IMAGE DATA ANALYSIS (6CFU)

MODULE OF

REMOTE SENSING

(9 CFU)

A.Y. 2022/23

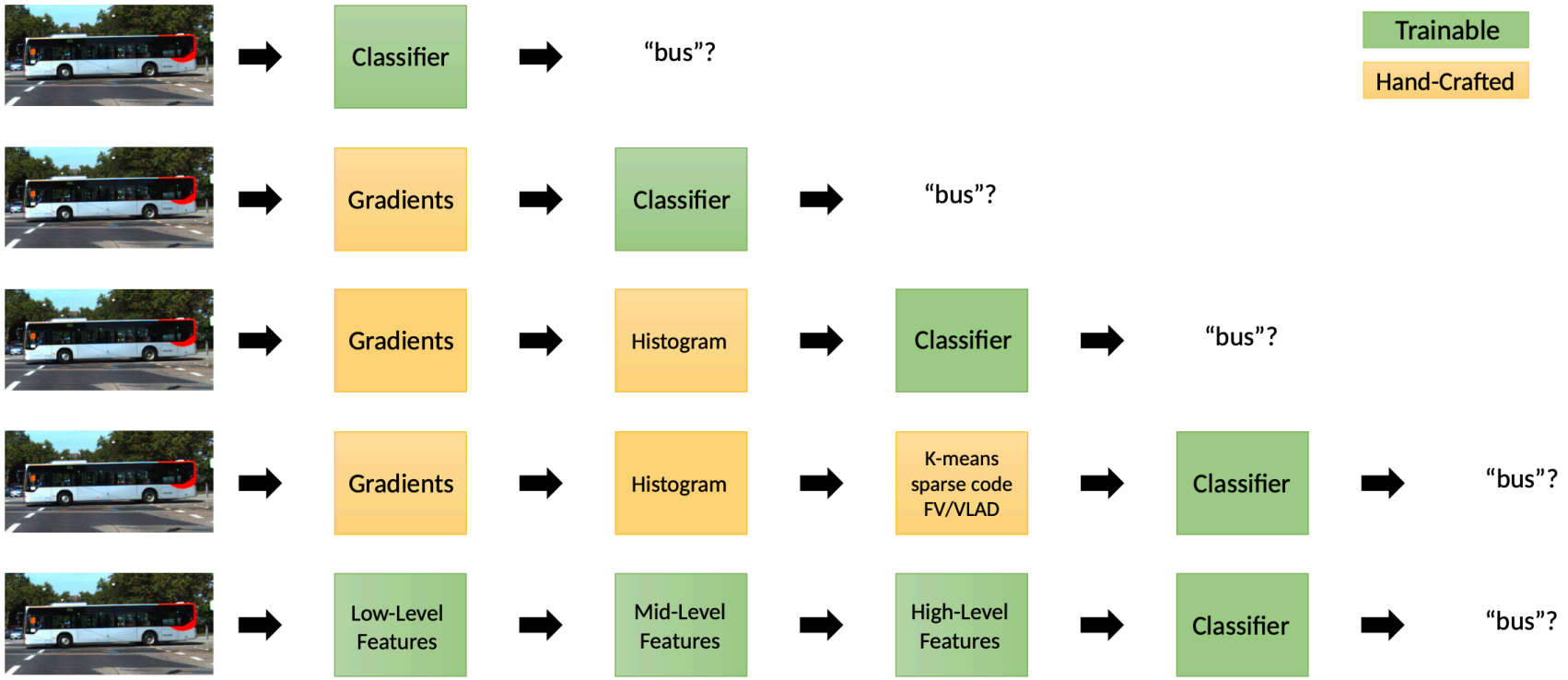
MASTER OF SCIENCE IN COMMUNICATION TECHNOLOGIES AND MULTIMEDIA
MASTER OF SCIENCE IN COMPUTER SCIENCE, LM INGEGNERIA INFORMATICA

PROF. ALBERTO SIGNORONI – DR. MATTIA SAVARDI

CONVOLUTIONAL NEURAL NETWORKS

Introduction

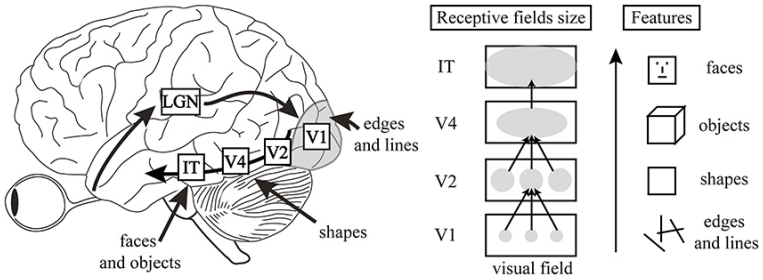
- **Representation learning for image classification**

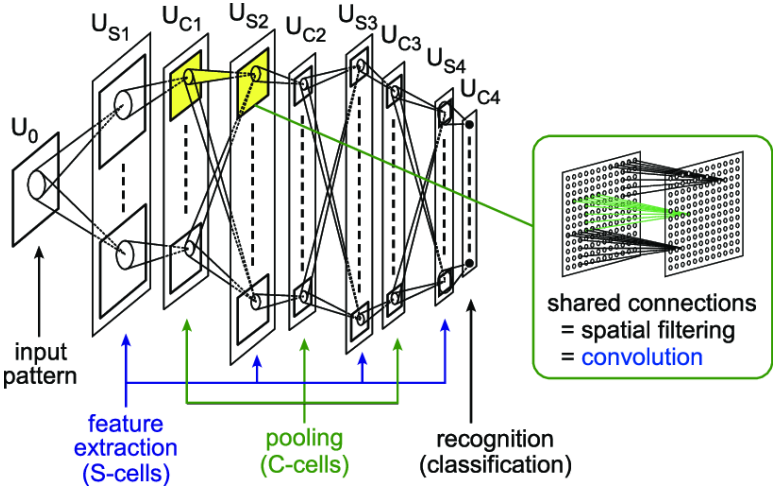


inductive "reasoning" to control vs **inductive "biases" for learning**

Introduction

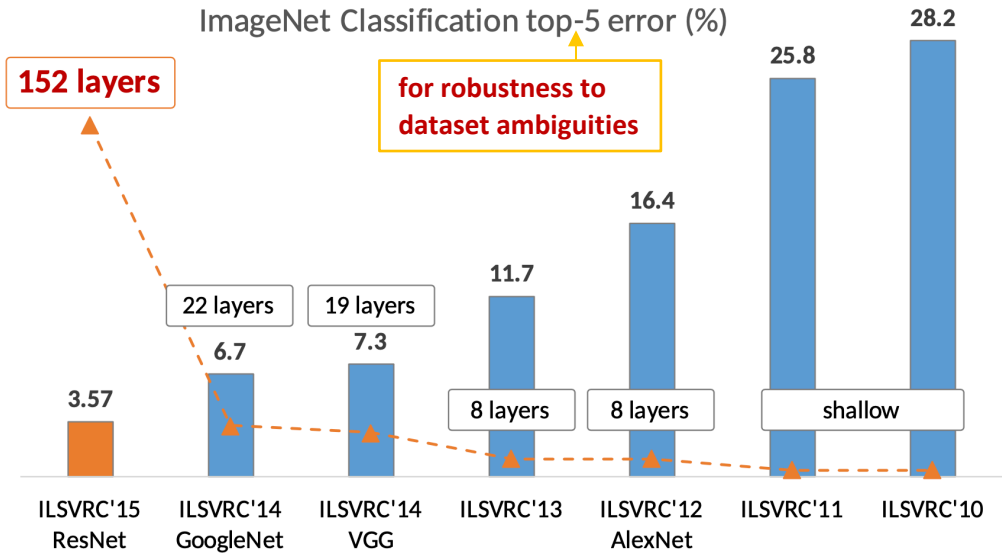
- **Convolutional Neural Network are key in "semantic" image interpretation** where images are typically:
  - 1 component 2D data, i.e. matrices of real (usually integer) numbers (grayscale)
  - 3 component 2D data (color)
  - N-component 2D data (multispectral, hyperspectral)
  - Conceptually easy generalizations allow considering 1D (e.g. audio) as well as 3D data (e.g. volumetric data from medical scan) and also 2D+t data (video).
- **Strong inspiration from 1. brain functioning and 2. past research results**
  1. Hierarchical organization of neural paths into the mammalian brain visual cortex
  2. From Neocognitron (Fukushima, 1980) to Convolutional Nets (LeCun, 1989)… to Imagenet (Krizhevsky, Sutskever, Hinton, 2012)
  - See historical notes on
    - http://cs231n.stanford.edu/slides/2021/lecture_5.pdf
    - on the textook of Goodfellow et al.





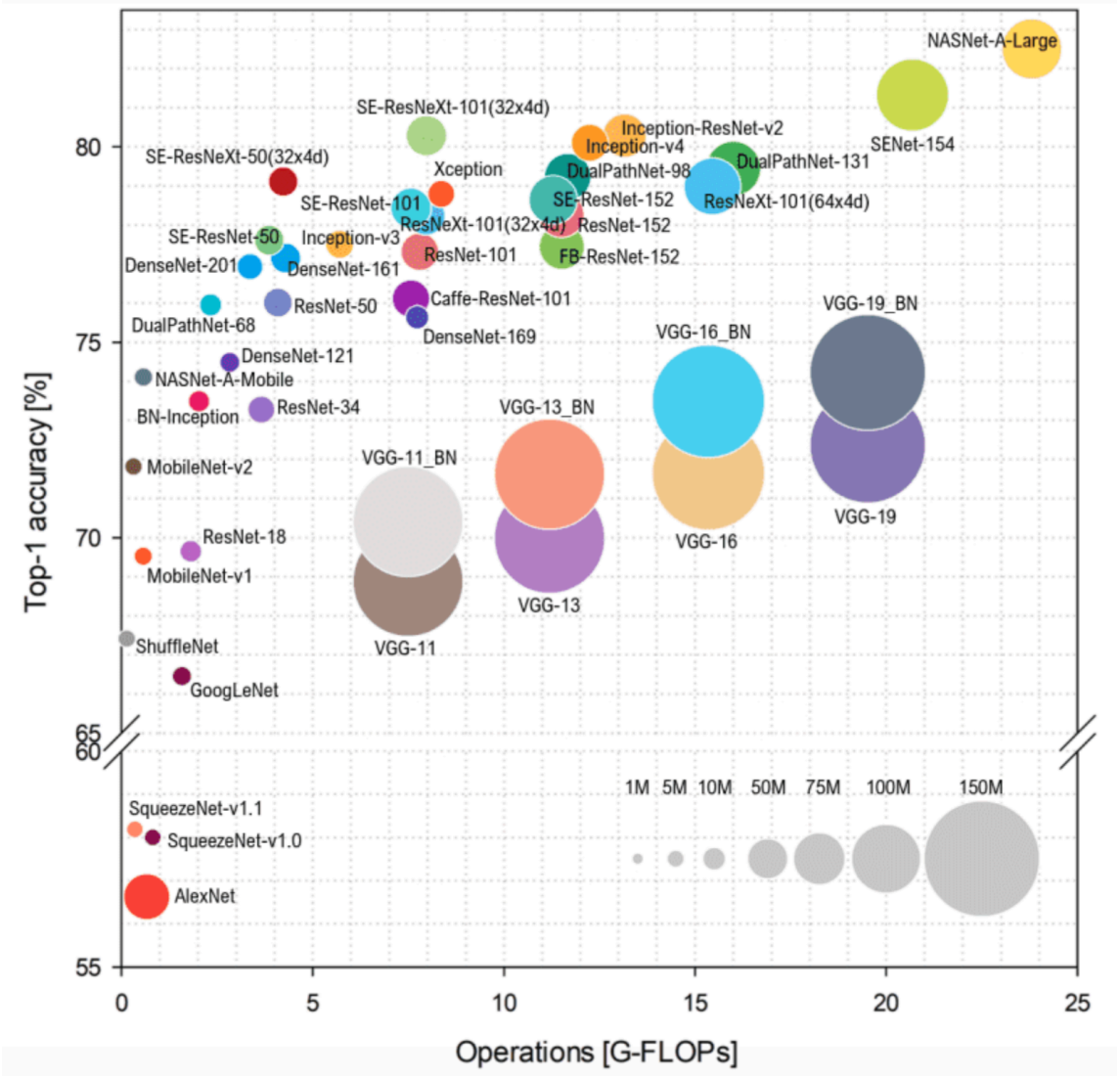
Introduction

- **Convolutional Neural Network are key in "semantic" image interpretation** where images are typically:
  - 1 component 2D data, i.e. matrices of real (usually integer) numbers (grayscale)
  - 3 component 2D data (color)
  - N-component 2D data (multispectral, hyperspectral)
  - Conceptually easy generalizations allow considering 1D (e.g. audio) as well as 3D data (e.g. volumetric data from medical scan) and also 2D+t data (video).
- **ImageNet Large Scale Visual Recognition Challenge**
  - Classification into 1000 object categories. Following picture is from 2015, current SoA is 1.3% (well over human performance ~5%)
  - CNN models, year (levels)
    - LeNet, 1990 (7)
    - Alexnet, 2012 (8)
    - VGG, 2014 (19)
    - GoogLeNet, 2014 (22)
    - ResNet, 2015 (152 but low cplx)
    - DenseNet , 2017
    - EfficientNet, 2019



