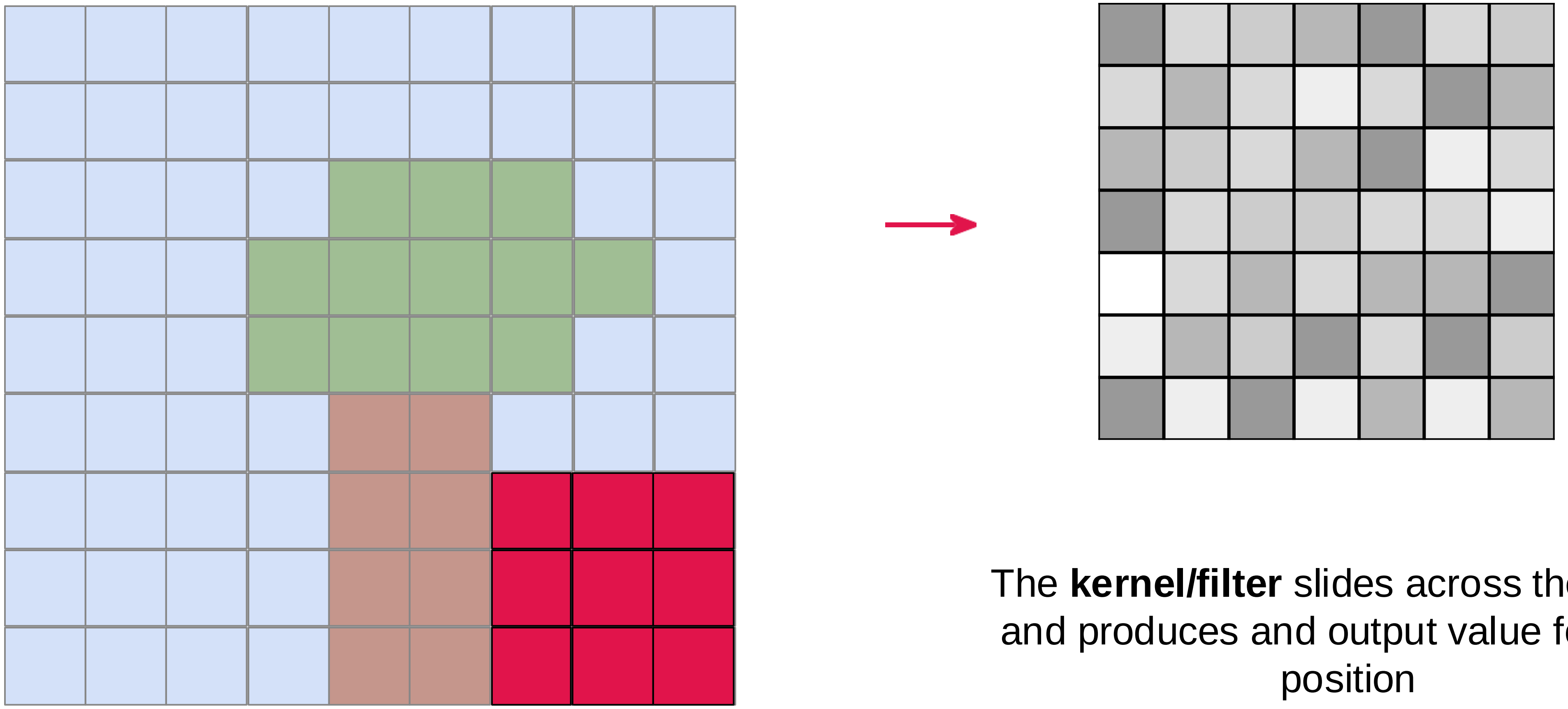
The **kernel/filter** slides across the image and produces and output value for each position

Implementing the convolution



The **kernel/filter** slides across the image and produces and output value for each position

Implementing the convolution

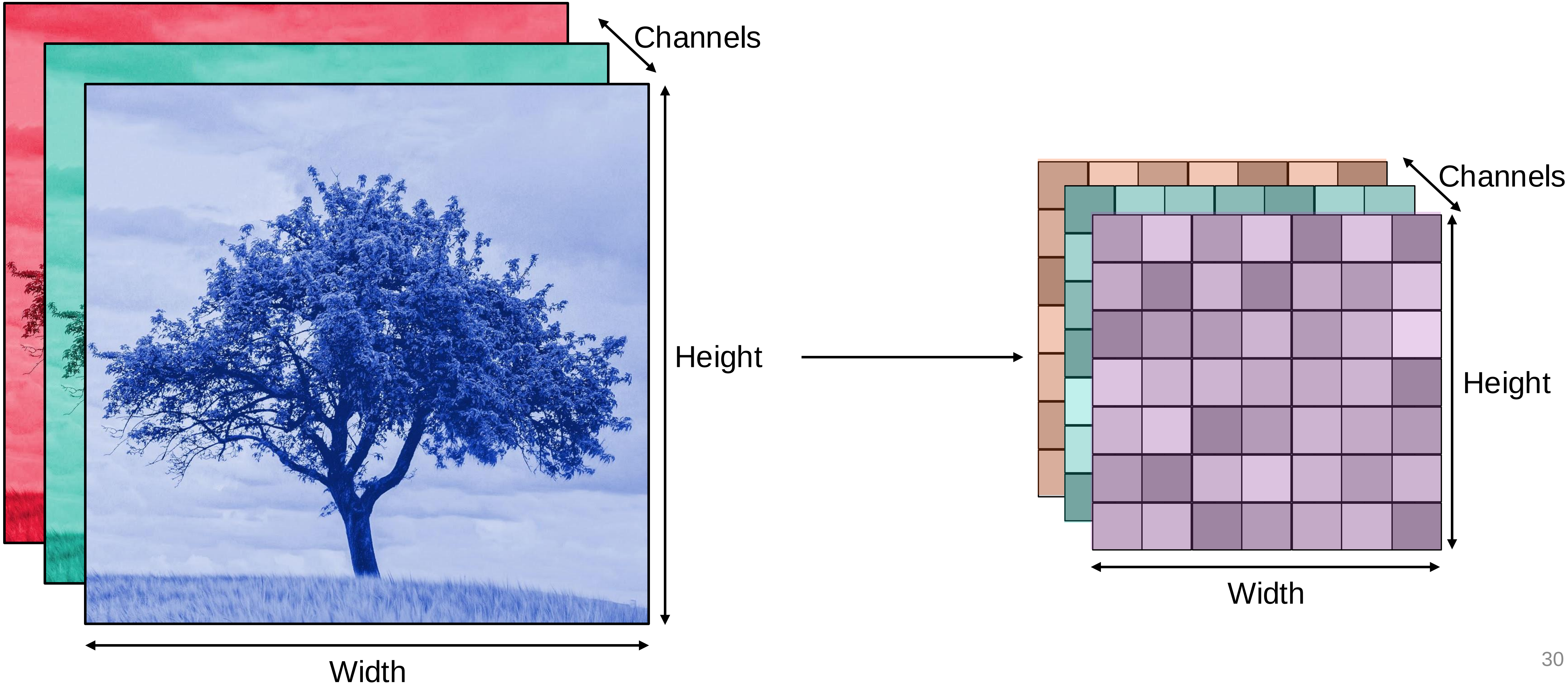


The **kernel/filter** slides across the image and produces and output value for each position

