OUTLINE

~~I. Introduction~~

~~II. Background~~

~~III. Fitting a Model~~

~~IV. Supervised Learning~~

~~V. Unsupervised Learning~~

VI. Convolutional Neural Networks

OUTLINE

~~I. Introduction~~

~~II. Background~~

~~III. Fitting a Model~~

~~IV. Supervised Learning~~

~~V. Unsupervised Learning~~

VI. Convolutional Neural Networks

- Definition
- Why ConvNets ?
- CNN layers
- Famous examples

Convolutional Neural Networks

- The idea of using such networks to mimic the **human visual system** dates back to K. Fukushima (1980). Y. Lecun was the first to train a CNN using backpropagation (1989)

Convolutional Neural Networks

- The idea of using such networks to mimic the **human visual system** dates back to K. Fukushima (1980). Y. Lecun was the first to train a CNN using backpropagation (1989)
- **Def:** a neural network that uses a linear operation called ***convolution*** in **at least one layer**, instead of a generic linear layer

Convolutional Neural Networks

- The idea of using such networks to mimic the **human visual system** dates back to K. Fukushima (1980). Y. Lecun was the first to train a CNN using backpropagation (1989)
- **Def:** a neural network that uses a linear operation called ***convolution*** in **at least one layer**, instead of a generic linear layer
- This amounts to **constraining the weight matrix** $\mathbf{W}$ to have a special structure called ***Toeplitz***

Convolutional Neural Networks

- The idea of using such networks to mimic the **human visual system** dates back to K. Fukushima (1980). Y. Lecun was the first to train a CNN using backpropagation (1989)
- **Def:** a neural network that uses a linear operation called ***convolution*** in **at least one layer**, instead of a generic linear layer
- This amounts to **constraining the weight matrix $\mathbf{W}$** to have a special structure called ***Toeplitz***

$$\mathbf{W} = \begin{bmatrix} v_0{=}u_0 & u_1 & u_2 & \cdots & u_{N-2} & u_{N-1} \\ v_1 & u_0 & u_1 & \ddots & u_{N-3} & u_{N-2} \\ v_2 & v_1 & u_0 & u_1 & \ddots & u_{N-3} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} & \vdots \\ v_{D-1} & v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} \end{bmatrix} = \text{Toeplitz}(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^{D \times N}$$

Convolutional Neural Networks

- The idea of using such networks to mimic the **human visual system** dates back to K. Fukushima (1980). Y. Lecun was the first to train a CNN using backpropagation (1989)
- **Def:** a neural network that uses a linear operation called ***convolution*** in **at least one layer**, instead of a generic linear layer
- This amounts to **constraining the weight matrix** $\mathbf{W}$ to have a special structure called ***Toeplitz***

$$\mathbf{W} = \begin{bmatrix} v_0{=}u_0 & u_1 & u_2 & \cdots & u_{N-2} & u_{N-1} \\ v_1 & u_0 & u_1 & \ddots & u_{N-3} & u_{N-2} \\ v_2 & v_1 & u_0 & u_1 & \ddots & u_{N-3} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} & \vdots \\ v_{D-1} & v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} \end{bmatrix} = \text{Toeplitz}(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^{D \times N}$$

- Typically, $\boldsymbol{u}$ and $\boldsymbol{v}$ only have only **a few nonzero values**, determined by the ***kernel size***

Convolutional Neural Networks

- The idea of using such networks to mimic the **human visual system** dates back to K. Fukushima (1980). Y. Lecun was the first to train a CNN using backpropagation (1989)
- **Def:** a neural network that uses a linear operation called ***convolution*** in **at least one layer**, instead of a generic linear layer
- This amounts to **constraining the weight matrix** $\mathbf{W}$ to have a special structure called ***Toeplitz***

$$\mathbf{W} = \begin{bmatrix} v_0{=}u_0 & u_1 & u_2 & \cdots & u_{N-2} & u_{N-1} \\ v_1 & u_0 & u_1 & \ddots & u_{N-3} & u_{N-2} \\ v_2 & v_1 & u_0 & \ddots & \ddots & u_{N-3} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} & \vdots \\ v_{D-1} & v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} \end{bmatrix} = \text{Toeplitz}(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^{D \times N}$$

- Typically, $\boldsymbol{u}$ and $\boldsymbol{v}$ only have only **a few nonzero values**, determined by the ***kernel size***
- In terms of **model fitting**, everything we saw so far remains valid!

Convolutional Neural Networks

- The idea of using such networks to mimic the **human visual system** dates back to K. Fukushima (1980). Y. Lecun was the first to train a CNN using backpropagation (1989)
- **Def:** a neural network that uses a linear operation called ***convolution*** in **at least one layer**, instead of a generic linear layer
- This amounts to **constraining the weight matrix** $\mathbf{W}$ to have a special structure called ***Toeplitz***

$$\mathbf{W} = \begin{bmatrix} v_0{=}u_0 & u_1 & u_2 & \cdots & u_{N-2} & u_{N-1} \\ v_1 & u_0 & u_1 & \ddots & u_{N-3} & u_{N-2} \\ v_2 & v_1 & u_0 & u_1 & \ddots & u_{N-3} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} & \vdots \\ v_{D-1} & v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} \end{bmatrix} = \text{Toeplitz}(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^{D \times N}$$

- Typically, $\boldsymbol{u}$ and $\boldsymbol{v}$ only have only **a few nonzero values**, determined by the ***kernel size***
- In terms of **model fitting**, everything we saw so far remains valid!

But what ***is*** (discrete) convolution?

Discrete Convolution (1D Case)

Discrete Convolution (1D Case)

- Given two sequences $x[n]$ and $k[n]$, the convolved sequence $a[n]$ is defined by the following linear operation:

$$a[n] = \sum_{\tau=-\infty}^{+\infty} x[\tau]k[n-\tau], \quad \text{denoted by} \quad \boldsymbol{a} = \boldsymbol{x} * \boldsymbol{k}$$

Discrete Convolution (1D Case)

- Given two sequences $x[n]$ and $k[n]$, the convolved sequence $a[n]$ is defined by the following linear operation:

$$a[n] = \sum_{\tau=-\infty}^{+\infty} x[\tau]k[n-\tau], \quad \text{denoted by} \quad \boldsymbol{a} = \boldsymbol{x} * \boldsymbol{k}$$

- In practice, in CNNs, $\boldsymbol{k}$ only has a **finite support**: the sum is finite

Discrete Convolution (1D Case)

- Given two sequences $x[n]$ and $k[n]$, the convolved sequence $a[n]$ is defined by the following linear operation:

$$a[n] = \sum_{\tau=-\infty}^{+\infty} x[\tau]k[n-\tau], \quad \text{denoted by} \quad \boldsymbol{a = x * k}$$

- In practice, in CNNs, $\boldsymbol{k}$ only has a **finite support**: the sum is finite
- Terminology:
  - $\boldsymbol{x}$: **input** (or signal)
  - $\boldsymbol{k}$: convolution **kernel** (or filter)
  - $\boldsymbol{a}$: **feature map** (analog to pre-activation)

Discrete Convolution (1D Case)

- Given two sequences $x[n]$ and $k[n]$, the convolved sequence $a[n]$ is defined by the following linear operation:

$$a[n] = \sum_{\tau=-\infty}^{+\infty} x[\tau]k[n-\tau], \quad \text{denoted by} \quad \boldsymbol{a = x * k}$$

- In practice, in CNNs, $\boldsymbol{k}$ only has a **finite support**: the sum is finite
- Terminology:
  - $\boldsymbol{x}$: **input** (or signal)
  - $\boldsymbol{k}$: convolution **kernel** (or filter)
  - $\boldsymbol{a}$: **feature map** (analog to pre-activation)

- 1D convolutions precisely describe ***Linear Time Invariant systems***

Discrete Convolution (1D Case)

- Given two sequences $x[n]$ and $k[n]$, the convolved sequence $a[n]$ is defined by the following linear operation:

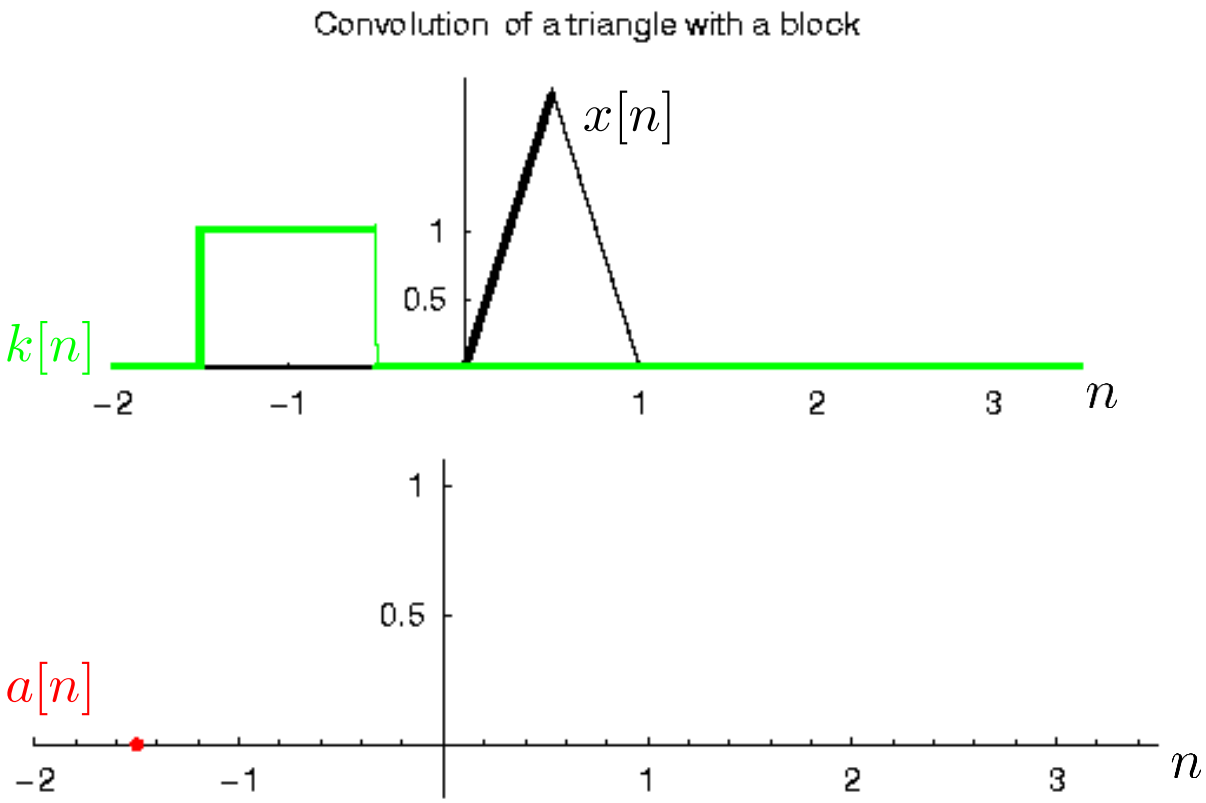
$$a[n] = \sum_{\tau=-\infty}^{+\infty} x[\tau]k[n-\tau], \quad \text{denoted by} \quad \boldsymbol{a = x * k}$$

- In practice, in CNNs, $\boldsymbol{k}$ only has a **finite support**: the sum is finite
- Terminology:
  - $\boldsymbol{x}$: **input** (or signal)
  - $\boldsymbol{k}$: convolution **kernel** (or filter)
  - $\boldsymbol{a}$: **feature map** (analog to pre-activation)

- 1D convolutions precisely describe ***Linear Time Invariant systems***
- **Ex:** moving average, smoothing, low-pass, band-pass, etc.