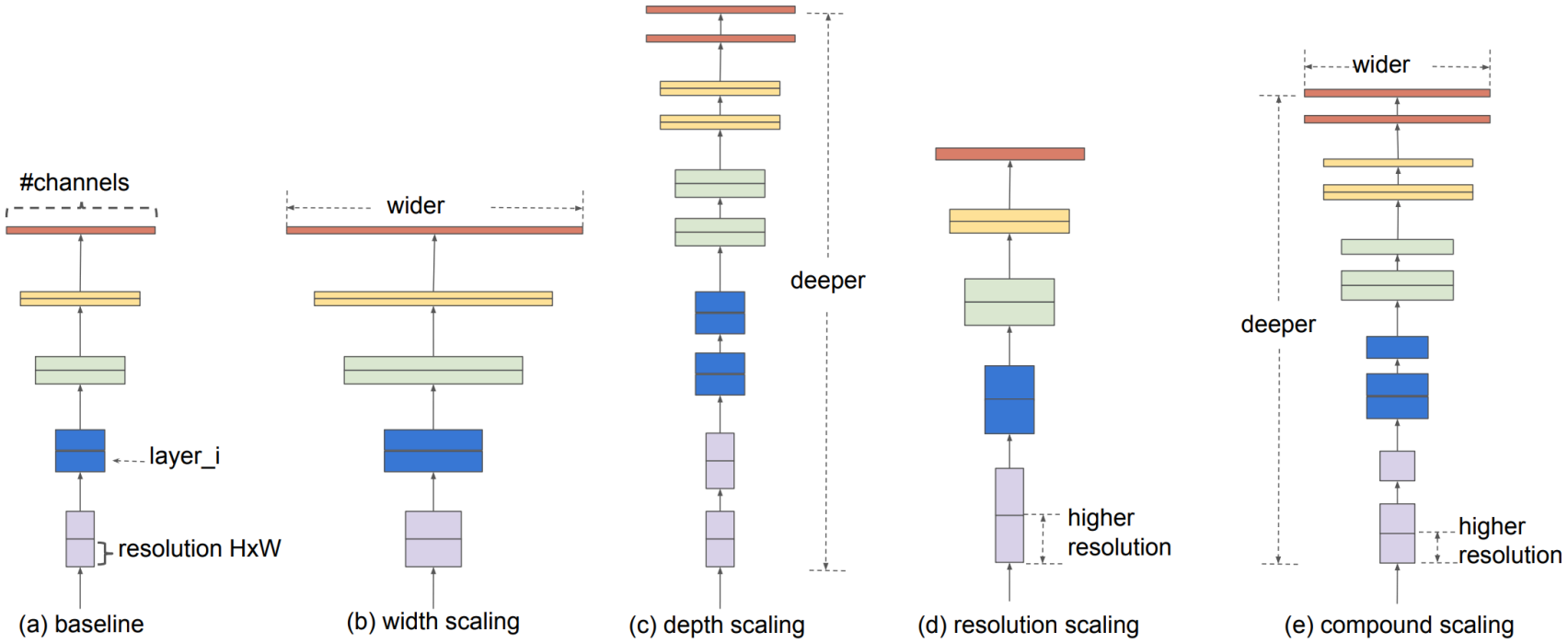
Introduction

- **CNN model complexity** (floating point operations for a single forward pass)
  - ImageNet Top-1 accuracy
  - https://arxiv.org/abs/1810.00736



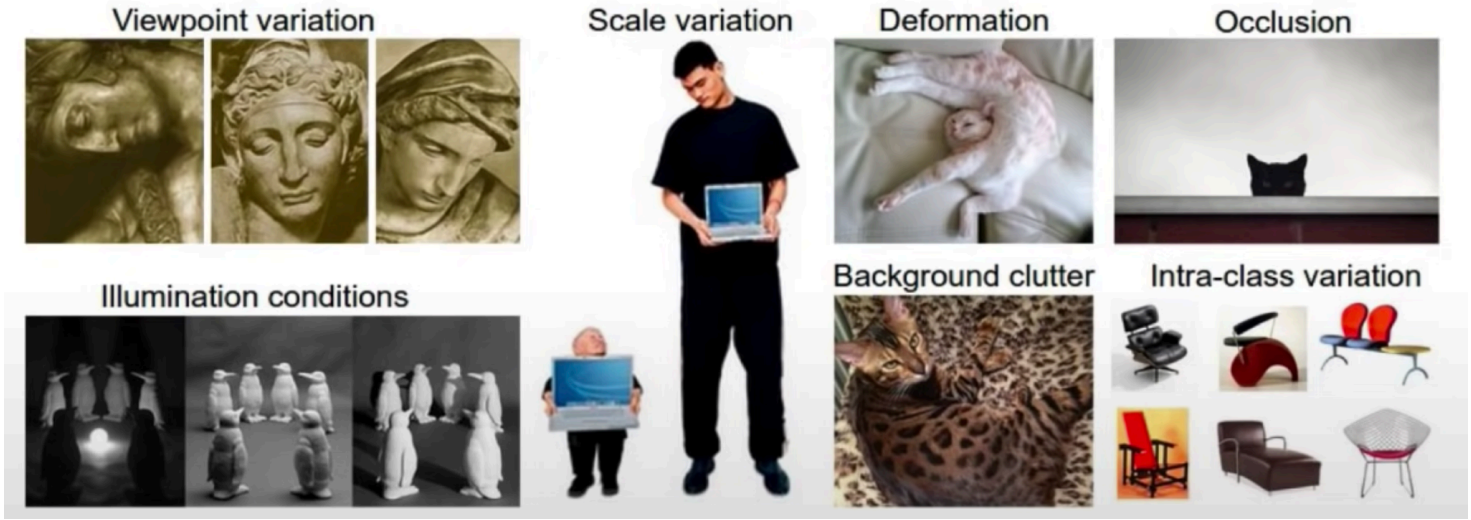
Introduction

- **CNN model scaling**
  - https://arxiv.org/pdf/1905.11946.pdf

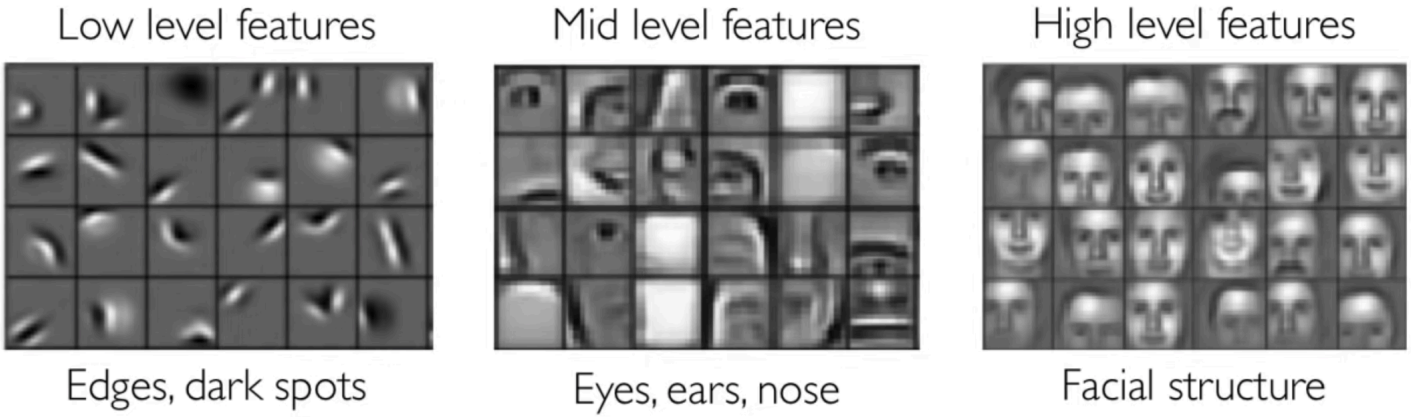


Introduction

- Manual (handcrafted) extraction of visual features is highly complex…

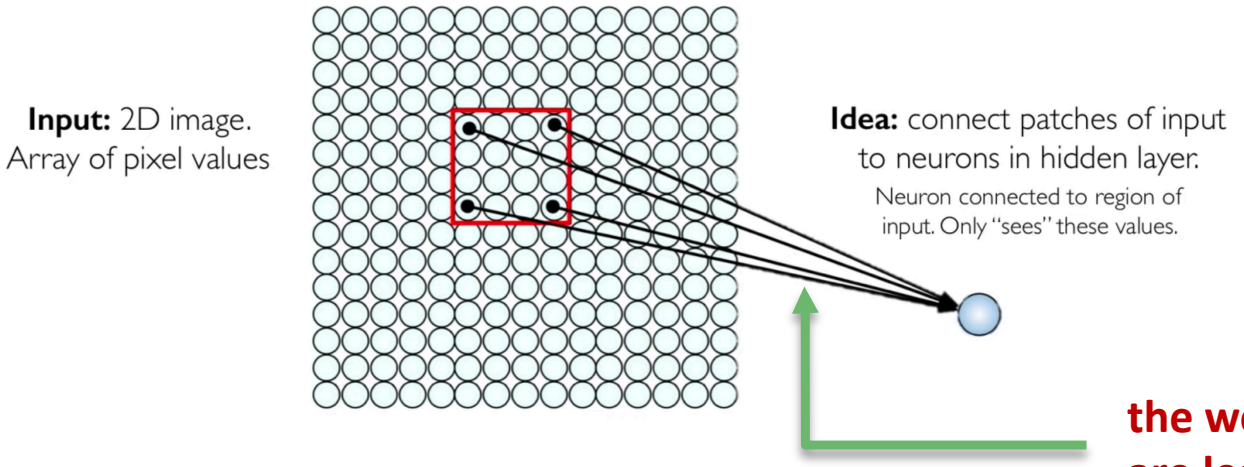


- CNN are successful (better than FCNN) in learning a hierarchy of visual features

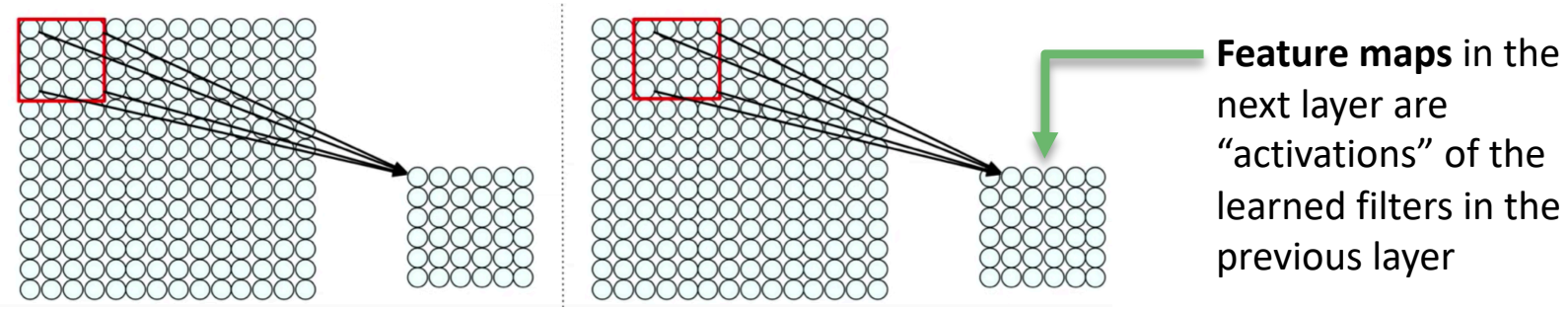


Introduction

- Main architectural idea in CNNs:
  - **sliding spatial patches from input** are connected to neurons of the **first hidden layer**
    - "weight sharing" imply performing "convolution" (or, more honestly, correlation)
    - the spatial patches can be many: multiple feature learning and feature map generation



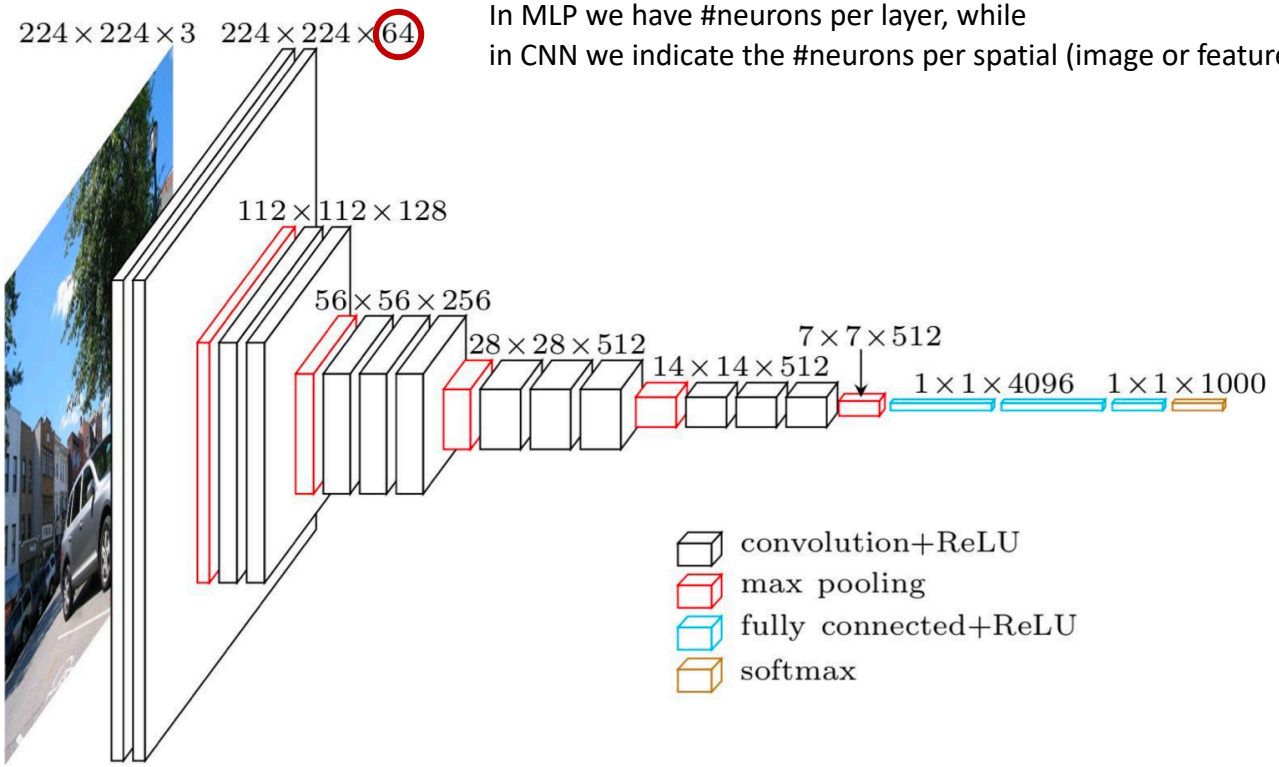
  - and **continuing doing the same** in the **subsequent hidden layers**



