Neural Networks and Deep Learning

Convolutional Neural Networks

Credit to Dimosthenis Karatzas

COMPUTER VISION

Solving computer vision over the summer

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence Group July 7, 1966
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

A few of the things that proved to be challenging

Segmentation

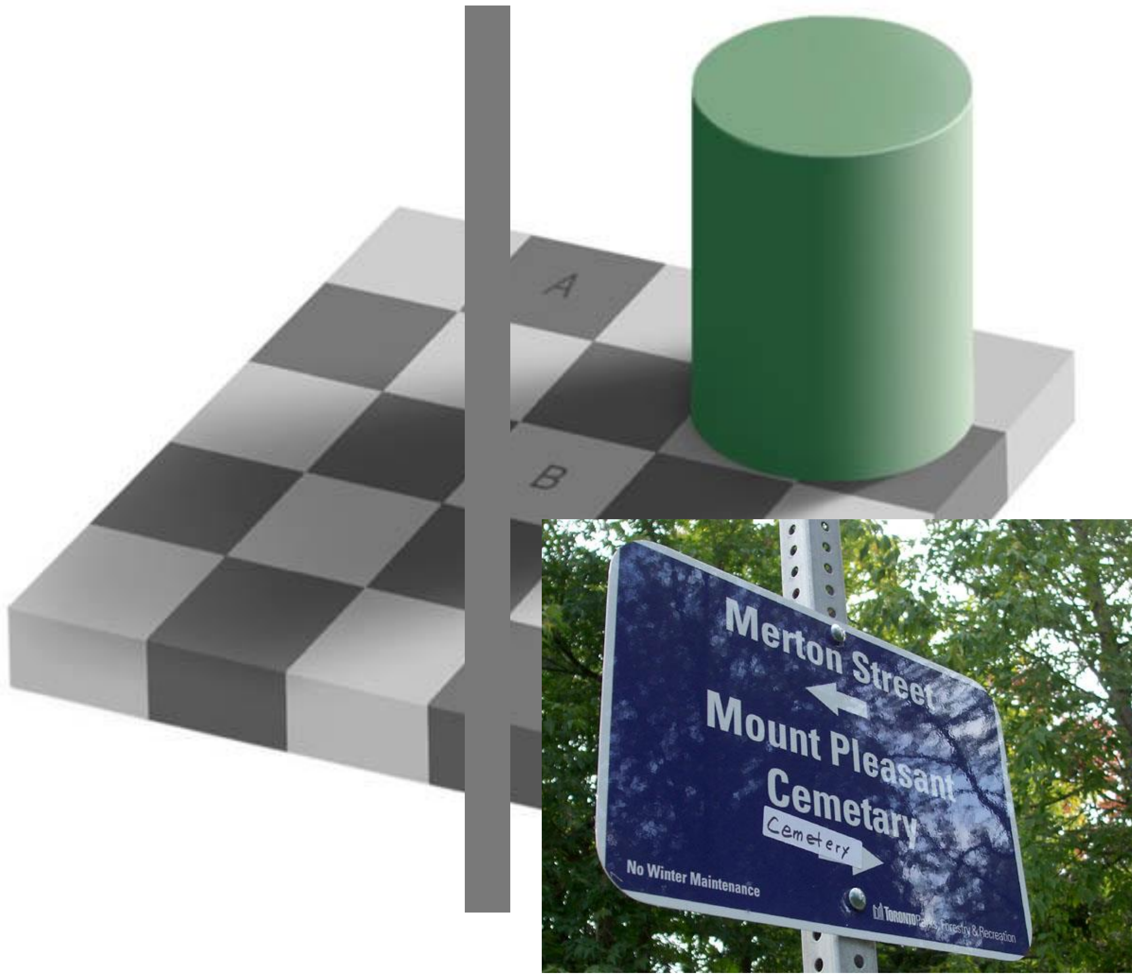
Where in the image is the object of interest? What pieces of the image go together? Which are the pixels that correspond to the object?

A few of the things that proved to be challenging

Segmentation

Lighting

The computer sees a matrix of values. The interpretation of these pixel intensities is not straightforward



A few of the things that proved to be challenging

Segmentation

Lighting

Deformations

Objects in real-life images should not be expected to appear in their canonical form

A few of the things that proved to be challenging

Segmentation

Lighting

Deformation

Class definition

Object classes are typically defined by their common affordances, not their visual similarity

Dealing with viewpoint changes

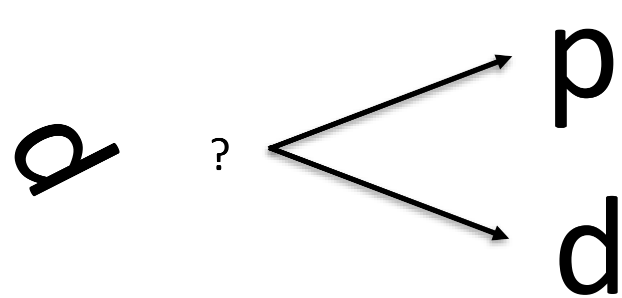
Segmentation-first approach

Try to localise the object and correct for the transformations

Attempt recognition only once the object is in its canonical form

Segmentation (localisation) is not easy – the brute force alternative is to try everything (see sliding window or object proposals)

In the general case we need to recognise in order to get the segmentation right…



Dealing with viewpoint changes

The Bag-of-Words approach

Do not attempt to localise or rectify the object

Extract instead features from all over the image (aim for redundancy) and “bag” them

Important for the features to be invariant under transformations

Ok for classifying a whole image, but difficult to separate different objects

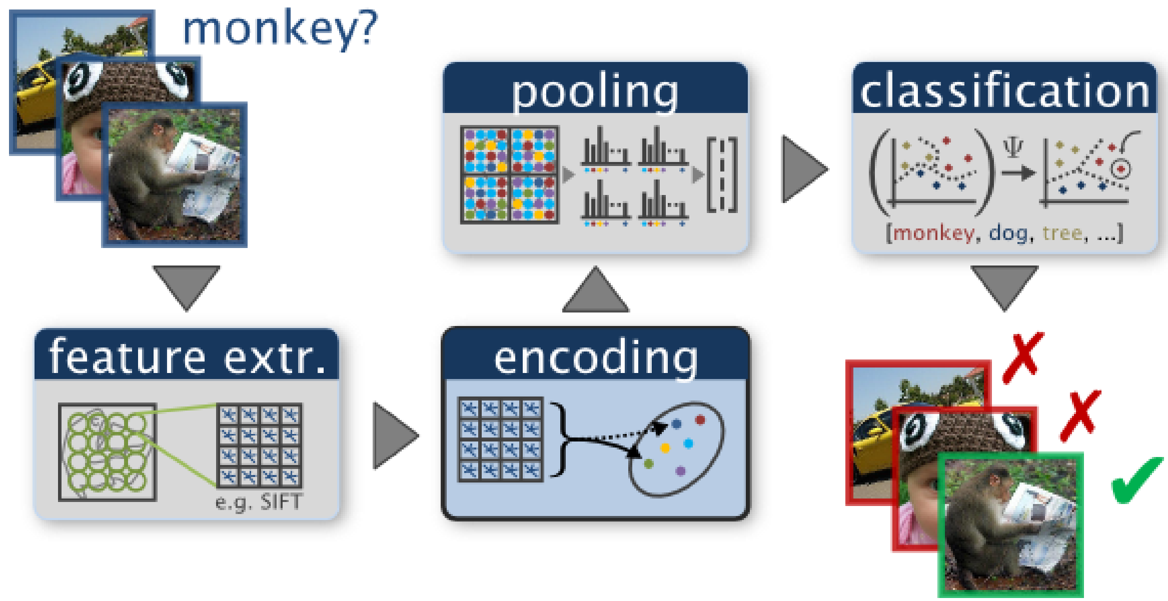
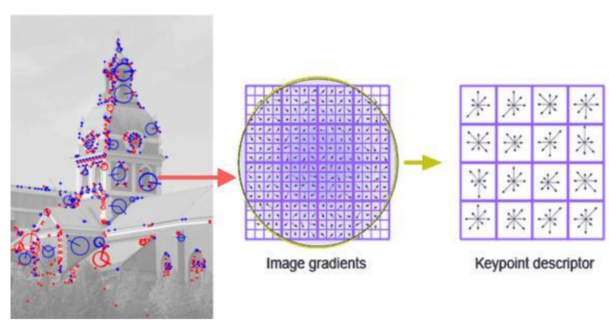


Image credit: Chatfield et al

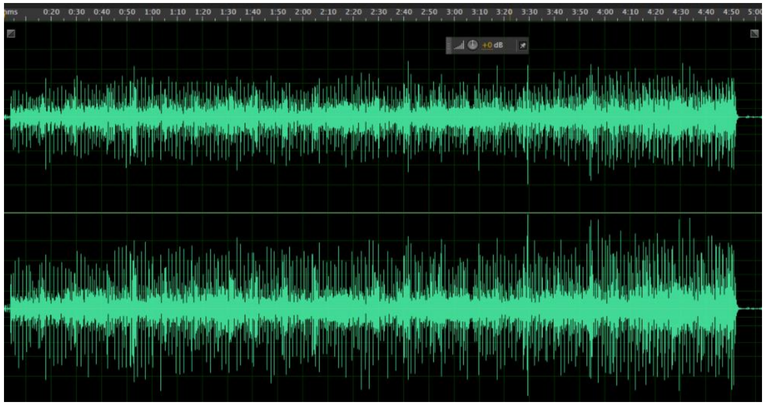


LIMITATIONS OF FULLY CONNECTED ARCHITECTURES

Intrinsic Structure

Images, sound clips, etc have an **intrinsic structure**:

- One or more axes for which ordering matters

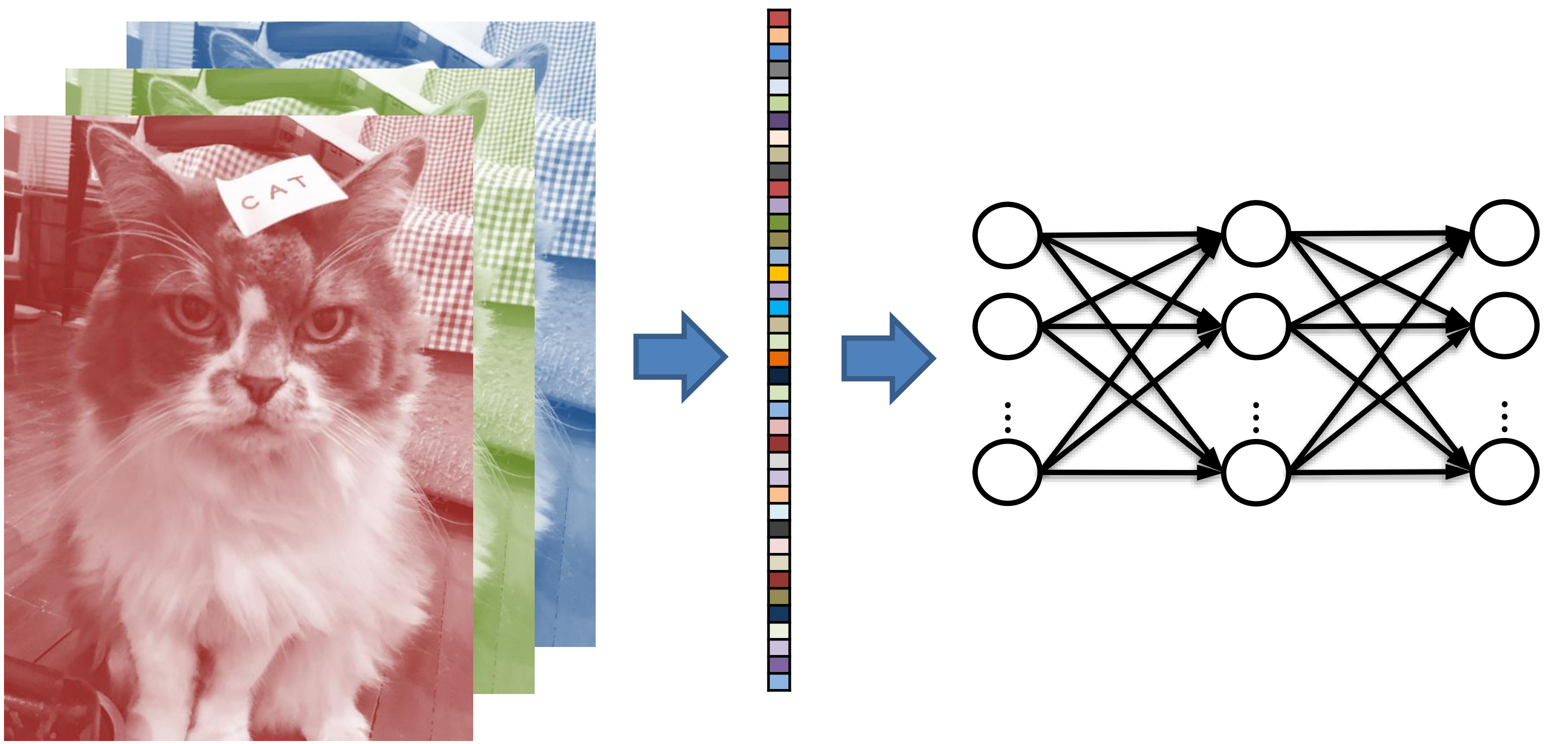


Intrinsic Structure

Images, sound clips, etc have an **intrinsic structure**:

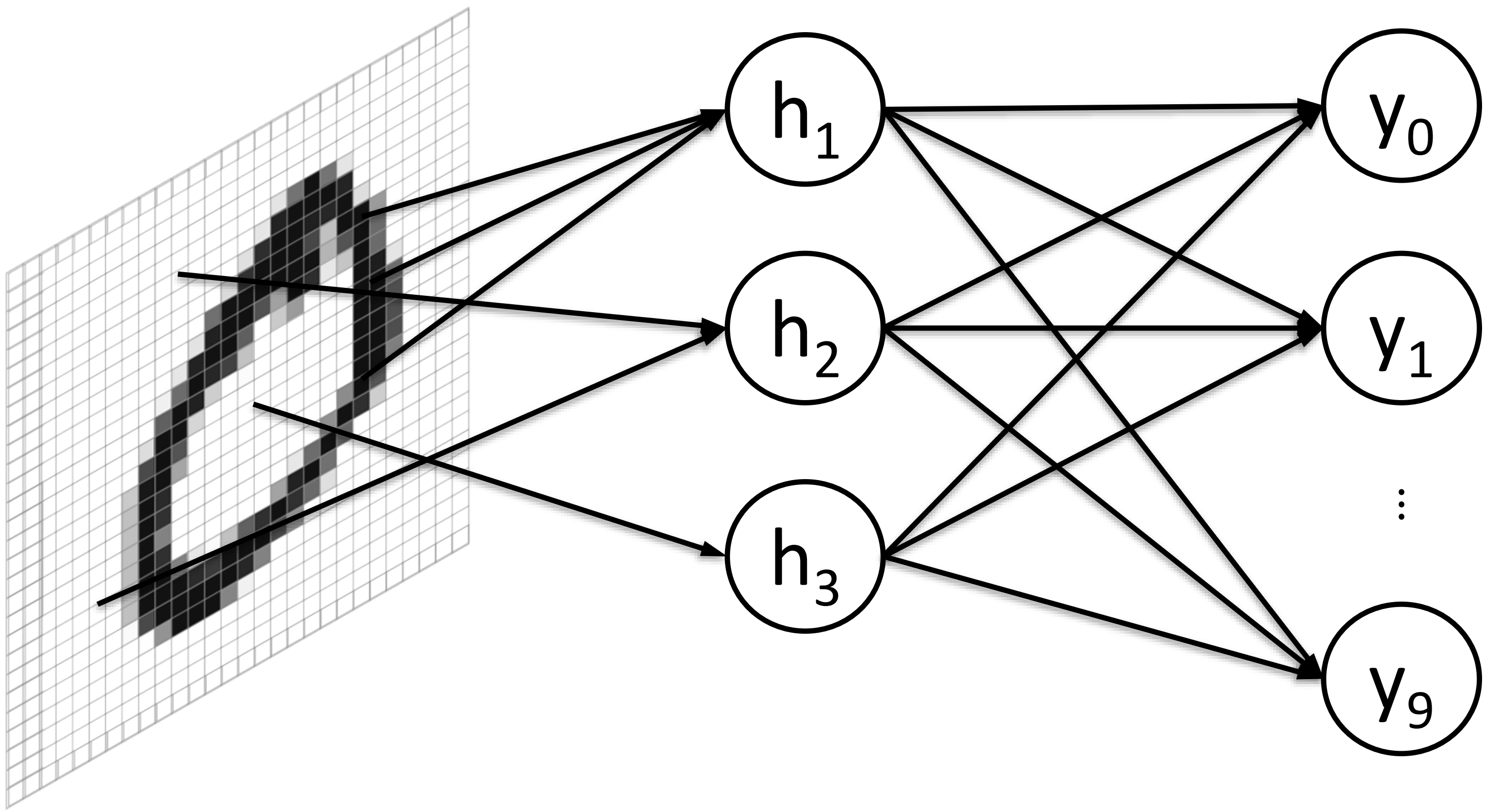
- One or more axes for which ordering matters
- One channel axis for different views of the data

MLPs have **no notion of spatial structure**. These properties are not exploited when an affine transformation is applied



Viewpoint changes

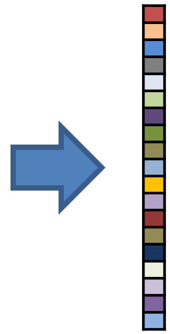
Changes in viewpoint cause changes in images that standard learning methods cannot cope with. E.g. if an object moves (rotates, scales, …) to a different location in the image, the object information moves to a different set of pixels



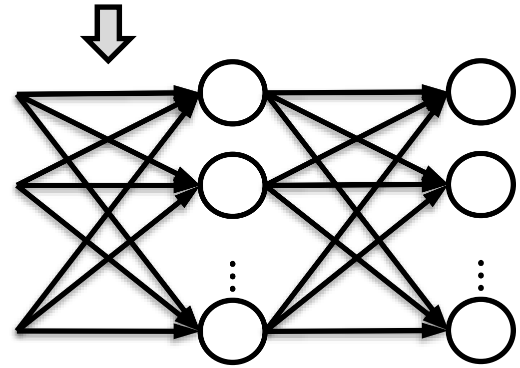
MLPs do not scale well

(64×48)

3,072 features

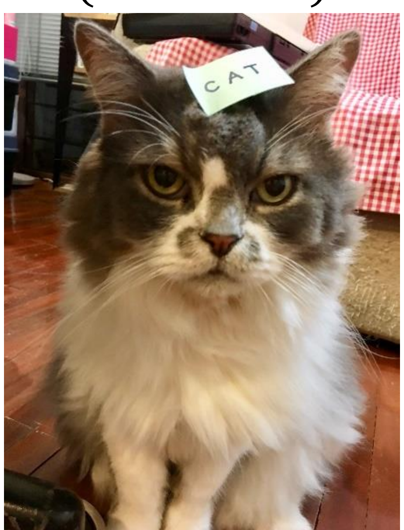


3,072,000 weights

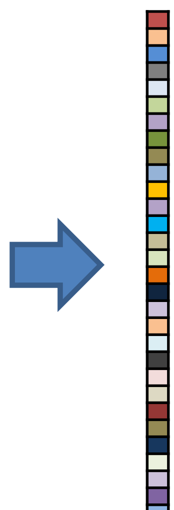


1,000 units

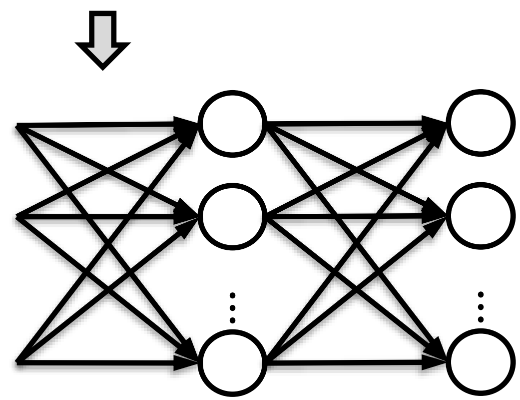
(640×480)



307,200 features



$307M$ weights



1,000 units

FILTERS AND CONVOLUTIONS

Looking for patterns in space

Simple recipe:

1. **Design** the pattern
2. Move it around the image
   (sound clip, sentence) and **compare**

Vertical edge

1	0	-1
1	0	-1
1	0	-1

Blob

0	1	0
1	1	1
0	1	0

Horizontal edge

1	1	1
0	0	0
-1	-1	-1

Corner

1	1	1
1	0	0
1	0	0

This pattern is called the "**filter**", "**kernel**" or "**mask**"

An eye?