Pictures as tensors

RGB

Pictures as tensors



CNN in pytorch

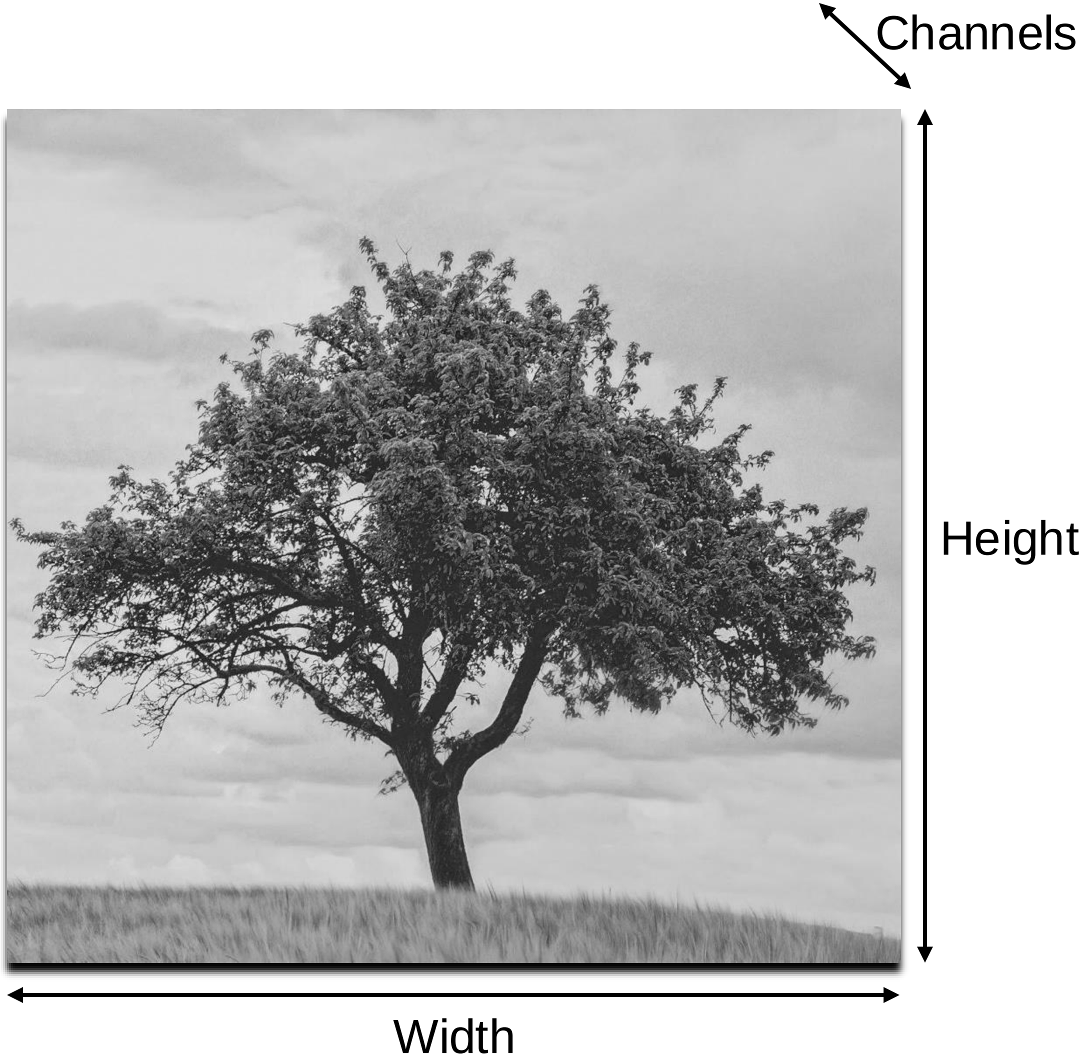
```python
class CNN(nn.Module):

    def __init__(self):
        super().__init__()
        module_list = nn.ModuleList()
        module_list.append(nn.Conv2d(
            in_channels=1,
            out_channels=1,
            kernel_size=(3, 3)
            ))

        module_list.append(nn.Flatten())

        module_list.append(
            nn.Linear(900,1)
        )
        module_list.append(
            nn.Sigmoid()
        )
        self.module_list = module_list

    def forward(self,x:torch.Tensor)->torch.Tensor:
        _x = x
        for l in self.module_list:
            _x = l(_x)
        return _x
```

```python
model = CNN()
summary(model, (1,1,32,32))
```

