Convolutional networks

UE de Master 2, AOS2
Fall 2023

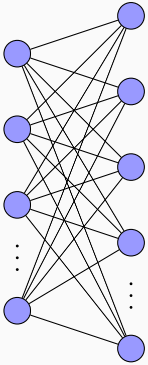
S. Rousseau

Introduction

Introduction

How can we apply neural models to computer vision?

- Flatten image as a vector and feed a MLP?
  - Spatial structure is lost
  - Color band is lost
  - Quadratic number of parameters wrt to number of neurons
- Special features of images:
  - **Translation equivariance:** translate an object should translate extracted features as well
  - **Locality**: Does it make sense to mix for example upper left and lower right pixels?



Which linear transform are **translation equivariant** and **local**?

Translation equivariance for 1-D signal

- What are the translation equivariant 1-D linear transforms?
  - Let $\boldsymbol{x} = (\dots, x_{-n}, \dots x_0, \dots, x_n, \dots)$ a (infinite) 1-D signal
  - L a linear transform of 1-D signals
  - S is the (right) shifting operator: $(S(\boldsymbol{x}))_j = x_{j-1}$
  - $S^k = S \circ \dots \circ S$, $k \in \mathbb{Z}$
- Translation equivariance reads: $L \circ S^k = S^k \circ L$. Linear transform of shifted signal is the shifted linear transform
  - Vector $\boldsymbol{x}$ can be written $\boldsymbol{x} = \sum_{i \in \mathbb{Z}} x_i S^i(\boldsymbol{e}_0)$
  - Then L is a convolution:

$$L_j(\boldsymbol{x}) = \sum_{i \in \mathbb{Z}} x_i y_{j-i} \quad \text{with} \quad \boldsymbol{y} = L(\boldsymbol{e}_0)$$

- A translation-equivariant linear transform is a convolution!

Proof

$$
\begin{aligned}
L_j(\boldsymbol{x}) &= \langle \boldsymbol{e}_j, L(\boldsymbol{x}) \rangle \\
&= \left\langle \boldsymbol{e}_j, L\left(\sum_{i \in \mathbb{Z}} x_i S^i(\boldsymbol{e}_0) \right) \right\rangle \\
&\qquad\qquad\qquad\qquad \text{(decomposition of } \boldsymbol{x}\text{)} \\
&= \left\langle \boldsymbol{e}_j, \sum_{i \in \mathbb{Z}} x_i L S^i(\boldsymbol{e}_0) \right\rangle \quad \text{(linearity of } L\text{)} \\
&= \left\langle \boldsymbol{e}_j, \sum_{i \in \mathbb{Z}} x_i S^i L(\boldsymbol{e}_0) \right\rangle \\
&\qquad\qquad\qquad\qquad \text{(equivariance of } L\text{)}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i \in \mathbb{Z}} x_i \left\langle \boldsymbol{e}_j, S^i \boldsymbol{y} \right\rangle \quad \text{(linearity of dot-product)} \\
&= \sum_{i \in \mathbb{Z}} x_i \left\langle S^{-i} \boldsymbol{e}_j, \boldsymbol{y} \right\rangle \quad \text{(isometry of } S\text{)} \\
&= \sum_{i \in \mathbb{Z}} x_i \left\langle \boldsymbol{e}_{j-i}, \boldsymbol{y} \right\rangle \\
&= \sum_{i \in \mathbb{Z}} x_i y_{j-i}
\end{aligned}
$$

Locality

- A translation-equivariant linear transform reads

$$L_j(\boldsymbol{x}) = \sum_i x_i y_{j-i}$$

- Locality implies that $L_j(\boldsymbol{x})$ must only depend on x_{j+k} for $k \in [\![-a, a]\!]$, $a \in \mathbb{N}^*$
- Translates to $y_k = 0$ except for when $k \in [\![-a, a]\!]$. Then we have

$$L_j(\boldsymbol{x}) = \sum_{k \in [\![-a,a]\!]} x_{j-k} y_k$$

- $\boldsymbol{y}$ must be a vector with a tiny contiguous support

Notations and properties

- The convolution operator is $*$:

$$(\mathbf{u} * \mathbf{v})_i = \sum_{k \in \mathbb{Z}} \mathbf{u}_k \mathbf{v}_{i-k}$$

- Linear wrt each argument: $\mathbf{u} * (\mathbf{v} + \boldsymbol{w}) = \mathbf{u} * \mathbf{v} + \mathbf{u} * \boldsymbol{w}$
- Symmetric: $\mathbf{u} * \mathbf{v} = \mathbf{v} * \mathbf{u}$
- Associativity: $(\mathbf{u} * \mathbf{v}) * \boldsymbol{w} = \mathbf{u} * (\mathbf{v} * \boldsymbol{w})$
- Equivalent to polynomial multiplication

$$(1, 2) * (2, -1, 2) = (2, 3, 0, 4) \quad \Longleftrightarrow \quad (1 + 2X)\left(2 - X + 2X^2\right) = 2 + 3X + 4X^3$$

- Easily generalisable to n-D signals:

$$(C * K)_{kl} = \sum_{(i,j) \in \mathbb{Z}^2} K_{ij} C_{k-i,l-j}$$

2-*D* convolution

2-D correlation

- For a matrix C of size $H_{\text{in}} \times W_{\text{in}}$ and a kernel K of size $k_h \times k_w$, 2-D convolution is defined as:

$$(C * K)_{kl} = \sum_{\substack{i=0,\dots,k_h-1 \\ j=0,\dots,k_w-1}} K_{ij} C_{k-i,l-j}$$

- In fact we use the **correlation** limited to a given window defined as:

$$C \circledast K = C * K^{\dagger} \quad \text{where} \quad K^{\dagger}_{ij} = K_{-i,-j}$$

limited to the indexes $k = 0, \dots, H_{\text{out}} - 1$ and $l = 0, \dots, W_{\text{out}} - 1$.

