Deep learning

4.4. Convolutions

François Fleuret

https://fleuret.org/dlc/
If they were handled as normal “unstructured” vectors, large-dimension signals such as sound samples or images would require models of intractable size.

For instance a linear layer taking a $256 \times 256$ RGB image as input, and producing an image of same size would require

$$(256 \times 256 \times 3)^2 \approx 3.87 \times 10^{10}$$

parameters, with the corresponding memory footprint ($\approx 150$Gb!), and excess of capacity.
Moreover, this requirement is inconsistent with the intuition that such large signals have some “invariance in translation”. A representation meaningful at a certain location can / should be used everywhere.
Moreover, this requirement is inconsistent with the intuition that such large signals have some “invariance in translation”. A representation meaningful at a certain location can / should be used everywhere.

A convolution layer embodies this idea. It applies the same linear transformation locally, everywhere, and preserves the signal structure.
\[
W - w + 1
\]

**Input**

\[
\begin{array}{cccccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1
\end{array}
\]
Input

1  4  -1  0  2  -2  1  3  3  1

Kernel

1  2  0  -1

Output

W - w + 1

François Fleuret
Deep learning / 4.4. Convolutions

3 / 23
\[ W - w + 1 \]
### 4.4. Convolutions

**Input**

| 1 | 4 | -1 | 0 | 2 | -2 | 1 | 3 | 3 | 1 |

**Kernel**

| 1 | 2 | 0 | -1 |

**Output**

| 9 | 0 |

$$W - w + 1$$
Input

\[
\begin{array}{cccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1 \\
\end{array}
\]

\[W\]

Kernel

\[
\begin{array}{cccc}
1 & 2 & 0 & -1 \\
\end{array}
\]

\[w\]

Output

\[
\begin{array}{ccc}
9 & 0 & 1 \\
\end{array}
\]

\[W - w + 1\]
\[ W - w + 1 \]

Input:

\[
\begin{array}{cccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1
\end{array}
\]

Kernel:

\[
\begin{array}{cccc}
1 & 2 & 0 & -1
\end{array}
\]

Output:

\[
\begin{array}{cccc}
9 & 0 & 1 & 3
\end{array}
\]
Input

\[ \begin{array}{cccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1
\end{array} \]

\[ W \]

\[ w \]

Output

\[ \begin{array}{cccc}
9 & 0 & 1 & 3 & -5
\end{array} \]

\[ W - w + 1 \]
Input

\[
\begin{array}{cccccccc}
1 & 4 & -1 & 0 & 2 & -2 & 1 & 3 & 3 & 1 \\
\end{array}
\]

\[W\]

\[w\]

Output

\[
\begin{array}{cccccc}
9 & 0 & 1 & 3 & -5 & -3 \\
\end{array}
\]

\[W - w + 1\]
\[ W - w + 1 \]
The diagram illustrates the convolution operation in the context of deep learning. The input sequence is 1, 4, -1, 0, 2, -2, 1, 3, 3, 1. A kernel of size $w = 3$ slides across the input, and the output is calculated as $W - w + 1$. The kernel used is [1, 2, 0, -1], and the resulting output sequence is 9, 0, 1, 3, -5, -3, 6.
Formally, in 1d, given

\[ x = (x_1, \ldots, x_W) \]

and a “convolution kernel” (or “filter”) of width \( w \)

\[ u = (u_1, \ldots, u_w) \]
Formally, in 1d, given

\[ x = (x_1, \ldots, x_W) \]

and a “convolution kernel” (or “filter”) of width \( w \)

\[ u = (u_1, \ldots, u_w) \]

the convolution \( x \otimes u \) is a vector of size \( W - w + 1 \), with

\[
(x \otimes u)_i = \sum_{j=1}^{w} x_{i-1+j} u_j \\
= (x_i, \ldots, x_{i+w-1}) \cdot u
\]

for instance

\[
(1, 2, 3, 4) \otimes (3, 2) = (3 + 4, 6 + 6, 9 + 8) = (7, 12, 17).
\]
Formally, in 1d, given
\[ x = (x_1, \ldots, x_W) \]
and a “convolution kernel” (or “filter”) of width \( w \)
\[ u = (u_1, \ldots, u_w) \]
the convolution \( x \odot u \) is a vector of size \( W - w + 1 \), with
\[
(x \odot u)_i = \sum_{j=1}^{w} x_{i-1+j} \cdot u_j
\]
\[ = (x_i, \ldots, x_{i+w-1}) \cdot u \]
for instance
\[
(1, 2, 3, 4) \odot (3, 2) = (3 + 4, 6 + 6, 9 + 8) = (7, 12, 17).
\]