✓ 0.0s

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN                                      [1, 1]                    --
├─ModuleList: 1-1                        --                        --
│    └─Conv2d: 2-1                       [1, 1, 30, 30]            10
│    └─Flatten: 2-2                      [1, 900]                  --
│    └─Linear: 2-3                       [1, 1]                    901
│    └─Sigmoid: 2-4                      [1, 1]                    --
==========================================================================================
Total params: 911
Trainable params: 911
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.01
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.00
Estimated Total Size (MB): 0.02
==========================================================================================
```

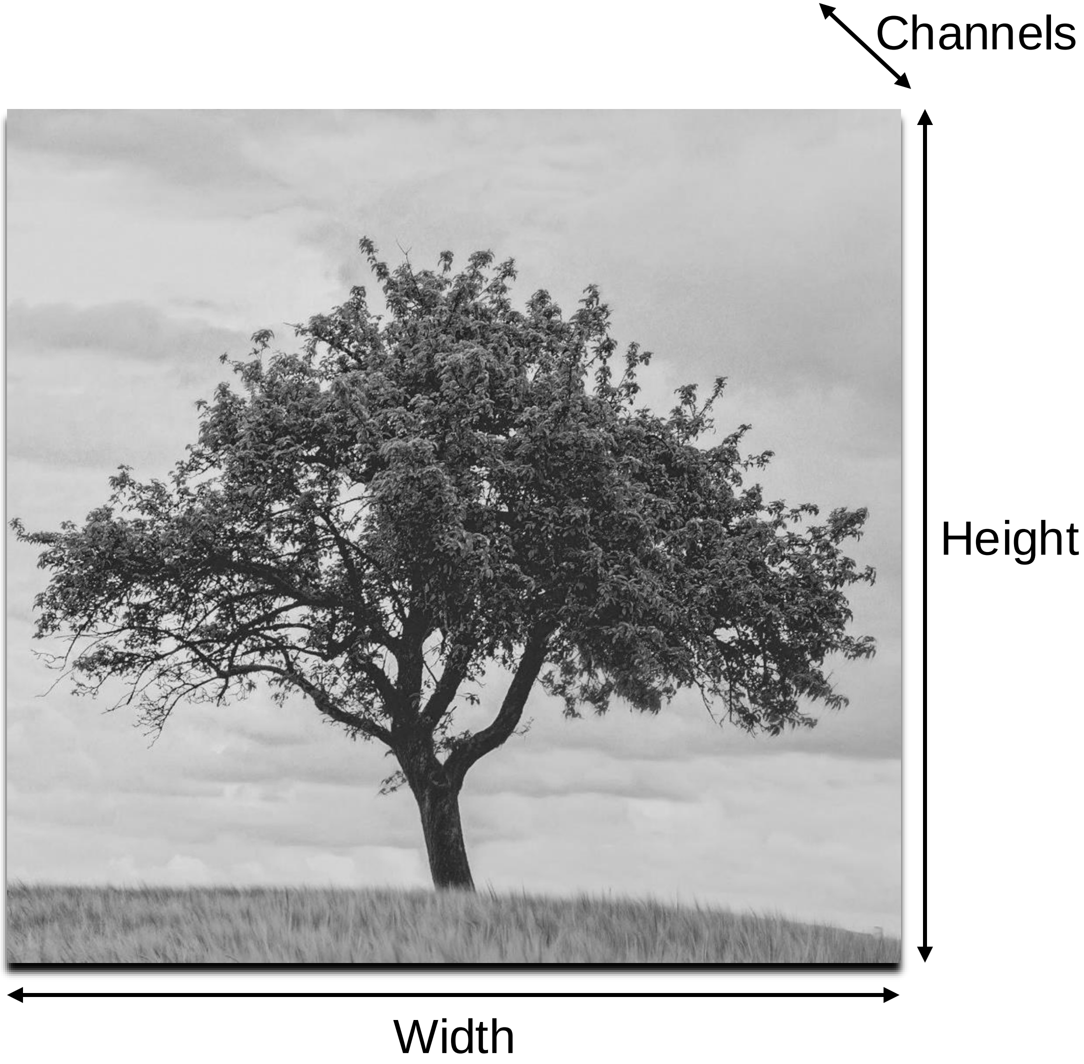
CNN in pytorch



Height Width

```
model = CNN()
summary(model, (1,1,32,32))
✓ 0.0s

==========================================================================
Layer (type:depth-idx)                     Output Shape
==========================================================================
CNN                                        [1, 1]
├─ModuleList: 1-1                          --
│    └─Conv2d: 2-1                         [1, 1, 30, 30]
│    └─Flatten: 2-2                        [1, 900]
│    └─Linear: 2-3                         [1, 1]
│    └─Sigmoid: 2-4                        [1, 1]
==========================================================================
Total params: 911
Trainable params: 911
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.01
==========================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.00
Estimated Total Size (MB): 0.02
==========================================================================
```

CNN in pytorch



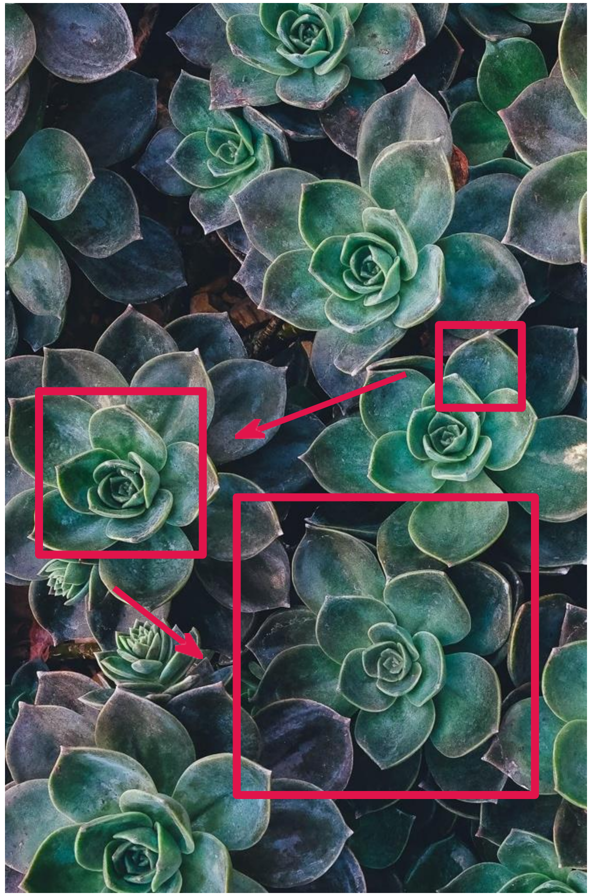
Batch size Channels Height Width