0	0	0	0	0	
0	0.6	0.5	0.5	0.2	0
0.7	0.8	0.8	0.6	0.4	0.4
0.7	0.5	0.9	0.9	0.3	0.4
0	0.6	0.5	0.5	0.4	0.1
0	0	0	0	0	0

0	0	0	0	0	0
0	0.6	0.5	0.5	0.2	0
0.7	0.8	0.8	0.6	0.4	0.4
0.7	0.5	0.9	0.9	0.3	0.4
0	0.6	0.5	0.5	0.4	0.1
0	0	0	0	0	0

Cross-correlation or "Convolution"

To "compare" a pattern (filter, kernel, mask) with the image we measure the cross-correlation of the pattern with each patch of the image

$$\begin{array}{|c|c|c|c|c|c|}\hline I_{(0,0)} & I_{(1,0)} & I_{(2,0)} & I_{(3,0)} & I_{(4,0)} & I_{(5,0)} \\ \hline I_{(0,1)} & I_{(1,1)} & I_{(2,1)} & I_{(3,1)} & I_{(4,1)} & I_{(5,1)} \\ \hline I_{(0,2)} & I_{(1,2)} & I_{(2,2)} & I_{(3,2)} & I_{(4,2)} & I_{(5,2)} \\ \hline I_{(0,3)} & I_{(1,3)} & I_{(2,3)} & I_{(3,3)} & I_{(4,3)} & I_{(5,3)} \\ \hline I_{(0,4)} & I_{(1,4)} & I_{(2,4)} & I_{(3,4)} & I_{(4,4)} & I_{(5,4)} \\ \hline I_{(0,5)} & I_{(1,5)} & I_{(2,5)} & I_{(3,5)} & I_{(4,5)} & I_{(5,5)} \\ \hline \end{array} \otimes \begin{array}{|c|c|c|}\hline w_{(-1,-1)} & w_{(0,-1)} & w_{(1,-1)} \\ \hline w_{(-1,0)} & w_{(0,0)} & w_{(1,0)} \\ \hline w_{(-1,1)} & w_{(0,1)} & w_{(1,1)} \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline O_{(1,1)} & O_{(2,1)} & O_{(3,1)} & O_{(4,1)} \\ \hline O_{(1,2)} & O_{(2,2)} & O_{(3,2)} & O_{(4,2)} \\ \hline O_{(1,3)} & O_{(2,3)} & O_{(3,3)} & O_{(4,3)} \\ \hline O_{(1,4)} & O_{(2,4)} & O_{(3,4)} & O_{(4,4)} \\ \hline \end{array}$$

For a 3x3 filter:

$$O(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w_{i,j} I(x+i, y+j)$$

Cross-correlation or “Convolution”

To “compare” a pattern (filter, kernel, mask) with the image we measure the cross-correlation of the pattern with each patch of the image

$I_{(0,0)}$	$I_{(1,0)}$	$I_{(2,0)}$	$I_{(3,0)}$	$I_{(4,0)}$	$I_{(5,0)}$
$I_{(0,1)}$	$I_{(1,1)}$	$I_{(2,1)}$	$I_{(3,1)}$	$I_{(4,1)}$	$I_{(5,1)}$
$I_{(0,2)}$	$I_{(1,2)}$	$I_{(2,2)}$	$I_{(3,2)}$	$I_{(4,2)}$	$I_{(5,2)}$
$I_{(0,3)}$	$I_{(1,3)}$	$I_{(2,3)}$	$I_{(3,3)}$	$I_{(4,3)}$	$I_{(5,3)}$
$I_{(0,4)}$	$I_{(1,4)}$	$I_{(2,4)}$	$I_{(3,4)}$	$I_{(4,4)}$	$I_{(5,4)}$
$I_{(0,5)}$	$I_{(1,5)}$	$I_{(2,5)}$	$I_{(3,5)}$	$I_{(4,5)}$	$I_{(5,5)}$

$\otimes$

$w_{(-1,-1)}$	$w_{(0,-1)}$	$w_{(1,-1)}$
$w_{(-1,0)}$	$w_{(0,0)}$	$w_{(1,0)}$
$w_{(-1,1)}$	$w_{(0,1)}$	$w_{(1,1)}$

$=$

$O_{(1,1)}$	$O_{(2,1)}$	$O_{(3,1)}$	$O_{(4,1)}$
$O_{(1,2)}$	$O_{(2,2)}$	$O_{(3,2)}$	$O_{(4,2)}$
$O_{(1,3)}$	$O_{(2,3)}$	$O_{(3,3)}$	$O_{(4,3)}$
$O_{(1,4)}$	$O_{(2,4)}$	$O_{(3,4)}$	$O_{(4,4)}$

For a 3x3 filter:

$$O(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w_{i,j} I(x+i, y+j)$$

Cross-correlation or "Convolution"

To "compare" a pattern (filter, kernel, mask) with the image we measure the cross-correlation of the pattern with each patch of the image

$I_{(0,0)}$	$I_{(1,0)}$	$I_{(2,0)}$	$I_{(3,0)}$	$I_{(4,0)}$	$I_{(5,0)}$
$I_{(0,1)}$	$I_{(1,1)}$	$I_{(2,1)}$	$I_{(3,1)}$	$I_{(4,1)}$	$I_{(5,1)}$
$I_{(0,2)}$	$I_{(1,2)}$	$I_{(2,2)}$	$I_{(3,2)}$	$I_{(4,2)}$	$I_{(5,2)}$
$I_{(0,3)}$	$I_{(1,3)}$	$I_{(2,3)}$	$I_{(3,3)}$	$I_{(4,3)}$	$I_{(5,3)}$
$I_{(0,4)}$	$I_{(1,4)}$	$I_{(2,4)}$	$I_{(3,4)}$	$I_{(4,4)}$	$I_{(5,4)}$
$I_{(0,5)}$	$I_{(1,5)}$	$I_{(2,5)}$	$I_{(3,5)}$	$I_{(4,5)}$	$I_{(5,5)}$

$\otimes$

$w_{(-1,-1)}$	$w_{(0,-1)}$	$w_{(1,-1)}$
$w_{(-1,0)}$	$w_{(0,0)}$	$w_{(1,0)}$
$w_{(-1,1)}$	$w_{(0,1)}$	$w_{(1,1)}$

$=$

$O_{(1,1)}$	$O_{(2,1)}$	$O_{(3,1)}$	$O_{(4,1)}$
$O_{(1,2)}$	$O_{(2,2)}$	$O_{(3,2)}$	$O_{(4,2)}$
$O_{(1,3)}$	$O_{(2,3)}$	$O_{(3,3)}$	$O_{(4,3)}$
$O_{(1,4)}$	$O_{(2,4)}$	$O_{(3,4)}$	$O_{(4,4)}$

For a 3x3 filter:

$$O(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w_{i,j} I(x+i, y+j)$$

Cross-correlation or "Convolution"

To "compare" a pattern (filter, kernel, mask) with the image we measure the cross-correlation of the pattern with each patch of the image

$I_{(0,0)}$	$I_{(1,0)}$	$I_{(2,0)}$	$I_{(3,0)}$	$I_{(4,0)}$	$I_{(5,0)}$
$I_{(0,1)}$	$I_{(1,1)}$	$I_{(2,1)}$	$I_{(3,1)}$	$I_{(4,1)}$	$I_{(5,1)}$
$I_{(0,2)}$	$I_{(1,2)}$	$I_{(2,2)}$	$I_{(3,2)}$	$I_{(4,2)}$	$I_{(5,2)}$
$I_{(0,3)}$	$I_{(1,3)}$	$I_{(2,3)}$	$I_{(3,3)}$	$I_{(4,3)}$	$I_{(5,3)}$
$I_{(0,4)}$	$I_{(1,4)}$	$I_{(2,4)}$	$I_{(3,4)}$	$I_{(4,4)}$	$I_{(5,4)}$
$I_{(0,5)}$	$I_{(1,5)}$	$I_{(2,5)}$	$I_{(3,5)}$	$I_{(4,5)}$	$I_{(5,5)}$

$\otimes$

$w_{(-1,-1)}$	$w_{(0,-1)}$	$w_{(1,-1)}$
$w_{(-1,0)}$	$w_{(0,0)}$	$w_{(1,0)}$
$w_{(-1,1)}$	$w_{(0,1)}$	$w_{(1,1)}$

$=$

$O_{(1,1)}$	$O_{(2,1)}$	$O_{(3,1)}$	$O_{(4,1)}$
$O_{(1,2)}$	$O_{(2,2)}$	$O_{(3,2)}$	$O_{(4,2)}$
$O_{(1,3)}$	$O_{(2,3)}$	$O_{(3,3)}$	$O_{(4,3)}$
$O_{(1,4)}$	$O_{(2,4)}$	$O_{(3,4)}$	$O_{(4,4)}$

For a 3x3 filter:

$$O(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w_{i,j} I(x+i, y+j)$$

Cross-correlation or "Convolution"

To "compare" a pattern (filter, kernel, mask) with the image we measure the cross-correlation of the pattern with each patch of the image

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
I_{(0,0)} & I_{(1,0)} & I_{(2,0)} & I_{(3,0)} & I_{(4,0)} & I_{(5,0)} \\ \hline
I_{(0,1)} & I_{(1,1)} & I_{(2,1)} & I_{(3,1)} & I_{(4,1)} & I_{(5,1)} \\ \hline
I_{(0,2)} & I_{(1,2)} & I_{(2,2)} & I_{(3,2)} & I_{(4,2)} & I_{(5,2)} \\ \hline
I_{(0,3)} & I_{(1,3)} & I_{(2,3)} & I_{(3,3)} & I_{(4,3)} & I_{(5,3)} \\ \hline
I_{(0,4)} & I_{(1,4)} & I_{(2,4)} & I_{(3,4)} & I_{(4,4)} & I_{(5,4)} \\ \hline
I_{(0,5)} & I_{(1,5)} & I_{(2,5)} & I_{(3,5)} & I_{(4,5)} & I_{(5,5)} \\ \hline
\end{array}
\otimes
\begin{array}{|c|c|c|}
\hline
w_{(-1,-1)} & w_{(0,-1)} & w_{(1,-1)} \\ \hline
w_{(-1,0)} & w_{(0,0)} & w_{(1,0)} \\ \hline
w_{(-1,1)} & w_{(0,1)} & w_{(1,1)} \\ \hline
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
O_{(1,1)} & O_{(2,1)} & O_{(3,1)} & O_{(4,1)} \\ \hline
O_{(1,2)} & O_{(2,2)} & O_{(3,2)} & O_{(4,2)} \\ \hline
O_{(1,3)} & O_{(2,3)} & O_{(3,3)} & O_{(4,3)} \\ \hline
O_{(1,4)} & O_{(2,4)} & O_{(3,4)} & O_{(4,4)} \\ \hline
\end{array}
$$

For a 3x3 filter:

$$O(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w_{i,j} I(x+i, y+j)$$

Cross-correlation or "Convolution"

