Data Analysis & Machine Learning: Lecture 10

Dr. Conor D. Rankine

Learning Objectives

At the end of this lecture, you'll be able to answer questions like:

- Why are fully-connected neural networks not equipped to handle image data for object classification/recognition tasks?
- What is a convolutional neural network?
- What are convolution and pooling operations, and how are they applied?
- What are detection and (semantic /instance) segmentation tasks?

Working With Image Data

Greyscale:

- $N \times M$-dimensional arrays $\boldsymbol{X}_i$ of pixel intensities $x^{(p,q)}$

$$\mathbf{X}_i = \begin{bmatrix} x^{(1,1)} & x^{(1,2)} & \dots & x^{(1,M)} \\ x^{(2,1)} & x^{(2,2)} & \dots & x^{(2,M)} \\ \vdots & \vdots & \ddots & \vdots \\ x^{(N,1)} & x^{(N,2)} & \dots & x^{(N,M)} \end{bmatrix}$$

`plt.imread('img.jpg')`

0.81	0.86	0.81	0.83	0.79	0.84	0.85	0.77	0.83	0.88	0.93	0.93
0.82	0.88	0.80	0.86	0.82	0.86	0.74	0.70	0.84	0.86	0.93	0.92
0.79	0.86	0.73	0.80	0.73	0.67	0.54	0.85	0.84	0.91	0.97	0.91
0.76	0.90	0.72	0.53	0.21	0.14	0.65	0.95	0.91	0.81	0.58	0.33
0.81	0.96	0.66	0.11	0.06	0.04	0.63	0.95	0.76	0.26	0.08	0.22
0.83	0.88	0.49	0.05	0.08	0.02	0.26	0.39	0.27	0.17	0.19	0.42
0.79	0.79	0.58	0.15	0.05	0.01	0.00	0.00	0.02	0.10	0.30	0.50
0.72	0.76	0.74	0.51	[illegible]	0.08	0.09	0.19	0.41	0.51	[illegible]	0.56
0.70	0.77	0.77	0.71	0.68	0.65	[illegible]	0.79	0.88	0.81	[illegible]	0.56
0.74	0.76	0.75	0.75	0.81	0.85	0.85	0.92	0.90	0.71	0.44	0.60
0.77	0.77	0.78	0.80	0.81	0.83	0.79	0.90	0.93	0.74	0.29	0.55
0.78	0.84	0.81	0.85	0.82	0.86	0.87	0.95	0.94	0.91	0.41	0.40

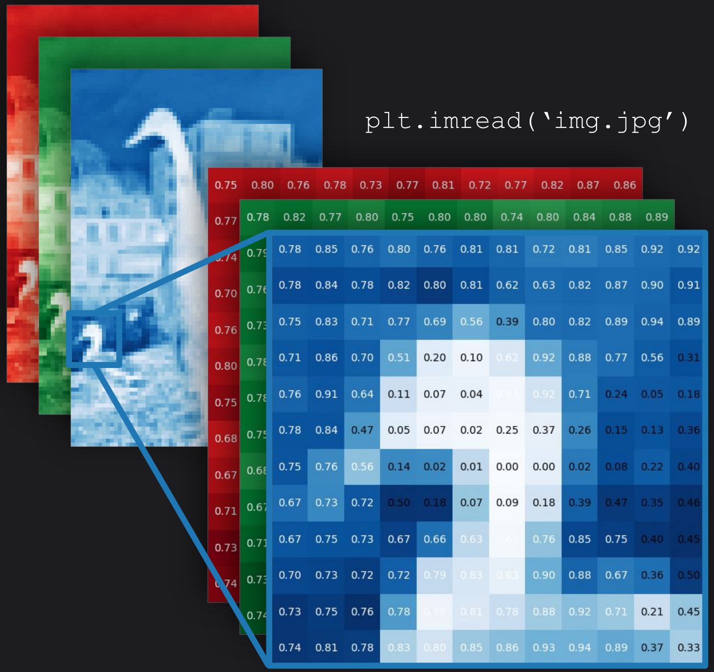
Working With Image Data

Color (RGB):

- $N \times M$-dimensional arrays $\boldsymbol{X}_i$ of red, green, and blue (RGB) pixel intensities $\boldsymbol{x}^{(p,q)}$

$$\mathbf{X}_i = \begin{bmatrix} \mathbf{x}^{(1,1)} & \mathbf{x}^{(1,2)} & \dots & \mathbf{x}^{(1,M)} \\ \mathbf{x}^{(2,1)} & \mathbf{x}^{(2.2)} & \dots & \mathbf{x}^{(2,M)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}^{(N,1)} & \mathbf{x}^{(N,2)} & \dots & \mathbf{x}^{(N,M)} \end{bmatrix}$$

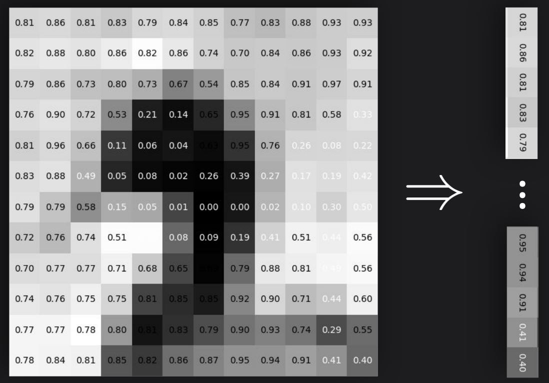
$$\mathbf{x}^{(p,q)} = \left\{ x_R^{(p,q)}, x_G^{(p,q)}, x_B^{(p,q)} \right\}$$



Working With Image Data

Preprocessing:

- fully-connected MLPs do not accept *N*-dimensional (*N*-D; *N*>1) array inputs
- *N*-D array inputs have to be **flattened** to 1-D arrays




```
>>> xi.shape
(128, 128)
>>> xi = xi.flatten()
>>> xi.shape
(16384)
```

$$\mathbf{X}_i = \begin{bmatrix} x^{(1,1)} & \cdots & x^{(1,M)} \\ \vdots & \ddots & \vdots \\ x^{(N,1)} & \cdots & x^{(N,M)} \end{bmatrix} \Rightarrow \left\{ x^{(1,1)}, x^{(1,2)}, \ldots, x^{(N,M)} \right\}$$

Image Data and Neural Networks

A fully-connected MLP is not equipped to handle image data for **object classification**/**recognition**.

Limitations:

- **handcoded features** are brittle and inextensible to new problem domains;
- **parameter scaling** with image/layer dimensions is extreme and unfavourable;
- **spatial information** is destroyed when image data are flattened for input.