```
model = CNN()
summary(model, (1,1,32,32))
✓ 0.0s

==========================================================================
Layer (type:depth-idx)                    Output Shape
==========================================================================
CNN                                       [1, 1]
├─ModuleList: 1-1                         --
│    └─Conv2d: 2-1                        [1, 1, 30, 30]
│    └─Flatten: 2-2                       [1, 900]
│    └─Linear: 2-3                        [1, 1]
│    └─Sigmoid: 2-4                       [1, 1]
==========================================================================
Total params: 911
Trainable params: 911
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.01
==========================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.00
Estimated Total Size (MB): 0.02
==========================================================================
```

Hierarchical local patterns

- **Hierarchical patterns**
  - Local low-level features are composed into larger, more abstract features

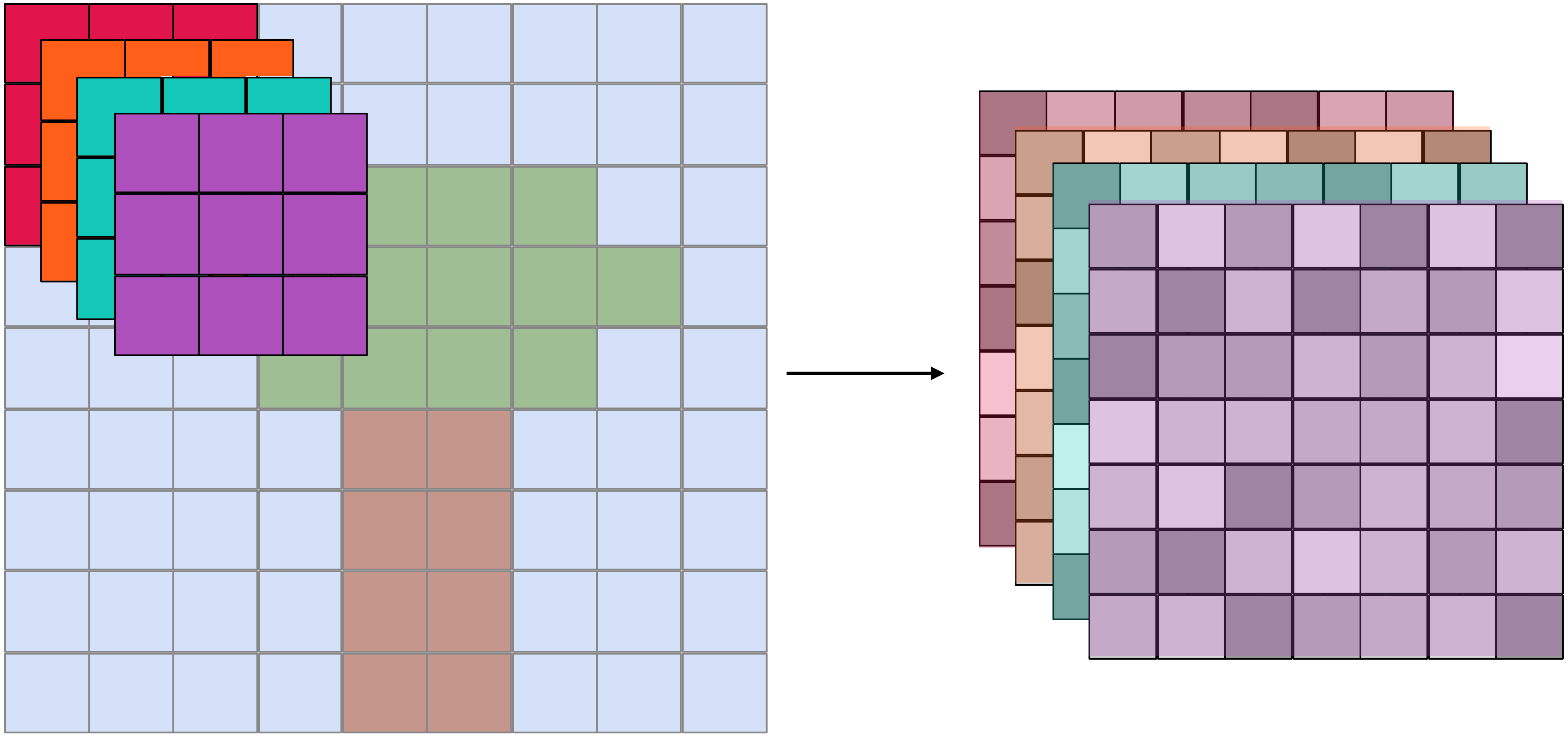
Edges and textures Object parts Objects



Deeper and more complex feature maps

Input dimension: (1,1,9,9)

Output dimension: (1,4,9,9)



Different filters (weights) activate for different patterns

Defining convolutional kernels

Conv2d

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

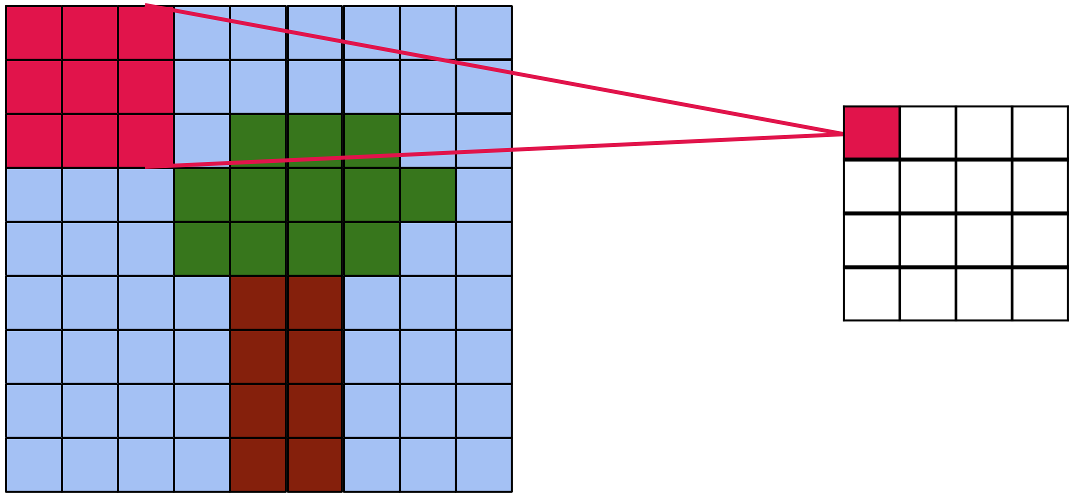
Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

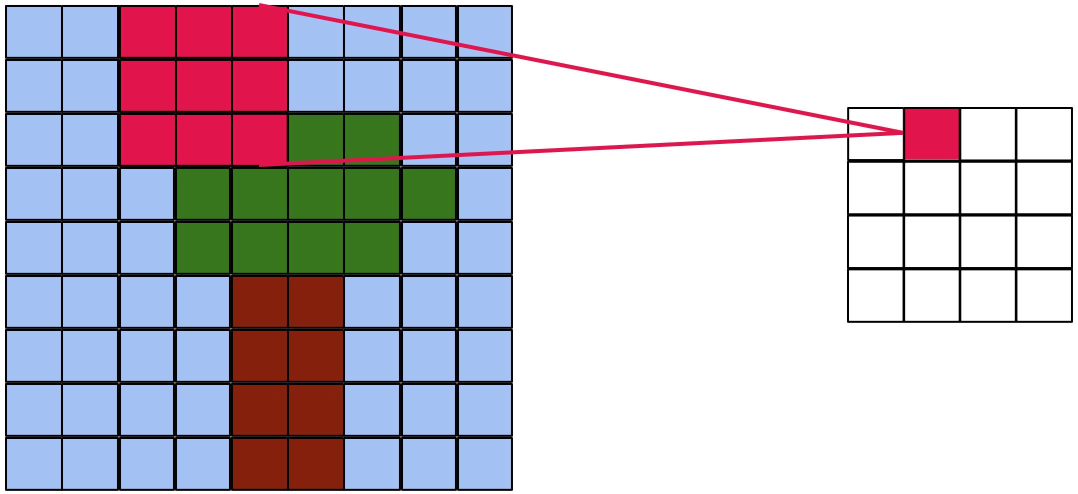
- **out_channels**
  - The number of different kernels;