Discrete Convolution (1D Case)

$$a[n] = \sum_{\tau=-\infty}^{+\infty} x[\tau]k[n-\tau]$$



Discrete Convolution (2D Case)

- Convolution can be generalized to **any** dimension

Discrete Convolution (2D Case)

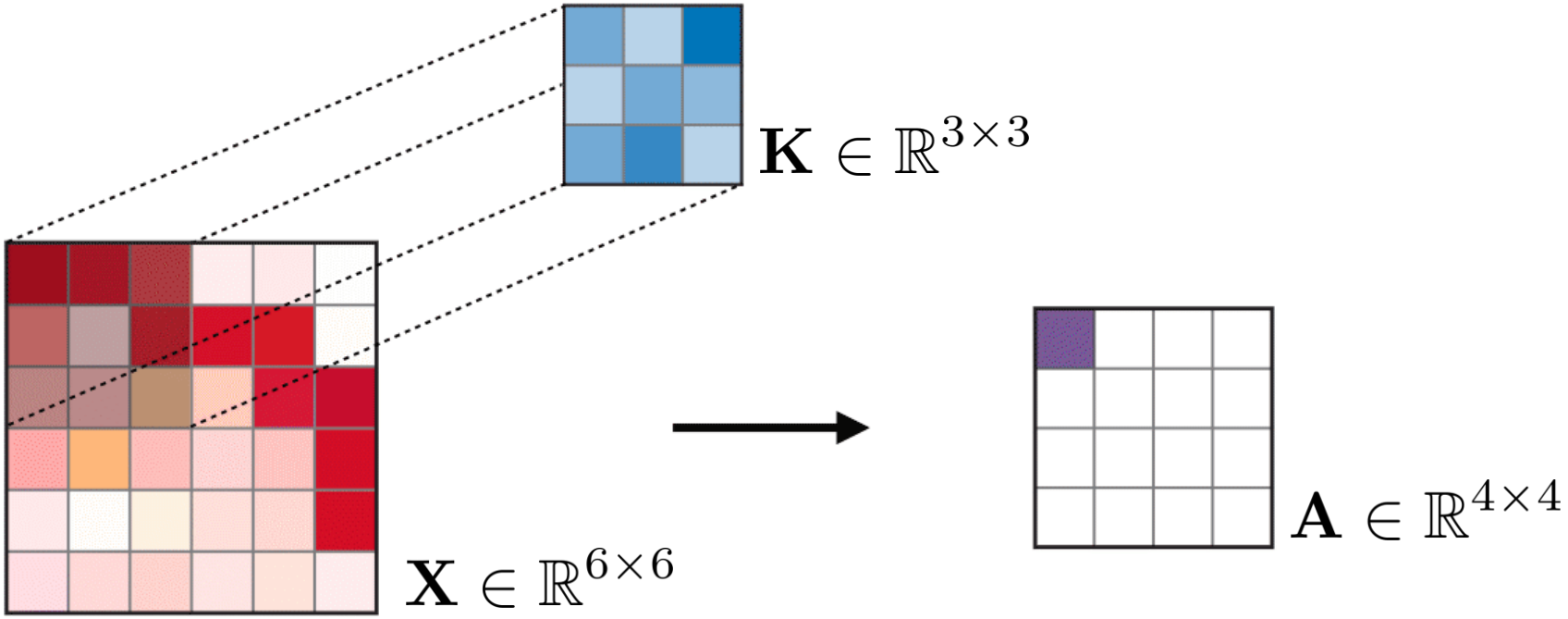
- Convolution can be generalized to **any** dimension
- For instance in 2D:

$$A[i,j] = (\mathbf{X} * \mathbf{K})[i,j] = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X[m,n]K[i-m, j-n]$$

Discrete Convolution (2D Case)

- Convolution can be generalized to **any** dimension
- For instance in 2D:

$$A[i,j] = (\mathbf{X} * \mathbf{K})[i,j] = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X[m,n]K[i-m, j-n]$$



Discrete Convolution (2D Case)

- **Ex:** edge detection, sharpening, blurring, ….

$$\boldsymbol{K} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{K} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \boldsymbol{K} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Discrete Convolution (2D Case)

- **Ex:** edge detection, sharpening, blurring, ….

$$\boldsymbol{K} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \boldsymbol{K} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \boldsymbol{K} = \frac{1}{16}\begin{bmatrix} -1 & -2 & -1 \\ -2 & 28 & -2 \\ -1 & -2 & -1 \end{bmatrix}$$

Convolution vs. Correlation

- The convolution operation (using proper ***zero-padding*** in the discrete finite case) is **associative** and **commutative**:

$$\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{b} * \boldsymbol{a}, \quad \boldsymbol{a} * (\boldsymbol{b} * \boldsymbol{c}) = (\boldsymbol{a} * \boldsymbol{b}) * \boldsymbol{c}$$

Convolution vs. Correlation

- The convolution operation (using proper ***zero-padding*** in the discrete finite case) is **associative** and **commutative**:

$$\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{b} * \boldsymbol{a}, \quad \boldsymbol{a} * (\boldsymbol{b} * \boldsymbol{c}) = (\boldsymbol{a} * \boldsymbol{b}) * \boldsymbol{c}$$

- This is not the case of the related ***cross-correlation*** operation:

$$A[i,j] = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X[m,n] K[i+m, j+n]$$

Convolution vs. Correlation

- The convolution operation (using proper ***zero-padding*** in the discrete finite case) is **associative** and **commutative**:

$$\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{b} * \boldsymbol{a}, \quad \boldsymbol{a} * (\boldsymbol{b} * \boldsymbol{c}) = (\boldsymbol{a} * \boldsymbol{b}) * \boldsymbol{c}$$

- This is not the case of the related ***cross-correlation*** operation:

$$A[i,j] = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X[m,n] K[i+m, j+n]$$

➡ Equivalent to **flipping** the kernel along all axes

Convolution vs. Correlation

- The convolution operation (using proper ***zero-padding*** in the discrete finite case) is **associative** and **commutative**:

$$\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{b} * \boldsymbol{a}, \quad \boldsymbol{a} * (\boldsymbol{b} * \boldsymbol{c}) = (\boldsymbol{a} * \boldsymbol{b}) * \boldsymbol{c}$$

- This is not the case of the related ***cross-correlation*** operation:

$$A[i,j] = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X[m,n] K[i+m, j+n]$$

➡ Equivalent to **flipping** the kernel along all axes

- Most machine learning libraries (eg. Pytorch, TensorFlow) implement cross-correlation but call it convolution

Convolution vs. Correlation

- The convolution operation (using proper ***zero-padding*** in the discrete finite case) is **associative** and **commutative**:

$$\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{b} * \boldsymbol{a}, \quad \boldsymbol{a} * (\boldsymbol{b} * \boldsymbol{c}) = (\boldsymbol{a} * \boldsymbol{b}) * \boldsymbol{c}$$

- This is not the case of the related ***cross-correlation*** operation:

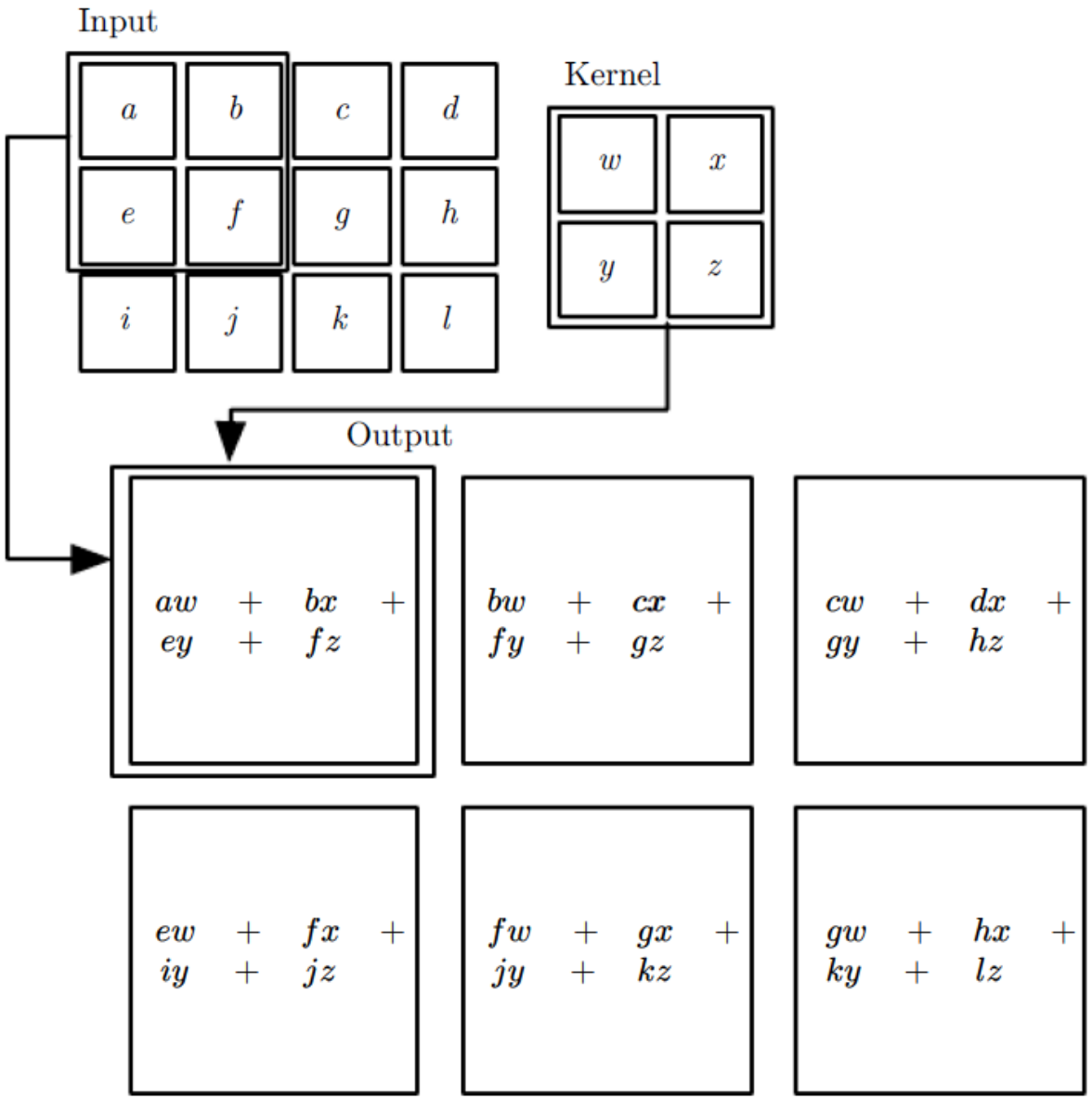
$$A[i,j] = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X[m,n] K[i+m, j+n]$$

➡ Equivalent to **flipping** the kernel along all axes

- Most machine learning libraries (eg. Pytorch, TensorFlow) implement cross-correlation but call it convolution

- In practice, since the kernel elements are **learned parameters**, it makes no difference (but worth keeping in mind!)