⚠️ This differs from the usual convolution since the kernel and the signal are both visited in increasing index order.
Convolution can implement in particular differential operators, e.g.

\[(0, 0, 0, 0, 1, 2, 3, 4, 4, 4) \ast (-1, 1) = (0, 0, 0, 1, 1, 1, 0, 0, 0, 0)\].
Convolution can implement in particular differential operators, e.g.

\[(0, 0, 0, 0, 1, 2, 3, 4, 4, 4) \ast (-1, 1) = (0, 0, 0, 1, 1, 1, 0, 0, 0).\]
Convolution can implement in particular differential operators, e.g.

\[(0, 0, 0, 0, 1, 2, 3, 4, 4, 4) \ast (-1, 1) = (0, 0, 0, 1, 1, 1, 0, 0, 0)\].

or crude “template matcher”, e.g.
It generalizes naturally to a multi-dimensional input, although specification can become complicated.
It generalizes naturally to a multi-dimensional input, although specification can become complicated.

Its most usual form for “convolutional networks” processes a 3d tensor as input (i.e. a multi-channel 2d signal) to output a 2d tensor. The kernel is not swiped across channels, just across rows and columns.

In this case, if the input tensor is of size $C \times H \times W$, and the kernel is $C \times h \times w$, the output is $(H - h + 1) \times (W - w + 1)$. 
It generalizes naturally to a multi-dimensional input, although specification can become complicated.

Its most usual form for “convolutional networks” processes a 3d tensor as input (i.e. a multi-channel 2d signal) to output a 2d tensor. The kernel is not swiped across channels, just across rows and columns.

In this case, if the input tensor is of size $C \times H \times W$, and the kernel is $C \times h \times w$, the output is $(H - h + 1) \times (W - w + 1)$.

⚠️ We say “2d signal” even though it has $C$ channels, since it is a feature vector indexed by a 2d location without structure on the feature indexes.
It generalizes naturally to a multi-dimensional input, although specification can become complicated.

Its most usual form for “convolutional networks” processes a 3d tensor as input (i.e. a multi-channel 2d signal) to output a 2d tensor. The kernel is not swiped across channels, just across rows and columns.

In this case, if the input tensor is of size $C \times H \times W$, and the kernel is $C \times h \times w$, the output is $(H - h + 1) \times (W - w + 1)$.

⚠️ We say “2d signal” even though it has $C$ channels, since it is a feature vector indexed by a 2d location without structure on the feature indexes.

In a standard convolution layer, $D$ such convolutions are combined to generate a $D \times (H - h + 1) \times (W - w + 1)$ output.
Kernel $K$:

$$D - h + 1$$

$$H - w + 1$$

Output $C = \frac{W - w + 1}{d}$$
\begin{align*}
    D & - h + 1 \\
    W & - w + 1
\end{align*}
François Fleuret

Deep learning / 4.4. Convolutions
Input

Kernel

Output

\[ D - h + 1 \times W - w + 1 \]
Kernel $K$ of size $h \times w \times C$ is applied to the input of size $H \times W \times C$ to produce the output of size $D \times (W-w+1) \times (H-h+1) \times C$. The convolution operation then slides the kernel across the input, computing the dot product at each position.
The formula for the number of output channels is given by:

\[ D - h + 1 \]
Kernel

\[ D - h + 1 \]

\[ W - w + 1 \]

\[ 1 \]
Kernel $K$:

$$h - h + 1 \times w - w + 1$$

Input $I$:

Output $O$:

François Fleuret

Deep learning / 4.4. Convolutions
$D - h + 1$

$W - w + 1$

$H$

$W$

$C$

$h$

$w$

$C$

Input

Kernel

Output
The equation for the output of a convolution is:

\[ D - h + 1 \]

\[ H - w + 1 \]

\[ W - w + 1 \]

where:
- \( D \) is the depth of the input tensor.
- \( H \) is the height of the input tensor.
- \( W \) is the width of the input tensor.
- \( h \) is the height of the kernel.
- \( w \) is the width of the kernel.
- \( C \) is the number of channels.

The convolution process can be visualized as sliding the kernel over the input tensor and performing element-wise multiplication and summation at each position.
Kernel

\[ D - h + 1 \]

\[ W - w + 1 \]

\[ H - h + 1 \]

Input

Output
Kernel

\[ K = \begin{bmatrix} D \\ H \end{bmatrix} - h + 1 \]

\[ W = W - w + 1 \]

\[ H = H - h + 1 \]
Affine

Input

$\begin{align*}
H &- h + 1 \\
W &- w + 1
\end{align*}$

Output

$\begin{align*}
H - h + 1 \\
W - w + 1
\end{align*}$

$C$

$D$
Note that a convolution preserves the signal support structure.