  - Increase to detect different *types* of local patterns
  - Each out_channel is a summation over input channels
- **kernel_size**
  - The size of the kernel e.g., (3,3)
  - Increase to capture more *global* patterns
- **stride**
  - ?
- **padding**
  - ?
- **dilation**
  - ?

Defining convolutional kernels: Stride



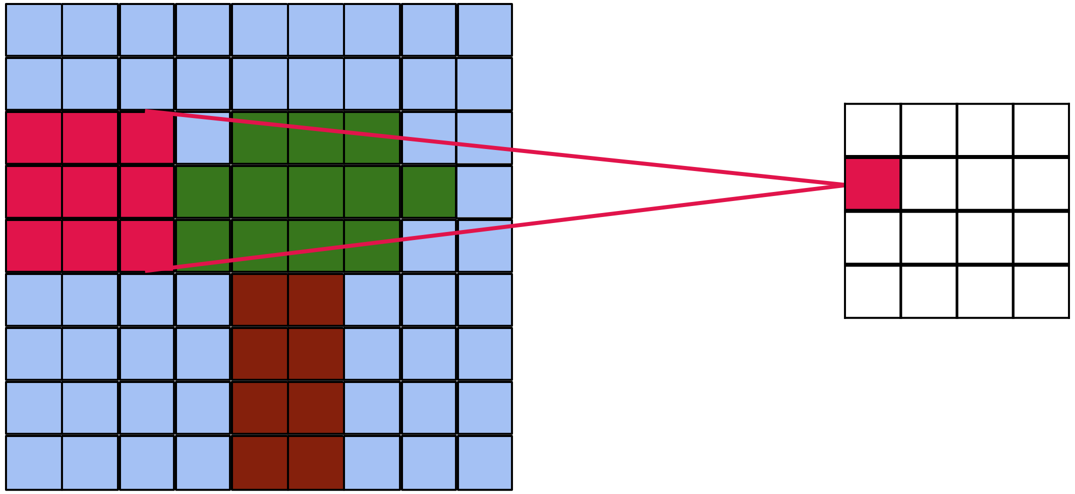
Stride = 2 ⇒ kernel is applied every other pixel

Defining convolutional kernels: Stride



Stride = 2 ⇒ kernel is applied every other pixel

Defining convolutional kernels: Stride



Stride = 2 ⇒ kernel is applied every other pixel

Defining convolutional kernels

Conv2d

CLASS torch.nn.Conv2d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None*) [SOURCE]

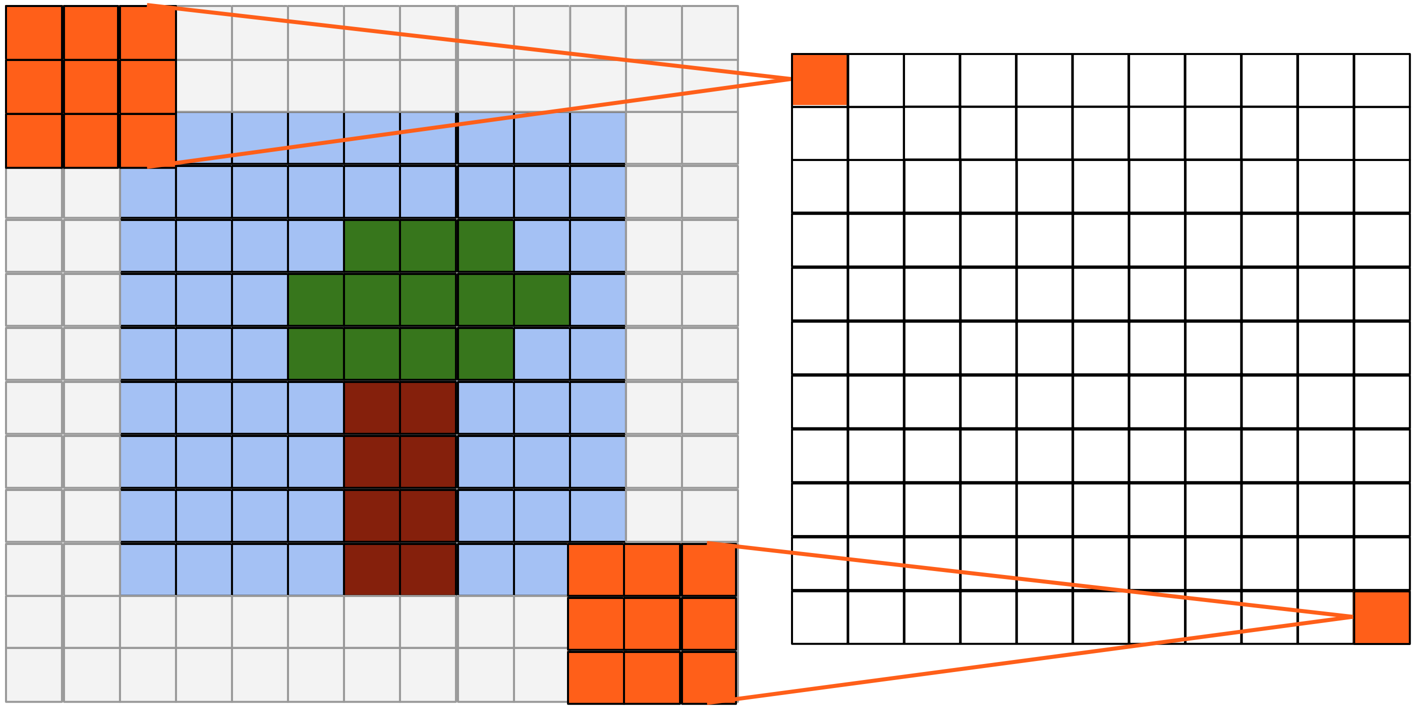
Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

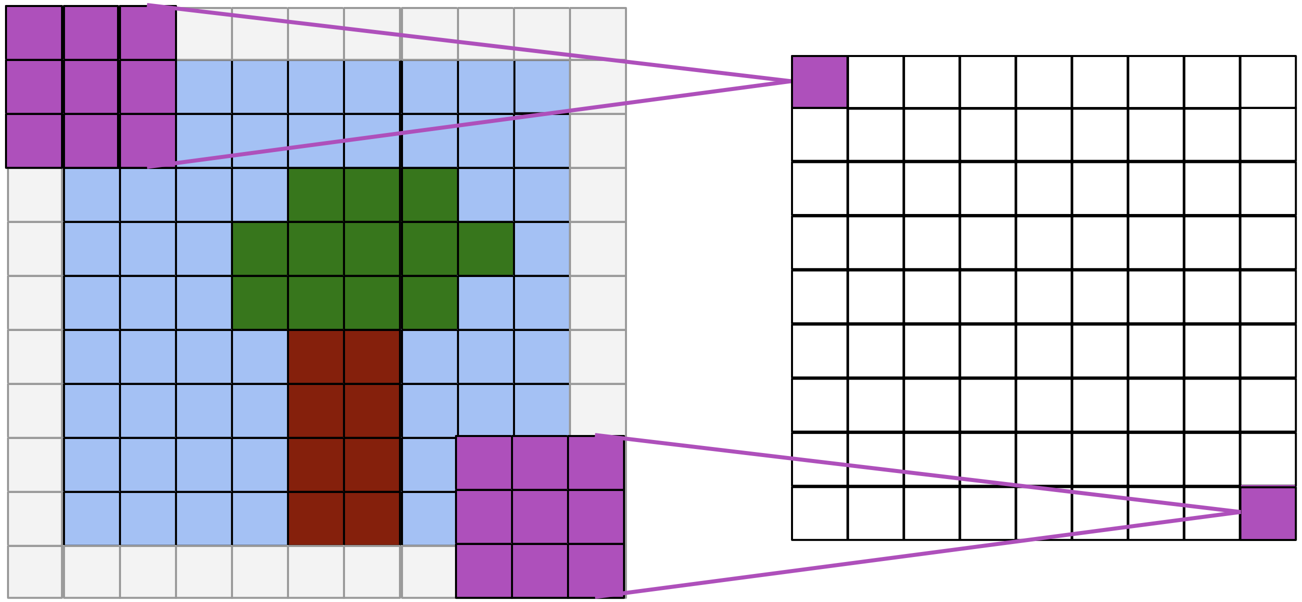
- **out_channels**
  - The number of different kernels;
  - Increase to detect different *types* of local patterns
  - Each out_channel is a summation over input channels
- **kernel_size**
  - The size of the kernel e.g., (3,3)
  - Increase to capture more *global* patterns
- **stride**
  - Defines the adjacency of consecutive kernel applications
  - Enables the output to be further down sampled without increasing the kernel size
- **padding**
  - ?
- **dilation**
  - ?

Defining convolutional kernels: Padding



Padding = 2 ⇒ dimensions are increased by 2 (of a provided value, generally 0). When combining padding of 2 and a kernel size of (3,3), the output dimension = input dimension + 1 (up sampling)