The maths:


Input
a b c d
e f g h
i j k l
Kernel
w x
y z
Output
aw + bx +
ey + fz
bw + cx +
fy + gz
cw + dx +
gy + hz
ew + fx +
iy + jz
fw + gx +
jy + kz
gw + hx +
ky + lz

Why Convolutional Neural Networks?

- Convolution leverages three key machine learning concepts:

  1) Sparse connectivity

  2) Parameter sharing

  3) Equivariant representation

Why Convolutional Neural Networks?

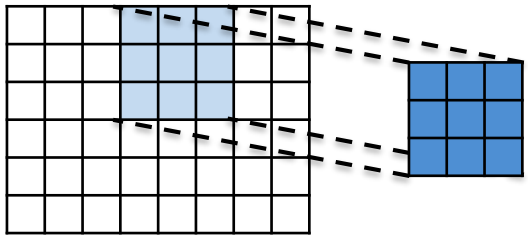
- Convolution leverages three key machine learning concepts:

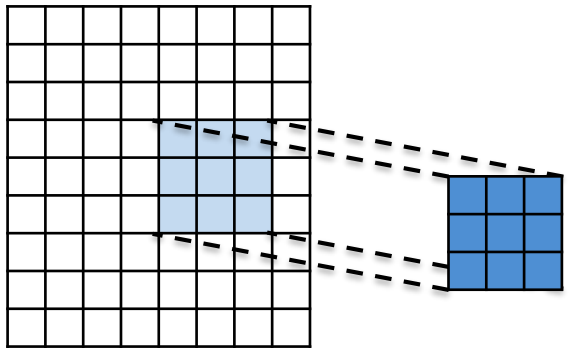
  1) Sparse connectivity

  2) Parameter sharing

  3) Equivariant representation

- Moreover, it provides a means for handling **variable size inputs**. The same kernel can be sled on signals/images of variable sizes:



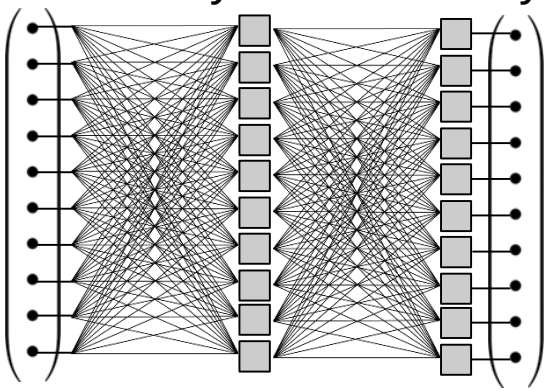


Why Convolutional Neural Networks?

1) Sparse connectivity

Two conventional fully connected layers:

$\boldsymbol{x} \in \mathbb{R}^N$

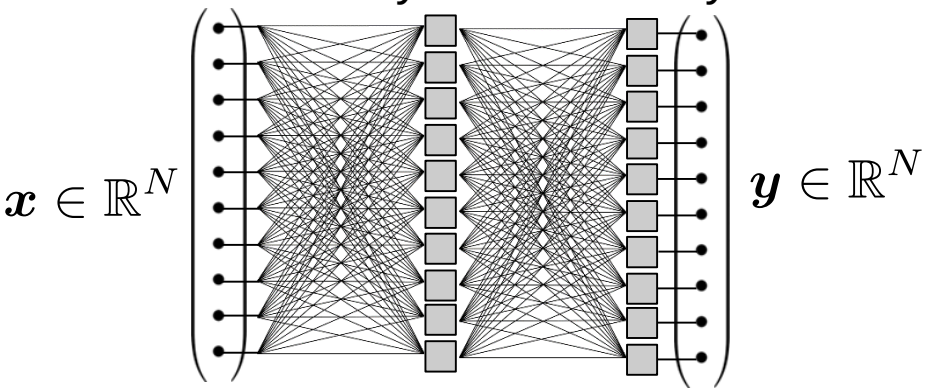


$\boldsymbol{y} \in \mathbb{R}^N$

Why Convolutional Neural Networks?

1) Sparse connectivity

Two conventional fully connected layers:

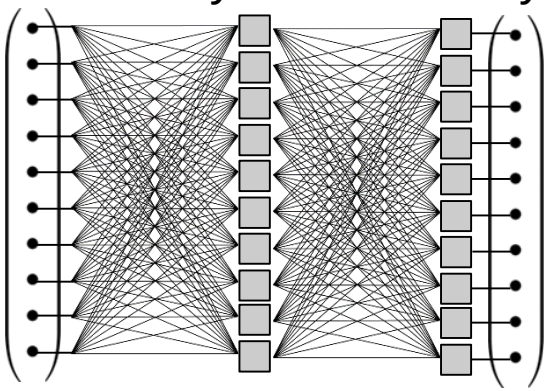


- Every output entry of a layer is connected to every input entry of the next

Why Convolutional Neural Networks?

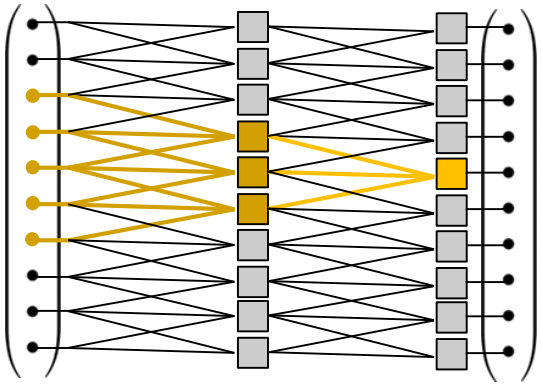
1) Sparse connectivity

Two conventional fully connected layers:

$\boldsymbol{x} \in \mathbb{R}^N$  $\boldsymbol{y} \in \mathbb{R}^N$

- Every output entry of a layer is connected to every input entry of the next

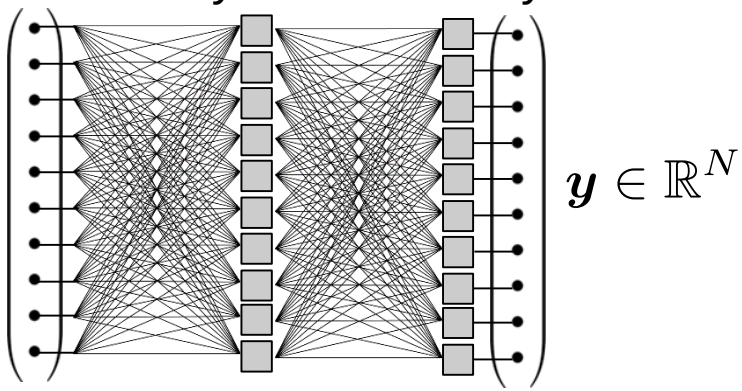
Two convolutional layers (kernel size S=3):

$\boldsymbol{x} \in \mathbb{R}^N$  $\boldsymbol{y} \in \mathbb{R}^N$

Why Convolutional Neural Networks?

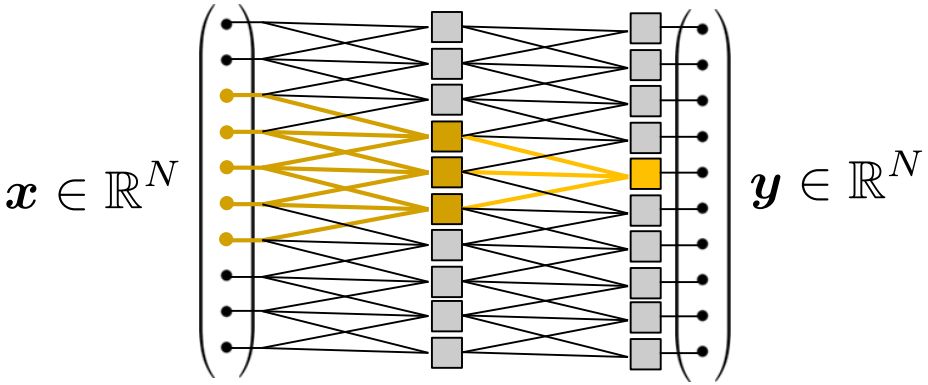
1) Sparse connectivity

Two conventional fully connected layers:



- Every output entry of a layer is connected to every input entry of the next

Two convolutional layers (kernel size S=3):

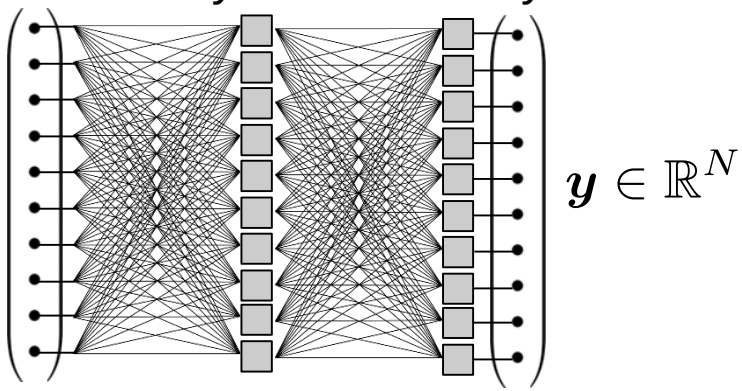


- Every output connected to only a **few neighboring inputs** due to the small kernel

Why Convolutional Neural Networks?