- This can be written

$$(C \circledast K)_{kl} = \sum_{\substack{i=0,\dots,k_h-1 \\ j=0,\dots,k_w-1}} K_{ij} C_{i+k,j+l}$$

where $k = 0, \dots, H_{\text{out}} - 1$ and $l = 0, \dots, W_{\text{out}} - 1$.

2-D “convolution”

- We use $*$ instead of $\circledast$ even if it is a correlation
- We use the term “convolution” even if it is a correlation

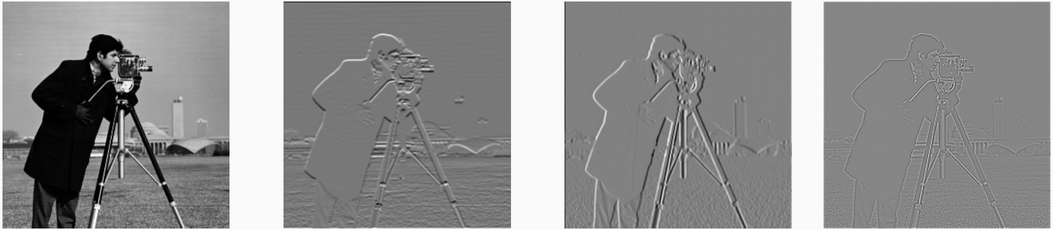
- Final formulation is

$$(C * K)_{kl} = \sum_{\substack{i=0,\dots,k_h-1 \\ j=0,\dots,k_w-1}} K_{ij} C_{i+k,j+l}$$

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 \\ 2 & 2 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}}_{C} * \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}}_{K} = \underbrace{\begin{bmatrix} 2 & 0 & 2 \\ 3 & 0 & 2 \\ 1 & 1 & 3 \end{bmatrix}}_{C * K}$$

Some handcrafted kernels used in computer vision:

$$K = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad K = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad K = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$



Padding

- Convolution operator **decreases size**
- Input of size: $H_{\text{in}} \times W_{\text{in}}$
- Kernel of size: $k_h \times k_w$
- Output of size:

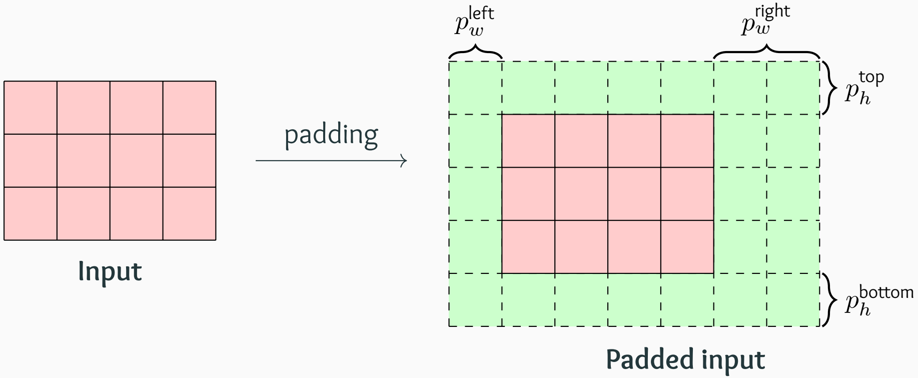
$$H_{\text{out}} = H_{\text{in}} - k_h + 1$$

$$W_{\text{out}} = W_{\text{in}} - k_w + 1$$

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 \\ 2 & 2 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}}_{C} * \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}}_{K} = \underbrace{\begin{bmatrix} 2 & 0 & 2 \\ 3 & 0 & 2 \\ 1 & 1 & 3 \end{bmatrix}}_{C * K}$$

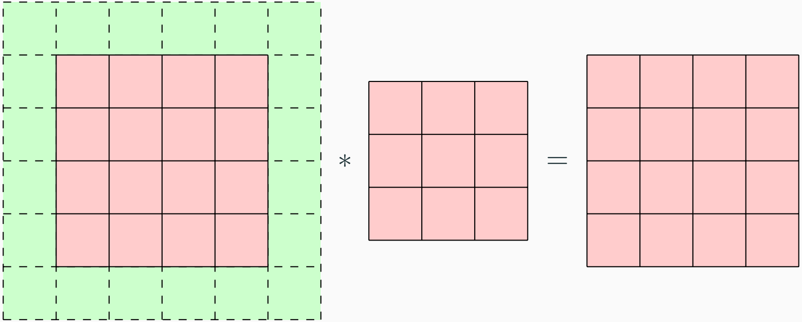
Padding

- **Enlarge size of input** by adding $p_h = p_h^{\text{top}} + p_h^{\text{bottom}}$ rows and $p_w = p_w^{\text{left}} + p_w^{\text{right}}$ columns at borders.
- For example $p_h = 2$ and $p_w = 3$.