Defining convolutional kernels: Padding



Padding = 1 ⇒ dimensions are increased by 1 (of a provided value, generally 0). When combining padding of 1 and a kernel size of (3,3), the output dimension = input dimension

Defining convolutional kernels

Conv2d

CLASS torch.nn.Conv2d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None*) [SOURCE]

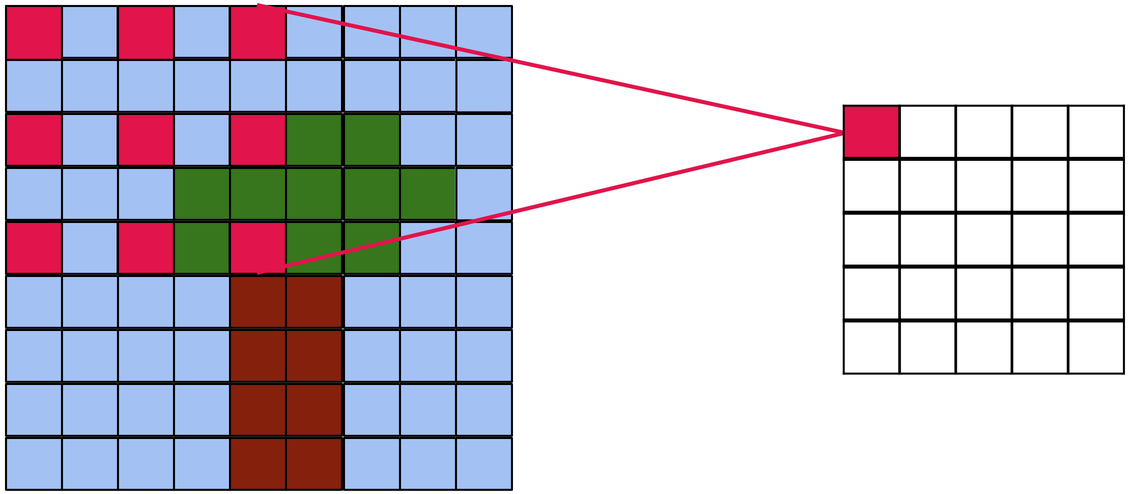
Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

- **out_channels**
  - The number of different kernels;
  - Increase to detect different *types* of local patterns
  - Each out_channel is a summation over input channels
- **kernel_size**
  - The size of the kernel e.g., (3,3)
  - Increase to capture more *global* patterns
- **stride**
  - Defines the adjacency of consecutive kernel applications
  - Enables the output to be further down sampled without increasing the kernel size
- **padding**
  - Adds additional values to each dimension
  - Manipulates the output dimensions (prevent down sampling\induce up sampling)
- **dilation**
  - ?

Defining convolutional kernels: Dilation



Dilation = 2 ⇒ 0 value, non–tunable weights of value 0 are interspersed every other weight. When combining dilation of 2 and a kernel size of (3,3), the kernel has *approximately* the same locality as a (5,5) kernel but with less parameters

Defining convolutional kernels

Conv2d

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

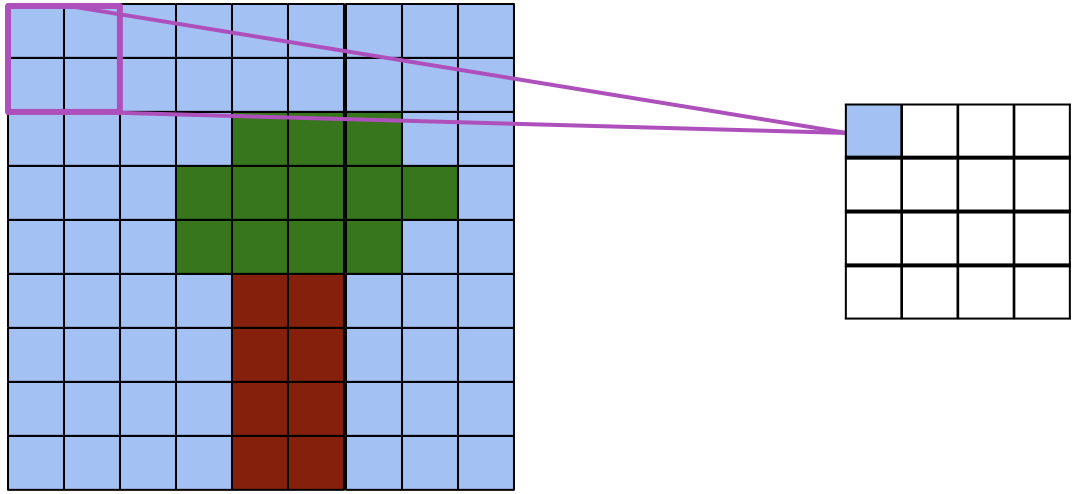
Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

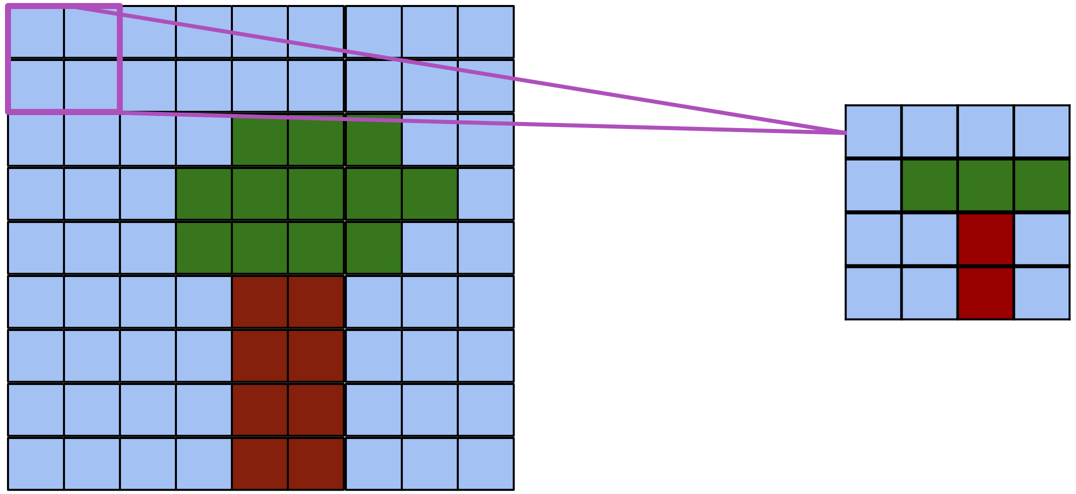
- **out_channels**