Introduction

- Typical CNN architecture: **VGG Network**
  - 3 Types of Layers
    - Convolution layers
    - Pooling Layers
    - Fully Connected Layers

In MLP we have #neurons per layer, while
in CNN we indicate the #neurons per spatial (image or feature map) location



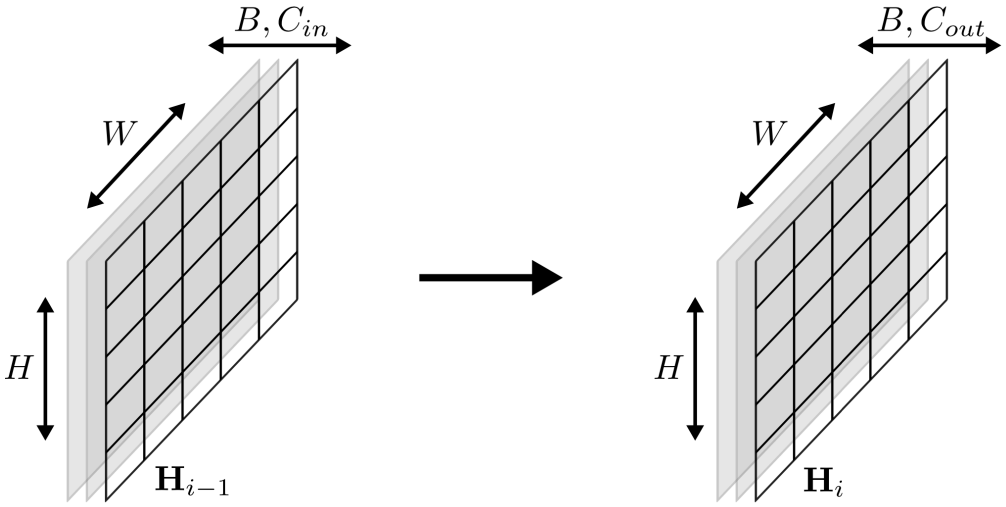
- https://arxiv.org/pdf/1409.1556.pdf

Notation

- We will use the following **notation:**
  - B: **Batch size**
  - W : **Width** of image or feature map
  - H: **Height** of image or feature map
  - C: Number of feature **channels** (=neurons per pixel)
    - C_{in}: Number of feature channels in current layer $\mathbf{H}_{i-1}$
    - C_{out}: Number of feature channels in next layer $\mathbf{H}_i$
      - C_{in} = 1 for first layer if input is a grayscale image
      - C_{in} = 3 for first layer if input is a color image
  - $K \times K$: Convolutional filter **kernel size**

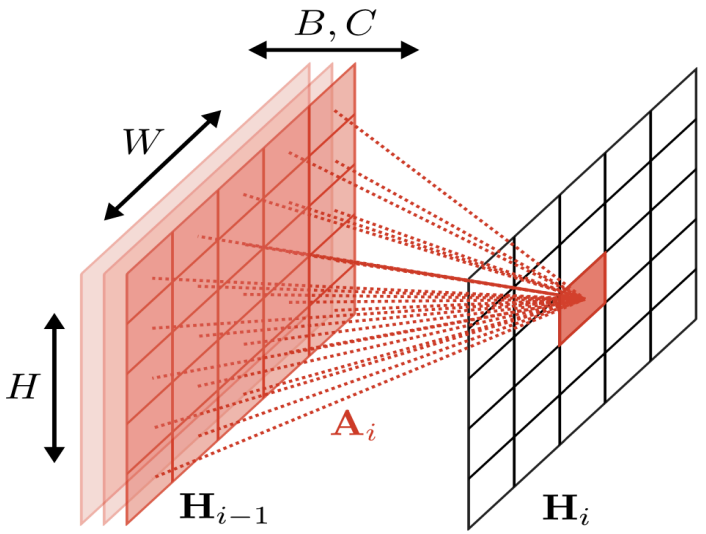
Layers have spatial extent and are represented as **tensors**

$$\mathbf{H}_i = \mathbf{H}_i(b,x,y,c)$$

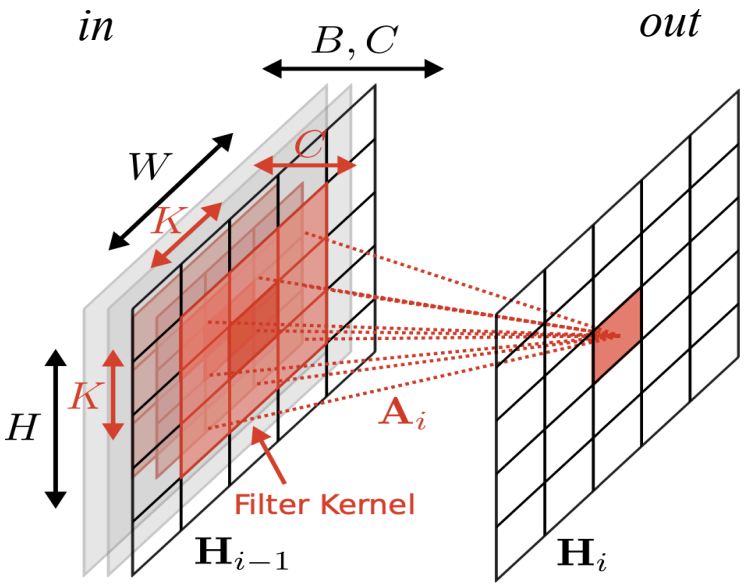


Fully Connected vs Convolutional layers

- **Fully connected layer:**
  - Number of weights for layer i =
  $= W_{out} \times H_{out} \times C_{out} \times (W_{in} \times H_{in} \times C_{in} + 1)$
    - C_{in} input and C_{out} output channels
    - $W \times H$ layer size (*in* or *h-1*, *out* or *h*)
    - +1 is for the bias
    - (images are flattened in vectors)

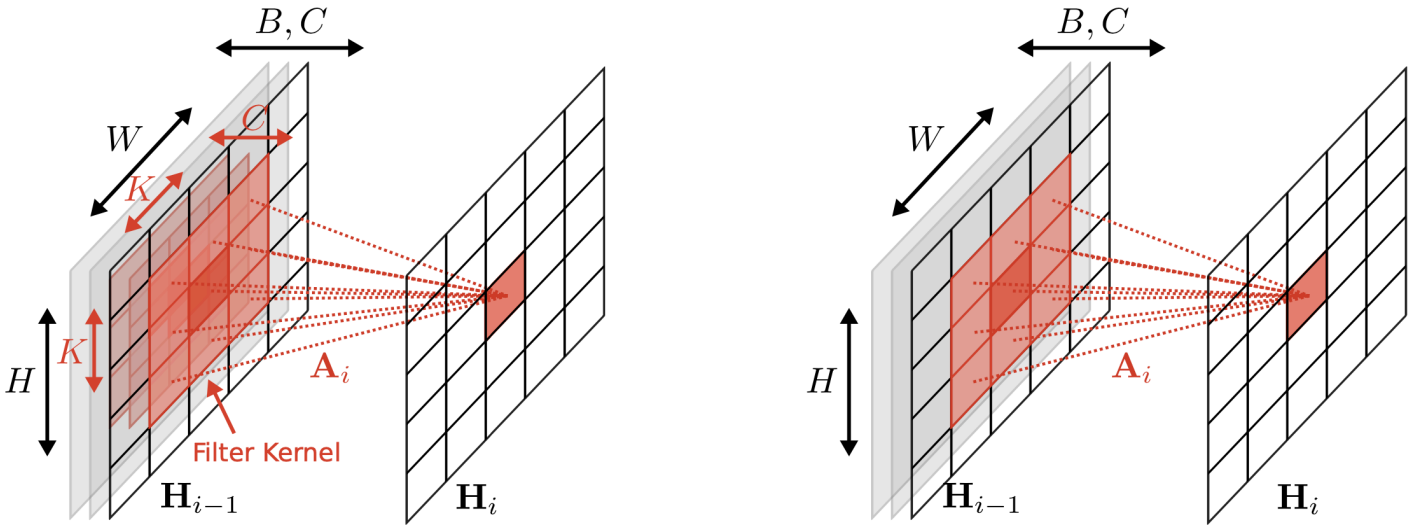


- **Convolutional layer:**
  - Number of weights for layer i =
  $= C_{out} \times (K \times K \times C_{in} + 1)$
    - $K \times K$ kernel size
    - one single filter per output channel
  - "***weight sharing***" (C_{out}): same weights no matter where we are in the image/feature map
  - "***localized focus***" ($K \times K$)
  - Convolution kernel is three-dimensional.



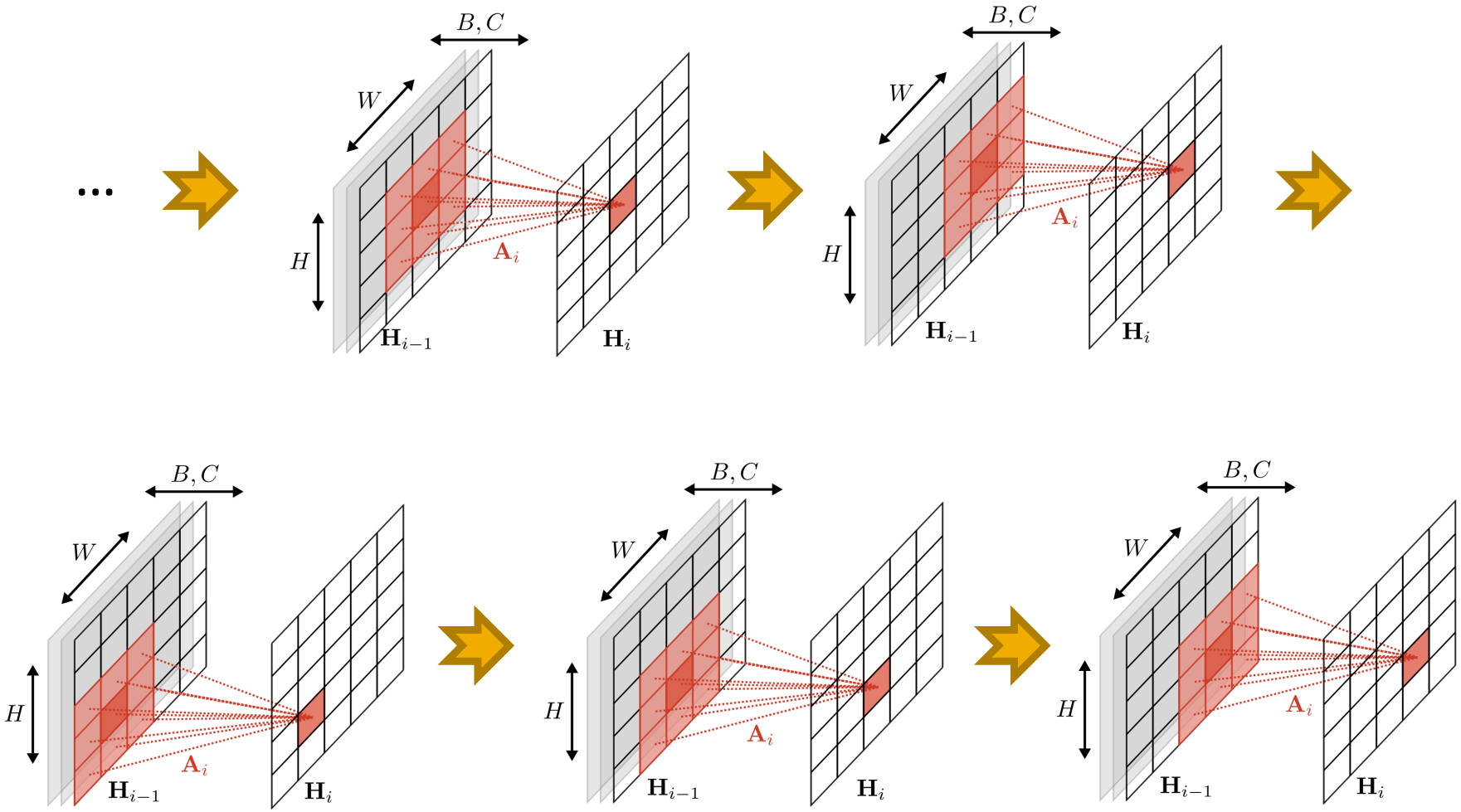
Convolutional layers

- Illustration convention
  - Convolution kernels are 3D on color image input and N-D for hidden layers seen as inputs, but in illustrations we drop the C dimension for clarity.
  - Usually, multiple convolution kernels are convolved with the input, each producing a different output channel. In illustrations, we show a single convolution for clarity.
  - We will use *Einstein notation* to make all operations formally clear and concise.