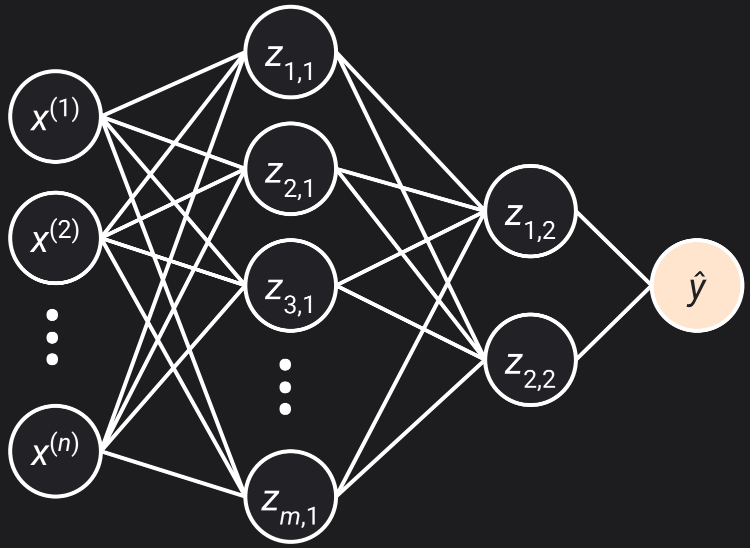
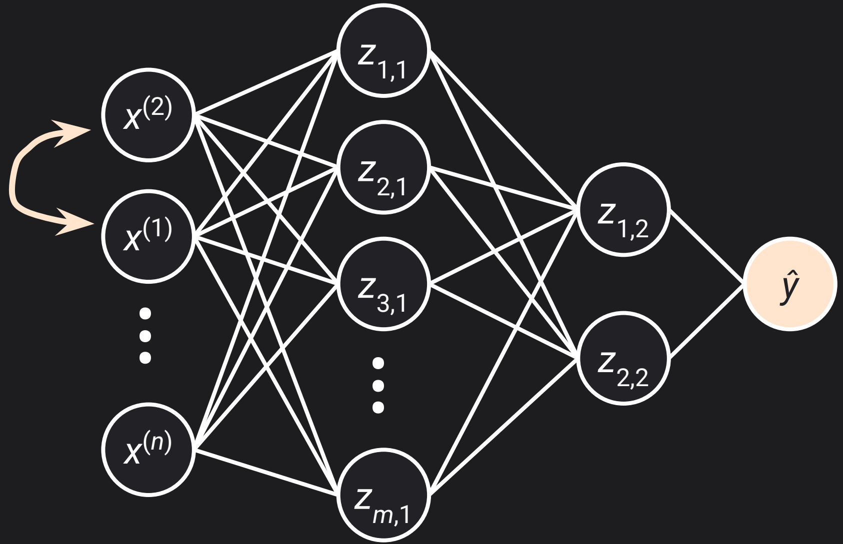


Image Data and Neural Networks

There is no concept of spatial information or correlation in a fully-connected MLP.

Swapping two inputs, $x^{(1)}$ and $x^{(2)}$, is equivalent to swapping two rows in the weight matrix, $\boldsymbol{W}_1$.

$$\mathbf{W}_1 = \begin{bmatrix} w^{(2,1)} & w^{(2,2)} & \cdots & w^{(2,m)} \\ w^{(1,1)} & w^{(1,2)} & \cdots & w^{(1,m)} \\ \vdots & \vdots & \ddots & \vdots \\ w^{(n,1)} & w^{(n,2)} & \cdots & w^{(n,m)} \end{bmatrix}$$

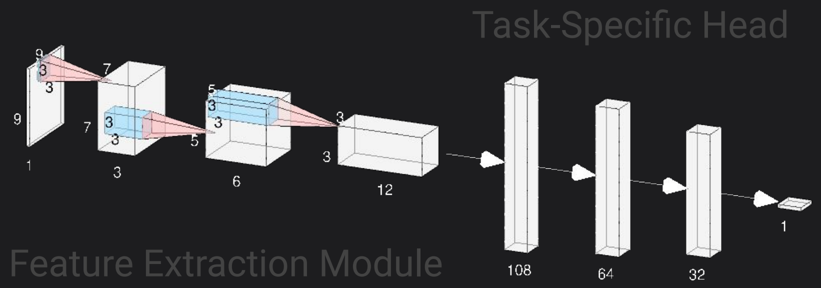


Convolutional Neural Networks

A **convolutional neural network** (**CNN**) is the base architecture of choice for object classification/recognition.

A CNN (usually) comprises a:

- feature extraction module;
    - convolutional layers;
    - pooling/downsampling layers;
- classification/regression head;
    - fully-connected/dense layers.



Convolutional Neural Networks

Recall the limitations of an MLP for tasks in the object classification/recognition domain:

- **handcoded features** are brittle and inextensible to new problem domains;
- **parameter scaling** with image/layer dimensions is extreme and unfavourable;
- **spatial information is destroyed** when image data are flattened for input.

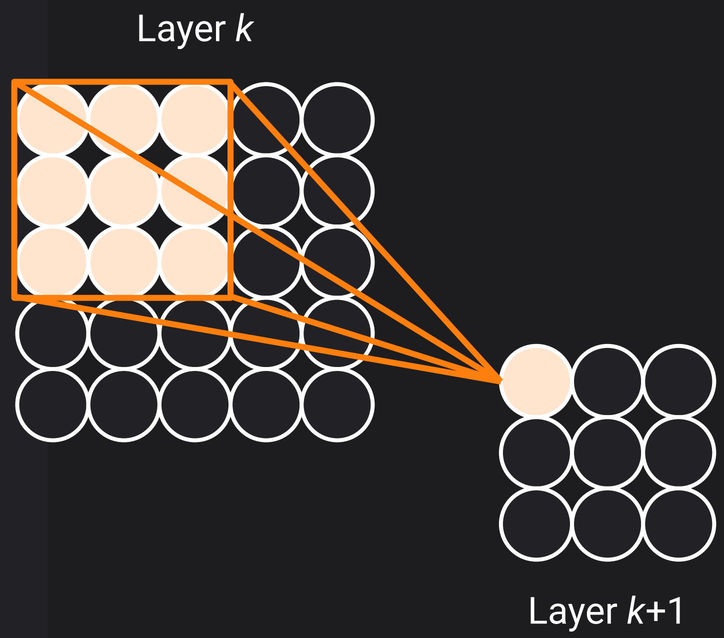
CNNs address these limitations:

- **learned features** are extracted from image data directly without handcoded heuristics;
- layers are not dense/fully connected; parameter scaling is favourable as neurons only have **patched connections**;
- **spatial information is preserved** as convolution can be carried out over arbitrarily-dimensioned (*e.g.* 2D, 3D) arrays.