  - The number of different kernels;
  - Increase to detect different *types* of local patterns
  - Each out_channel is a summation over input channels
- **kernel_size**
  - The size of the kernel e.g., (3,3)
  - Increase to capture more *global* patterns
- **stride**
  - Defines the adjacency of consecutive kernel applications
  - Enables the output to be further down sampled without increasing the kernel size
- **padding**
  - Adds additional values to each dimension
  - Manipulates the output dimensions (prevent down sampling\induce up sampling)
- **dilation**
  - Adds non-tunable weights to the filter
  - Allows for larger locality without adding additional parameters

Local pooling



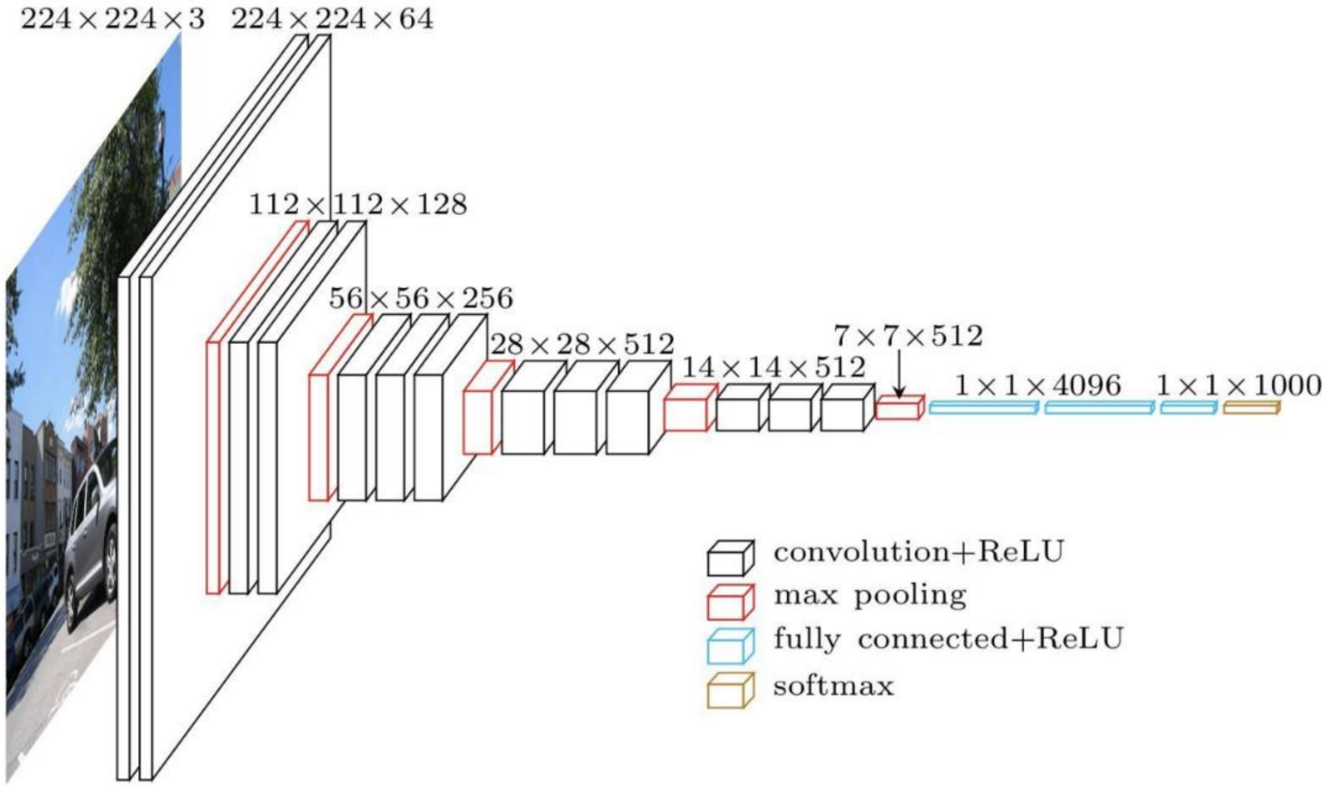
Computes summary statistics e.g, max and mean over filters (rather than convolving). Reduces resolution/coarsens the image

Local pooling



Computes summary statistics e.g, max and mean over filters (rather than convolving). Reduces resolution/coarsens the image

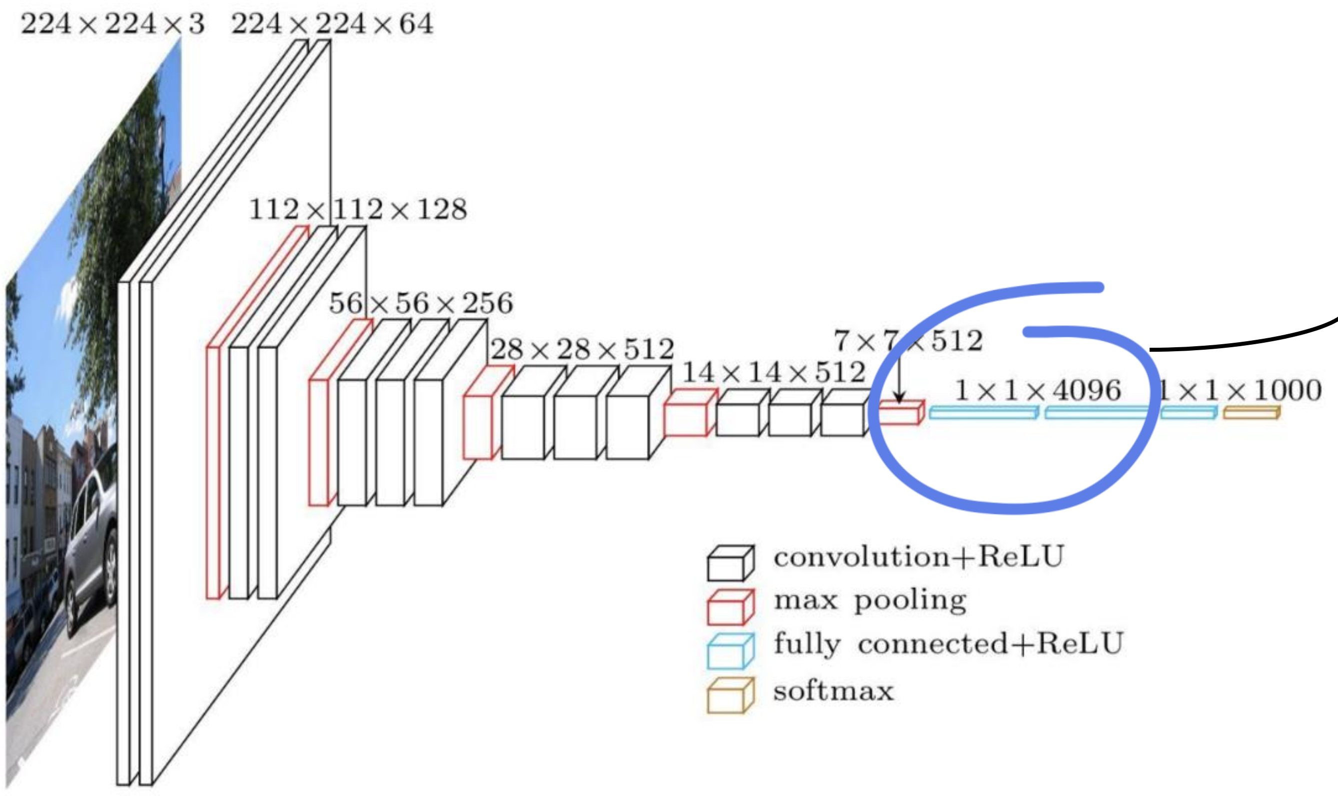
VGG



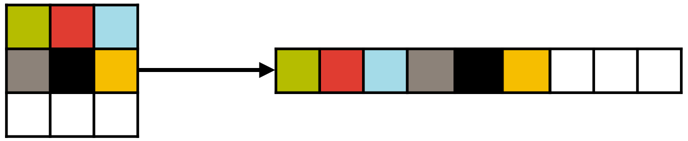
- Uses small kernels (3,3) with non-linear ReLU activations between conv layers
- Hugely upscales the number of filters whilst down sampling the image

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION, 2015

To MLPs (flatten/global pooling)



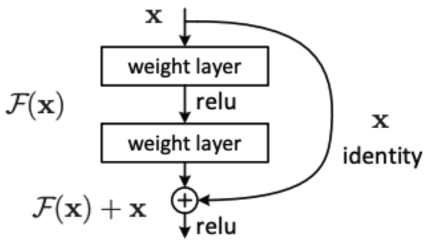
- MLPs require 1D input however, output of convolutions/pooling is 3D (7,7,512)
- Generally, “flattening” is used i.e.,



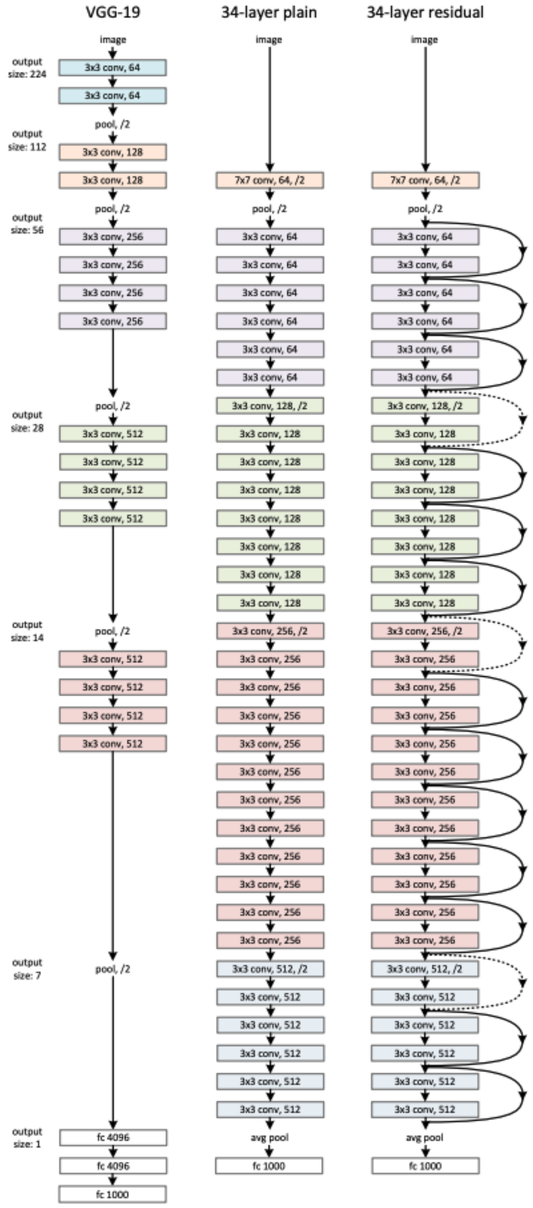