Convolutional layers

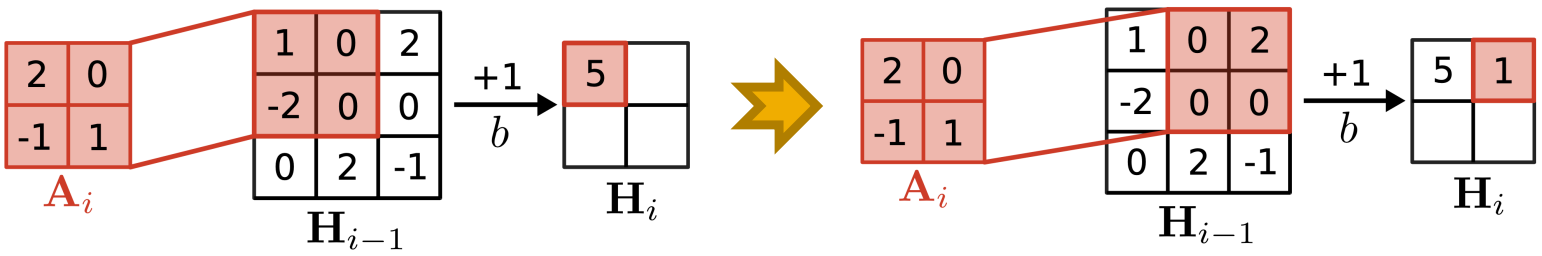
- **Convolution in action**

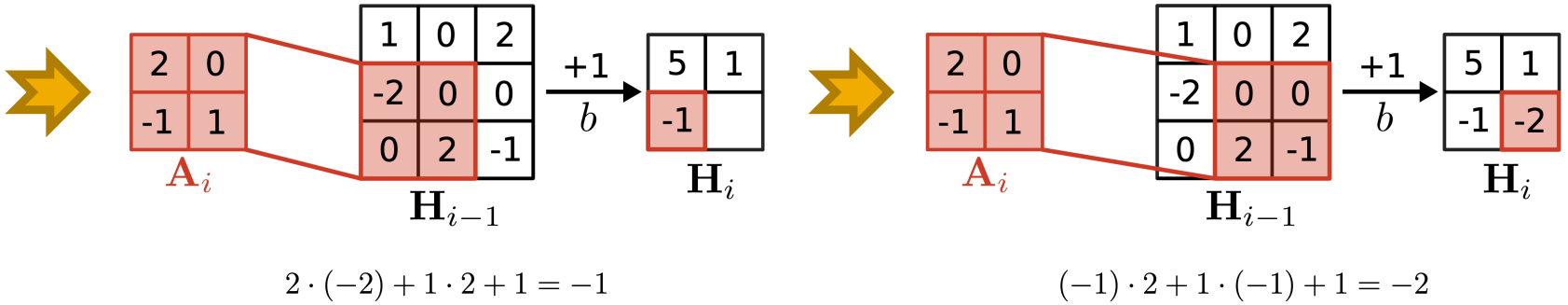


Convolutional layers

Convolution example

- Actually, this is a correlation (convolution would require flip along dimensions of the filter coefficients) but this does not change the substance and the expressivity of the operation
- If filters are separable (low rank kernels) computation can be made more efficient





Convolutional layers

- **Einstein notation:**
  - we use a modified form of **Einstein Notation** to represent convolutions (correlations) more compactly, where **capital letters** are used to denote **slices of a tensor** (es. raws of a matrix)
  - $A[i,j]$ denotes **one element** of the matrix A
  - $A[i,J]$ denotes the **i'th row** of matrix A
  - $A[I,j]$ denotes the **j'th column** of matrix A
  - $A[I,J]$ denotes the **full matrix** A
  - $H[i,j,k]$ denotes **one element** of the tensor H
  - $\ldots$
- Repeated capital letters in a product denote **summation** over those letters:

$$\begin{aligned} \mathbf{y} = \mathbf{A}\mathbf{x} \quad &\equiv \quad y[i] = \sum_j A[i,j]x[j] \\ &\equiv \quad y[i] = A[i,J]x[J] \end{aligned}$$

$$\begin{aligned} \mathbf{y} = \mathbf{x}^\top\mathbf{A} \quad &\equiv \quad y[j] = \sum_i A[i,j]x[i] \\ &\equiv \quad y[j] = A[I,j]x[I] \end{aligned}$$

Note the order of terms in the product is always the same: this simplify / disambiguate the use of the transpose operators in tensor products

Convolutional layers

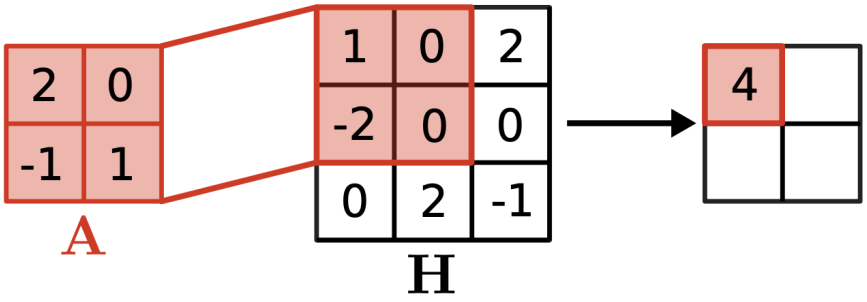
- Multiple capital letters denote summation over the **combination** of indices:

$$c = \sum_{i,j} A[i,j]B[i,j] = A[I,J]B[I,J]$$

- The indeces of tensors typically have **types** such as "batch index", "x coordinate", "y coordinate" or "channel index"
- Writing a tensor as $A[B,X,Y,C]$ with batch B, coordinates X,Y and feature channels C makes the type of the tensor elements and the order of the indeces **explicit.** It therefore also disambiguates $\mathbf{A}$ from $\mathbf{A}^\top$.
- We will therefore use Einstein notation to describe the (hidden) layers and filter weights in a CNN
- Einstein notation can also be used in **NumPy** (`numpy.einsum`)

Convolutional layers

- **Convolution…** in the context of CNN



Definition of Convolution Operator:

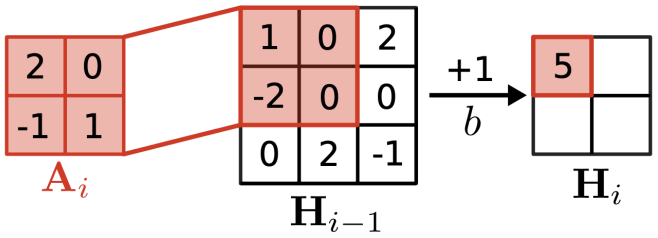
$$[\mathbf{A} * \mathbf{H}](\mathbf{x}) = \sum_{\Delta\mathbf{x} \in \mathbb{Z}^2} \mathbf{A}(\Delta\mathbf{x})\, \mathbf{H}(\mathbf{x} + \Delta\mathbf{x})$$

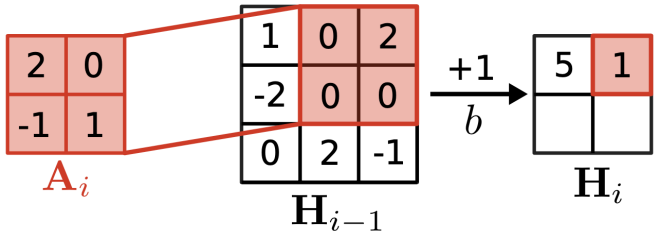
- Convolution of feature map $\mathbf{H}$ with filter kernel $\mathbf{A}$
- Convolutions are **translation equivariant**

Translate the same way as the input does

Convolutional layers

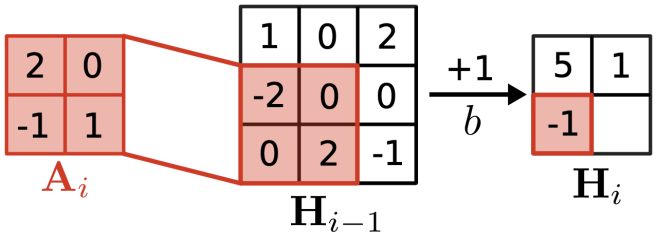
- **Convolution example** (assuming **0-based** indexing):

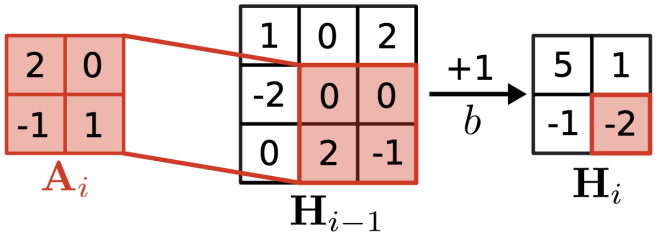




$$[\mathbf{A} * \mathbf{H}]\begin{pmatrix}0\\0\end{pmatrix} = \mathbf{A}\begin{pmatrix}0\\0\end{pmatrix}\mathbf{H}\begin{pmatrix}0+0\\0+0\end{pmatrix} + \mathbf{A}\begin{pmatrix}0\\1\end{pmatrix}\mathbf{H}\begin{pmatrix}0+0\\0+1\end{pmatrix} + \mathbf{A}\begin{pmatrix}1\\0\end{pmatrix}\mathbf{H}\begin{pmatrix}0+1\\0+0\end{pmatrix} + \mathbf{A}\begin{pmatrix}1\\1\end{pmatrix}\mathbf{H}\begin{pmatrix}0+1\\0+1\end{pmatrix}$$

$$[\mathbf{A} * \mathbf{H}]\begin{pmatrix}0\\1\end{pmatrix} = \mathbf{A}\begin{pmatrix}0\\0\end{pmatrix}\mathbf{H}\begin{pmatrix}0+0\\1+0\end{pmatrix} + \mathbf{A}\begin{pmatrix}0\\1\end{pmatrix}\mathbf{H}\begin{pmatrix}0+0\\1+1\end{pmatrix} + \mathbf{A}\begin{pmatrix}1\\0\end{pmatrix}\mathbf{H}\begin{pmatrix}0+1\\1+0\end{pmatrix} + \mathbf{A}\begin{pmatrix}1\\1\end{pmatrix}\mathbf{H}\begin{pmatrix}0+1\\1+1\end{pmatrix}$$

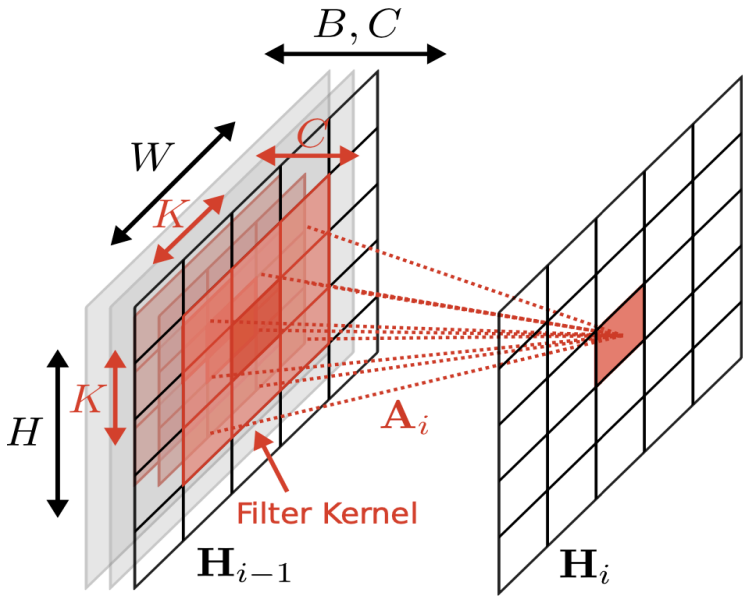




$$[\mathbf{A} * \mathbf{H}]\begin{pmatrix}1\\0\end{pmatrix} = \mathbf{A}\begin{pmatrix}0\\0\end{pmatrix}\mathbf{H}\begin{pmatrix}1+0\\0+0\end{pmatrix} + \mathbf{A}\begin{pmatrix}0\\1\end{pmatrix}\mathbf{H}\begin{pmatrix}1+0\\0+1\end{pmatrix} + \mathbf{A}\begin{pmatrix}1\\0\end{pmatrix}\mathbf{H}\begin{pmatrix}1+1\\0+0\end{pmatrix} + \mathbf{A}\begin{pmatrix}1\\1\end{pmatrix}\mathbf{H}\begin{pmatrix}1+1\\0+1\end{pmatrix}$$

$$[\mathbf{A} * \mathbf{H}]\begin{pmatrix}1\\1\end{pmatrix} = \mathbf{A}\begin{pmatrix}0\\0\end{pmatrix}\mathbf{H}\begin{pmatrix}1+0\\1+0\end{pmatrix} + \mathbf{A}\begin{pmatrix}0\\1\end{pmatrix}\mathbf{H}\begin{pmatrix}1+0\\1+1\end{pmatrix} + \mathbf{A}\begin{pmatrix}1\\0\end{pmatrix}\mathbf{H}\begin{pmatrix}1+1\\1+0\end{pmatrix} + \mathbf{A}\begin{pmatrix}1\\1\end{pmatrix}\mathbf{H}\begin{pmatrix}1+1\\1+1\end{pmatrix}$$

Convolutional layers



$$\underbrace{\mathbf{H}_i[b, x, y, c_{out}]}_{\text{Current Layer}} = g\left(\underbrace{\mathbf{A}_i[\Delta X, \Delta Y, C_{in}, c_{out}]}_{\text{Weights}} \underbrace{\mathbf{H}_{i-1}[b, x + \Delta X, y + \Delta Y, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}}\right)$$

- b: Batch index x, y: Spatial locations c_{in}, c_{out}: Feature channels g: Act. func.
- $\mathbf{H}_i[b, x, y, d]$ denotes feature d at image position x, y in layer i of batch b
- $\mathbf{A}_i$ and b_i depend on i as every layer has its own parameters
- Einstein summation considers all combinations over a range of indices $(\Delta X, \Delta Y)$

Convolutional layers

- **Convolutions…** are translation equivariant

Invariance: $f(\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$

same result for any transformed input

Equivariance: $\mathcal{T}_\theta[f](\mathbf{H}) = f(\mathcal{T}_\theta[\mathbf{H}])$

output transforms as its input for some specific transformation type (in the case of CNN: translation)