Padding

- Size of input: $H_{\text{in}} \times W_{\text{in}}$
- Size after padding: $(H_{\text{in}} + p_h) \times (W_{\text{in}} + p_w)$
- Size of output: $H_{\text{out}} = H_{\text{in}} + p_h - k_h + 1$, $W_{\text{out}} = W_{\text{in}} + p_w - k_w + 1$
- Preserve input size when: $p_h = k_h - 1$ and $p_w = k_w - 1$



Zero padding

Pad with zero

$$\begin{array}{|c|c|c|c|c|c|c|}\hline 0&0&0&0&0&0&0\\\hline 0&0&1&0&1&1&0\\\hline 0&0&0&1&2&2&0\\\hline 0&0&1&1&1&0&0\\\hline 0&1&1&0&1&2&0\\\hline 0&0&1&0&0&2&0\\\hline 0&0&0&0&0&0&0\\\hline\end{array} * \begin{array}{|c|c|c|}\hline 1&0&1\\\hline 0&1&2\\\hline 0&1&0\\\hline\end{array} = \begin{array}{|c|c|c|c|c|}\hline 2&1&3&5&3\\\hline 1&3&8&8&3\\\hline 3&5&5&5&4\\\hline 4&3&4&6&5\\\hline 3&2&2&6&3\\\hline\end{array}$$

Reflection padding

Pad using reflections

$$
\begin{array}{|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 1 & 2 & 2 & 2 \\ \hline
1 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline
0 & 0 & 0 & 1 & 2 & 2 & 2 \\ \hline
1 & 0 & 1 & 1 & 1 & 0 & 1 \\ \hline
1 & 1 & 1 & 0 & 1 & 2 & 1 \\ \hline
1 & 0 & 1 & 0 & 0 & 2 & 0 \\ \hline
1 & 1 & 1 & 0 & 1 & 2 & 1 \\ \hline
\end{array}
*
\begin{array}{|c|c|c|}
\hline
1 & 0 & 1 \\ \hline
0 & 1 & 2 \\ \hline
0 & 1 & 0 \\ \hline
\end{array}
=
\begin{array}{|c|c|c|c|c|}
\hline
2 & 2 & 5 & 8 & 9 \\ \hline
2 & 3 & 8 & 8 & 8 \\ \hline
3 & 5 & 5 & 5 & 8 \\ \hline
5 & 3 & 4 & 6 & 8 \\ \hline
5 & 3 & 2 & 7 & 6 \\ \hline
\end{array}
$$

Symmetric padding

Pad using symmetry

$$
\begin{array}{|c|c|c|c|c|c|c|}
\hline
0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline
0 & 0 & 1 & 0 & 1 & 1 & 1 \\ \hline
0 & 0 & 0 & 1 & 2 & 2 & 2 \\ \hline
0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline
1 & 1 & 1 & 0 & 1 & 2 & 2 \\ \hline
0 & 0 & 1 & 0 & 0 & 2 & 2 \\ \hline
0 & 0 & 1 & 0 & 0 & 2 & 2 \\ \hline
\end{array}
*
\begin{array}{|c|c|c|}
\hline
1 & 0 & 1 \\ \hline
0 & 1 & 2 \\ \hline
0 & 1 & 0 \\ \hline
\end{array}
=
\begin{array}{|c|c|c|c|c|}
\hline
3 & 1 & 5 & 6 & 7 \\ \hline
1 & 3 & 8 & 8 & 8 \\ \hline
3 & 5 & 5 & 5 & 6 \\ \hline
4 & 3 & 4 & 6 & 9 \\ \hline
4 & 3 & 2 & 6 & 11 \\ \hline
\end{array}
$$

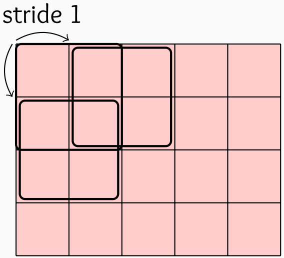
Stride

Input size is either slowly decreasing or constant. How can we reduce input size?

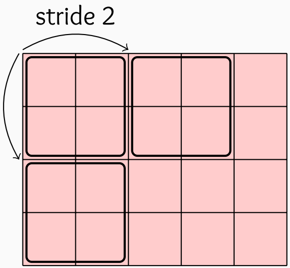
- **Strided convolution:** increasing step
- **Pooling:** summarize locally