A 1d signal is converted into a 1d signal, a 2d signal into a 2d, and neighboring parts of the input signal influence neighboring parts of the output signal.
Note that a convolution preserves the signal support structure.

A 1d signal is converted into a 1d signal, a 2d signal into a 2d, and neighboring parts of the input signal influence neighboring parts of the output signal.

A 3d convolution can be used if the channel index has some metric meaning, such as time for a series of grayscale video frames. Otherwise swiping across channels makes no sense.
We usually refer to one of the channels generated by a convolution layer as an activation map.

The sub-area of an input map that influences a component of the output as the receptive field of the latter.
We usually refer to one of the channels generated by a convolution layer as an **activation map**.

The sub-area of an input map that influences a component of the output as the **receptive field** of the latter.

In the context of convolutional networks, a standard linear layer is called a **fully connected layer**, sometime a **dense layer**, since every input influences every output.
The autograd-compliant function

\[
\text{F.conv2d}(\text{input}, \text{weight}, \text{bias}={\text{None}}, \text{stride}=1, \text{padding}=0, \text{dilation}=1, \text{groups}=1)
\]

Implements a 2d convolution, where \text{weight} is of dimension \( D \times C \times h \times w \) and contains the kernels, \text{bias} is of dimension \( D \), \text{input} is of dimension

\[
N \times C \times H \times W
\]

and the result is of dimension

\[
N \times D \times (H - h + 1) \times (W - w + 1).
\]
The autograd-compliant function

\[
F.\text{conv2d}(\text{input}, \text{weight}, \text{bias}=\text{None}, \text{stride}=1, \text{padding}=0, \text{dilation}=1, \text{groups}=1)
\]

Implements a 2d convolution, where \text{weight} is of dimension \( D \times C \times h \times w \)
and contains the kernels, \text{bias} is of dimension \( D \), \text{input} is of dimension

\[
N \times C \times H \times W
\]

and the result is of dimension

\[
N \times D \times (H - h + 1) \times (W - w + 1).
\]

```python
>>> weight = torch.empty(5, 4, 2, 3).normal_()
>>> bias = torch.empty(5).normal_()
>>> input = torch.empty(117, 4, 10, 3).normal_()
>>> output = F.conv2d(input, weight, bias)
>>> output.size()
torch.Size([117, 5, 9, 1])
```
The autograd-compliant function

```
F.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)
```

Implements a 2d convolution, where `weight` is of dimension $D \times C \times h \times w$
and contains the kernels, `bias` is of dimension $D$, `input` is of dimension

$$N \times C \times H \times W$$

and the result is of dimension

$$N \times D \times (H - h + 1) \times (W - w + 1).$$

```py
>>> weight = torch.empty(5, 4, 2, 3).normal_()
>>> bias = torch.empty(5).normal_()
>>> input = torch.empty(117, 4, 10, 3).normal_()
>>> output = F.conv2d(input, weight, bias)
>>> output.size()
torch.Size([117, 5, 9, 1])
```

Similar functions implement 1d and 3d convolutions.
\[
x = \text{mnist\_train\_data[12].float().view(1, 1, 28, 28)}
\]

\[
\text{weight} = \text{torch\_empty(5, 1, 3, 3)}
\]

\[
\text{weight[0, 0]} = \text{torch\_tensor([[ 0., 0., 0. ],
[ 0., 1., 0. ],
[ 0., 0., 0. ]])}
\]

\[
\text{weight[1, 0]} = \text{torch\_tensor([[ 1., 1., 1. ],
[ 1., 1., 1. ],
[ 1., 1., 1. ]])}
\]

\[
\text{weight[2, 0]} = \text{torch\_tensor([[ -1., 0., 1. ],
[ -1., 0., 1. ],
[ -1., 0., 1. ]])}
\]

\[
\text{weight[3, 0]} = \text{torch\_tensor([[ -1., -1., -1. ],
[ 0., 0., 0. ],
[ 1., 1., 1. ]])}
\]

\[
\text{weight[4, 0]} = \text{torch\_tensor([[ 0., -1., 0. ],
[ -1., 4., -1. ],
[ 0., -1., 0. ]])}
\]

\[
y = F.\text{conv2d}(x, \text{weight})
\]
Francois Fleuret

Deep learning / 4.4. Convolutions
Francois Fleuret
class torch.nn.Conv2d(in_channels, out_channels,
    kernel_size, stride=1, padding=0, dilation=1,
    groups=1, bias=True)

Wraps the convolution into a Module, with the kernels and biases as Parameter properly randomized at creation.