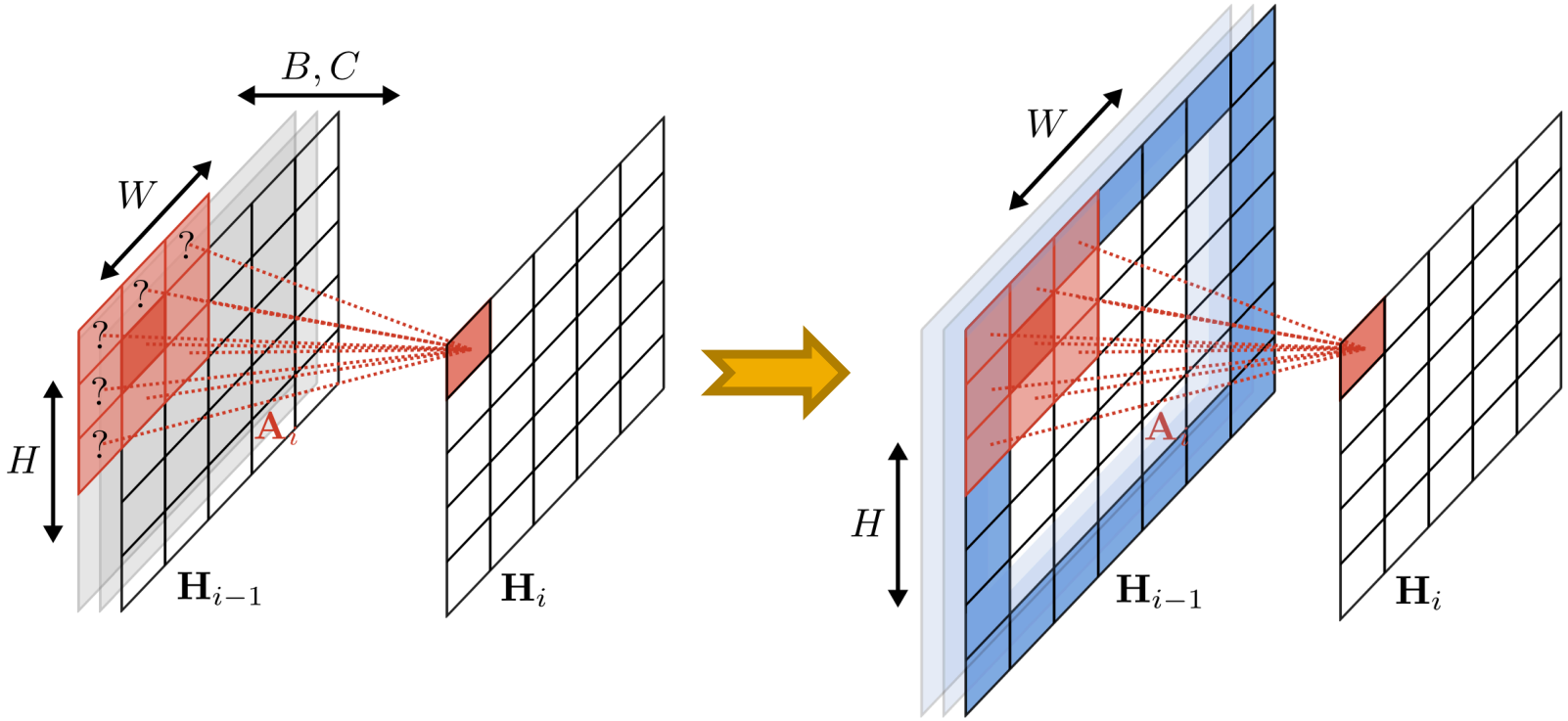
Proof:

$$
\begin{aligned}
[\mathbf{A} * \mathcal{T}_\mathbf{t}[\mathbf{H}]](\mathbf{x}) &= \sum_{\Delta\mathbf{x}\in\mathbb{Z}^2} \mathbf{A}(\Delta\mathbf{x})\, \mathcal{T}_\mathbf{t}[\mathbf{H}](\mathbf{x}+\Delta\mathbf{x}) && \text{definition of convolution} \\
&= \sum_{\Delta\mathbf{x}\in\mathbb{Z}^2} \mathbf{A}(\Delta\mathbf{x})\, \mathbf{H}(\mathbf{x}+\Delta\mathbf{x}-\mathbf{t}) && \text{expanding translation operator} \\
&= \sum_{\Delta\mathbf{x}\in\mathbb{Z}^2} \mathbf{A}(\Delta\mathbf{x})\, \mathbf{H}((\mathbf{x}-\mathbf{t})+\Delta\mathbf{x}) && \text{rearranging} \\
&= [\mathbf{A} * \mathbf{H}](\mathbf{x}-\mathbf{t}) = \mathcal{T}_\mathbf{t}[\mathbf{A} * \mathbf{H}](\mathbf{x}) && \text{definition of convolution}
\end{aligned}
$$

- Convolutions are **translation equivariant** but not translation invariant
- Shifting the input, results in a **shifted output** (exception: image boundaries)
- Interpretation as **inductive bias** built into ConvNet architecture

Convolutional layers

- **Padding**
  - Convolutions can only be executed if kernel lies entirely **within input domain**
  - **Idea of padding**: add boundary of appropriate size with zeros (blue) around input tensor

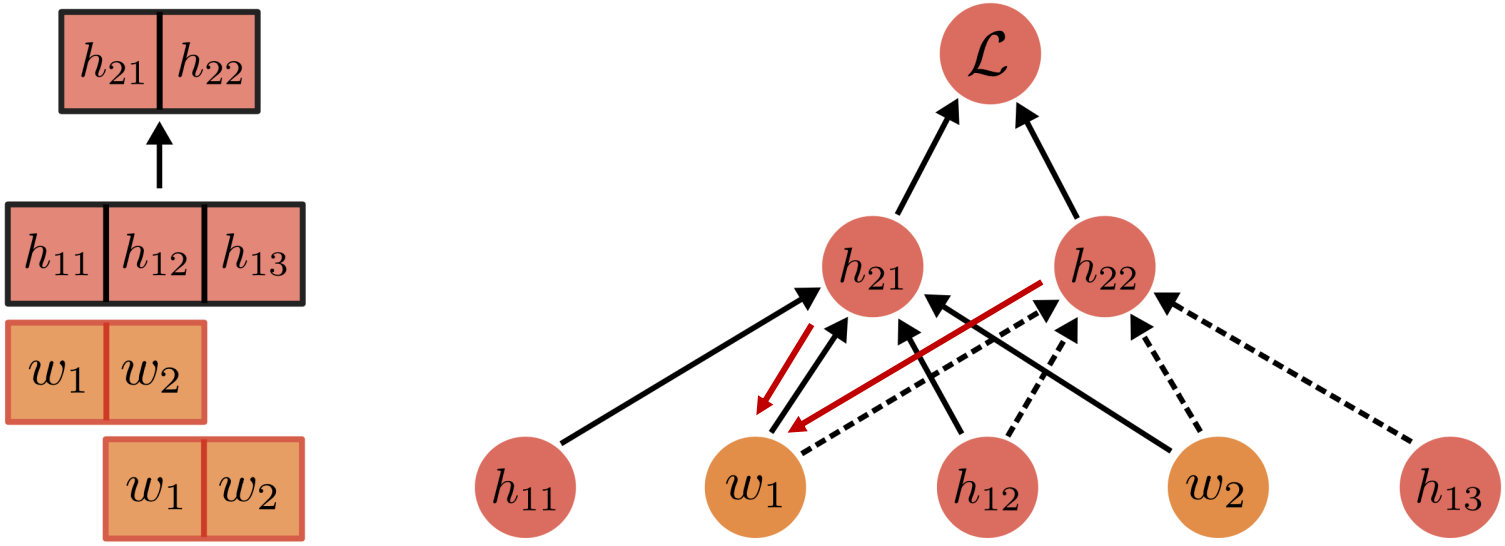


  - In NumPy, this can be achieved as follows (with P the padding size):

```
H_pad = np.zeros(H+2P,W+2P)
H_pad[P:H+P,P:W+P] = H
```

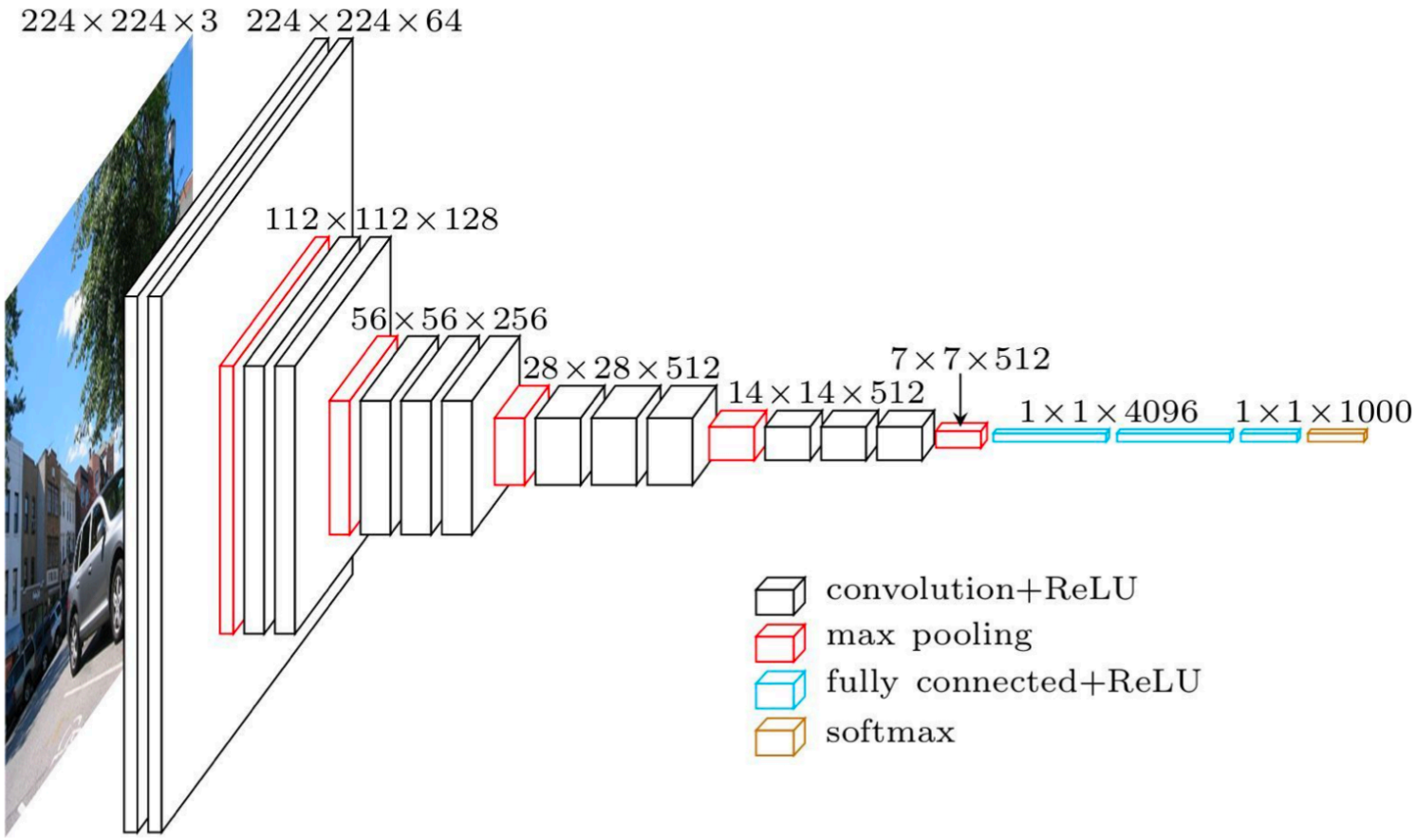
Convolutional layers

- **Backpropagation on the computation graph**
  - Gradients get **accumulated** (summed up) across locations many times to the same weight parameters due to **weight sharing**



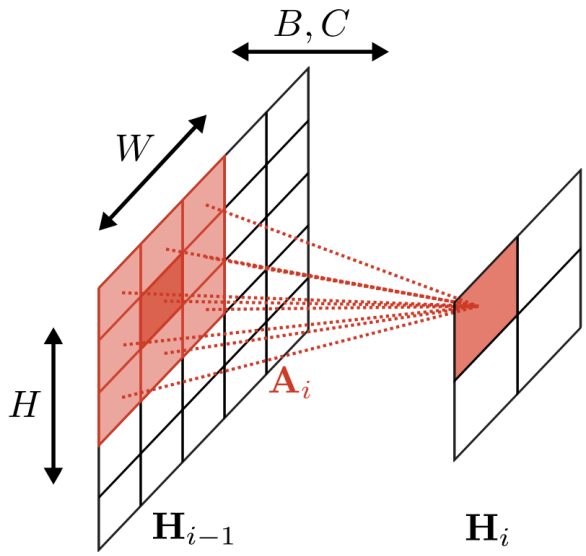
Downsampling

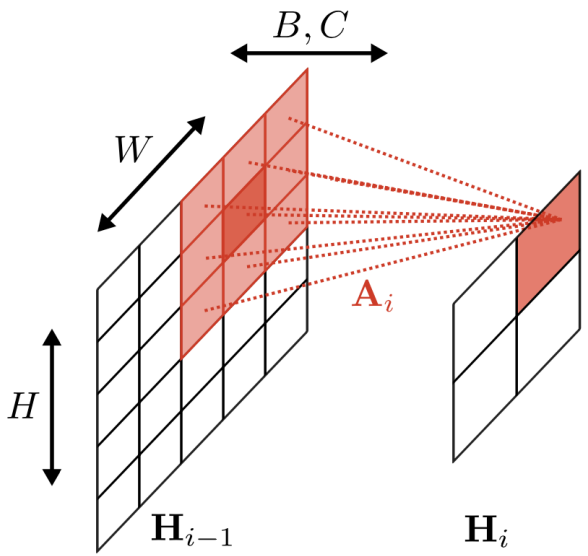
- Downsampling **reduces the spatial resolution** (e.g., for image level predictions)
- Downsampling **increases the receptive field** (which pixels influence a neuron)