Convolutional Neural Networks

Patching:

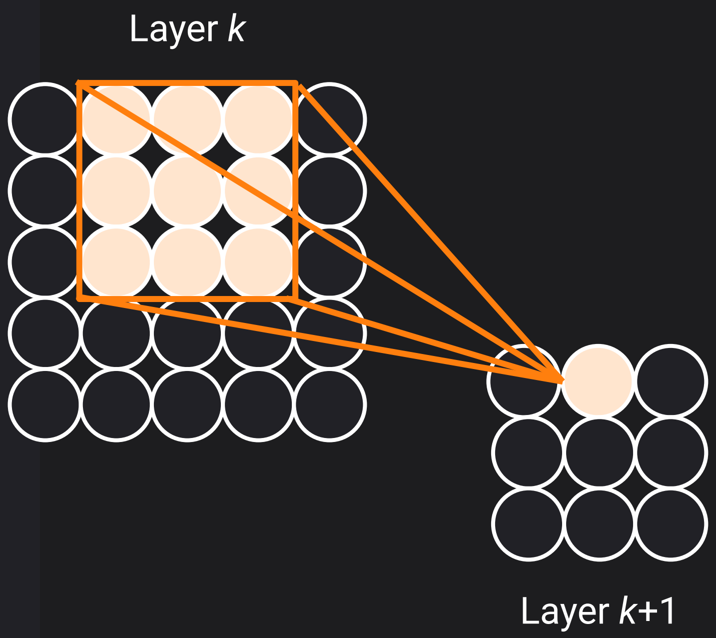
- a neuron in the k+1th layer of the CNN is only connected to a defined **patch** of neurons in the kth layer of the CNN;
- a patch has a defined size (n×m);
    - 3×3 is a common choice;
- the k+1th layer is obtained by **rastering** the patch over the kth layer.



Convolutional Neural Networks

Patching:

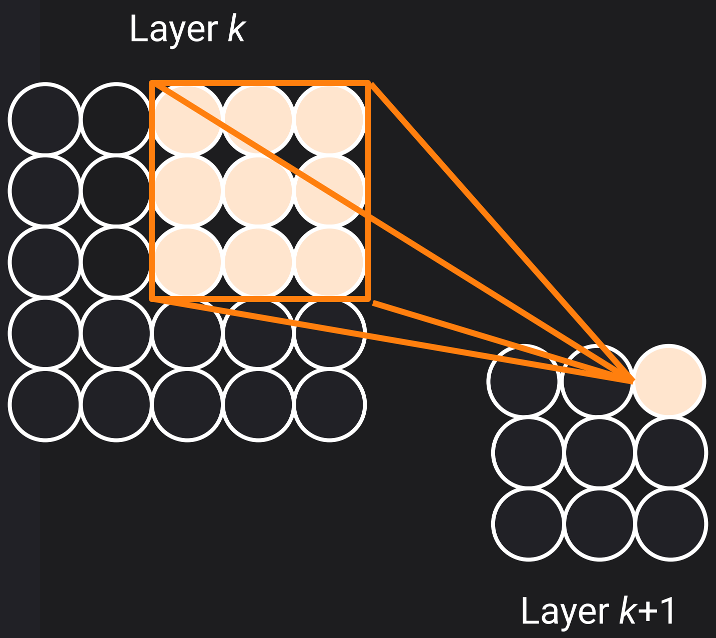
- a neuron in the k+1th layer of the CNN is only connected to a defined **patch** of neurons in the k^{th} layer of the CNN;
- a patch has a defined size ($n \times m$);
  - 3×3 is a common choice;
- the k+1th layer is obtained by **rastering** the patch over the k^{th} layer.



Convolutional Neural Networks

Patching:

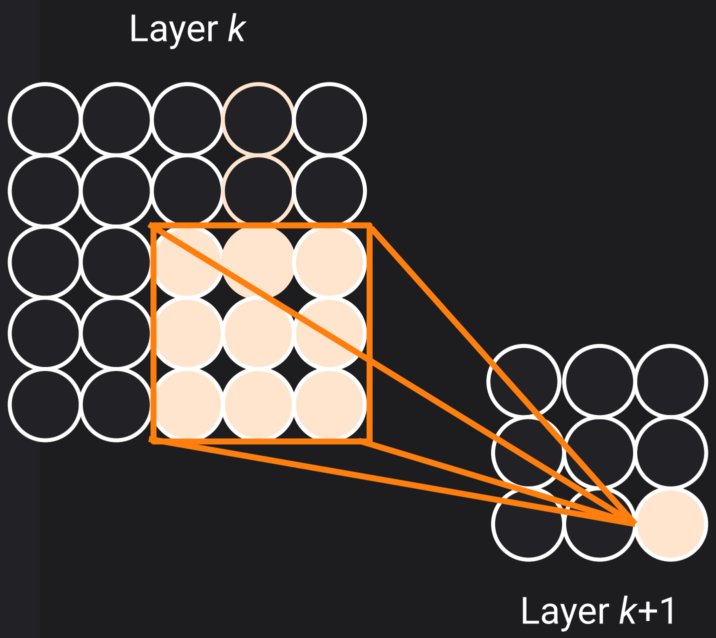
- a neuron in the k+1th layer of the CNN is only connected to a defined **patch** of neurons in the k^{th} layer of the CNN;
- a patch has a defined size (n×m);
  - 3×3 is a common choice;
- the k+1th layer is obtained by **rastering** the patch over the k^{th} layer.



Convolutional Neural Networks

Patching:

- a neuron in the $k+1^{th}$ layer of the CNN is only connected to a defined **patch** of neurons in the k^{th} layer of the CNN;
- a patch has a defined size ($n \times m$);
  - 3×3 is a common choice;
- the $k+1^{th}$ layer is obtained by **rastering** the patch over the k^{th} layer.



Convolutional Neural Networks

An $n \times m$ **filter**, or **convolutional kernel**, is applied to each sliding patch to produce the output for the $k+1^{th}$ layer.

This process is called **convolution**.

Filters can be designed for, *e.g.*:

- **sharpening** images;
- **detecting edges** in images;
- …and many more applications!

Convolutional Neural Networks

Let’s imagine we’re counting votes: can we classify whether or not an image contains an ‘╳’?