Strided convolution

- Shifting by more than one step



No stride



Stride 2

- Strided convolution is equivalent to classic convolution + subsampling

Examples

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
0 & 1 & 0 & 1 & 1 & 0 \\ \hline
0 & 1 & 2 & 2 & 0 & 1 \\ \hline
1 & 1 & 0 & 1 & 1 & 0 \\ \hline
1 & 2 & 0 & 1 & 0 & 0 \\ \hline
2 & 1 & 0 & 1 & 0 & 1 \\ \hline
2 & 0 & 1 & 0 & 1 & 1 \\ \hline
\end{array}
*
\begin{array}{|c|c|}
\hline
0 & 2 \\ \hline
0 & 2 \\ \hline
\end{array}
=
$$

$$
\begin{array}{|c|c|c|c|c|}
\hline
4 & 4 & 6 & 2 & 2 \\ \hline
4 & 4 & 6 & 2 & 2 \\ \hline
6 & 0 & 4 & 2 & 0 \\ \hline
6 & 0 & 4 & 0 & 2 \\ \hline
2 & 2 & 2 & 2 & 4 \\ \hline
\end{array}
$$

Stride 1

$$
\begin{array}{|c|c|c|}
\hline
4 & 6 & 2 \\ \hline
6 & 4 & 0 \\ \hline
2 & 2 & 4 \\ \hline
\end{array}
$$

Stride 2

$$
\begin{array}{|c|c|}
\hline
4 & 2 \\ \hline
6 & 0 \\ \hline
\end{array}
$$

Stride 3

Stride formula, no padding

- Kernel k_h, k_w and stride s_h, s_w

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} - k_h + s_h}{s_h} \right\rfloor$$
$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} - k_w + s_w}{s_w} \right\rfloor$$

- No stride ($s_h = s_w = 1$) yields previous formula
- Input size is divided by stride: $H_{\text{out}} \sim \frac{1}{s_h} H_{\text{in}}$ and $W_{\text{out}} \sim \frac{1}{s_w} W_{\text{in}}$

Padding and stride formula

- Kernel k_h, k_w, padding p_h, p_w and stride s_h, s_w

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} - k_h + p_h + s_h}{s_h} \right\rfloor$$
$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} - k_w + p_w + s_w}{s_w} \right\rfloor$$

Pooling

Locally summarizing data:

- Same mechanism as for convolution
- No kernel, just a parameterless function operating on a window

Two functions are used:

- Max-pooling
- Average-pooling

Max-pooling

- Take maximum value in window:

$$\text{MaxPool}\left(\begin{array}{|c|c|c|c|c|c|}\hline 0 & 1 & 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 2 & 2 & 0 & 1 \\ \hline 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 1 & 2 & 0 & 1 & 0 & 0 \\ \hline 2 & 1 & 0 & 1 & 0 & 1 \\ \hline 2 & 0 & 1 & 0 & 1 & 1 \\ \hline \end{array}, \text{window\_size} = (2, 2)\right) = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 1 & 1 \\ \hline 2 & 1 & 1 \\ \hline \end{array}$$

- Usually the stride is equal to the kernel size

Average pooling

- Take average value in window:

$$\text{AvgPool}\left(\begin{array}{|c|c|c|c|c|c|}\hline 0 & 1 & 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 2 & 2 & 0 & 1 \\ \hline 1 & 1 & 0 & 1 & 1 & 0 \\ \hline 1 & 2 & 0 & 1 & 0 & 0 \\ \hline 2 & 1 & 0 & 1 & 0 & 1 \\ \hline 2 & 0 & 1 & 0 & 1 & 1 \\ \hline \end{array}, \text{window\_size} = (3,3)\right) = \begin{array}{|c|c|}\hline 0.67 & 0.78 \\ \hline 1.0 & 0.56 \\ \hline \end{array}$$

- Usually the stride is equal to the kernel size

3-*D* convolution

From 2-D convolution to 3-D convolution

Both C and K are now 3-D tensors with **same number of channels:**

- 3-D convolution is the sum of 2-D convolutions channel-wise

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 1 \\ 2 & 2 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}}_{C} * \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}}_{K} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 2 & 2 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 1 & 0 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 0 & 2 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \underbrace{\begin{bmatrix} 2 & 0 & 2 \\ 3 & 0 & 2 \\ 1 & 1 & 3 \end{bmatrix}}_{C * K}$$

- Whatever the number of channels there is only one channel after 3-D convolution!

From 2-D convolution to 3-D convolution

Mathematical formulation

- As a sum of simple 2-D convolutions channel-wise

$$C * K = \sum_{k=0,\ldots,c_{in}-1} K_{\cdot\cdot k} * C_{\cdot\cdot k}$$

- Expanded version

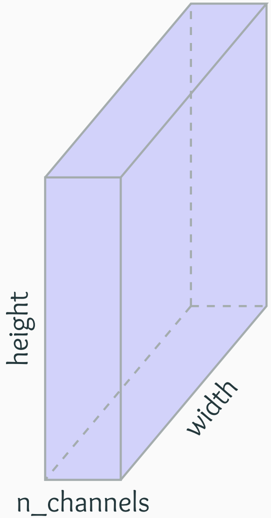
$$(C * K)_{ab} = \sum_{\substack{i=0,\ldots,k_h-1 \\ j=0,\ldots,k_w-1 \\ k=0,\ldots,c_{in}-1}} K_{ijk} C_{i+a,j+b,k}$$