- Channel wise global pooling can also be used to reduce the dimensionality further however, this is not commonly used
    - Also induces invariance

Deeper CNNs: ResNet

- Performance of deep networks (>10 layers) can saturate
  - Even when using batch norm/correct initialisation etc.
- ResNet proposes modelling
  - $g(z) - x$ where $z = \alpha + \beta x$
  - The **residual** change is hypothesised to be easier to model than the full transformation



Deep Residual Learning for Image Recognition, 2015



Learning invariance and equivariance

Learning invariance and equivariance

- Convolutions enable an **analytic expression of translational equivariance**
- When detecting objects **other desirable invariances and equivariances** included:
  - **Rotational**
    - Bounding boxes should equally rotate (equivariance)
    - Classification results should be the same (invariance)
  - **Scale**
    - Bounding boxes should become larger/smaller (equivariance)
    - Classification results should be the same (invariance)

Data augmentations

- When there is no analytic expression, it might be possible to **augment** the input data with **simulated samples** to ensure that the model **learns the equ/invariance**
  - Images can be easily **rotated/cropped** and **scaled**
  - **Noise** can be overlayed to mimic ‘poor quality’ images or **parts of the image can be obscured**
- Not all domains are **as** amenable to data augmentations e.g., healthcare data
- Data augmentations are key for contrastive self-supervised learning of images (studied in week 5)

Anonymous feedback

- https://forms.gle/c3fFtxGG1aAUSrU56