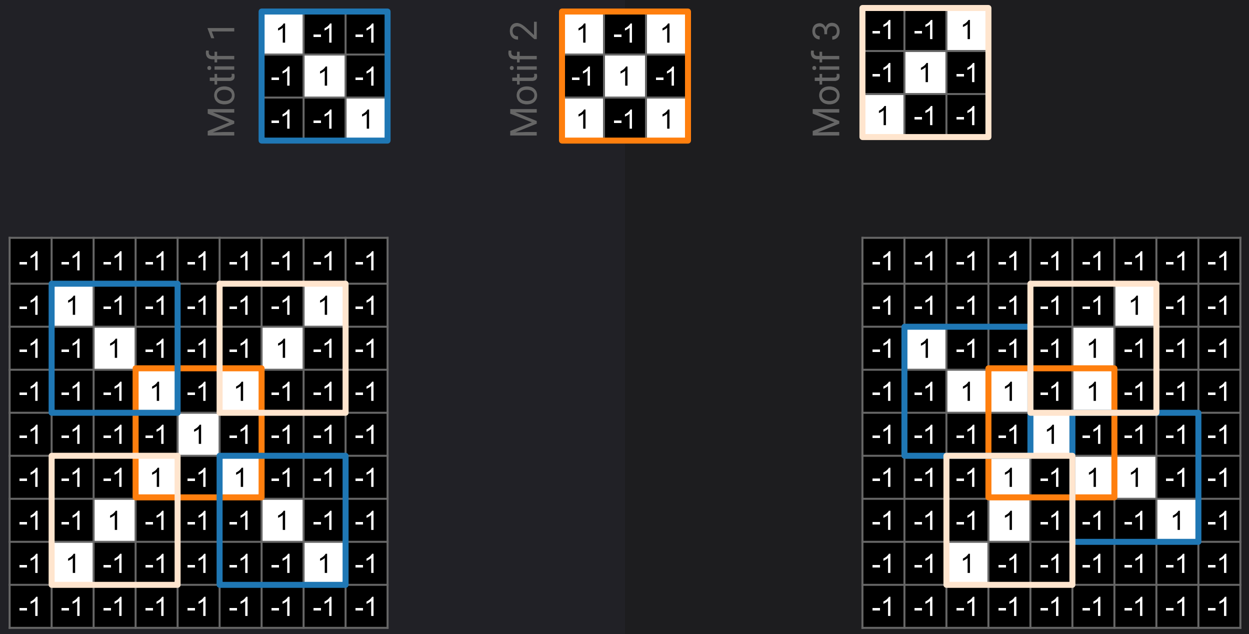
Assumptions:

- the images are binary black (-1) and white (1), *i.e.* not grayscale;
- the ‘╳’ can appear:
  - anywhere (*translation*);
  - in any orientation (*rotation*);
  - in any size; or misshapen.

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

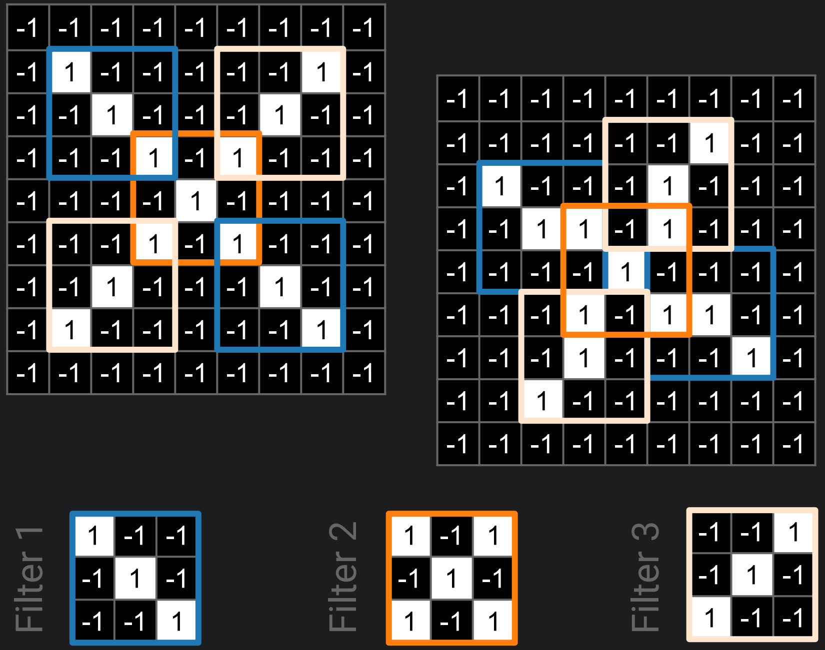
Convolutional Neural Networks



Convolutional Neural Networks

Strategy:

- The common motifs can be used as **detection filters**.
- It is reasonable to assume that these motifs will occur in any image of an '╳'.
- Detection (or absence) of the common motifs in an image is a feature that is informative for classifying that image.



The Convolution Operation

Convolution:

1) slide/raster the filter over the image;

2) compute the element-wise multiplication of the patch and the filter;

3) sum the result.

$$
\begin{bmatrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{bmatrix}
\otimes
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1
\end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
$$

$$
\sum \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix} = 9
$$

The Convolution Operation

Convolution:

1) slide/raster the filter over the image;
2) compute the element-wise multiplication of the patch and the filter;
3) sum the result.

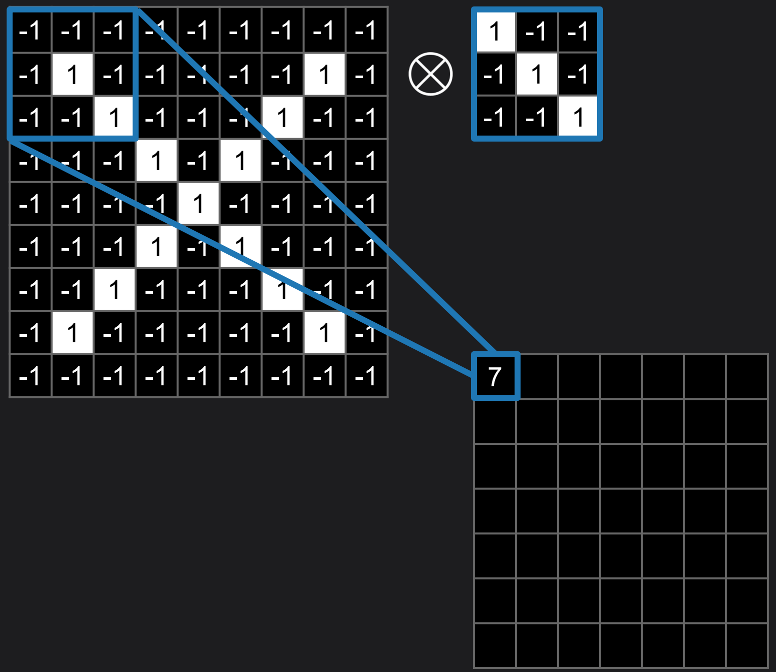
$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 1 \\ -1 & -1 & 1 \\ 1 & -1 & -1 \end{bmatrix}$$

$$\sum \begin{bmatrix} -1 & 1 & 1 \\ -1 & -1 & 1 \\ 1 & -1 & -1 \end{bmatrix} = -1$$

The Convolution Operation

The k+1$^{\text{th}}$ layer, or **feature map**, is constructed *via* convolution.