- Result is a 2-D tensor because C and K have the same number of channels

3-D input representation

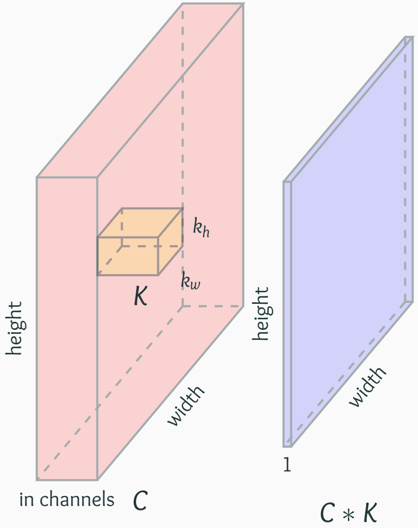
Input tensor is represented as a block of size:
$height \times width \times n\_channels$

- Input is a color image
  - n_channels $= 3$
- Input is a grayscale image
  - n_channels $= 1$



Represention of 3-D convolution

- Input tensor is represented as a 3-D block of size *height* $\times$ *width* $\times$ *in channels*
- Output is 1 channel wide

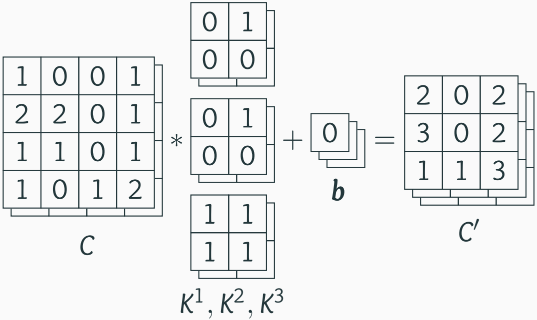


Convolutional layer

Convolutional layer

A **convolutional layer** consists in several 3-D convolutions + bias stacked as channels:

- C' gathers 3-D convolution with filters $K^1, \ldots, K^{C_{out}}$
- Channels of C' are called **feature maps**
- Kernel + bias is called a **filter**
- Number of out channels is number of filters



Convolutional layer

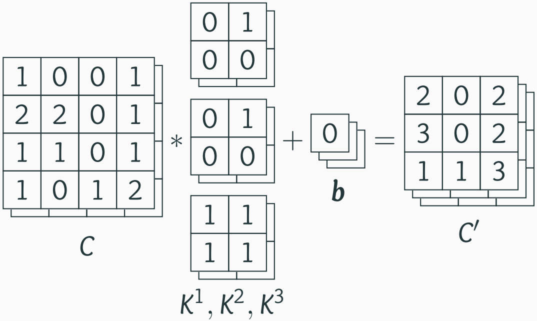
Mathematical formulation

- Per output channel

$$C'_{\cdot\cdot c} = b^c + K^c * C$$

- Expanded version

$$C'_{abc} = b^c + \sum_{\substack{i=0,\dots,k_h-1 \\ j=0,\dots,k_w-1 \\ k=0,\dots,c_{in}-1}} K^c_{ijk} C_{i+a,j+b,k}$$

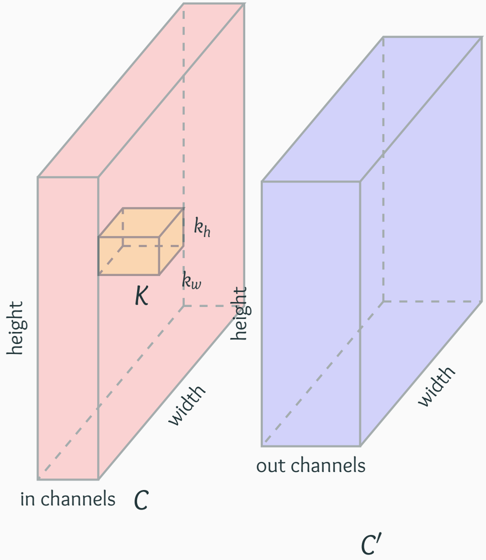


3-D representation of a convolutional layer

- Convolutional layers are often represented as consecutive blocks of size $height \times width \times channels$
- Only one kernel is represented
- Number of learnable parameters is

$$(k_h \times k_w \times c_{in} + 1) \times c_{out}$$

- Biases are not represented



First convolutional networks

LeNet-5 from LeCun et al. 1998

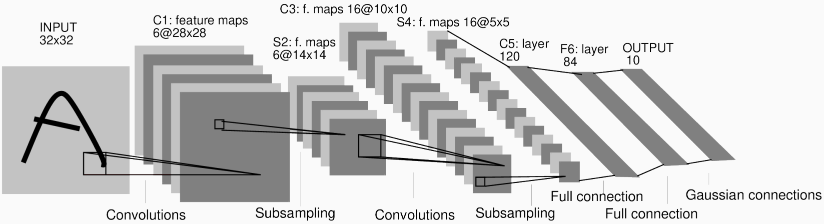
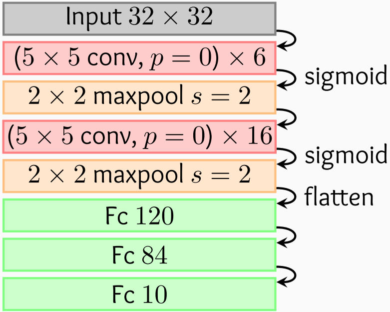


Figure 1: From LeCun et al. 1998

- Consists in two parts:
    - Features: 2 convolutional layers
    - Classification: 3 fully connected layers