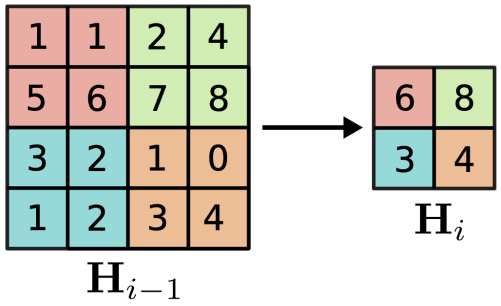
- https://arxiv.org/pdf/1409.1556.pdf

Downsampling: pooling



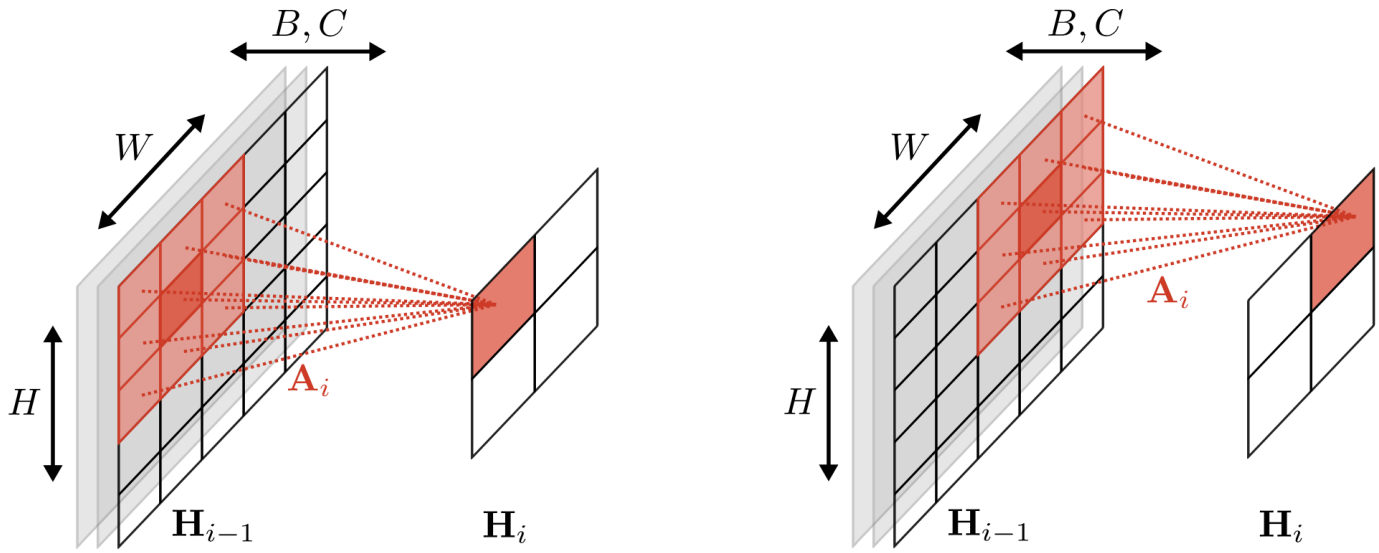


$$\underbrace{\mathbf{H}_i[b, x, y, c]}_{\text{Current Layer}} = \max_{\Delta x, \Delta y} \underbrace{\mathbf{H}_{i-1}[b, s \cdot x + \Delta x, s \cdot y + \Delta y, c]}_{\text{Prev. Layer}}$$



- Typically, **stride** s = 2 and kernel size 2 × 2 ⇒ **reduces spatial dimensions** by 2
- Pooling has **no parameters** (typical pooling operations: max, min, mean)
- Pooling is **applied to each channel separately** ⇒ keeps number of channels
- Remark: Max pooling provides some **invariance** to small translations of the input

Downsampling: strided convolution

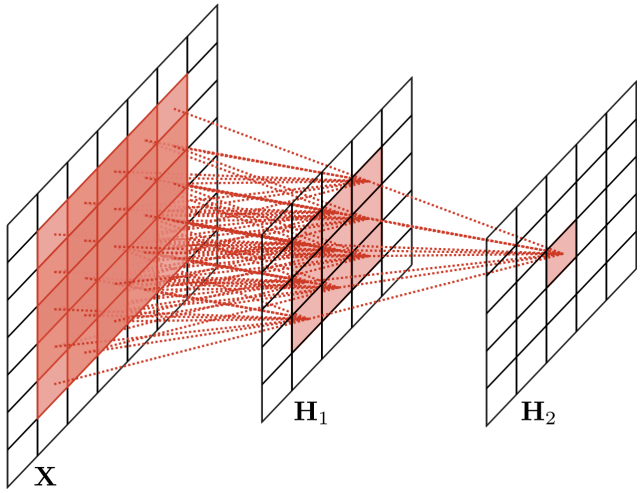


$$\underbrace{\mathbf{H}_i[b, x, y, c_{out}]}_{\text{Current Layer}} = g\left(\underbrace{\mathbf{A}_i[\Delta X, \Delta Y, C_{in}, c_{out}]}_{\text{Weights}} \underbrace{\mathbf{H}_{i-1}[b, s \cdot x + \Delta X, s \cdot y + \Delta Y, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}}\right)$$

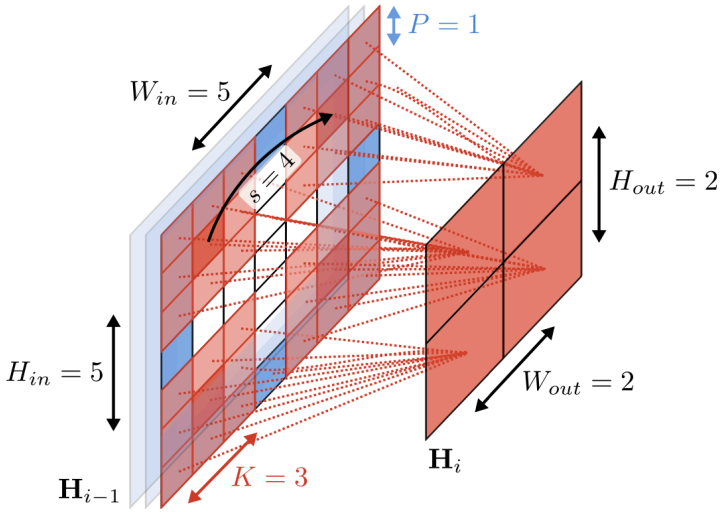
- **Move convolution filter** (with parameters A and b) by stride s
- **Learned downsampling.** Often replaces max pooling today (e.g., in ResNet)

Receptive field arithmetic

- **Receptive field** of a neuron
  - defined as all pixels in the input $\mathbf{X}$ which influence it
  - With pooling the receptive field grows faster



- Size arithmetic
  - **Assuming:**
    - Input: $W_{in} \times H_{in}$
    - Filter: $K \times K$
    - Padding: P
    - Stride: s

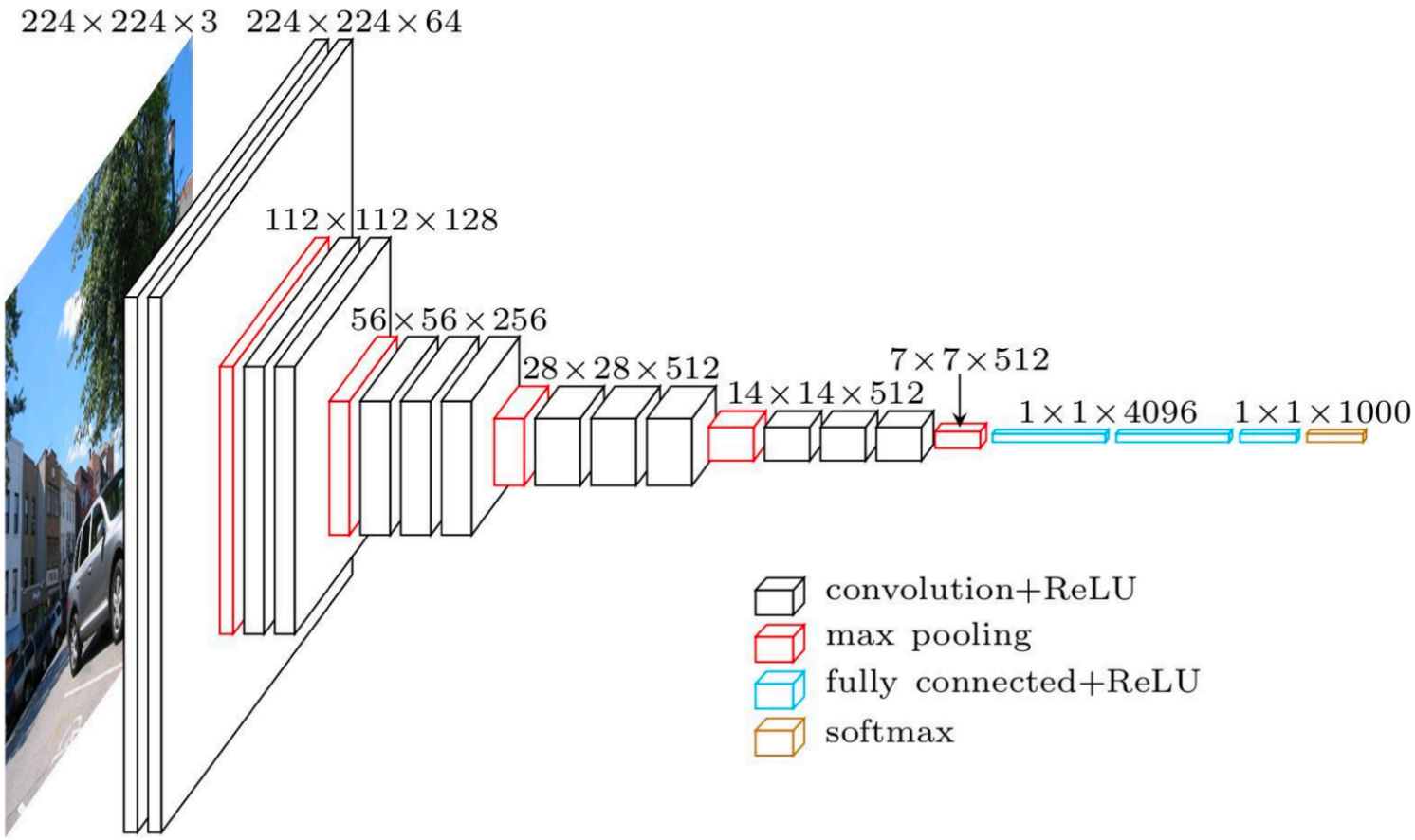


Then, the output has size:

$$\underbrace{\left(\left\lfloor \frac{W_{in}+2P-K}{s} \right\rfloor + 1\right)}_{W_{out}} \times \underbrace{\left(\left\lfloor \frac{H_{in}+2P-K}{s} \right\rfloor + 1\right)}_{H_{out}}$$