To "compare" a pattern (filter, kernel, mask) with the image we measure the cross-correlation of the pattern with each patch of the image

$$
\begin{array}{|c|c|c|c|c|c|}
\hline
I_{(0,0)} & I_{(1,0)} & I_{(2,0)} & I_{(3,0)} & I_{(4,0)} & I_{(5,0)} \\
\hline
I_{(0,1)} & I_{(1,1)} & I_{(2,1)} & I_{(3,1)} & I_{(4,1)} & I_{(5,1)} \\
\hline
I_{(0,2)} & I_{(1,2)} & I_{(2,2)} & I_{(3,2)} & I_{(4,2)} & I_{(5,2)} \\
\hline
I_{(0,3)} & I_{(1,3)} & I_{(2,3)} & I_{(3,3)} & I_{(4,3)} & I_{(5,3)} \\
\hline
I_{(0,4)} & I_{(1,4)} & I_{(2,4)} & I_{(3,4)} & I_{(4,4)} & I_{(5,4)} \\
\hline
I_{(0,5)} & I_{(1,5)} & I_{(2,5)} & I_{(3,5)} & I_{(4,5)} & I_{(5,5)} \\
\hline
\end{array}
\otimes
\begin{array}{|c|c|c|}
\hline
w_{(-1,-1)} & w_{(0,-1)} & w_{(1,-1)} \\
\hline
w_{(-1,0)} & w_{(0,0)} & w_{(1,0)} \\
\hline
w_{(-1,1)} & w_{(0,1)} & w_{(1,1)} \\
\hline
\end{array}
=
\begin{array}{|c|c|c|c|}
\hline
O_{(1,1)} & O_{(2,1)} & O_{(3,1)} & O_{(4,1)} \\
\hline
O_{(1,2)} & O_{(2,2)} & O_{(3,2)} & O_{(4,2)} \\
\hline
O_{(1,3)} & O_{(2,3)} & O_{(3,3)} & O_{(4,3)} \\
\hline
O_{(1,4)} & O_{(2,4)} & O_{(3,4)} & O_{(4,4)} \\
\hline
\end{array}
$$

For a 3x3 filter:

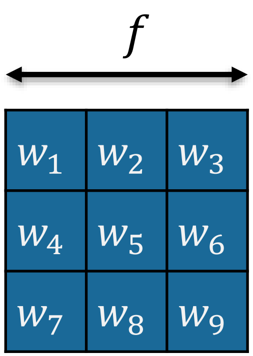
$$O(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w_{i,j} I(x+i, y+j)$$

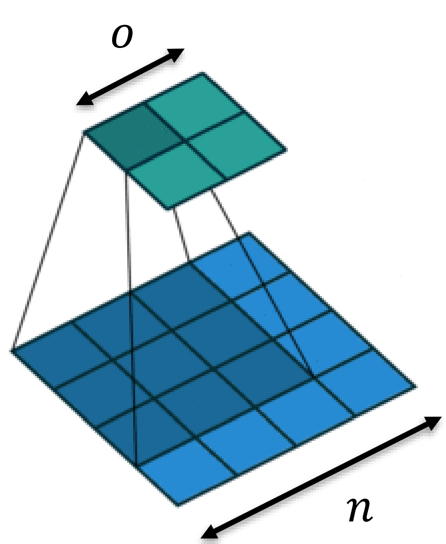
Convolution in Deep Learning

The term "**convolution**" in deep learning notation refers to the calculation of the cross-correlation between the filter and a part of the input.

- A linear transformation
- A dot product

It is **sparse** (only a few input units contribute to a given output unit) and **reuses parameters** (the same weights are applied to multiple locations in the input).





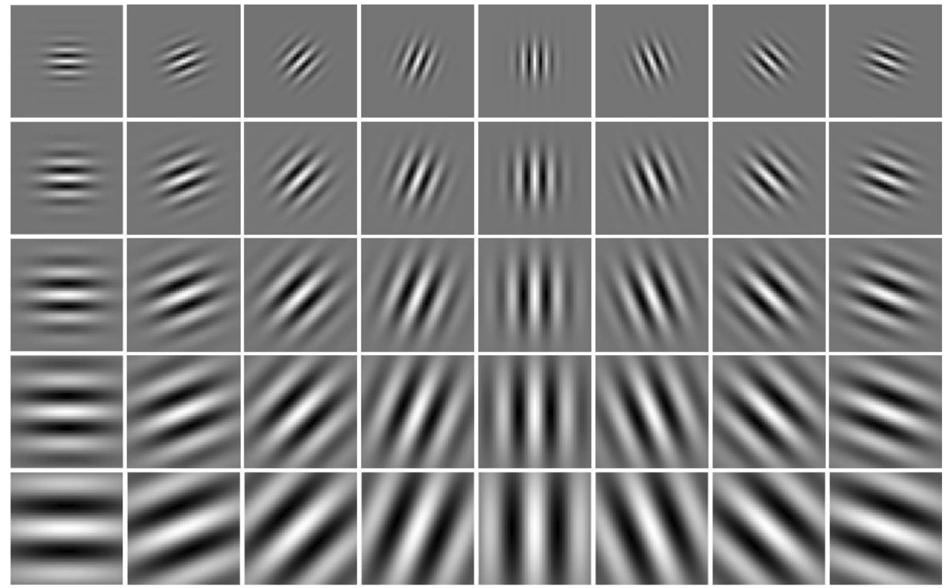
Input size: $n \times n$
Filter size: $f \times f$
Output size: $o \times o$

$$o = n - f + 1$$

Image credit: V. Dumoulin, F. Visin, "guide to convolution arithmetic for deep learning", 2018

Image processing filters

Frequency decomposition



Gabor

Calculate edges / gradients

1	0	−1
1	0	−1
1	0	−1

1	1	1
0	0	0
−1	−1	−1

Prewitt

1	0	−1
2	0	−2
1	0	−1

1	2	1
0	0	0
−1	−2	−1

Sobel

Image smoothing / denoising

$\frac{1}{K^2}$

1	1	$\cdots$	1
1	1	$\cdots$	1
$\vdots$	$\vdots$	1	$\vdots$
1	1	$\cdots$	1

Box

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Bilinear

$\frac{1}{256}$

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

Gaussian

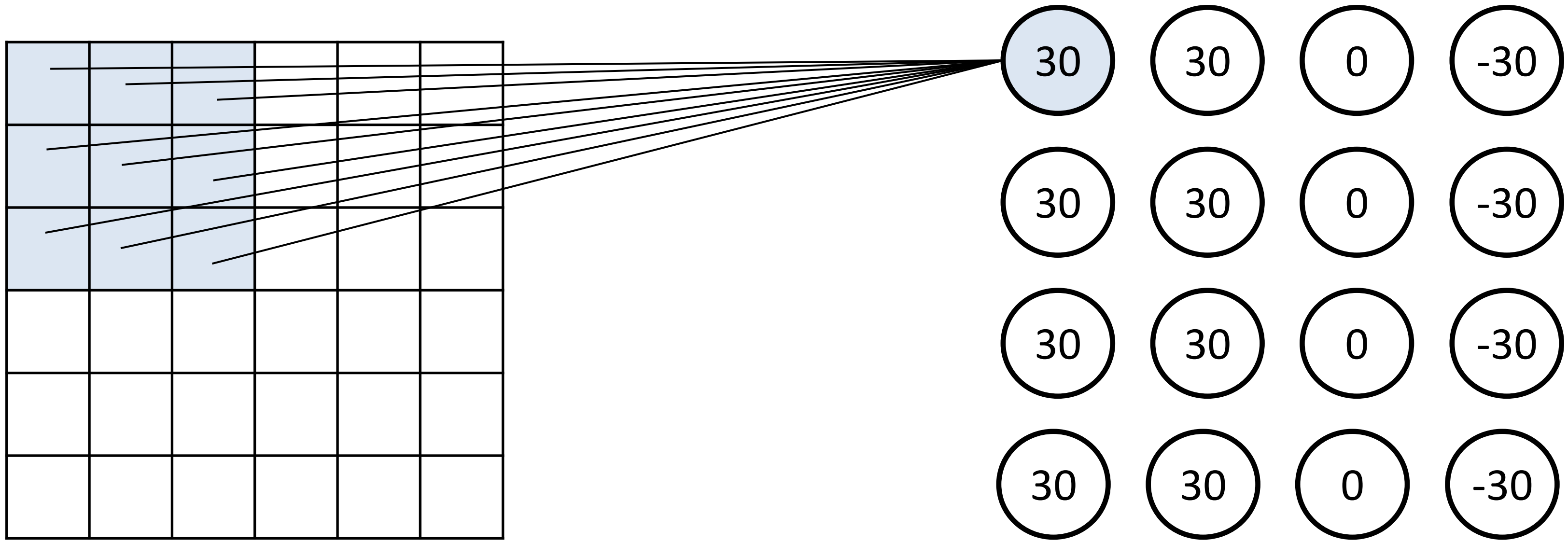
Learning filters

$$\begin{bmatrix} 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 30 & 30 & 0 & -30 \\ 30 & 30 & 0 & -30 \\ 30 & 30 & 0 & -30 \\ 30 & 30 & 0 & -30 \end{bmatrix}$$

?

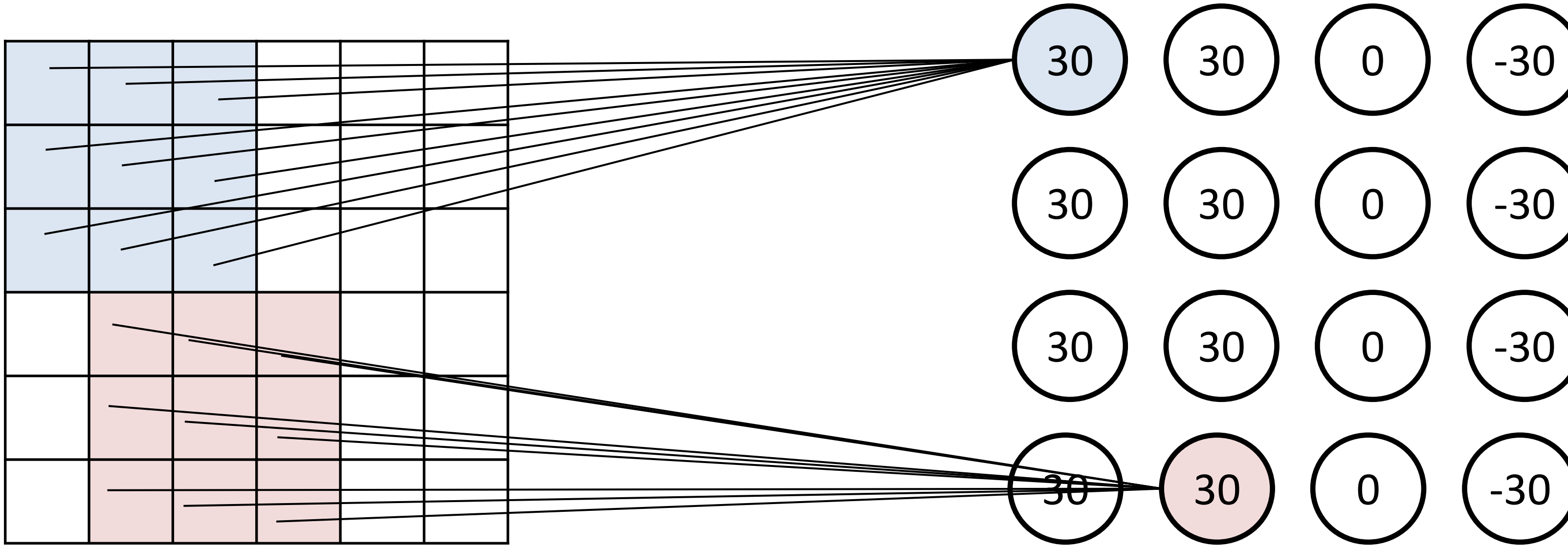
$$\begin{bmatrix} 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \\ 10 & 10 & 0 & 0 & 0 & 10 \end{bmatrix} \otimes \begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} = \begin{bmatrix} 30 & 30 & 0 & -30 \\ 30 & 30 & 0 & -30 \\ 30 & 30 & 0 & -30 \\ 30 & 30 & 0 & -30 \end{bmatrix}$$