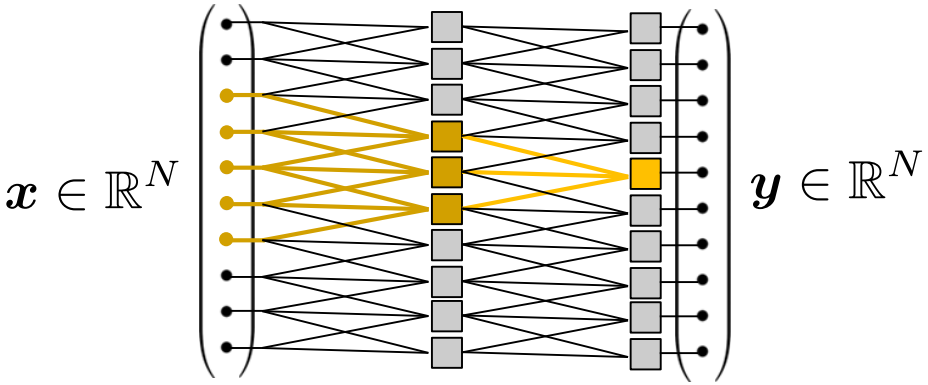
1) Sparse connectivity

Two conventional fully connected layers:



- Every output entry of a layer is connected to every input entry of the next

Two convolutional layers (kernel size S=3):

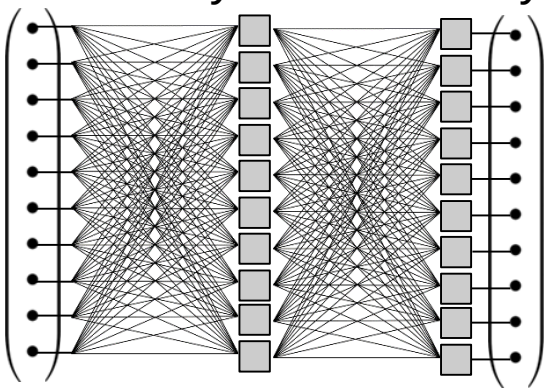


- Every output connected to only a **few neighboring inputs** due to the small kernel
- Still, in a deep CNN, each unit in late layers can interact with many inputs, forming a **receptive field**

Why Convolutional Neural Networks?

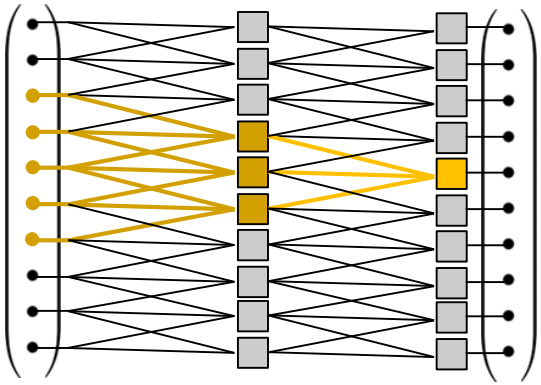
1) Sparse connectivity

Two conventional fully connected layers:

$\boldsymbol{x} \in \mathbb{R}^N$  $\boldsymbol{y} \in \mathbb{R}^N$

- Every output entry of a layer is connected to every input entry of the next

Two convolutional layers (kernel size S=3):

$\boldsymbol{x} \in \mathbb{R}^N$  $\boldsymbol{y} \in \mathbb{R}^N$

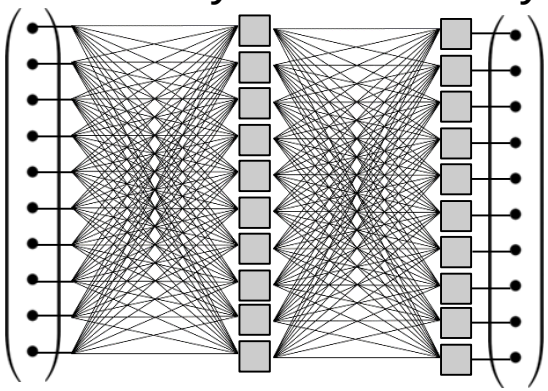
- Every output connected to only a **few neighboring inputs** due to the small kernel
- Still, in a deep CNN, each unit in late layers can interact with many inputs, forming a **receptive field**

Less connections** = **smaller model** = **less overfitting** & **faster computation

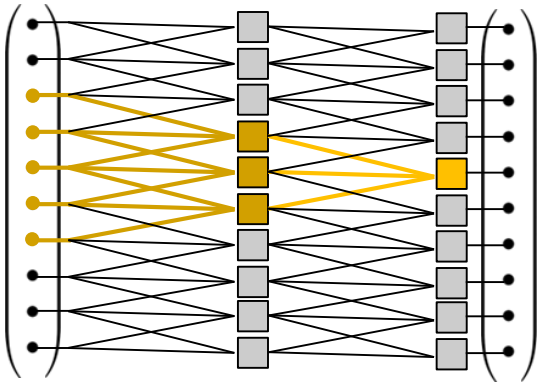
Why Convolutional Neural Networks?

2) Parameter sharing

Two conventional fully connected layers:

$\boldsymbol{x} \in \mathbb{R}^N$  $\boldsymbol{y} \in \mathbb{R}^N$

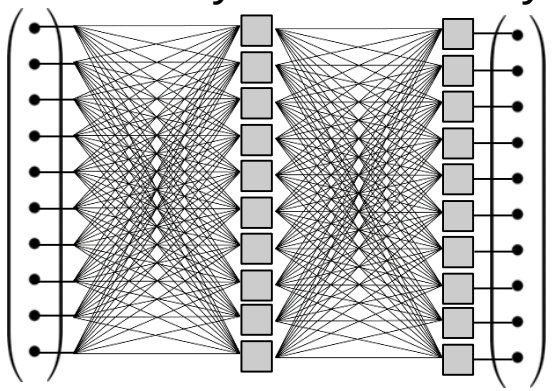
Two convolutional layers (kernel size S=3):

$\boldsymbol{x} \in \mathbb{R}^N$  $\boldsymbol{y} \in \mathbb{R}^N$

Why Convolutional Neural Networks?

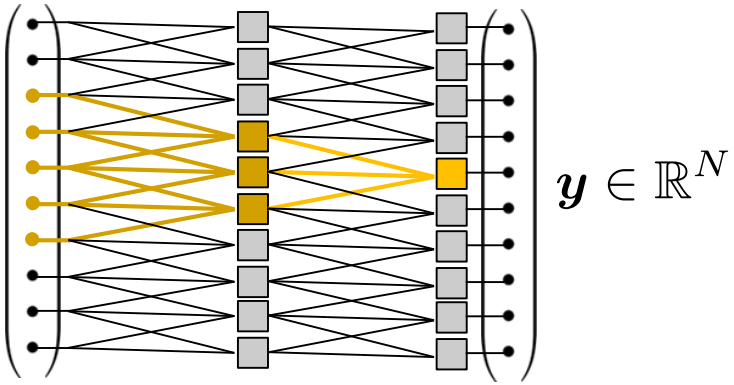
2) Parameter sharing

Two conventional fully connected layers:



- Every parameter is used **exactly once**
- ➡ $\mathcal{O}(LN^2)$ parameters

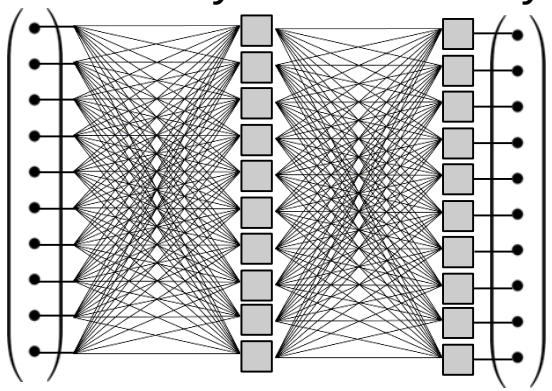
Two convolutional layers (kernel size S=3):



Why Convolutional Neural Networks?

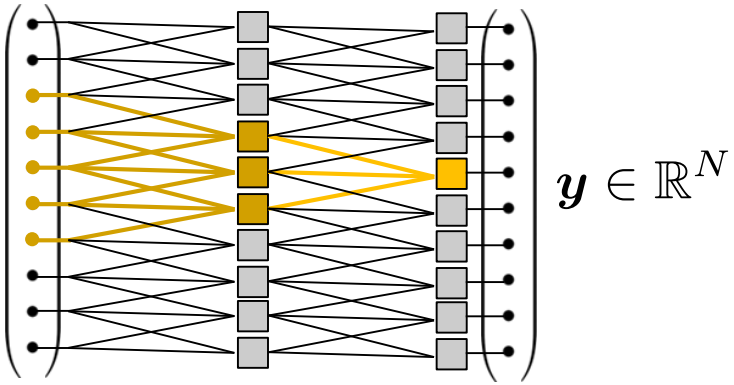
2) Parameter sharing

Two conventional fully connected layers:



- Every parameter is used **exactly once**
- ➡ $\mathcal{O}(LN^2)$ parameters

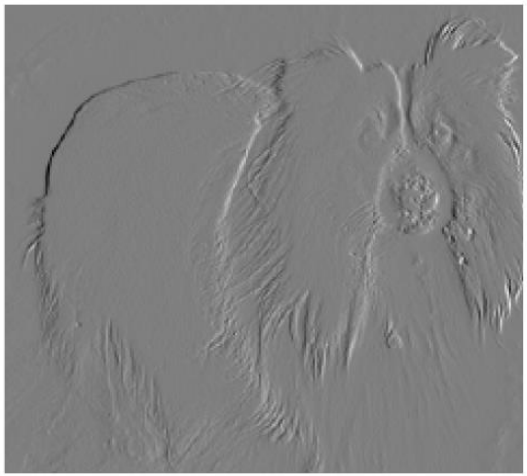
Two convolutional layers (kernel size S=3):



- Parameters **shared** (or ***tied***) across every position of the input
- ➡ $\mathcal{O}(LS)$ parameters
- Even **smaller model**, even **less overfitting**

Why Convolutional Neural Networks?

Computational efficiency



$$\mathbf{K} = \begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array}$$

- Example: horizontal edge detection (280 x 320 pixels)
  - **Convolutional:** 1 x 2 kernel, 280 x 319 ≈ 2 x 10^5 operations
  - **Fully connected**: 280 x 320 x 280 x 319 ≈ 8 x 10^9 weights ≈ 16 x 10^9 operations

Why Convolutional Neural Networks?

3) Equivariance

- **Definition:** If the input is transformed, then the output is **transformed the same way**

Why Convolutional Neural Networks?

3) Equivariance

- **Definition:** If the input is transformed, then the output is **transformed the same way**
- **/!\ Different from invariance**: If the input is transformed, then the output is **unchanged**.

Why Convolutional Neural Networks?

3) Equivariance

- **Definition:** If the input is transformed, then the output is **transformed the same way**
- **/!\ Different from invariance**: If the input is transformed, then the output is **unchanged**.
- Convolution is **equivariant to translation**: If the input signal is shifted (along its axes) by τ, then the output is also shifted by τ (convolution can even be **defined** by this).

Why Convolutional Neural Networks?

3) Equivariance

- **Definition:** If the input is transformed, then the output is **transformed the same way**
- **/!\ Different from invariance**: If the input is transformed, then the output is **unchanged**.
- Convolution is **equivariant to translation**: If the input signal is shifted (along its axes) by τ, then the output is also shifted by τ (convolution can even be **defined** by this).
- Hence, the **feature map** can be interpreted as indicating **where** some features (matching the kernels) appear in the input

Why Convolutional Neural Networks?

3) Equivariance

- **Example 1:** To process an **image**, because the objects will **look the same** no matter **where** they are in the image, it makes sense to use 2D convolution

Why Convolutional Neural Networks?

3) Equivariance

- **Example 1:** To process an **image**, because the objects will **look the same** no matter **where** they are in the image, it makes sense to use 2D convolution
- **Example 2:** To process a **speech signal**, because speech will **sound the same** no matter **when** it occurs in the signal, it makes sense to use 1D convolution

Why Convolutional Neural Networks?