Fully connected layers

- Fully connected layers are most **memory intensive** part of VGG architecture

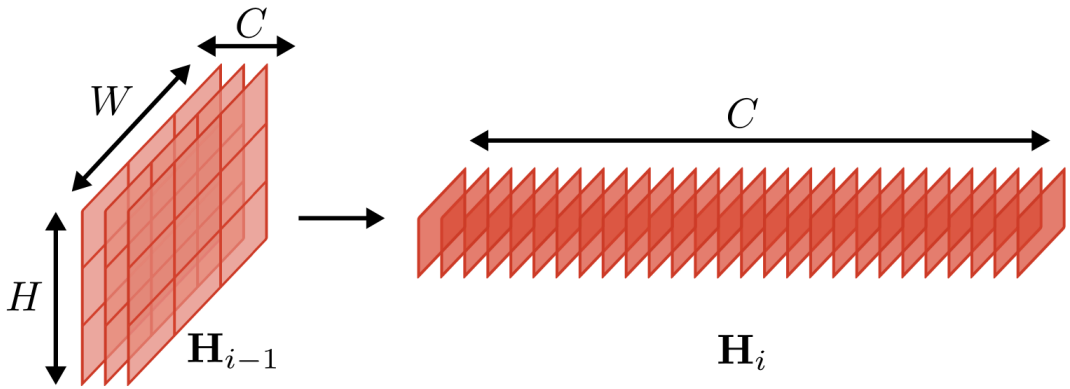


- https://arxiv.org/pdf/1409.1556.pdf

Fully connected layers

- Fully connected layers are most **memory intensive** part of VGG architecture

- Reshape $\mathbf{H}_{i-1}[B, X, Y, C]$ into $\mathbf{H}_i[B, C]$



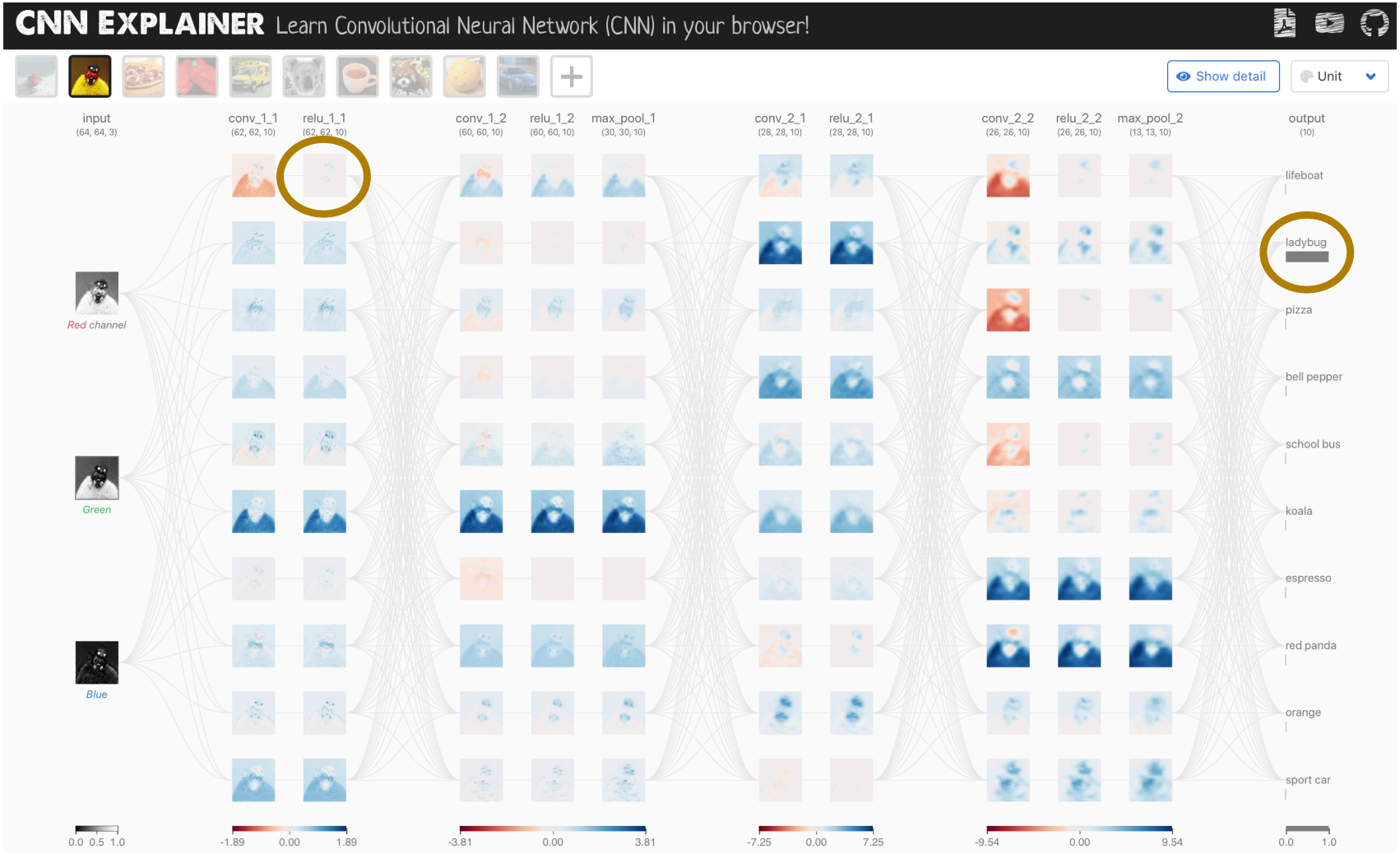
- Now X and Y are **reshaped** into the feature channel dimension C
- A fully connected layer in **Einstein notation** can be written as:

$$\underbrace{\mathbf{H}_i[b, c_{out}]}_{\text{Current Layer}} = g\left(\underbrace{\mathbf{A}_i[c_{out}, C_{in}]}_{\text{Weights}}\ \underbrace{\mathbf{H}_{i-1}[b, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}}\right)$$

Convolutional Network Example

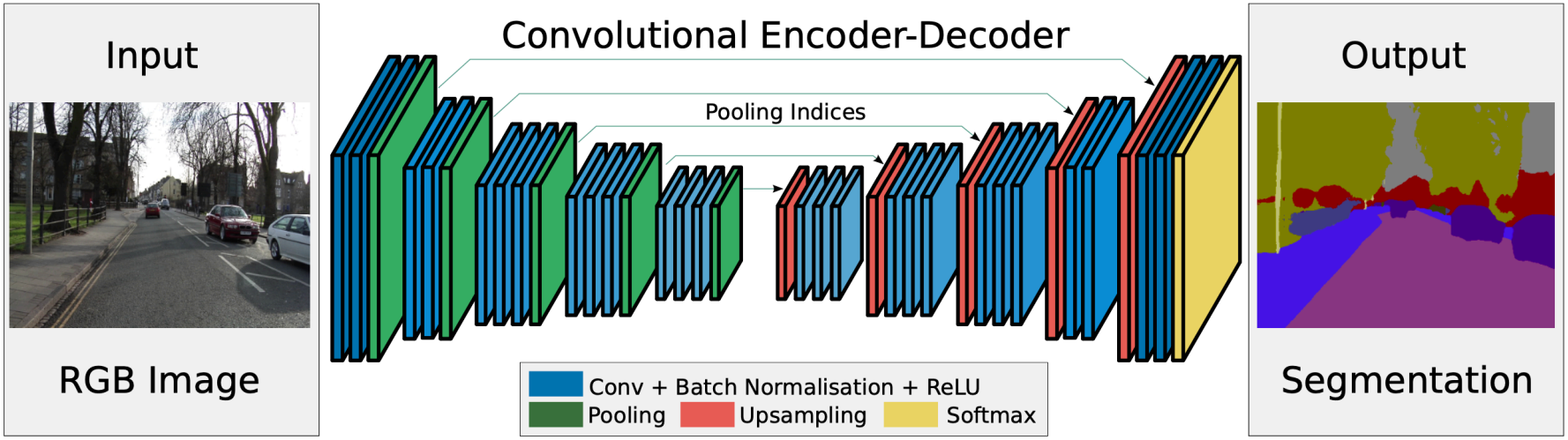
- http://cs231n.stanford.edu/2017/index.html

Convolutional Network Example



- https://poloclub.github.io/cnn-explainer/
- https://arxiv.org/abs/2004.15004

Upsampling



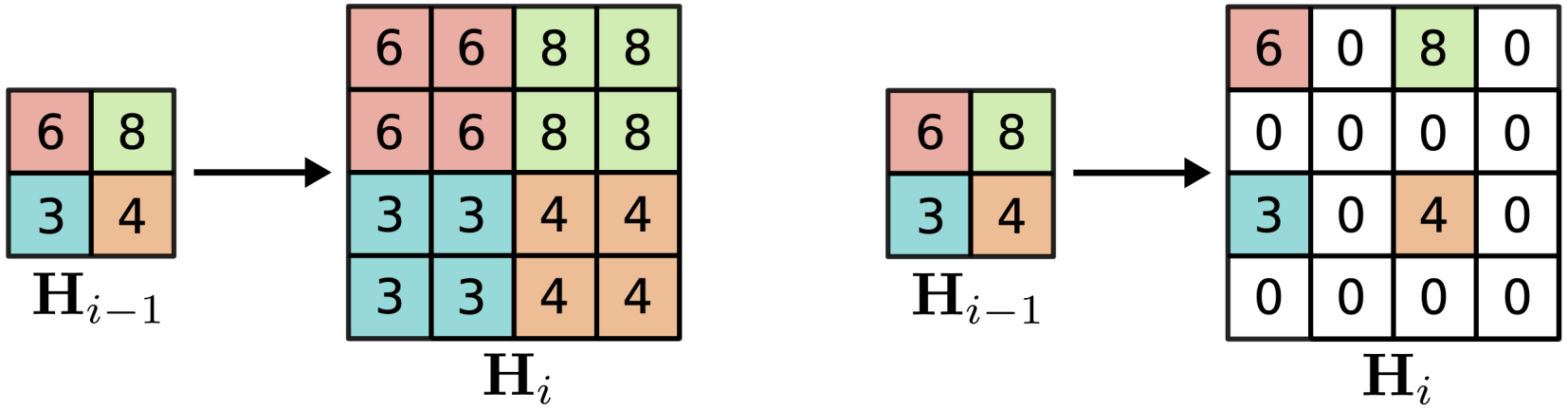
- If **pixel-level outputs** are desired, we have to **upsample** the features again
  - e.g. to obtain depth maps, or motion maps or semantic level map (pixel-wise semantic classification or segmentation)
- Why we need downsampling (coding) first and then upsampling (decoding)?
  - Downsampling provides strong features with large receptive field (quick information aggregation with limited number of network parameters)
  - Upsampling yields output at the same resolution as input
  - Usually skip connections are used, like in ResNet, but longer as we will see in U-Nets
    - Skip connections allow maintaining high level of accuracy related e.g. to object boundaries

Badrinarayanan, Kendall and Cipolla: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEE Trans. PAMI, 2017.

Upsampling

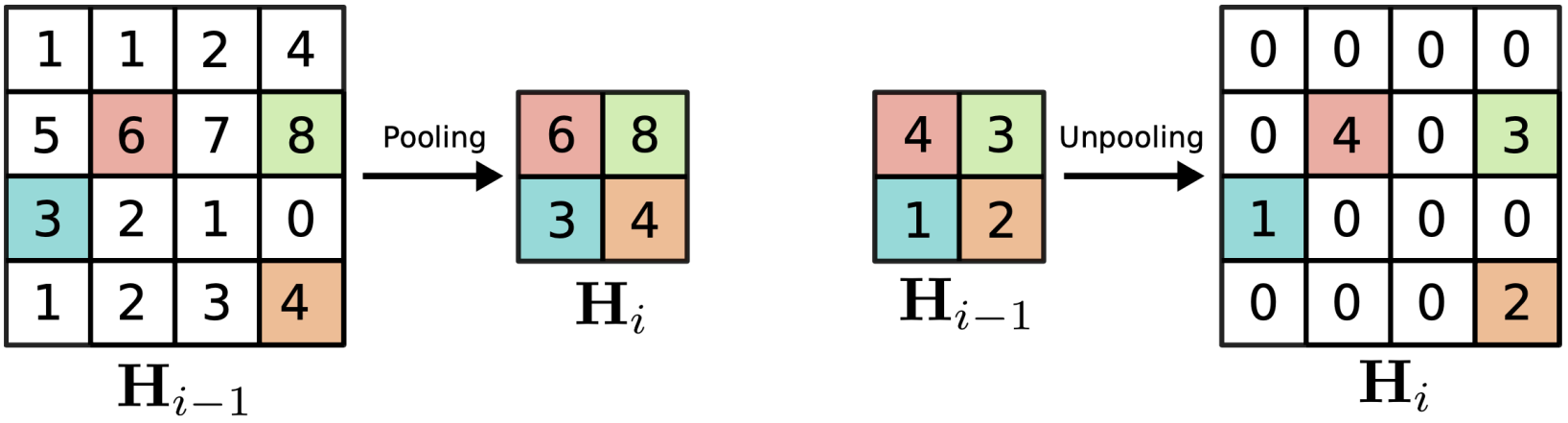
As for downsampling, there are multiple ways to implement Upsampling:

- **Nearest neighbor:** scale each channel using nearest neighbor interpolation
- **Bilinear:** scale each channel using bilinear neighbor interpolation
  - possible smoothing artefacts
- **Bed of nails:** insert elements at sparse location (followed by convolutional layer)
  - possible nail fingerprint artefacts



Badrinarayanan, Kendall and Cipolla: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEE Trans. PAMI, 2017.

Max Unpooling

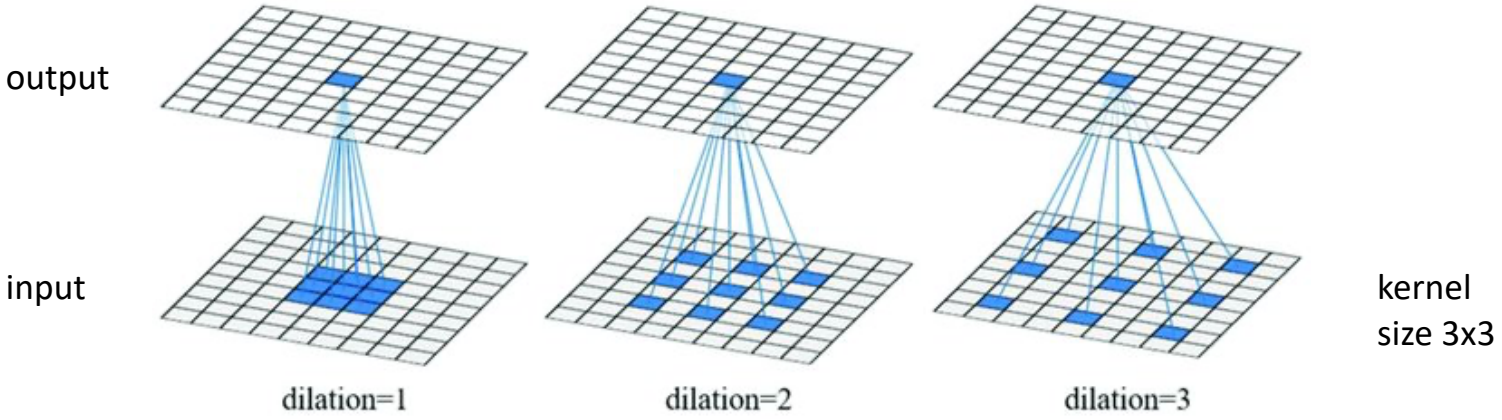


- For unpooling, **remember** which element was **maximum** during pooling
- Always requires corresponding pairs of downsampling and upsampling layers
- This approach has been used in SegNet (a foundational architecture for semantic segmentation)

Badrinarayanan, Kendall and Cipolla: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEE Trans. PAMI, 2017.