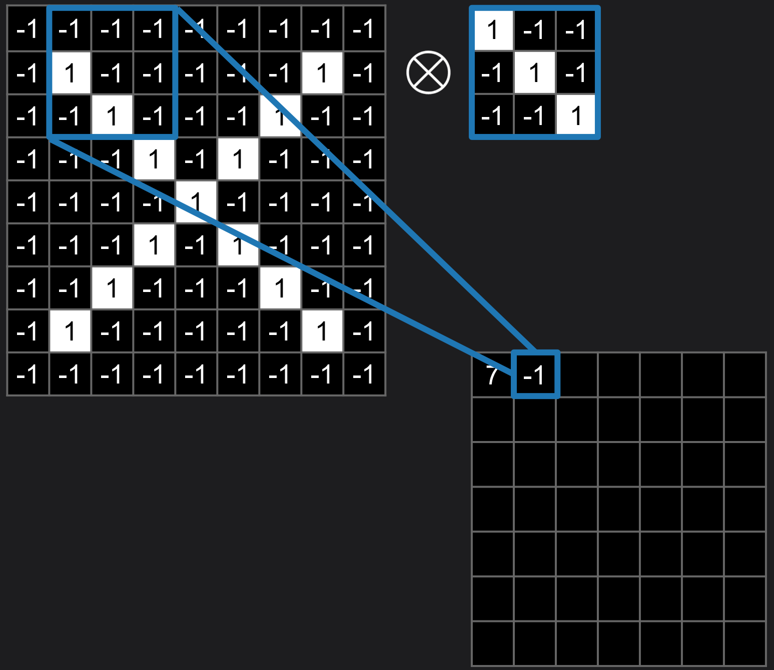
$$z_{k+1}^{(p,q)} = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{F}^{(i,j)} x_k^{(p+i,q+j)}$$



The Convolution Operation

The k+1$^{\text{th}}$ layer, or **feature map**, is constructed *via* convolution.

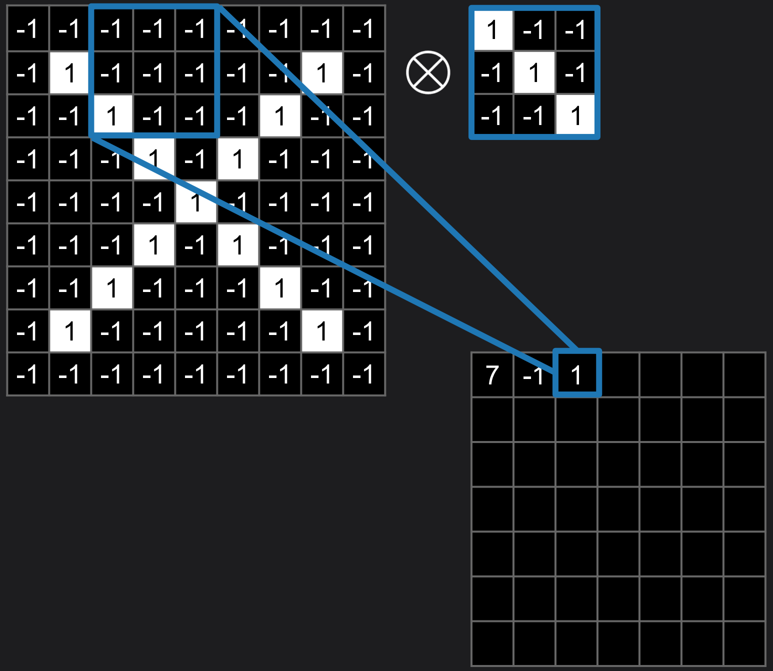
$$z_{k+1}^{(p,q)} = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{F}^{(i,j)} x_k^{(p+i,q+j)}$$



The Convolution Operation

The k+1th layer, or **feature map**, is constructed *via* convolution.

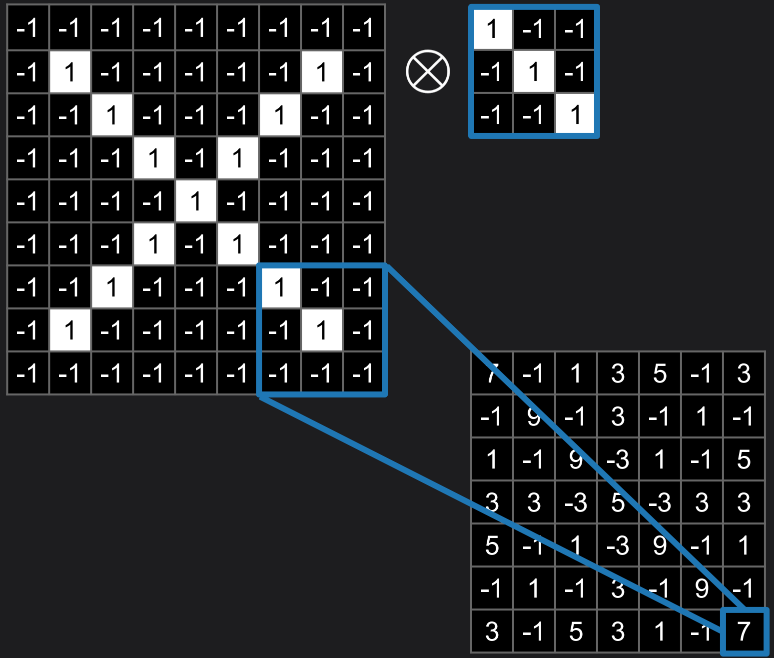
$$z_{k+1}^{(p,q)} = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{F}^{(i,j)} x_k^{(p+i,q+j)}$$



The Convolution Operation

The k+1th layer, or **feature map**, is constructed *via* convolution.

$$z_{k+1}^{(p,q)} = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{F}^{(i,j)} x_k^{(p+i,q+j)}$$