The kernel size is either a pair \((h, w)\) or a single value \(k\) interpreted as \((k, k)\).
```python
class torch.nn.Conv2d(in_channels, out_channels,
    kernel_size, stride=1, padding=0, dilation=1,
    groups=1, bias=True)
```

Wraps the convolution into a Module, with the kernels and biases as Parameter properly randomized at creation.

The kernel size is either a pair \((h, w)\) or a single value \(k\) interpreted as \((k, k)\).

```python
>>> f = nn.Conv2d(in_channels = 4, out_channels = 5, kernel_size = (2, 3))
>>> for n, p in f.named_parameters(): print(n, p.size())
... weight torch.Size([5, 4, 2, 3])
  bias torch.Size([5])
>>> x = torch.empty(117, 4, 10, 3).normal_()
>>> y = f(x)
>>> y.size()
torch.Size([117, 5, 9, 1])
```
Padding, stride, and dilation
Convolutions have three additional parameters:

- The **padding** specifies the size of a zeroed frame added around the input,
- the **stride** specifies a step size when moving the kernel across the signal,
- the **dilation** modulates the expansion of the filter without adding weights.
Here with $C \times 3 \times 5$ as input
Here with $C \times 3 \times 5$ as input, a padding of $(2,1)$
Here with $C \times 3 \times 5$ as input, a padding of $(2,1)$, a stride of $(2,2)$
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$. 

\begin{figure}
\centering
\includegraphics[width=\textwidth]{convolution_example.png}
\caption{Output of convolution with $C \times 3 \times 3$ kernel.}
\end{figure}
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$. 

Input

Output
Here with $C \times 3 \times 5$ as input, a padding of $(2,1)$, a stride of $(2,2)$, and a kernel of size $C \times 3 \times 3$.
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$.
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$.
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$.
Here with $C \times 3 \times 5$ as input, a padding of $(2,1)$, a stride of $(2,2)$, and a kernel of size $C \times 3 \times 3$.
Here with $C \times 3 \times 5$ as input, a padding of $(2, 1)$, a stride of $(2, 2)$, and a kernel of size $C \times 3 \times 3$.
Here with $C \times 3 \times 5$ as input, a padding of $(2,1)$, a stride of $(2,2)$, and a kernel of size $C \times 3 \times 3$, the output is $1 \times 3 \times 3$. 
A convolution with a stride greater than 1 may not cover the input map entirely, hence may ignore some of the input values.
The dilation modulates the expansion of the filter support by adding rows and
columns of zeros between coefficients (Yu and Koltun, 2015).

It is 1 for standard convolutions, but can be greater, in which case the resulting
operation can be envisioned as a convolution with a regularly sparsified filter.
The dilation modulates the expansion of the filter support by adding rows and columns of zeros between coefficients (Yu and Koltun, 2015).

It is 1 for standard convolutions, but can be greater, in which case the resulting operation can be envisioned as a convolution with a regularly sparsified filter.

This notion comes from signal processing, where it is referred to as *algorithme à trous*, hence the term sometime used of “convolution à trous”.
Dilation = 1

François Fleuret

Deep learning / 4.4. Convolutions
Dilation = 1

Input

Output
Dilation = 1
Input

Output

Dilation = 1
Dilation = 1
Dilation = 1
Dilation = 1

Francois Fleuret Deep learning / 4.4. Convolutions
Dilation $= 1$
Dilation = 2

Input

Output
Dilation = 2

Input

Output
Dilation = 2

Input

Output
Dilation = 2
Dilation = 2
Dilation = 2

Input

Output
Dilation = 2
Dilation = 2

Francois Fleuret Deep learning / 4.4. Convolutions
Dilation = 2

Francois Fleuret

Deep learning / 4.4. Convolutions
A convolution with a kernel of size $k$ and dilation $d$ can be interpreted as a convolution with a filter of size $1 + (k - 1)d$ with only $k$ non-zero coefficients.

For example with $k = 3$ and $d = 4$, the difference between the input map size and the output map size is $1 + (3 - 1)4 - 1 = 8$.

```python
>>> x = torch.empty(1, 1, 20, 30).normal_()
>>> l = nn.Conv2d(1, 1, kernel_size = 3, dilation = 4)
>>> l(x).size()
torch.Size([1, 1, 12, 22])
```
Having a dilation greater than one increases the units' receptive field size without increasing the number of parameters.

Convolutions with stride or dilation strictly greater than one reduce the activation map size, for instance to make a final classification decision.
The end
References