Learning filters



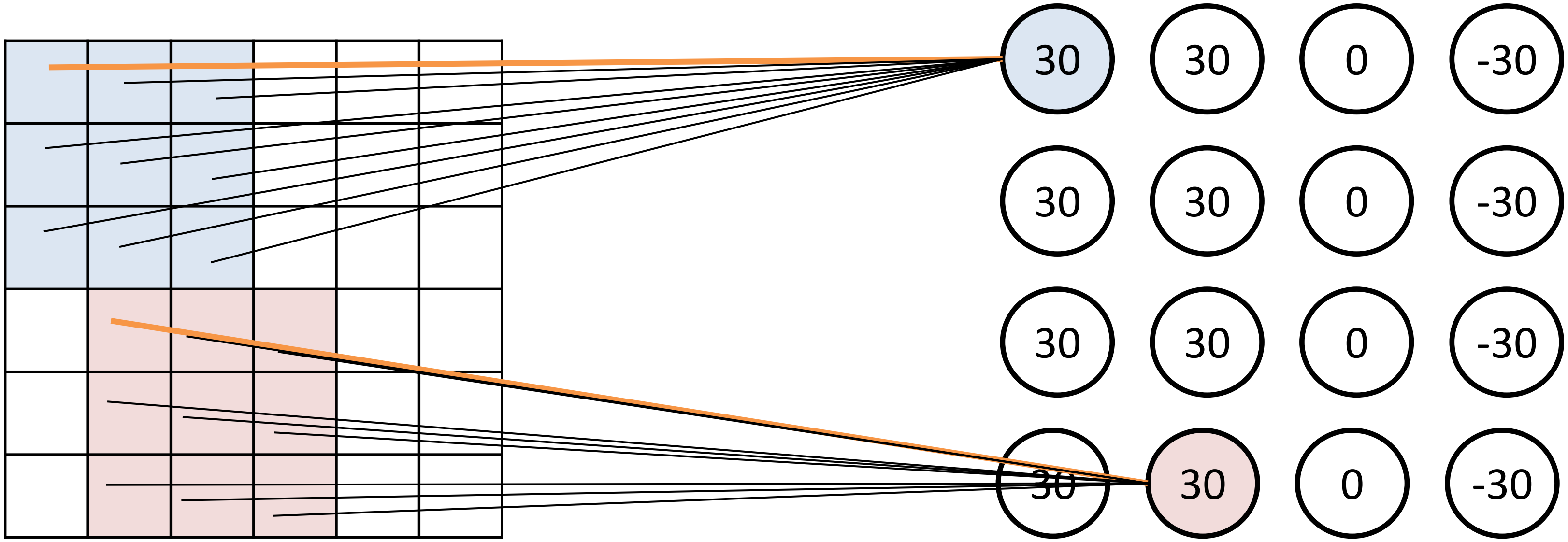
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Learning filters



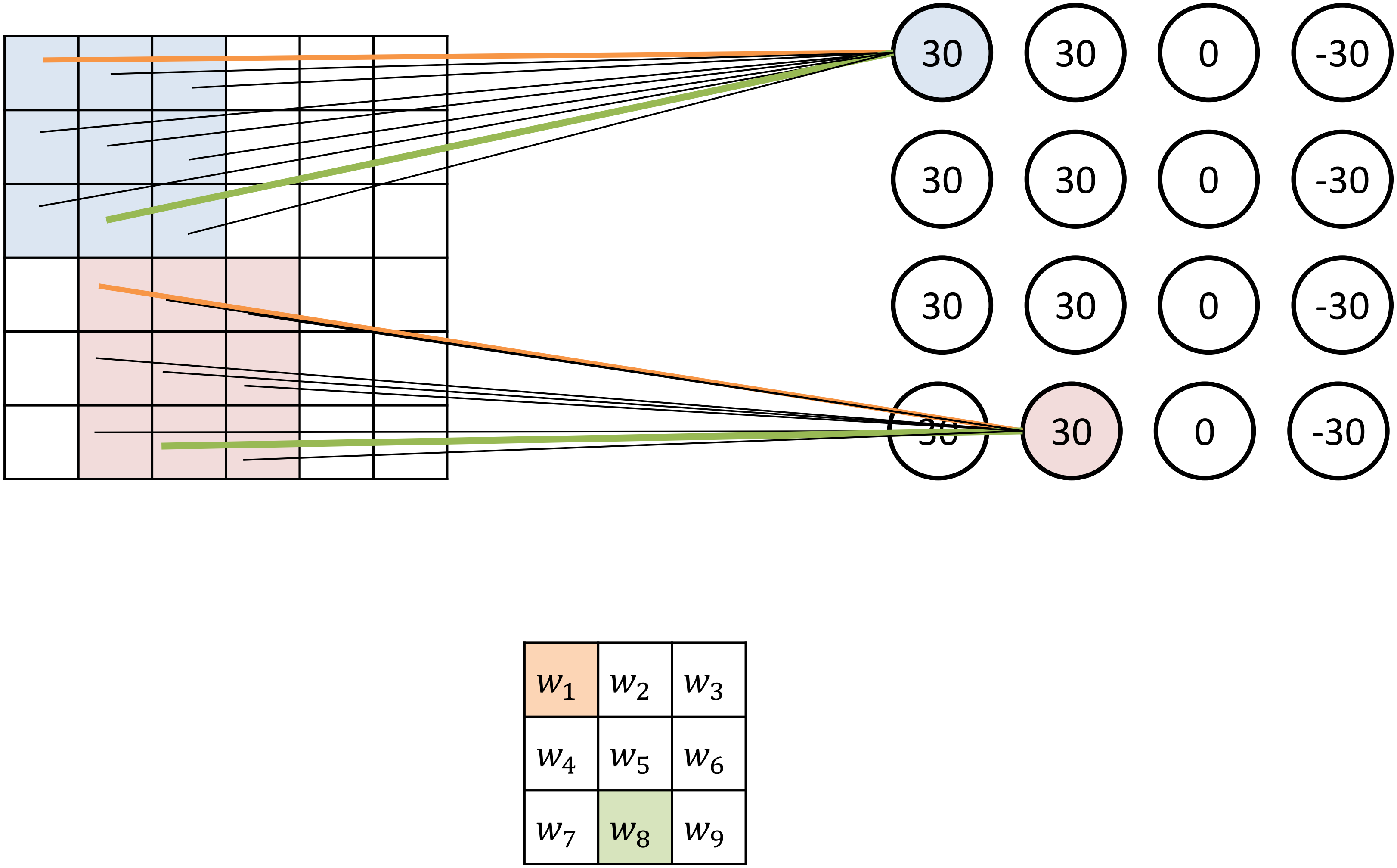
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Learning filters



w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Learning filters

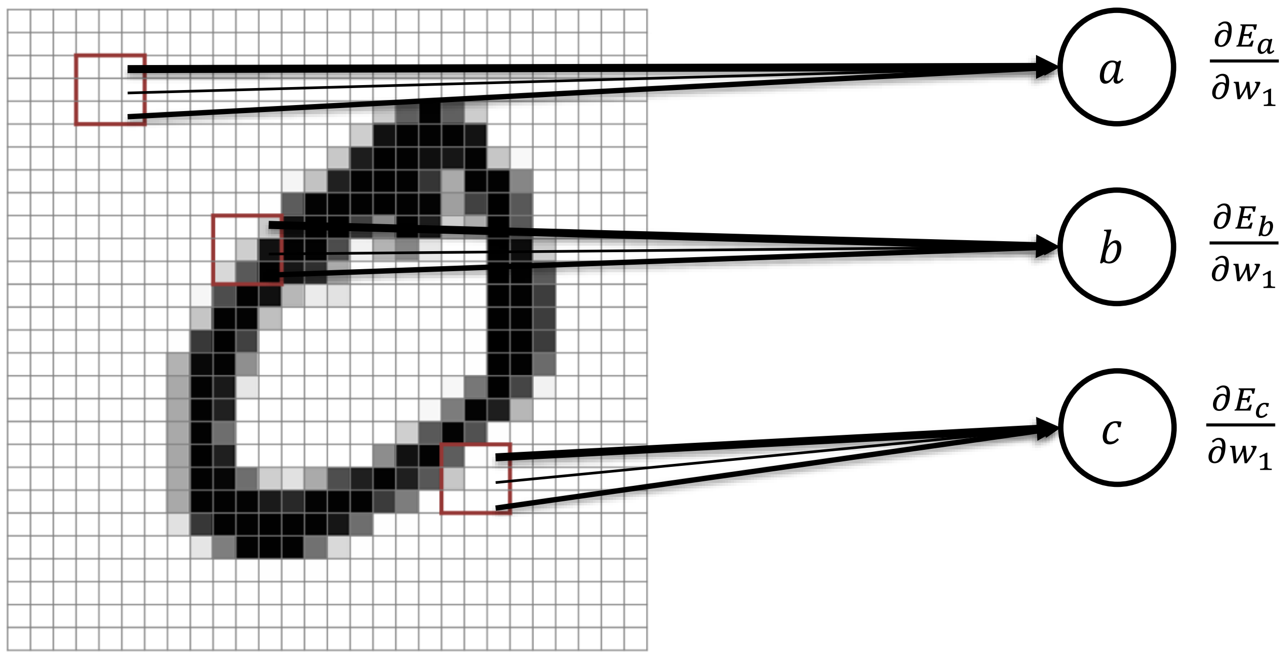


Local Receptive Fields

Instead of fully-connecting the input with the every hidden layer unit, we connect only a small region of the image with each unit, called the **local receptive field** of the unit

We replicate by applying **the same weights** over different patches (local receptive fields) of the image

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9



Backpropagation and replicated features

The backpropagation algorithm can be easily modified to incorporate linear constraints between the weights

We need to make sure that all copies of a weight w_1 change always in the same way