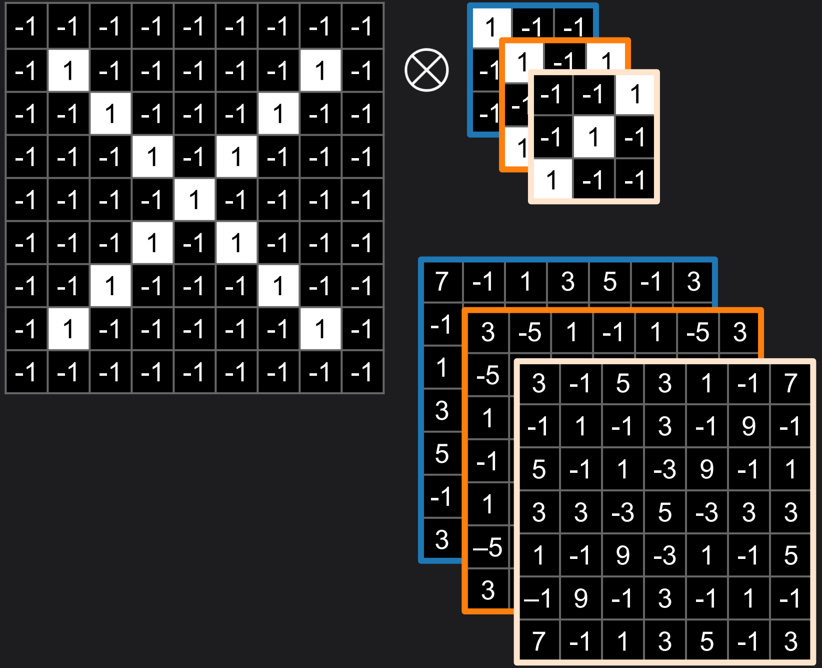
The Convolution Operation

Number of Filters:

- multiple filters are used to extract multiple features (*e.g.* eyes, noses, mouths, *etc.*);
- each filter constructs a feature map.

Stride:

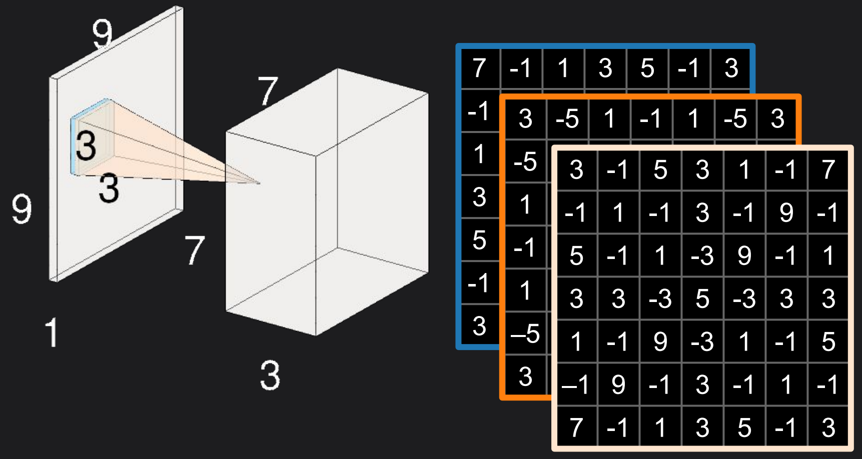
- the ‘*step size*’ with which the filter slides /rasters over the image.



The Convolution Operation

The output of a convolutional layer is described as a **convolutional volume**:

- the **width** and **height** of the convolutional volume are determined by the **stride**;
  - a larger stride gives a smaller width and height;
- the **depth** of the convolutional volume is determined by the **number of filters**;
  - a larger number of filters gives a greater depth.

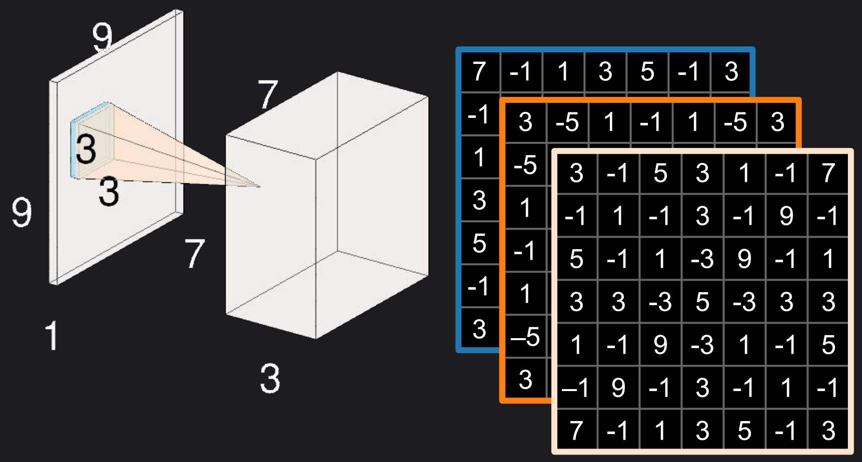


The Convolution Operation

The dimensions (h_{k+1}, w_{k+1}, d_{k+1}) of the *k*+1th layer can be determined from:

- the number of filters, N_F;
- the filter stride, s_F;
- the dimensions, $n \times m$, of the filters.

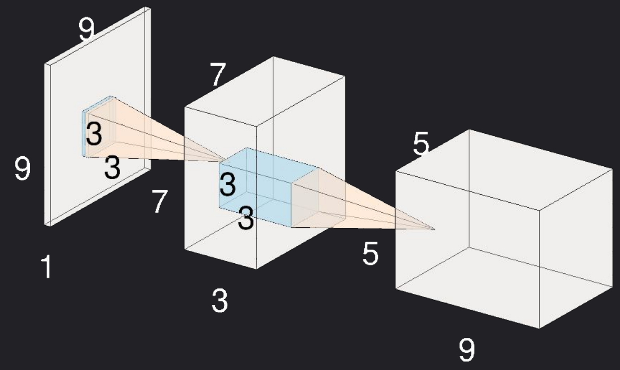
$$h_{k+1} = \frac{(h_k - n)}{s_{\mathbf{F}} + 1} \qquad w_{k+1} = \frac{(w_k - m)}{s_{\mathbf{F}} + 1}$$