3) Equivariance

- **Example 1:** To process an **image**, because the objects will **look the same** no matter **where** they are in the image, it makes sense to use 2D convolution
- **Example 2:** To process a **speech signal**, because speech will **sound the same** no matter **when** it occurs in the signal, it makes sense to use 1D convolution
- **Note**: Convolution is **not** equivariant to rotation or scaling

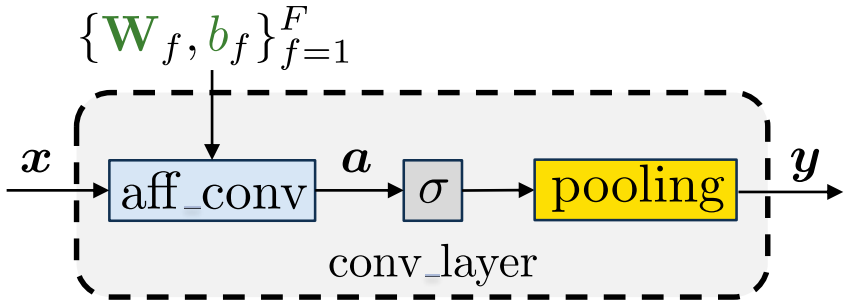
Why Convolutional Neural Networks?

3) Equivariance

- **Example 1:** To process an **image**, because the objects will **look the same** no matter **where** they are in the image, it makes sense to use 2D convolution
- **Example 2:** To process a **speech signal**, because speech will **sound the same** no matter **when** it occurs in the signal, it makes sense to use 1D convolution
- **Note**: Convolution is **not** equivariant to rotation or scaling
- **Exercise:** Supposed my input signal is a list of incomes in a given city, sorted in ascending order? Supposed my input is a 2D map of the humidity of the soil around a given location?

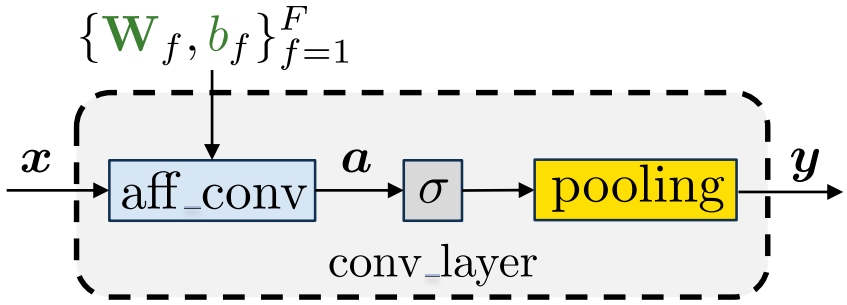
CNN layers

- A **CNN layer** typically consists of 3 stages

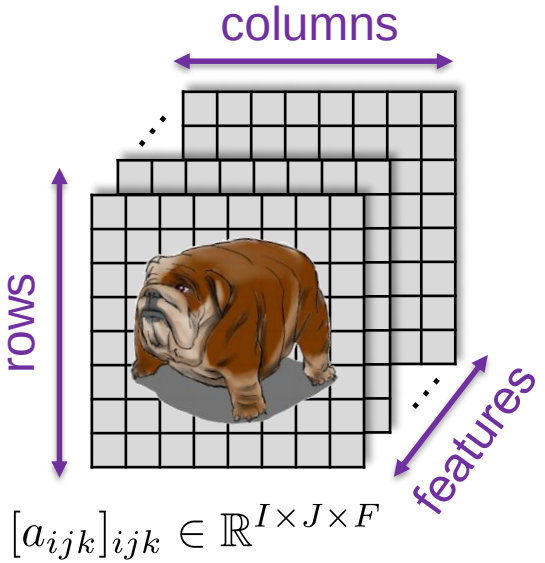


CNN layers

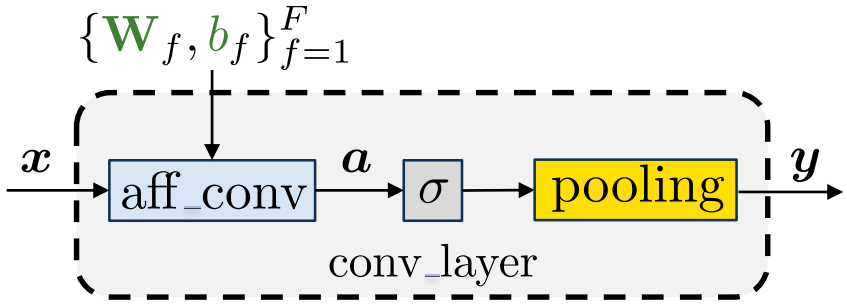
- A **CNN layer** typically consists of 3 stages



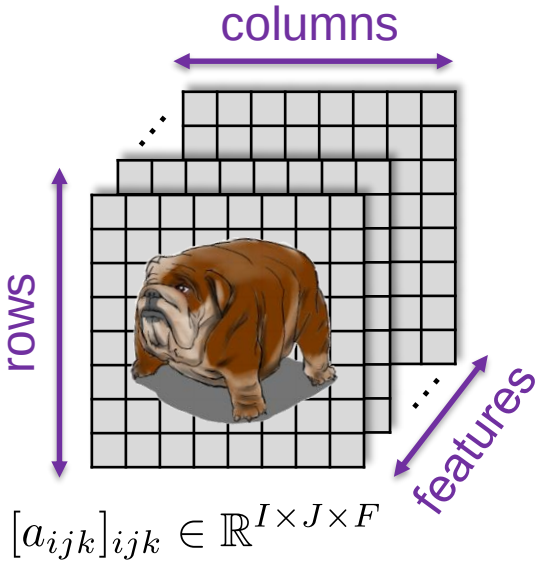
- The **affine convolution** stage applies **multiple kernels** $\{\mathbf{W}_f\}_{f=1}^F$ to the input, yielding F feature maps, arranged in a **multi-way** tensor $\boldsymbol{a}$.



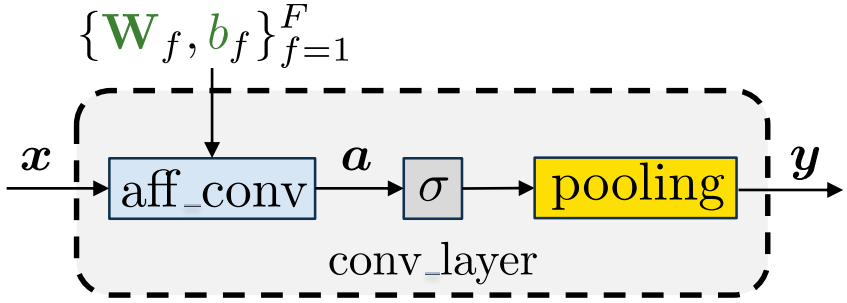
CNN layers



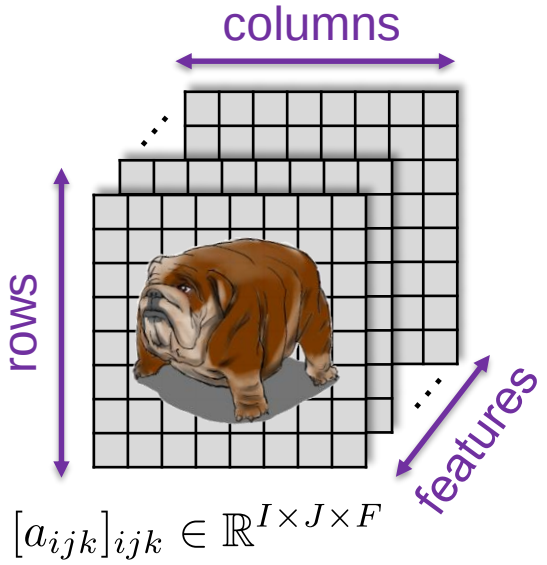
- A **CNN layer** typically consists of 3 stages
- The **affine convolution** stage applies **multiple kernels** $\{\mathbf{W}_f\}_{f=1}^F$ to the input, yielding F feature maps, arranged in a **multi-way** tensor $\boldsymbol{a}$.
- A **bias term** b_f is added to each feature map



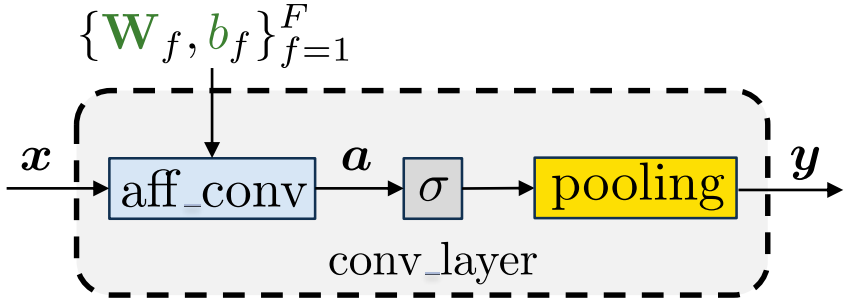
CNN layers



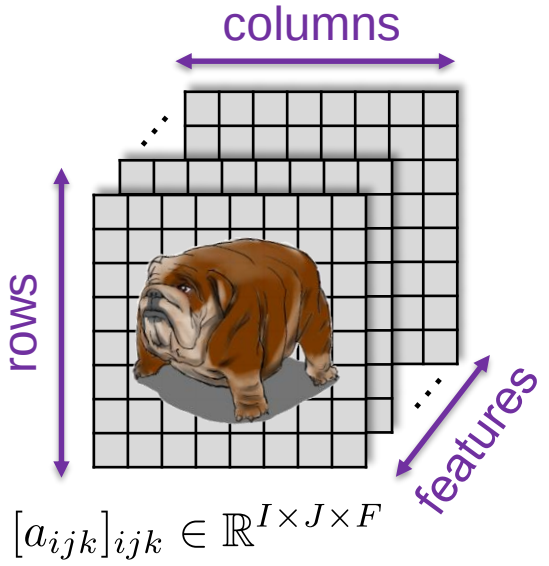
- A **CNN layer** typically consists of 3 stages
- The **affine convolution** stage applies **multiple kernels** $\{\mathbf{W}_f\}_{f=1}^F$ to the input, yielding F feature maps, arranged in a **multi-way** tensor $\boldsymbol{a}$.
- A **bias term** b_f is added to each feature map
- A **nonlinear activation function** σ is then applied to all feature maps (e.g. ReLU)



CNN layers



- A **CNN layer** typically consists of 3 stages
- The **affine convolution** stage applies **multiple kernels** $\{\mathbf{W}_f\}_{f=1}^F$ to the input, yielding F feature maps, arranged in a **multi-way** tensor $\boldsymbol{a}$.
- A **bias term** b_f is added to each feature map
- A **nonlinear activation function** σ is then applied to all feature maps (e.g. ReLU)
- There is then a **pooling** stage, that replaces every **neighborhood** in a given feature map by a **summary statistics** (ex: mean or max)



Pooling

- Pooling makes the representation **more invariant** to **small translations** of the input

Pooling

- Pooling makes the representation **more invariant** to **small translations** of the input

- Translation invariance is useful when we care more about whether some feature is **present** than exactly **where** it is

Pooling

- Pooling makes the representation **more invariant** to **small translations** of the input

- Translation invariance is useful when we care more about whether some feature is **present** than exactly **where** it is

- **Example:** to detect a face, we search for an eye on the left side and an eye on the right side, but we don't need to locate them with pixel-level accuracy.