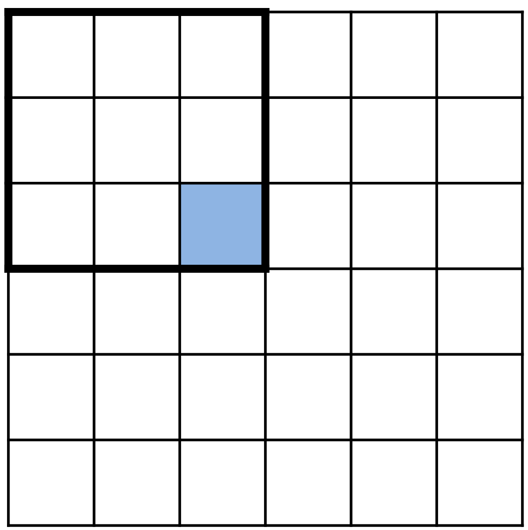
Compute gradients $\frac{\partial E_a}{\partial w_1}, \frac{\partial E_b}{\partial w_1}, \ldots$ as usual for all the units

Use a different rule for updating, e.g. use the quantity $\frac{\partial E_a}{\partial w_1} + \frac{\partial E_b}{\partial w_1}$

Padding

A problem with applying convolutions is that

- the output is shrinking, and
- we are throwing away information from the edge pixels

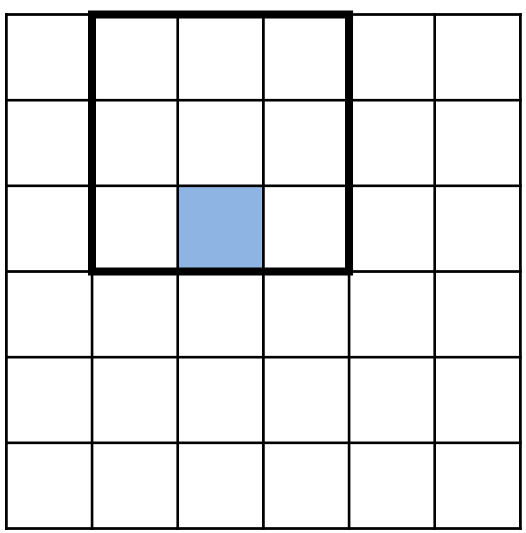


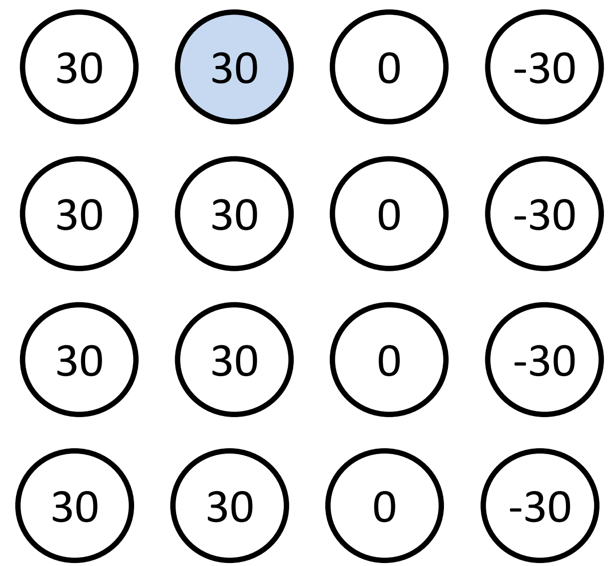
30	30	0	-30
30	30	0	-30
30	30	0	-30
30	30	0	-30

Padding

A problem with applying convolutions is that

- the output is shrinking, and
- we are throwing away information from the edge pixels

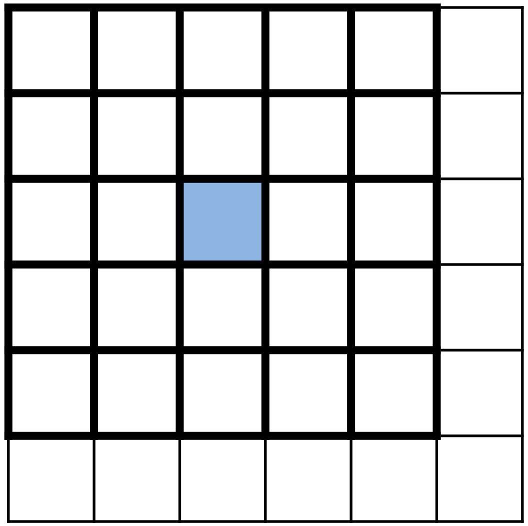




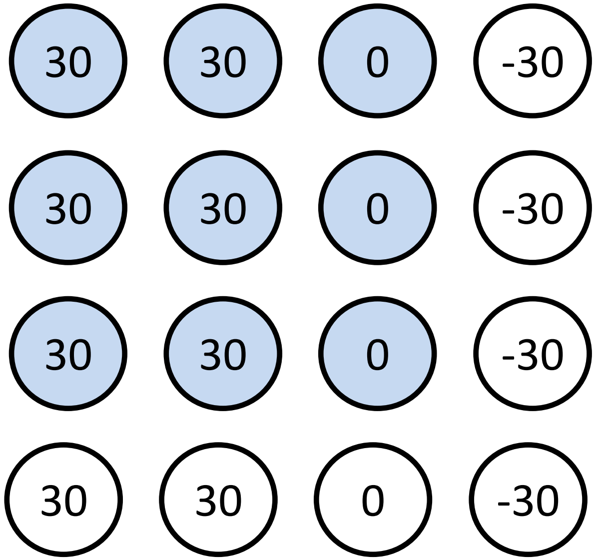
Padding

A problem with applying convolutions is that

- the output is shrinking, and
- we are throwing away information from the edge pixels



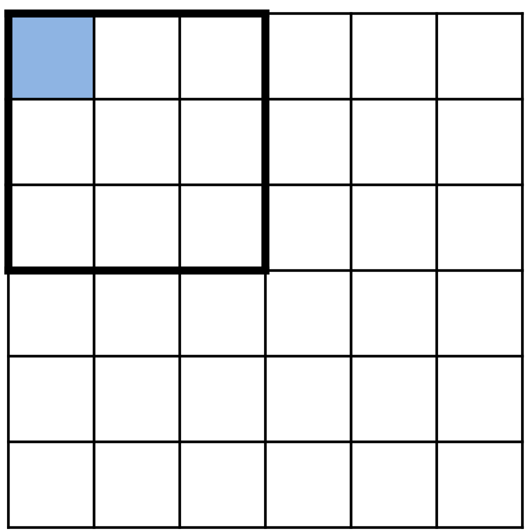
Pixel affects
9 units



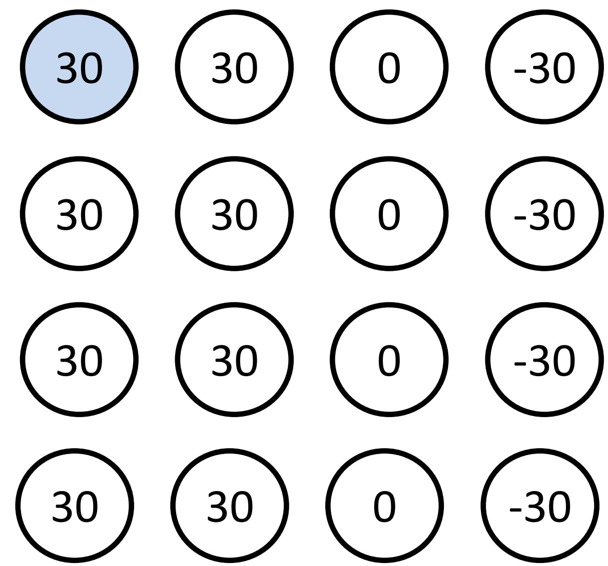
Padding

A problem with applying convolutions is that

- the output is shrinking, and
- we are throwing away information from the edge pixels



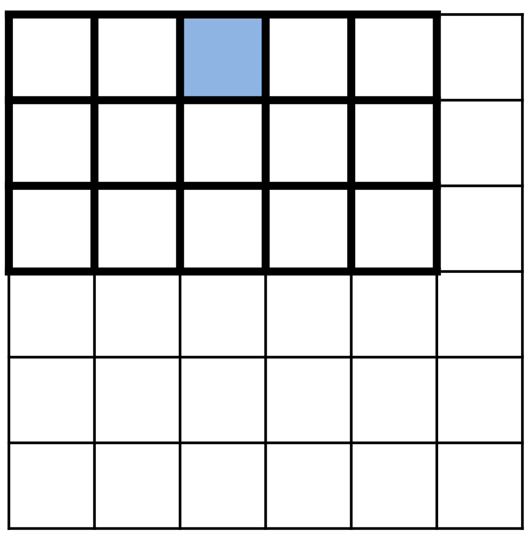
Pixel affects
1 unit



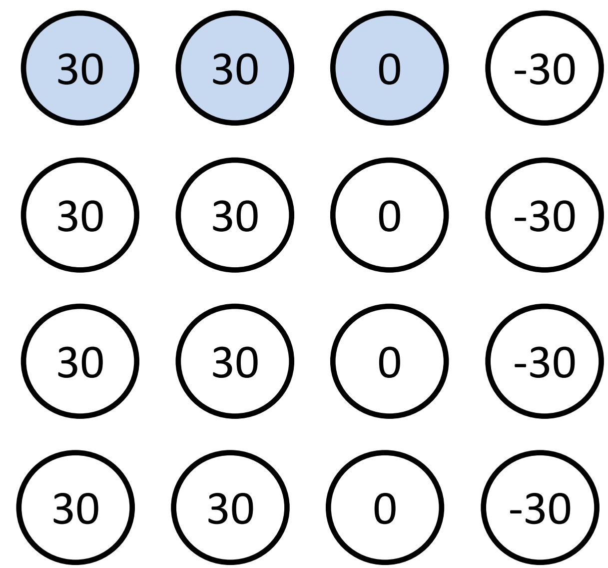
Padding

A problem with applying convolutions is that

- the output is shrinking, and
- we are throwing away information from the edge pixels



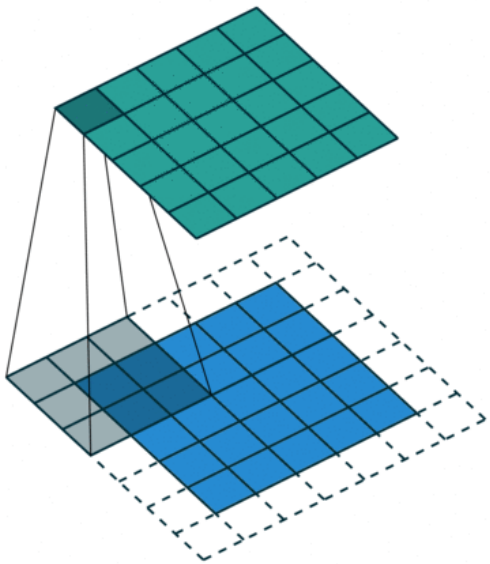
Pixel affects
3 units



Padding

Padding aims to

- Correct the shrinking output
- Do not throw away info from edges (make all pixels count the same)



Input size:	$n \times n$
Filter size:	$f \times f$
Output size:	$o \times o$
Padding:	$p \times p$

$$o = n + 2p - f + 1$$

Image credit: V. Dumoulin, F. Visin, "guide to convolution arithmetic for deep learning", 2018