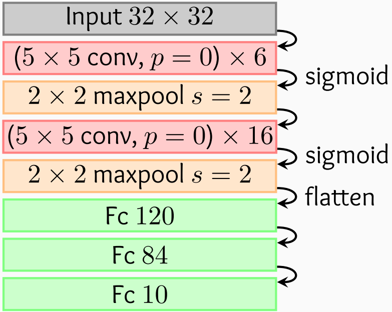
LeNet-5

- Parameters: 60k
- Activation function: sigmoid
- 5 weight layers



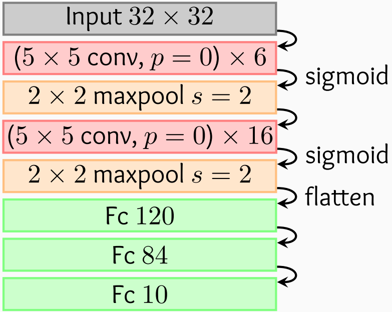
LeNet-5 number of parameters

- First layer: $32 \times 32 \times 1 \to 28 \times 28 \times 6$
  - 6 filters of size $5 \times 5 \times 1$
  - # of parameters is $(5 \times 5 \times 1 + 1) \times 6 = 156$
- Second layer: $14 \times 14 \times 6 \to 10 \times 10 \times 16$
  - 16 filters of size $5 \times 5 \times 6$
  - # of parameters is $(5 \times 5 \times 6 + 1) \times 16 = 2416$



LeNet-5 number of parameters

- Third layer: flattened $5 \times 5 \times 16 \to 120$
  $(5 \times 5 \times 16 + 1) \times 120 = 48120$
- Fourth layer: $120 \to 84$
  $(120 + 1) \times 84 = 10164$
- Fifth layer: $84 \to 10$
  $(84 + 1) \times 10 = 850$
- Total # of parameters: $61706 \approx 60k$

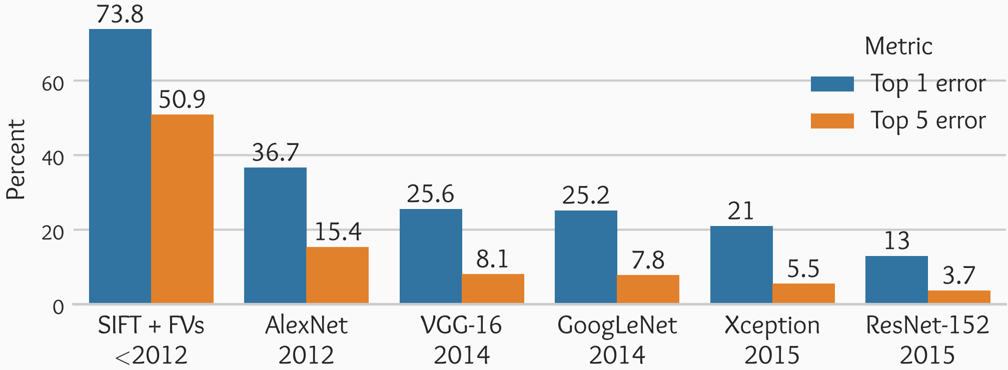


Modern convolutional networks

The ImageNet challenge from Russakovsky et al. 2015

- Since 2010 the Imagenet dataset is used in a the ILSVRC challenge (Large Scale Visual Recognition Challenge)
- Object classification/detection
- Classification task:
    - $>$ 1.2M annotated images of various size
    - 1000 classes

Classification error on ImageNet



AlexNet from Krizhevsky, Sutskever, and Hinton 2012

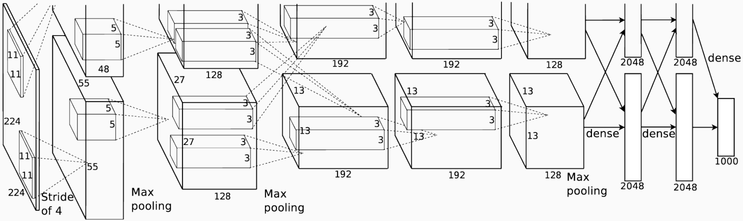
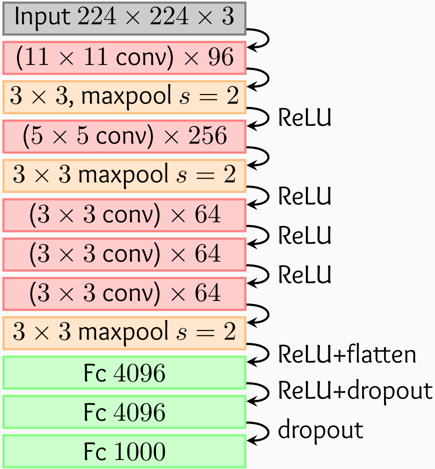


Figure 2: From Krizhevsky, Sutskever, and Hinton 2012

- Won ILSVRC 2012 by a large margin!