Pooling

- Pooling makes the representation **more invariant** to **small translations** of the input

- Translation invariance is useful when we care more about whether some feature is **present** than exactly **where** it is

- **Example:** to detect a face, we search for an eye on the left side and an eye on the right side, but we don't need to locate them with pixel-level accuracy.

- **Counter-example:** to denoise an image, we must preserve the location of the features. In that situation pooling is not desirable..

Pooling

- Since pooling summarizes the responses over a **neighborhood**, we often report summary statistics **every** *S* **pixels** instead of every 1 pixel

Pooling

- Since pooling summarizes the responses over a **neighborhood**, we often report summary statistics **every** *S* **pixels** instead of every 1 pixel
- *S* is called the **stride**. It can be different for different spatial directions

Pooling

- Since pooling summarizes the responses over a **neighborhood**, we often report summary statistics **every** *S* **pixels** instead of every 1 pixel
- *S* is called the **stride**. It can be different for different spatial directions
- This drastically improves the **computational efficiency**
  - Typical stride values are 2 or 3 along each axes.
  - For an image, this amounts to a **x4** or **x9** **dimensionality reduction** over feature maps

Pooling

- Since pooling summarizes the responses over a **neighborhood**, we often report summary statistics **every** *S* **pixels** instead of every 1 pixel
- *S* is called the **stride**. It can be different for different spatial directions
- This drastically improves the **computational efficiency**
  - Typical stride values are 2 or 3 along each axes.
  - For an image, this amounts to a **x4** or **x9** **dimensionality reduction** over feature maps
- Also useful when dealing with **variable size** input: adjust the pooling and stride to obtain a fixed-size output

Pooling

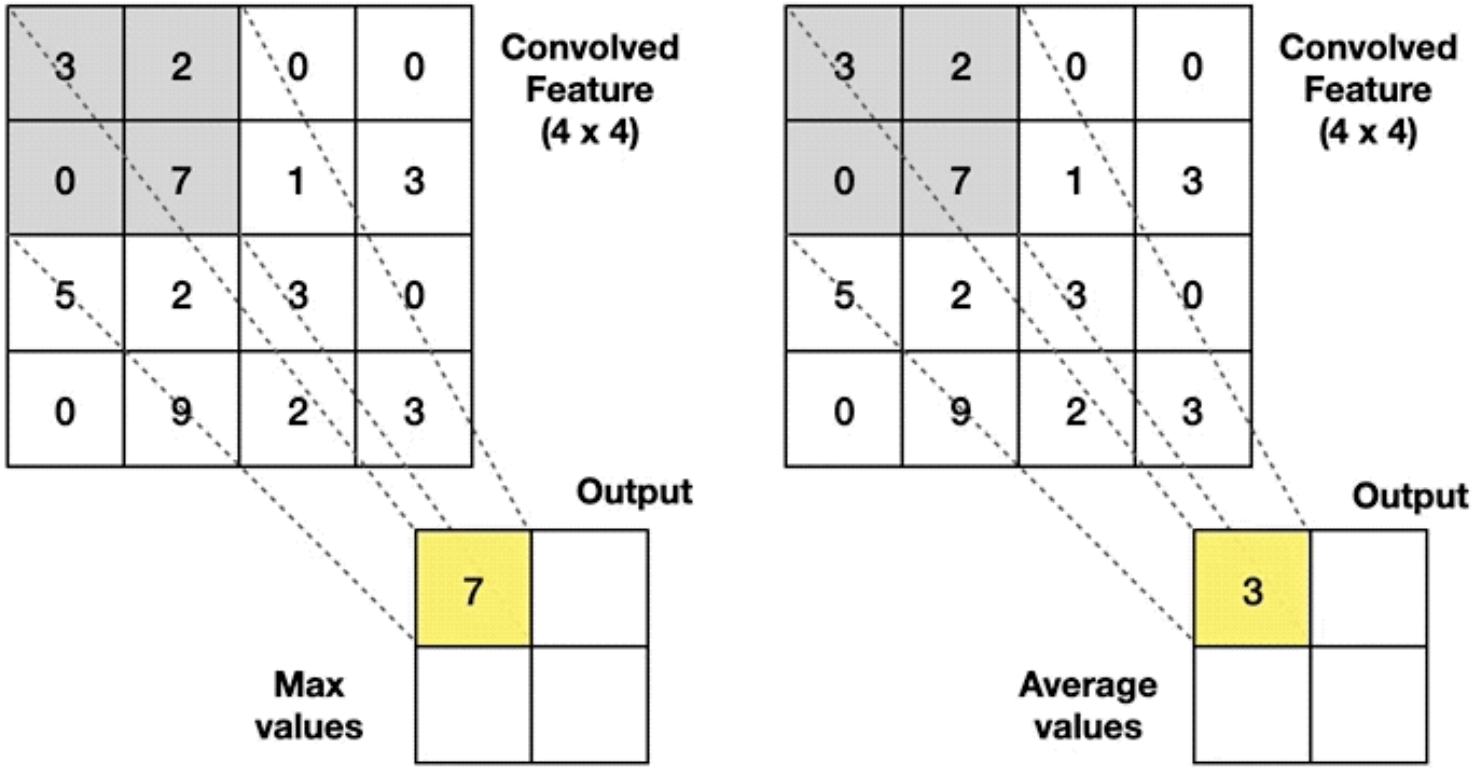
Max Pooling

Take the **highest** value from the area covered by the kernel

Average Pooling

Calculate the **average** value from the area covered by the kernel

Example: Kernel of size 2 x 2; stride=(2,2)



Dealing with multichannel input

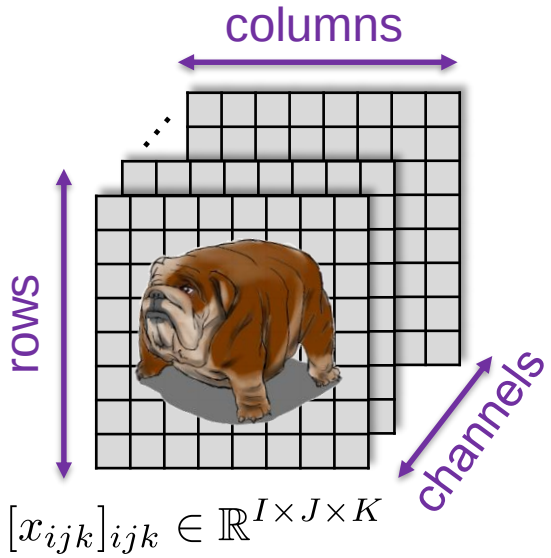
- In contrast to classical convolution, each **entry** of the input of a **convolution layer** in a CNN is usually a **vector**, not a **scalar**.

Dealing with multichannel input

- In contrast to classical convolution, each **entry** of the input of a **convolution layer** in a CNN is usually a **vector**, not a **scalar**.
  - **Ex1**: RGB **channels** in an image
  - **Ex2**: Multiple **feature maps** obtained in a previous layer

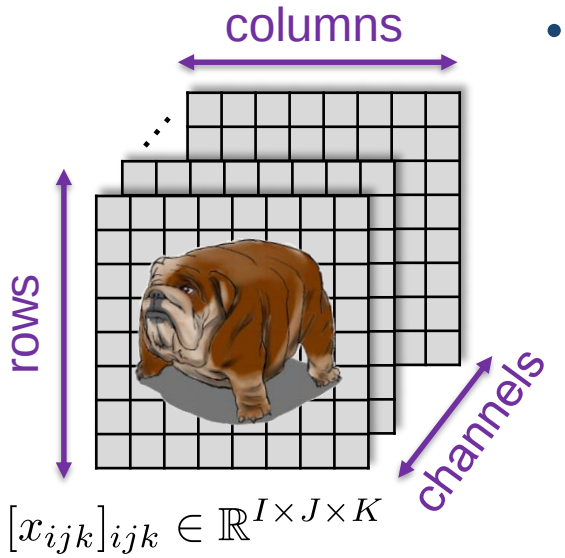
Dealing with multichannel input

- In contrast to classical convolution, each **entry** of the input of a **convolution layer** in a CNN is usually a **vector**, not a **scalar**.
  - **Ex1**: RGB **channels** in an image
  - **Ex2**: Multiple **feature maps** obtained in a previous layer
- Hence, the **input** of a 2D conv. layer is also a **3-way tensor**:



Dealing with multichannel input

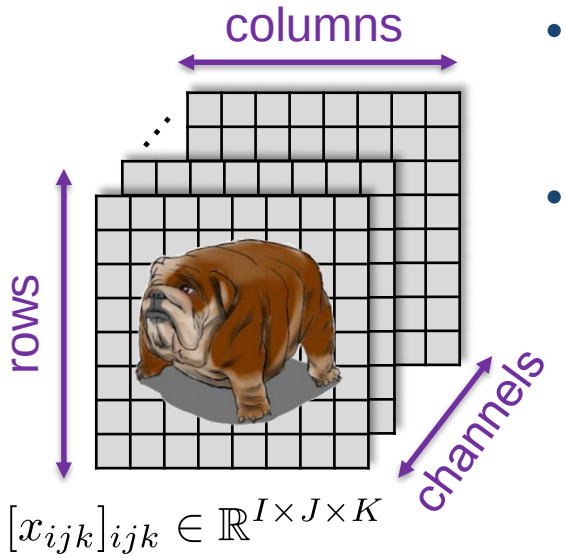
- In contrast to classical convolution, each **entry** of the input of a **convolution layer** in a CNN is usually a **vector**, not a **scalar**.
  - **Ex1**: RGB **channels** in an image
  - **Ex2**: Multiple **feature maps** obtained in a previous layer
- Hence, the **input** of a 2D conv. layer is also a **3-way tensor**:



- Convolution over the channels (ie. 3D conv.) does not make sense (by def. of “channels”)

Dealing with multichannel input

- In contrast to classical convolution, each **entry** of the input of a **convolution layer** in a CNN is usually a **vector**, not a **scalar**.
  - **Ex1**: RGB **channels** in an image
  - **Ex2**: Multiple **feature maps** obtained in a previous layer
- Hence, the **input** of a 2D conv. layer is also a **3-way tensor**:

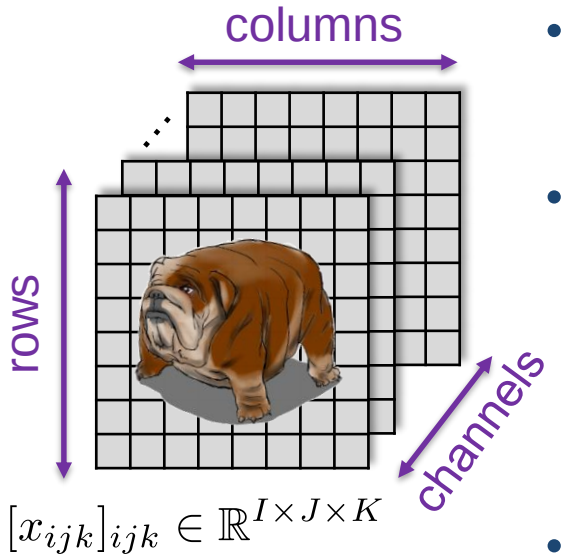


- Convolution over the channels (ie. 3D conv.) does not make sense (by def. of “channels”)
- In standard multichannel convolution, each of the K **input channel** is **fully connected** to all of the F feature maps, i.e., **output channels**:

$$a_{ijf} = (\mathbf{X} *_{1,2} \mathbf{W})_{ijf} = \sum_k \sum_m \sum_n x_{ijk} w_{i+m,j+n,f,k}$$

Dealing with multichannel input

- In contrast to classical convolution, each **entry** of the input of a **convolution layer** in a CNN is usually a **vector**, not a **scalar**.
  - **Ex1**: RGB **channels** in an image
  - **Ex2**: Multiple **feature maps** obtained in a previous layer
- Hence, the **input** of a 2D conv. layer is also a **3-way tensor**:

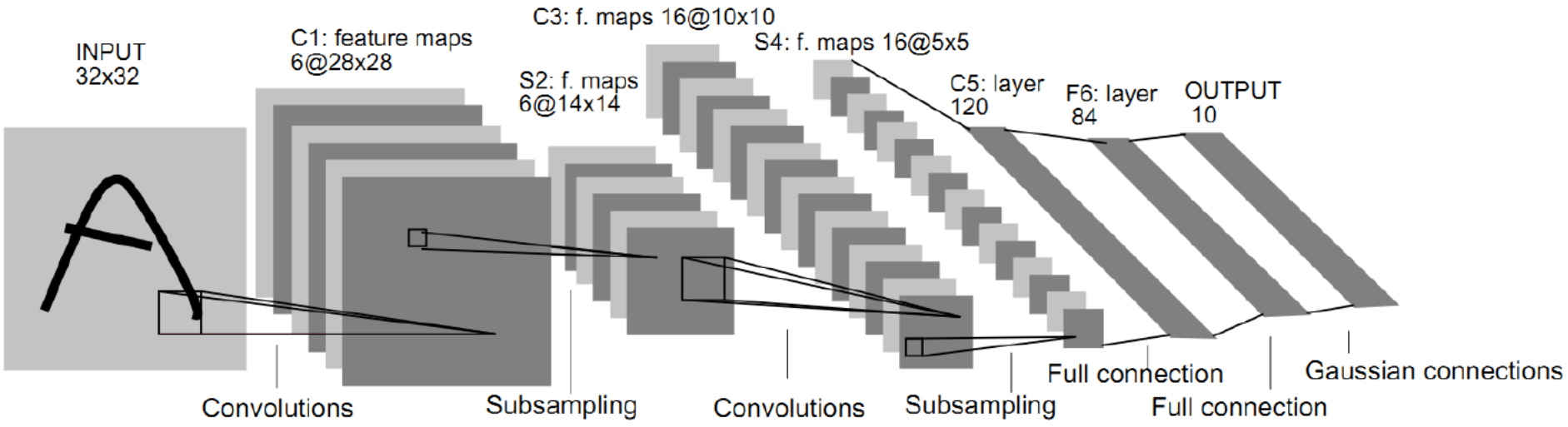


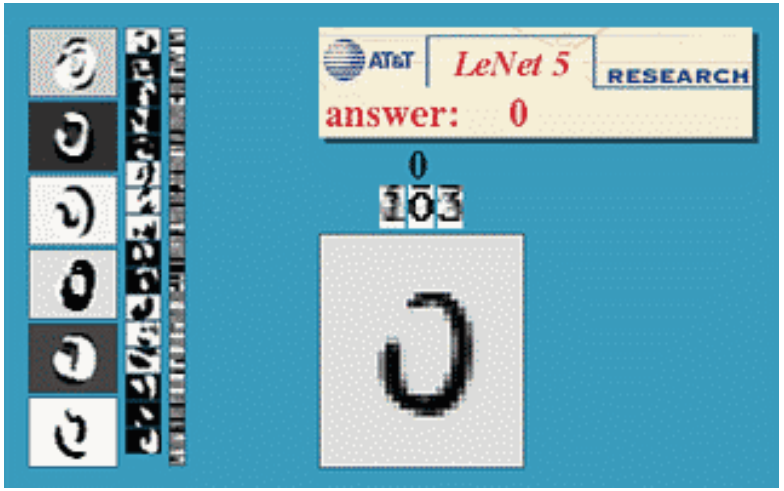
- Convolution over the channels (ie. 3D conv.) does not make sense (by def. of "channels")
- In standard multichannel convolution, each of the K **input channel** is **fully connected** to all of the F feature maps, i.e., **output channels**:

$$a_{ijf} = (\mathbf{X} *_{1,2} \mathbf{W})_{ijf} = \sum_k \sum_m \sum_n x_{ijk} w_{i+m,j+n,f,k}$$

- Each **entry** of the kernel acts as a $F \times K$ matrix: the list of kernels $\mathbf{W}$ form a **4-way** tensor

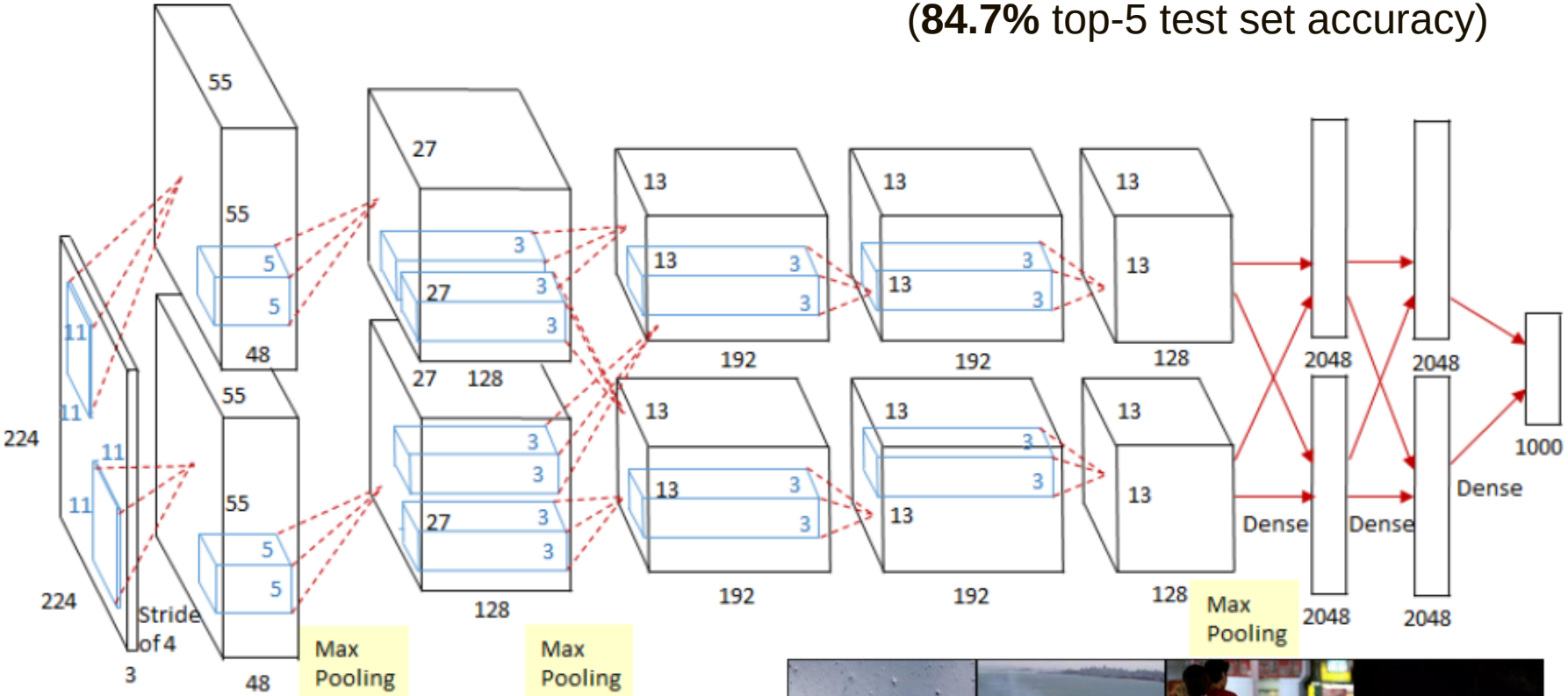
Famous example: LeNet-5 for digit recognition (1989)





Famous example: AlexNet for ImageNet classification (2012)

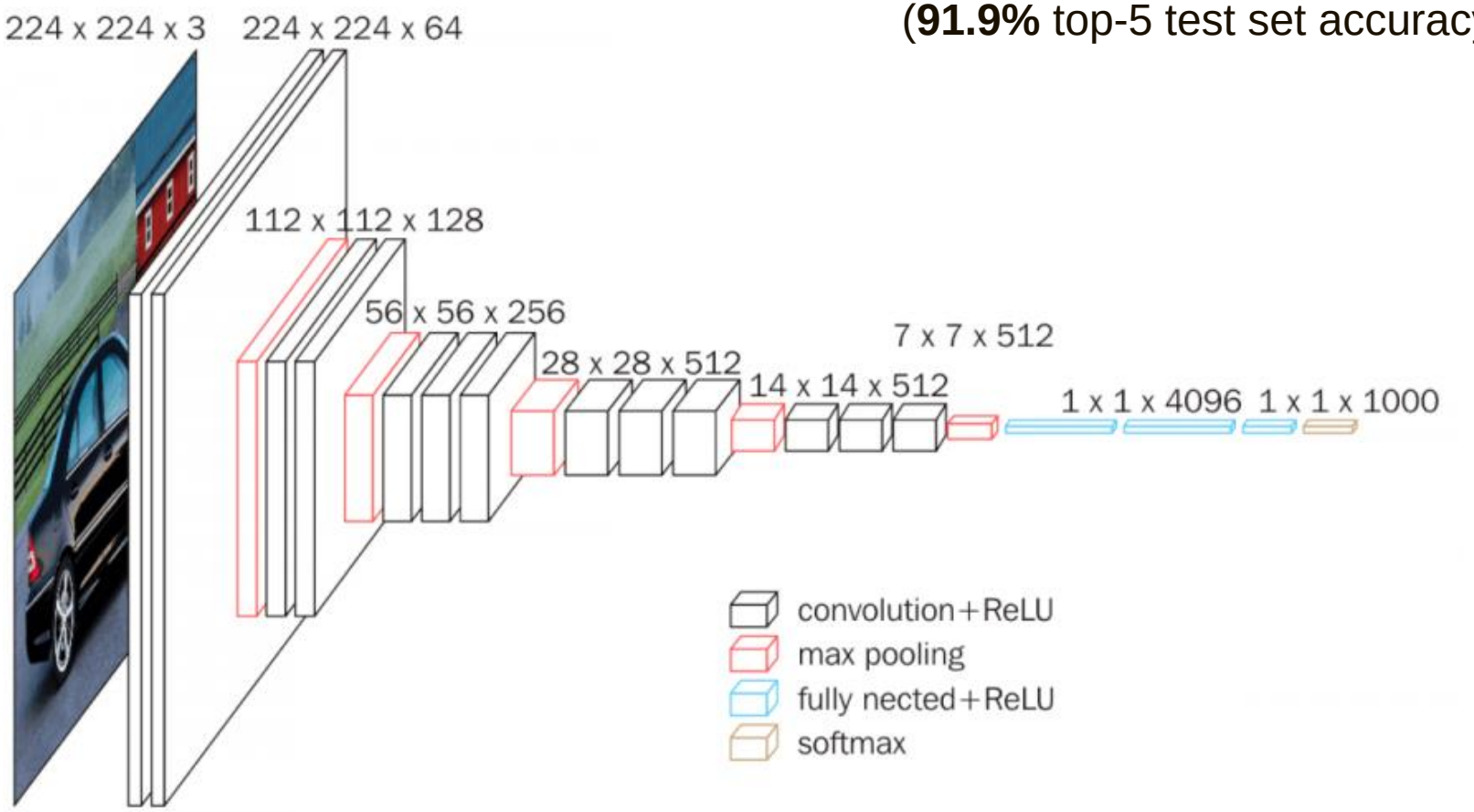
(**84.7%** top-5 test set accuracy)