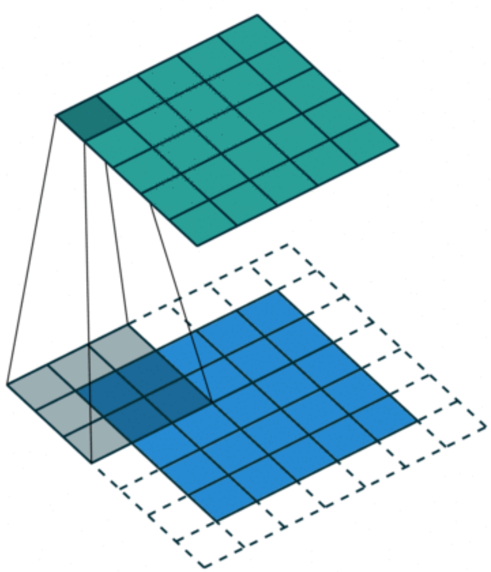
Padding

Valid padding (no padding): when $p = 0$ ⇨ Shrinks

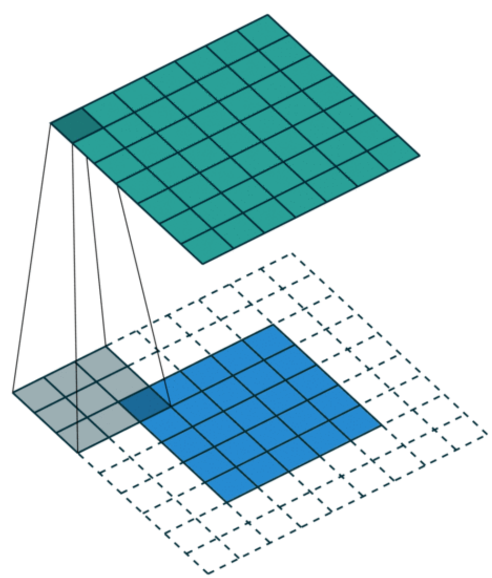
Same (or **half**) padding: when output size is the same as the input size, $p = (f-1)/2$ ⇨ No change

Full padding: when $p = f - 1$ introduces padding so that all pixels are visited the same amount of times ⇨ Grows

OUTPUT



Half (Same) Padding
$p = 1, f = 3$



Full Padding
$p = 2, f = 3$

Input size:	$n \times n$
Filter size:	$f \times f$
Output size:	$o \times o$
Padding:	$p \times p$

$$o = n + 2p - f + 1$$

Padding with what?

Constant padding

c	c	c	c	c	c	c
c	c	3	8	9	c	c
c	c	6	4	1	c	c
c	c	4	3	2	c	c
c	c	c	c	c	c	c

Zero padding

0	0	0	0	0	0	0
0	0	3	8	9	0	0
0	0	6	4	1	0	0
0	0	4	3	2	0	0
0	0	0	0	0	0	0

Reflection padding

1	5	6	5	1	5	6
9	8	7	8	9	8	7
1	5	6	5	1	5	6
2	3	4	3	2	3	4
1	5	6	5	1	5	6

Replication padding

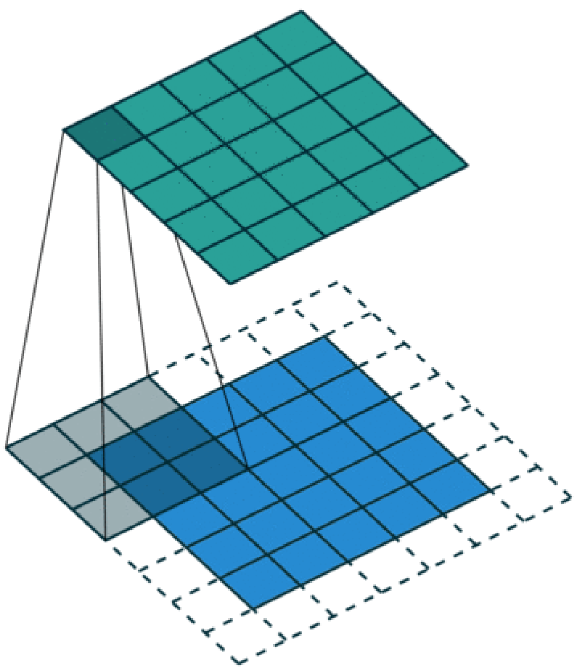
3	3	3	8	9	9	9
3	3	3	8	9	9	9
6	6	6	4	1	1	1
4	4	4	3	2	2	2
4	4	4	3	2	2	2

Stride

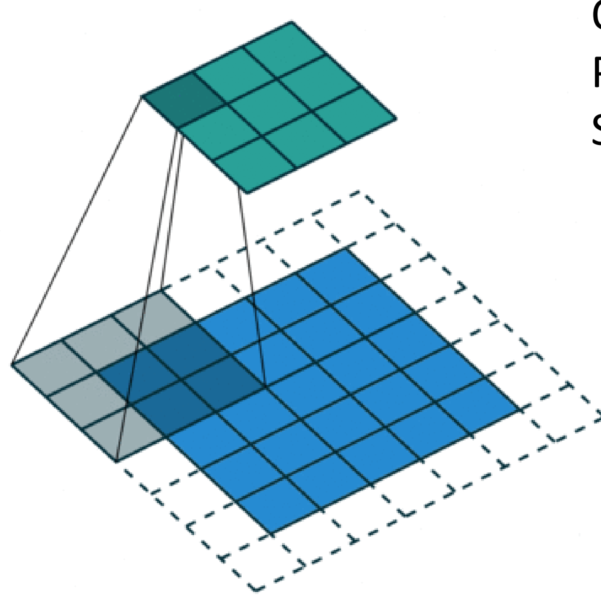
The **stride length** defines the step by which we move the local receptive field.

Input size: $n \times n$
Filter size: $f \times f$
Output size: $o \times o$
Padding: $p \times p$
Stride: $s \times s$

$$o = \frac{n + 2p - f}{s} + 1$$



$s = 1, p = 1, f = 3$



$s = 2, p = 1, f = 3$

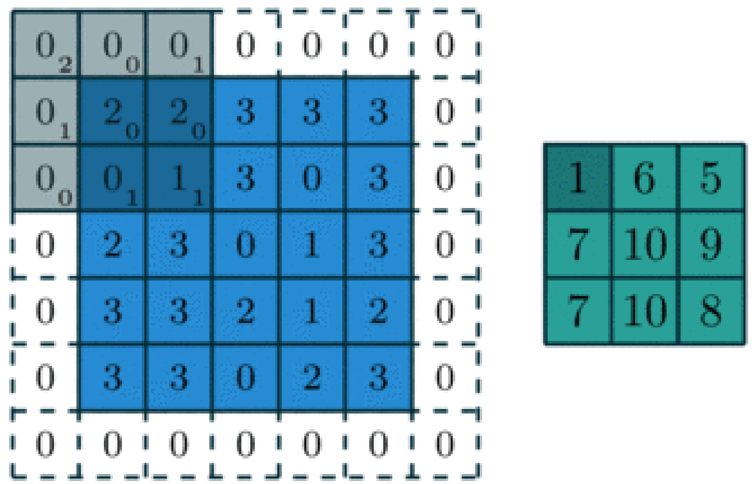
https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Image credit: V. Dumoulin, F. Visin, “guide to convolution arithmetic for deep learning”, 2018

For example

What would be the output size if a 3×3 kernel is applied to a 5×5 input padded with a 1×1 border of zeros using 2×2 strides?

$$o = \frac{n + 2p - f}{s} + 1$$

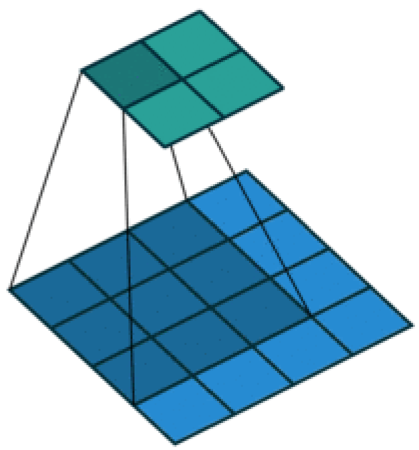


Answer: 3 x 3

Image credit: V. Dumoulin, F. Visin, “guide to convolution arithmetic for deep learning”, 2018

Transposed Convolutions

A **transposed convolution** (also called **fractionally strided convolution** or **deconvolution** in the NN literature) is the reverse process of convolution.



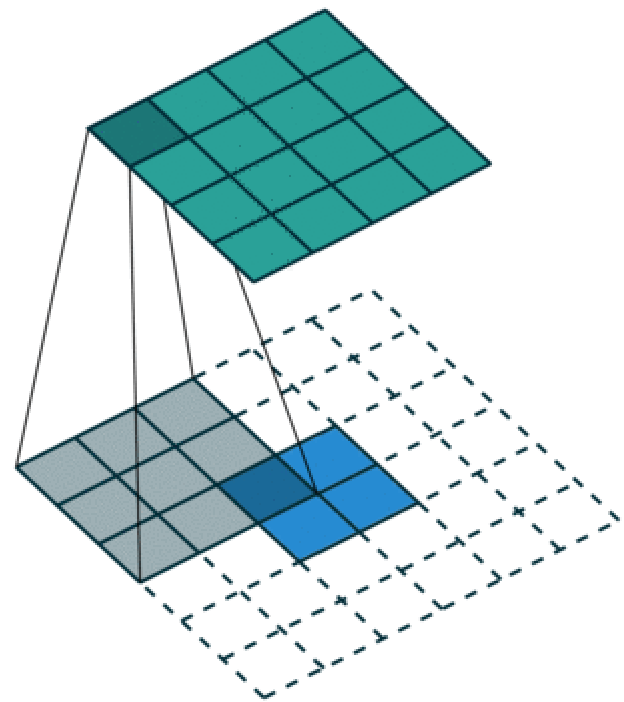
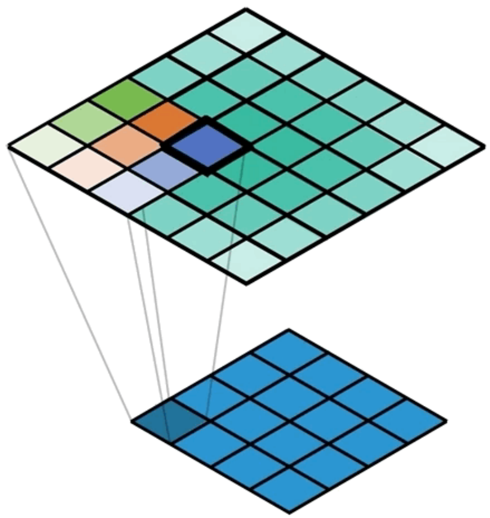


Image credit: V. Dumoulin, F. Visin, “guide to convolution arithmetic for deep learning”, 2018

Transposed Convolutions

A **transposed convolution** (also called **fractionally strided convolution** or **deconvolution**) is the reverse process of convolution.