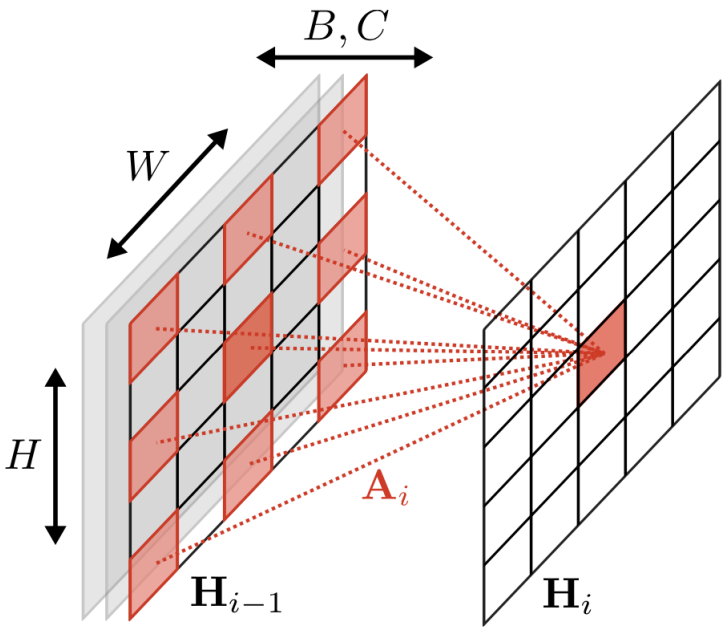
Dilated Convolution

- **Dilated convolutions** are an alternative to combining downsampling and upsampling opterations in order to **reach a large receptive field size quickly**
- Dilated convolutions increase the receptive field of standard convolutions **without increasing the number of parameters**
- A network with dilated convolutions is able to **maintain feature channel dimension** and to perform image-level predictions (e.g., semantic segmentation) without upsampling and downsampling
  - Without dilated convolutions, a very large number of layers of standard convolutions would be required to reach the same receptive field size without using upsampling and downsampling (as in a U-Net)



- **Remark**: Dilated convolutions are not upsampling operations but makes use of upsampled filter kernels

Dilated Convolution



$$\underbrace{H_i[b, x, y, c_{out}]}_{\text{Current Layer}} = g\left(\underbrace{A_i[\Delta X, \Delta Y, C_{in}, c_{out}]}_{\text{Weights}}\ \underbrace{H_{i-1}[b, x + d \cdot \Delta X, y + d \cdot \Delta Y, C_{in}]}_{\text{Prev. Layer}} + \underbrace{b_i[c_{out}]}_{\text{Bias}}\right)$$

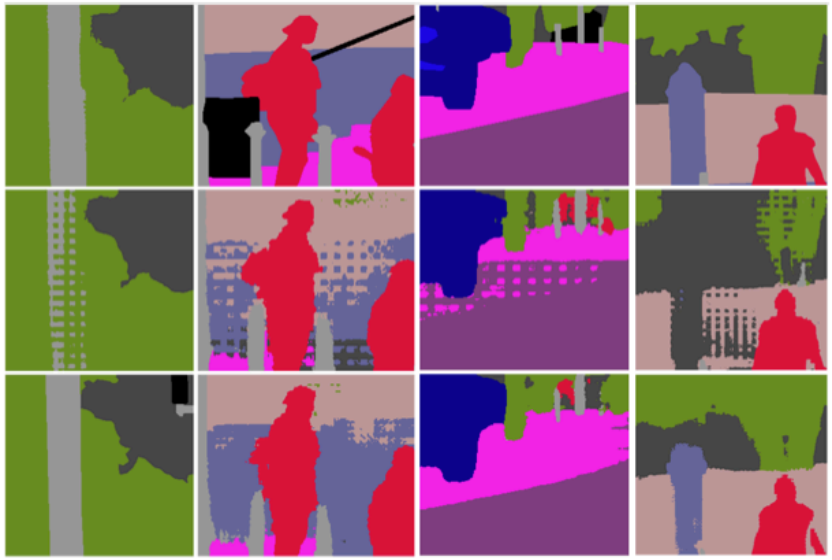
Same #parameters to learn

Higher dispersion over the input
(like to have a larger kernel)

- Dilation factor d (here d = 2) leads to more "distributed" sampling of the input
- **Increases receptive field** without increasing parameters or reducing spatial dimension of the feature channels

Dilated Convolution: gridding effects

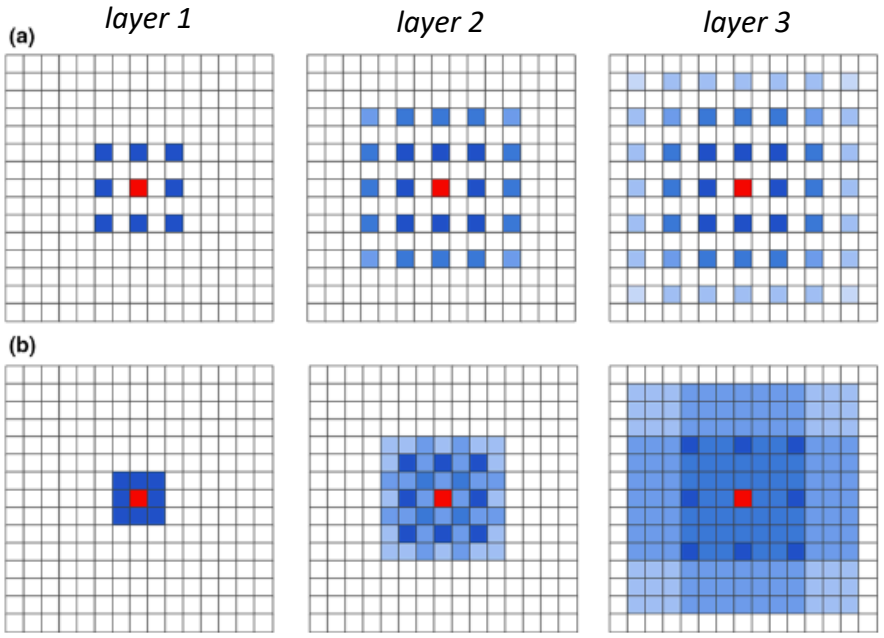
- Careful selection of the dilation factors d for subsequent dilated convolutional layers to avoid «gridding effects»



Ground truth patches

Gridding effects (same dilation factor)

Better results (**progressive dilation** factors)



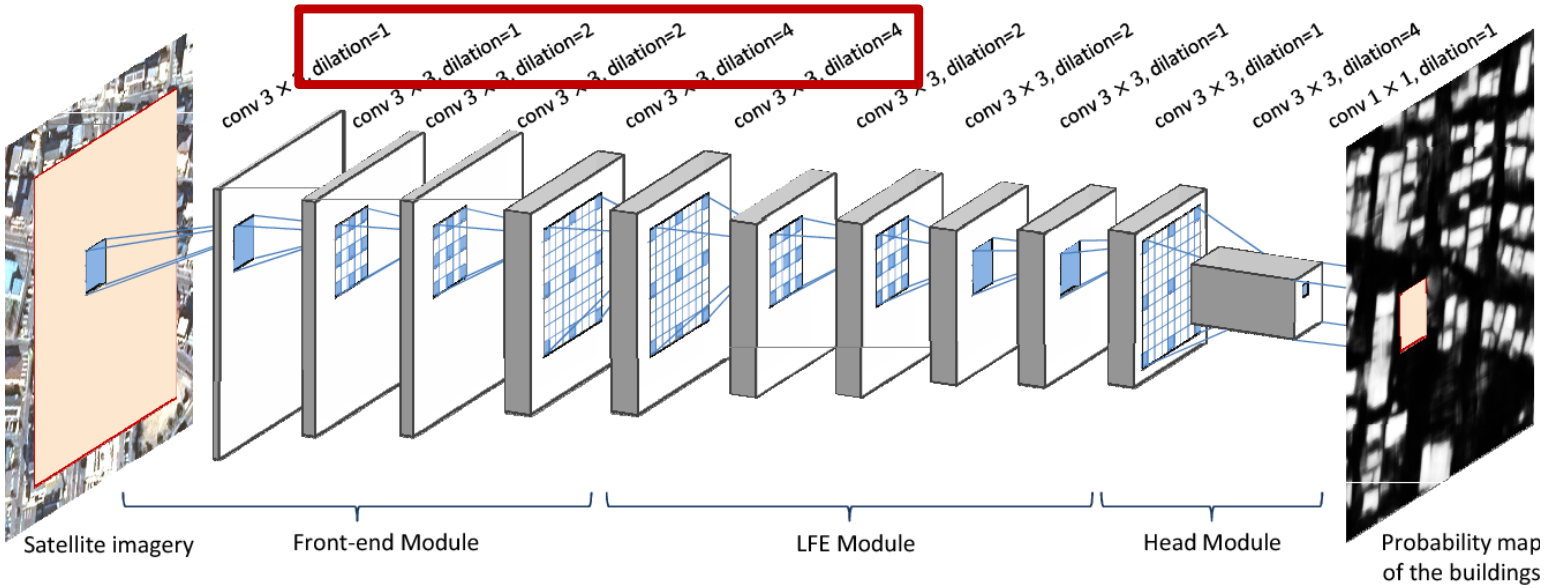
■ output node at layer 1
■ receptive field (nodes that contributes to the computation of ■)
Filter kernel **3x3** for every layers

(a) **same dilation** factor d = 2 for each layer

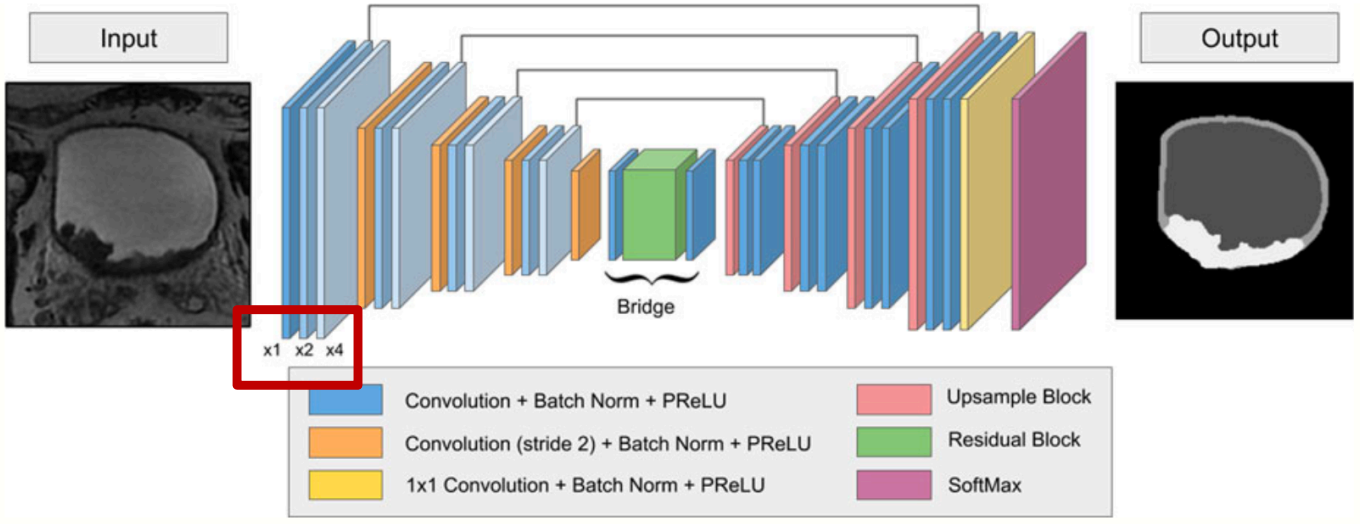
(b) **Progressive dilation** (exponential)

- *layer 1* dilation d = 1 (no dilation)
- *layer 2* dilation d = 2
- *layer 3* dilation d = 4

Dilated Convolution: architectures



Effective Use of Dilated Convolutions for Segmenting Small Object Instances in Remote Sensing Imagery, R. Hamaguchi et al., IEEE Winter Conf. on Applications of Computer Vision, 2018



Multiregion segmentation of bladder cancer structures in MRI with progressive dilated convolutional networks, J.Dolz et al., Medical Physics, 2018

Some remarks about CNNs

- **Is there anything wrong in CNNs?**
  - CNN identify not just one but a family of deep learning architectures implementing some clever inductive biases related to the idea of convolutional layers (learnable shared local weights).
  a. They reach unprecedented high performance on a variety of challenging tasks.
  b. However, they have many drawbacks that made them still very different from brain learning.
  - For the above two reasons, sometimes they are celebrated while other times they are heavily criticized.
    - But, why did we not criticize other classifiers with same emphasis?
    - Maybe because higher potentials lead to higher expectations.

- More technically, in CNNs we still **do not have** **pose invariance response** of the network in object recognition, and this could lead to two attitudes:
  a. "CNN are rubbish" : we can find several videos of Goffrey Hinton saying this, and this is perfectly coherent with the attitude of basic research aimed at discovering new architectures,
  b. CNN does a good job, provided we have enough training samples: this is the **applied research perspective** where we must know and take care of the fact that we must use large datasets to compensate for lack of pose variance, other than for to avoid overfitting

Some remarks about CNNs

- We observe children learning with less samples than CNNs, why this?
  - Human brain can implement **pose and light invariant learning mechanisms** we still do not know how to replicate
  - To obtain pose invariance we need to hack the deep neural network architecture to introduce more advanced inductive biases
    - a tentative in this direction has been made by Hinton and his group, where CNN neurons are organized into “capsules” dedicated to the recognition of a hierarchy of parts and some part-to-the-whole routing is implemented to assemble parts thanks to explicit pose estimation mechanisms and maximization of the agreement and consistency of the process → however clear/effective learning mechanisms have not yet been found
  - Luckily there is room in CNN to compensate for its drawbacks by using enough training samples: way less than in case of fully connected networks but still too much in a viewpoint invariance and part to the whole perspectives
    - Translation equivariance of convolutional layers is (partially) lost with pooling, but pooling is important to go from the local to the global view.
- Many CNN architectural ideas and variants have been proposed that are able to improve on specific aspects (e.g. on vanishing gradient) or on global task performance:
  - learned pooling (strided convolution) is better than max pooling
  - dilated convolutions are interesting even if they are to fit well in the context of the whole architecture (see e.g. https://doi.org/10.1016/j.compbiomed.2022.105891)
  - other solutions: residual (or skip) connections, small kernel sizes, inception modules,…