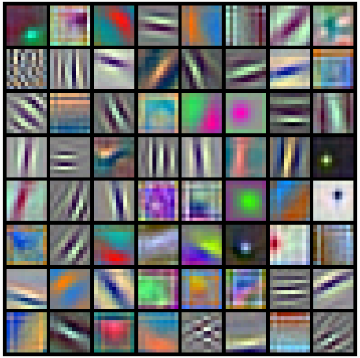
AlexNet from Krizhevsky, Sutskever, and Hinton 2012

- Number of parameters: 60M
- Deeper than LeNet
- ReLU activation instead of sigmoid
- 8 learnable layers



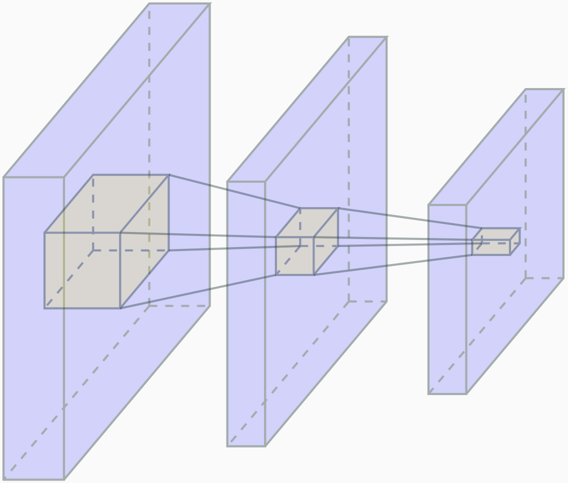
AlexNet: first layer filters

- Learned filters are Gabor-like
- 64 filters of size 11×11



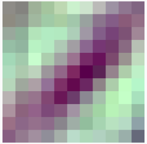
Receptive field

- Given a feature the receptive field is the window in the input that created that feature.



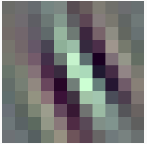
AlexNet: receptive fields

Some $11 \times 11 \times 3$ filters and 9 receptive fields corresponding to best activation across all training set:

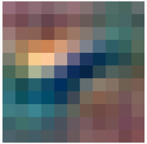


(a) filter #7

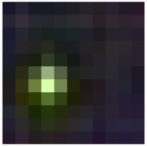
(b) filter #10



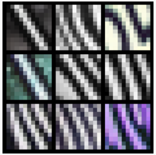
(c) filter #19

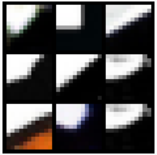


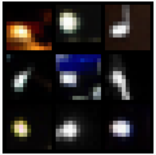
(d) filter #27



(e) filter #32







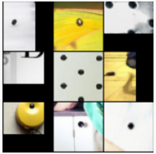
AlexNet: receptive fields

Receptive fields of best activations in feature maps

- Second convolutional layer: 51×51 receptive field

(a) filter #25

(b) filter #41



(c) filter #107

- Third convolutional layer: 99×99 receptive field

(a) filter #90

(b) filter #165

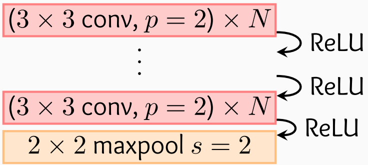
(c) filter #377

VGG networks from Simonyan and Zisserman 2015

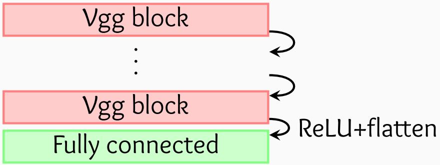
Evolution from AlexNet:

- Replace 11×11 by sequence of 3×3
- Use a block that is repeated
- Same fully connected layers

VGG block with N filters:



Sequence of VGG blocks:



VGG-16 from Simonyan and Zisserman 2015

- Example of VGG-16:
  - 16 weight layers
  - 133–144 M parameters
- Drawbacks:
  - Too many parameters
  - Hard to train

Input $224 \times 224 \times 3$

$(3 \times 3$ conv, $p = 2) \times 64$ $\times 2$

2×2 maxpool $s = 2$

$(3 \times 3$ conv, $p = 2) \times 128$ $\times 2$

2×2 maxpool $s = 2$

$(3 \times 3$ conv, $p = 2) \times 256$ $\times 3$

2×2 maxpool $s = 2$

$(3 \times 3$ conv, $p = 2) \times 512$ $\times 3$

2×2 maxpool $s = 2$

$(3 \times 3$ conv, $p = 2) \times 512$ $\times 3$

Fc 4096

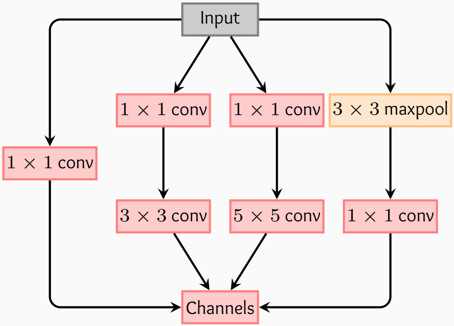
Fc 4096

Fc 1000

GoogLeNet (Inception-v1) from Szegedy et al. 2015

GoogleNet won ILSVRC 2015, main ingredients are:

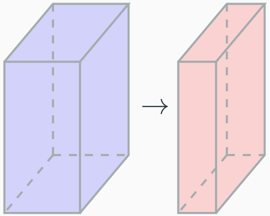
- Use 1×1 *convolution*
- Use *global average pooling* instead of fully connected layers
- Propose an *inception module* implementing a *split-transform-merge* strategy:
    - Mix filters of different sizes
    - Height and width unchanged
    - Concatenated along channel dimension
- Parametrized by 6 hyperparameters



1×1 convolution from Lin, Chen, and Yan 2014

Convolution with a kernel of size 1×1

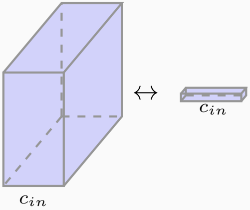
- Properties:
  - No spatial transformation
  - Height and width are unchanged
  - Change de number of channels
  - Each output channel is a linear combination of input channels
- Can be used to:
  - Reduce the number of channels
  - Reduce number of parameters
  - Apply an MLP pixel-wise



Global average pooling from Lin, Chen, and Yan 2014

Average pooling with maximum window

- Properties
  - Same as averaging each channel
  - $H_{\text{in}} \times W_{\text{in}} \times c_{in}$ becomes $1 \times 1 \times c_{in}$
- Is used to
  - Replace flatten + fully connected layer



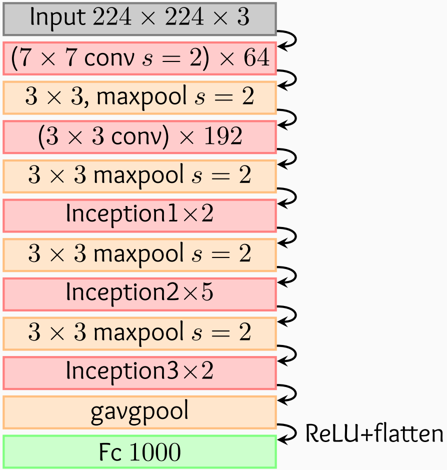
GoogLeNet (Inception-v1)

Inception-v1:

- Parameters $\simeq$ 6.8 M
- ReLU activation

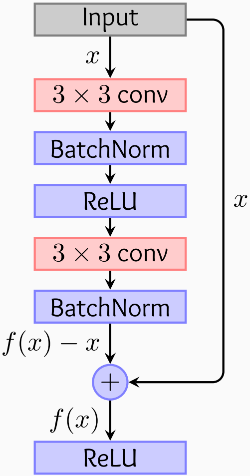
Improvements (Inception-v2, Inception-v3)

- Replace 5×5 by two 3×3 convolution layers
- Spatially separable convolutions
- Batch normalization



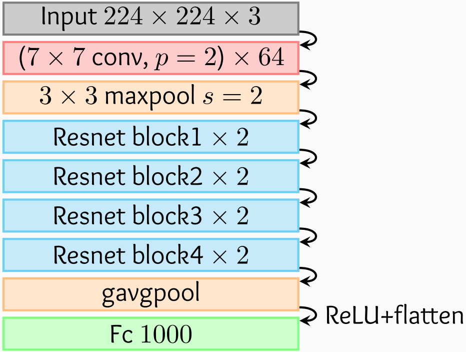
Residual Networks (ResNets) from He et al. 2016

- Use **skip connections** around VGG-like block
- Learn residual mapping instead of full mapping



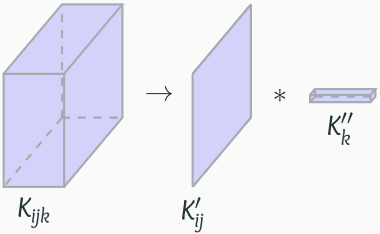
Resnet-18

- 18 learnable layers
- 11M parameters
- Deeper models by changing multipliers



Depthwise Separable Convolution

- Make convolution separable to reduce parameters:
$$K_{ijk} \quad \to \quad K'_{ij} * K''_{k}$$
    - K'_{ij} is applied to each channel
    - K''_{k} is a 1×1 convolution
- Number of parameters:
$$k_h k_w c_{in} \quad \to \quad k_h k_w + c_{in}$$



[1] Yann LeCun et al. **"Gradient-Based Learning Applied to Document Recognition".** In: *PROC. OF THE IEEE* (1998), p. 1.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. **"Imagenet Classification with Deep Convolutional Neural Networks".** In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[3] Min Lin, Qiang Chen, and Shuicheng Yan. **"Network In Network".** Mar. 4, 2014. arXiv: `1312.4400 [cs]`. URL: `http://arxiv.org/abs/1312.4400` (visited on 08/24/2021).

[4] Olga Russakovsky et al. **"ImageNet Large Scale Visual Recognition Challenge".** Jan. 29, 2015. arXiv: `1409.0575 [cs]`. URL: `http://arxiv.org/abs/1409.0575` (visited on 11/23/2021).

[5] Karen Simonyan and Andrew Zisserman. **"Very Deep Convolutional Networks for Large-Scale Image Recognition".** Apr. 10, 2015. arXiv: 1409.1556 [cs]. URL: http://arxiv.org/abs/1409.1556 (visited on 08/30/2021).

[6] Christian Szegedy et al. **"Going Deeper with Convolutions".** In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[7] Kaiming He et al. **"Deep Residual Learning for Image Recognition".** In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778. arXiv: 1512.03385.