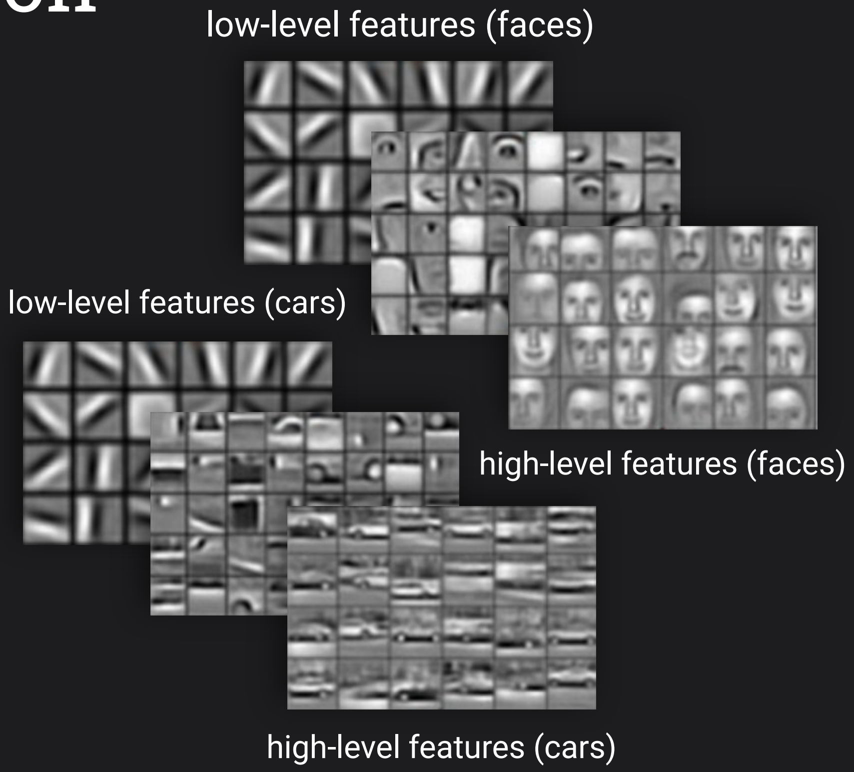


Fractional convolutional volumes are not allowed, *i.e.* h_{k+1}, w_{k+1}, and d_{k+1} have to be integers!

The Convolution Operation

- subsequent convolutional layers operate on the feature maps constructed by previous convolutional layers;
- features are extracted **hierarchically**.





Nonlinear Activation

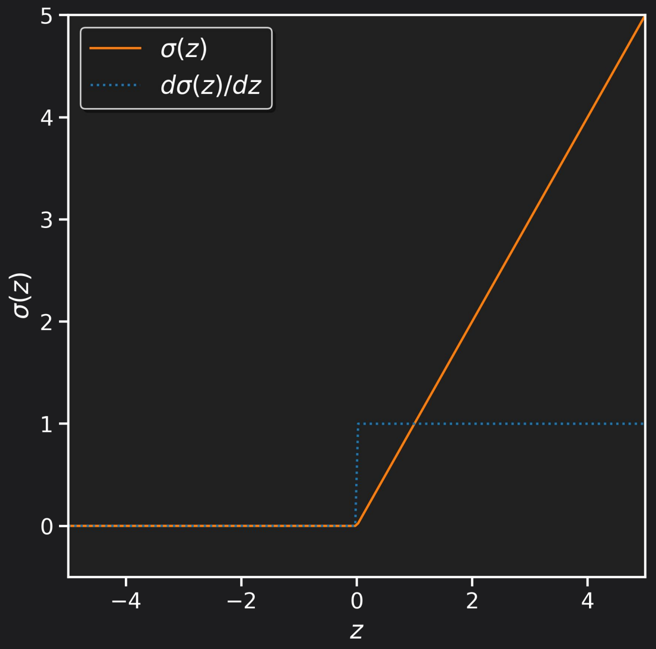
After feature extraction by a convolutional layer, a **nonlinear activation function** is applied to the feature maps.

Popular Choices:

- rectified linear unit (**ReLU**):

$$\sigma(z) = \max(0,\ z)$$

- leaky ReLU
- parametric ReLU



Nonlinear Activation

$$
\begin{bmatrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{bmatrix}
\otimes
\begin{bmatrix}
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & 1
\end{bmatrix}
$$

pre-activation feature map

post-activation feature map

$$
\sigma_{\mathrm{RELU}}\left(
\begin{bmatrix}
3 & -5 & 1 & -1 & 1 & -5 & 3 \\
-5 & 5 & -5 & 3 & -5 & 5 & -5 \\
1 & -5 & 5 & -7 & 5 & -5 & 1 \\
-1 & 3 & -7 & 9 & -7 & 3 & -1 \\
1 & -5 & 5 & -7 & 5 & -5 & 1 \\
-5 & 5 & -5 & 3 & -5 & 5 & -5 \\
3 & -5 & 1 & -1 & 1 & -5 & 3
\end{bmatrix}
\right)
=
\begin{bmatrix}
3 & 0 & 1 & 0 & 1 & 0 & 3 \\
0 & 5 & 0 & 3 & 0 & 5 & 0 \\
1 & 0 & 5 & 0 & 5 & 0 & 1 \\
0 & 3 & 0 & 9 & 0 & 3 & 0 \\
1 & 0 & 5 & 0 & 5 & 0 & 1 \\
0 & 5 & 0 & 3 & 0 & 5 & 0 \\
3 & 0 & 1 & 0 & 1 & 0 & 3
\end{bmatrix}
$$

Learning Convolutional Filters

Wouldn't it be better if we didn't have to design the filters by hand?

We don't!

- filters are **arrays of weights**, ***W***; these are the parameters that the CNN learns during the training process.

$$z_{k+1}^{(p,q)} = w_{k+1}^{(0)} + \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{W}_{k+1}^{(i,j)} x_k^{(p+i,q+j)}$$

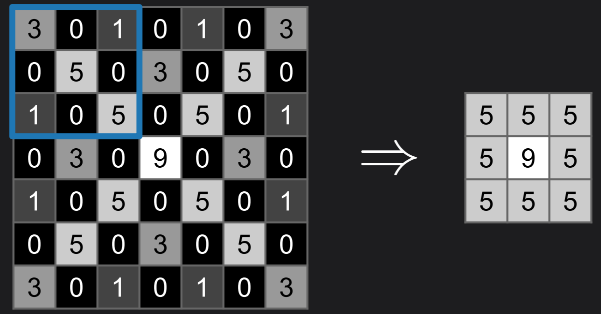
$$\underset{\boldsymbol{X}_i}{\begin{bmatrix} x^{(1,1)} & x^{(1,2)} & x^{(1,3)} & x^{(1,4)} & x^{(1,5)} \\ x^{(2,1)} & x^{(2,2)} & x^{(2,3)} & x^{(2,4)} & x^{(2,5)} \\ x^{(3,1)} & x^{(3,2)} & x^{(3,3)} & x^{(3,4)} & x^{(3,5)} \\ x^{(4,1)} & x^{(4,2)} & x^{(4,3)} & x^{(4,4)} & x^{(4,5)} \\ x^{(5,1)} & x^{(5,2)} & x^{(5,3)} & x^{(5,4)} & x^{(5,5)} \end{bmatrix}} \otimes \underset{\boldsymbol{W}_1}{\begin{bmatrix} w^{(1,1)} & w^{(1,2)} & w^{(1,3)} \\ w^{(2,1)} & w^{(2,2)} & w^{(2,3)} \\ w^{(3,1)} & w^{(3,2)} & w^{(3,3)} \end{bmatrix}} + w^{(0)}$$

The Pooling Operation

Pooling is a **downsampling** operation applied after a convolutional layer.