Proven not so useful in the end

Famous example: VGG-16 for ImageNet classification (2014)

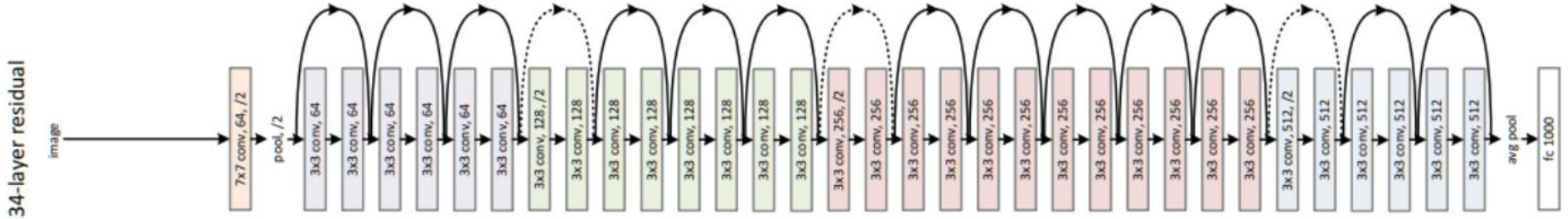
(**91.9%** top-5 test set accuracy)



- Uses only 3x3 kernels, but more depth (16 vs. 7 layers)

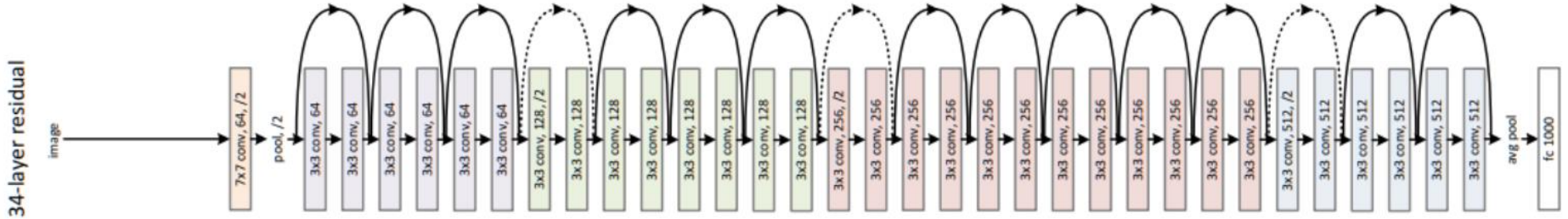
Famous example: ResNet for ImageNet classification (2015)

94.3% top-5 for ResNet-152 / **95.2%** for ResNet-200 (2016)



Famous example: ResNet for ImageNet classification (2015)

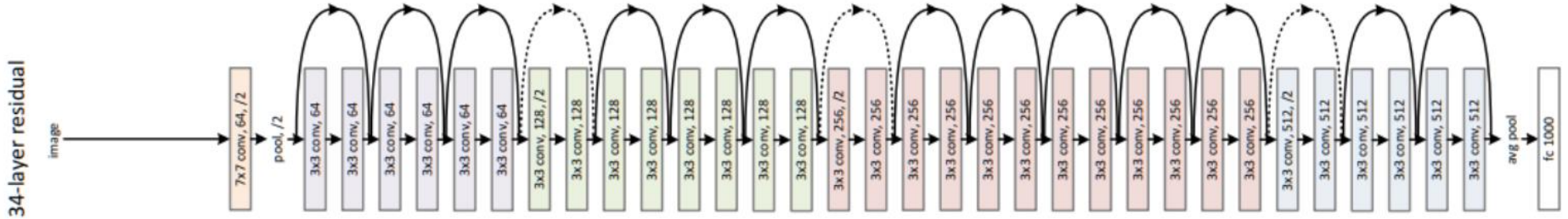
94.3% top-5 for ResNet-152 / **95.2%** for ResNet-200 (2016)



Note:
- Human top-5 accuracy reported at **94.9%**
- Perf. on ImageNet ≈plateaus around **98%** since 2018

Famous example: ResNet for ImageNet classification (2015)

94.3% top-5 for ResNet-152 / **95.2%** for ResNet-200 (2016)



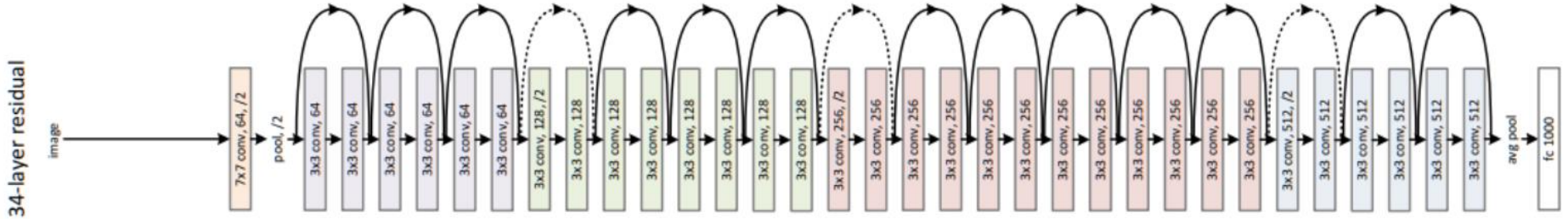
Note:
- Human top-5 accuracy reported at **94.9%**
- Perf. on ImageNet ≈plateaus around **98%** since 2018

Key idea:

- **Residual** or ***skip* connections** that "passes through" several layers

Famous example: ResNet for ImageNet classification (2015)

94.3% top-5 for ResNet-152 / **95.2%** for ResNet-200 (2016)



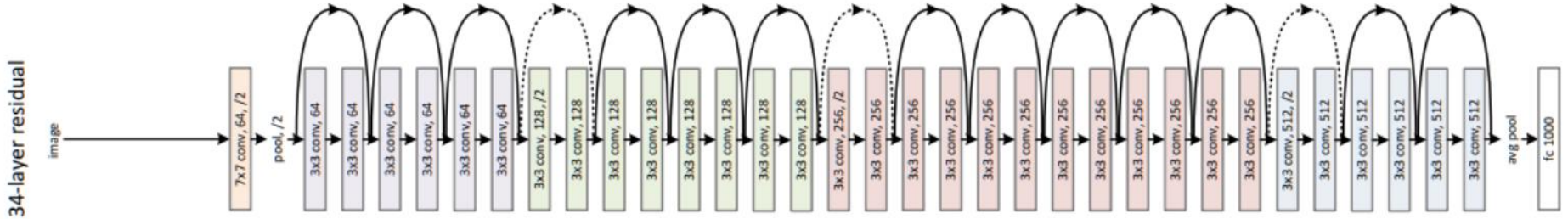
Note:
- Human top-5 accuracy reported at **94.9%**
- Perf. on ImageNet ≈plateaus around **98%** since 2018

Key idea:

- **Residual** or ***skip* connections** that "passes through" several layers
- Solves the problem of ***vanishing gradient***

Famous example: ResNet for ImageNet classification (2015)

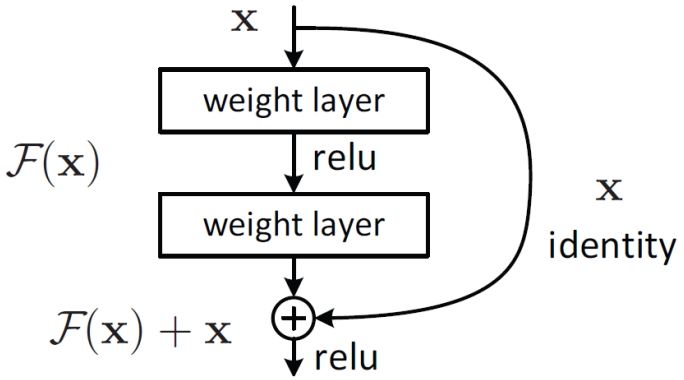
94.3% top-5 for ResNet-152 / **95.2%** for ResNet-200 (2016)



Note:
- Human top-5 accuracy reported at **94.9%**
- Perf. on ImageNet ≈plateaus around **98%** since 2018

Key idea:

- **Residual** or ***skip* connections** that "passes through" several layers
- Solves the problem of ***vanishing gradient***
- **Recall:**



$$G_{b^1} = \sigma'(a_t^1) \odot \mathbf{W}^{2\top} \sigma'(a_t^2) \odot \mathbf{W}^{3\top} \ldots \sigma'(a_t^3) \odot \mathbf{W}^{L\top} \sigma'(a_t^L) \odot \nabla_{\boldsymbol{x}^L} \ell(\boldsymbol{x}_t^L, \boldsymbol{y}_t)$$

OUTLINE

I. Introduction
A.I., Machine Learning, Deep Learning: What, How, Why and When

II. Background
Tensors and Multivariate Calculus

III. Fitting a Model
Optimization techniques, Backpropagation, Gradient Descent, PyTorch

IV. Supervised Learning
Regression, Classification, Loss Design, Over & Underfitting

V. Unsupervised Learning
From K-means and PCA to Deep Clustering and Autoencoders

VI. Convolutional Neural Networks
Definition, Why, Layers, Famous Examples