Purpose:

- reducing the dimensionality of feature maps to lower compute requirements;
- enforcing spatial invariance, *e.g.*, to translation/rotation;
- increasing the receptive field of the filters by pooling the feature maps.



The Pooling Operation

Types:

Slide/raster over the patches of an image and output:

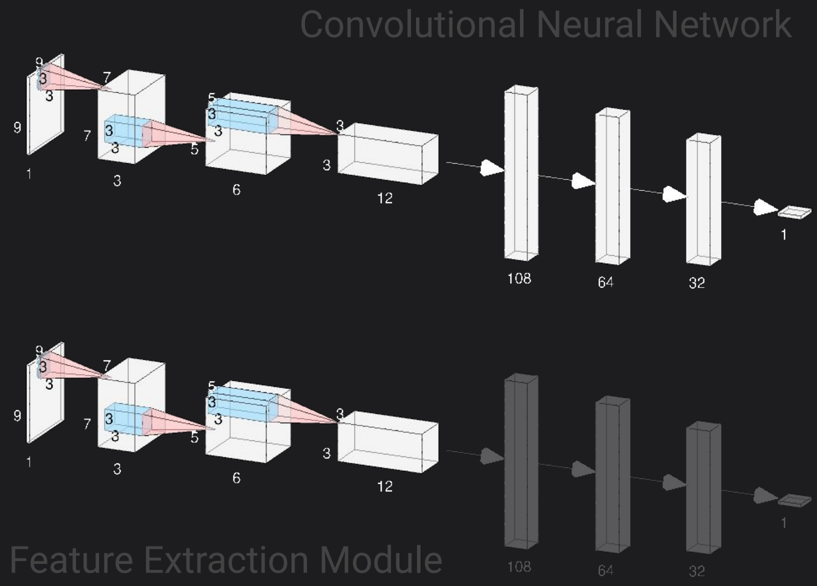
- **max pooling:** the maximum value;
- **average pooling:** the average value;
- **min pooling:** the minimum value.

A common choice is to use a 2×2 or 3×3 pooling patch with a stride of 2 (if compatible).

Convolutional Neural Networks

Feature Extraction Module:

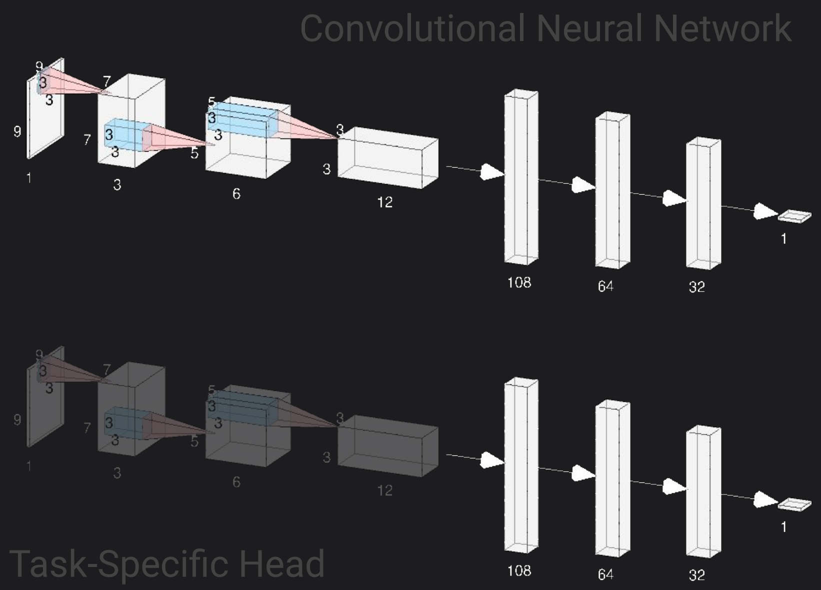
- **comprises:**
  - convolutional layers;
  - pooling/downsampling layers.
- **purpose:**
  - extracting features from image data directly without handcoded heuristics;
  - expressing non-linearity.



Convolutional Neural Networks

Task-Specific Head:

- **comprises:**
    - fully-connected/dense layers;
    - other classifiers or regressors, *e.g.* SVMs, logistic regressors, *etc.*
- **purpose:**
    - classification or regression.



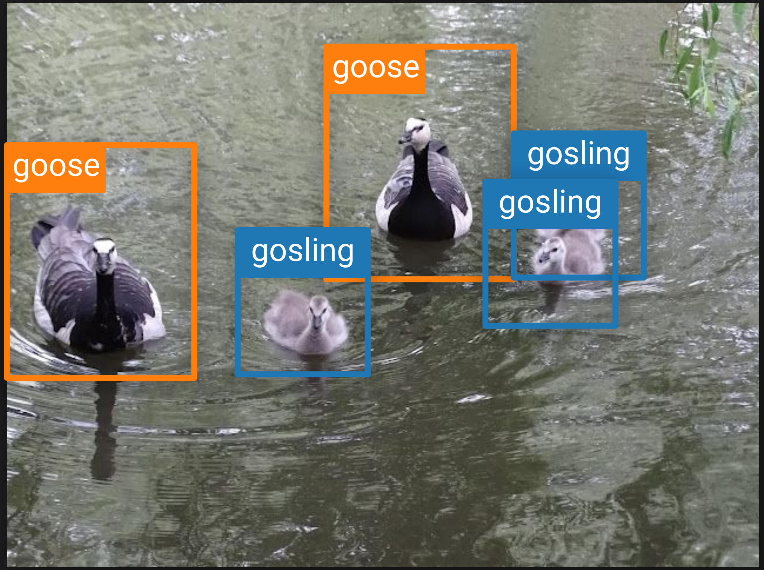
Advanced Tasks: Object Detection

An **object detection** task is a mixed classification-regression problem, since we want to identify:

- **what** the object(s) is/are (classification);
- **where** the object(s) is/are (regression).

Challenges:

- how do we propose the locations of the object(s)/bounding box(es) in images?
  - (fast/faster) R-CNNs



Advanced Tasks: Segmentation

Segmentation is a pixel-level classification task:

- **semantic segmentation:** classify pixels into categories without distinguishing between instances of those categories;
- **instance segmentation:** classify pixels into instances of categories.