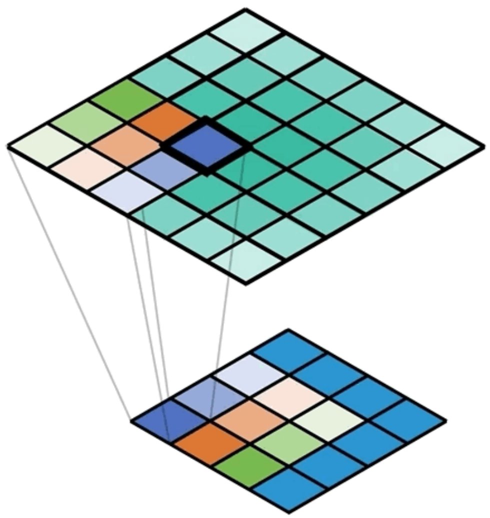
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

The easiest way of thinking about it is to take each value in your input and distribute it (using the corresponding weights of a kernel) to a local region in the output

Image credit: T. Lane, 2018, https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

Transposed Convolutions

A **transposed convolution** (also called **fractionally strided convolution** or **deconvolution**) is the reverse process of convolution.



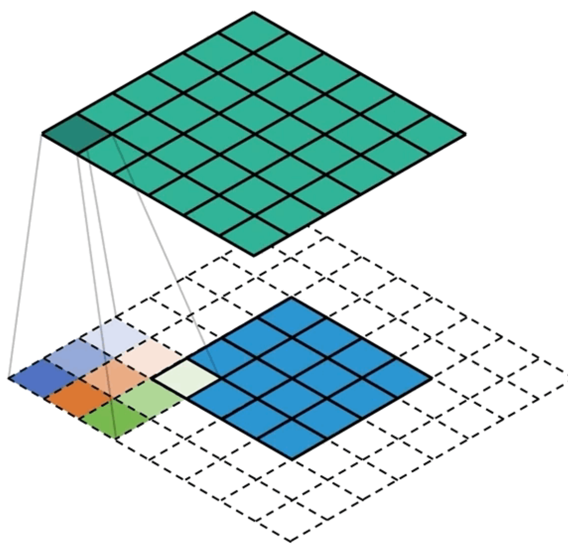
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Note how “distributing” the input values ends up being equivalent to applying a normal convolution with the transposed kernel

Image credit: T. Lane, 2018, https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

Transposed Convolutions

A **transposed convolution** (also called **fractionally strided convolution** or **deconvolution**) is the reverse process of convolution.



w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

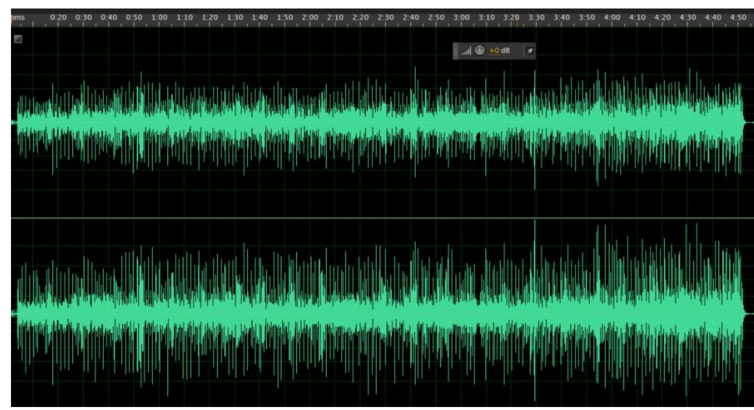
Applying a forward convolution with a transposed kernel, actually results in a much smaller output. What we are really doing here is equivalent to this transposed convolution but with full padding

Image credit: T. Lane, 2018, https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

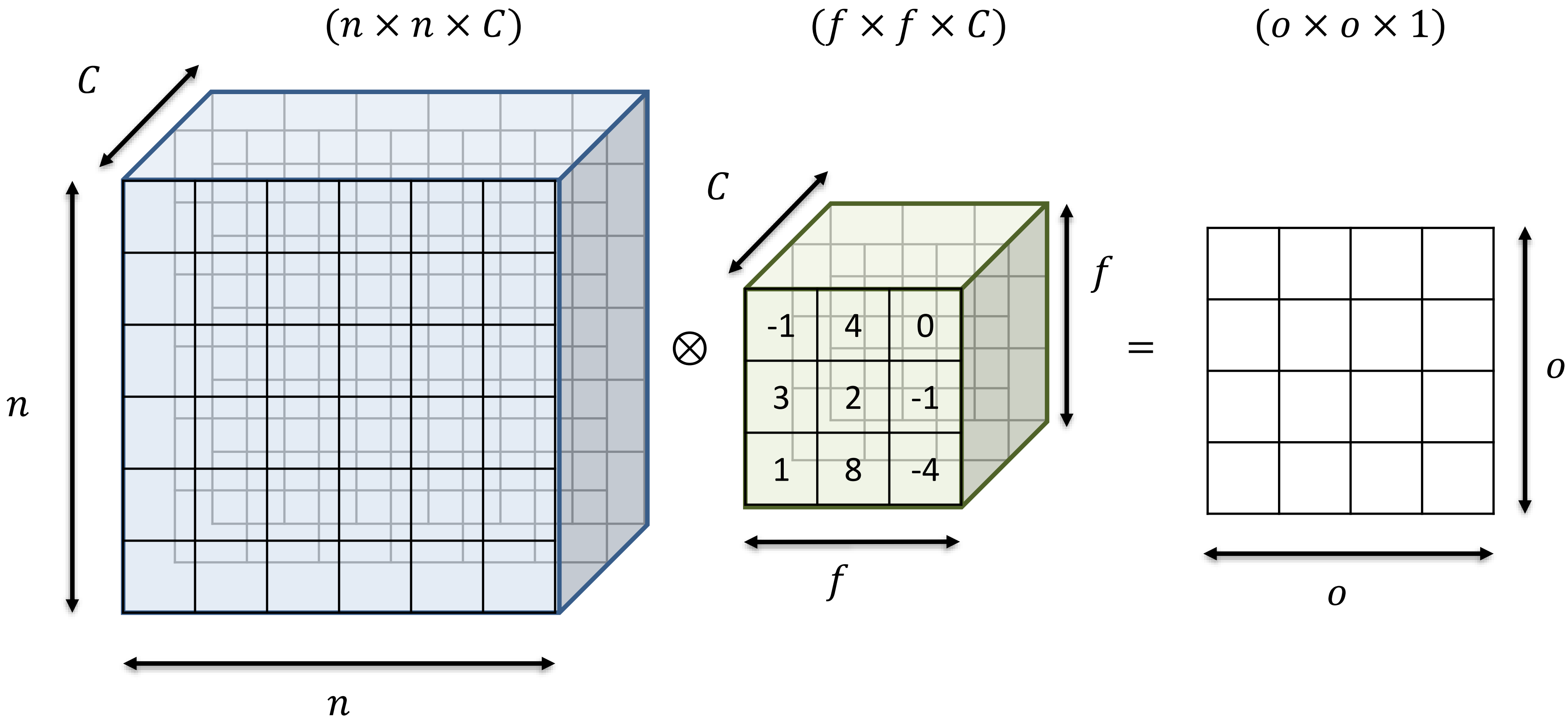
Channels

Images, sound clips, etc have an **intrinsic structure**:

- One or more axes for which ordering matters
- One **channel** axis for different views of the data

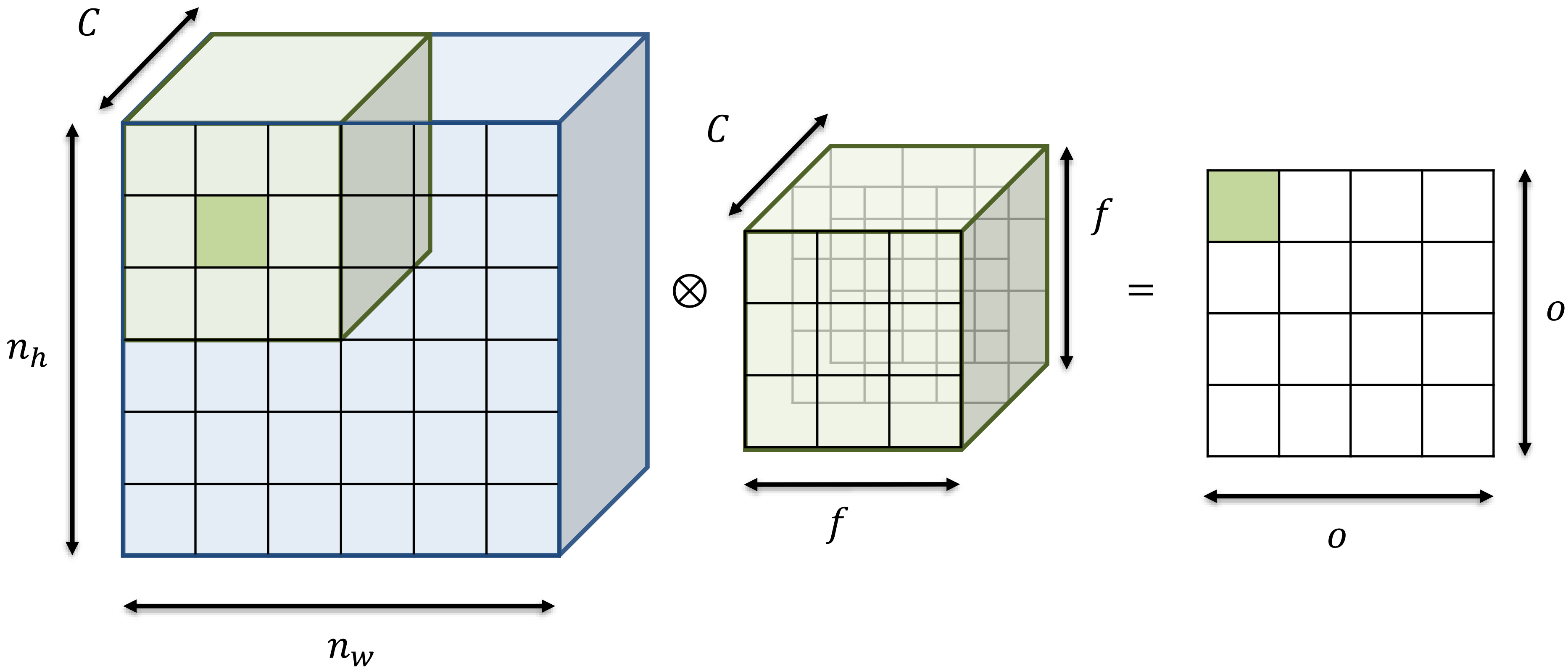


Channels



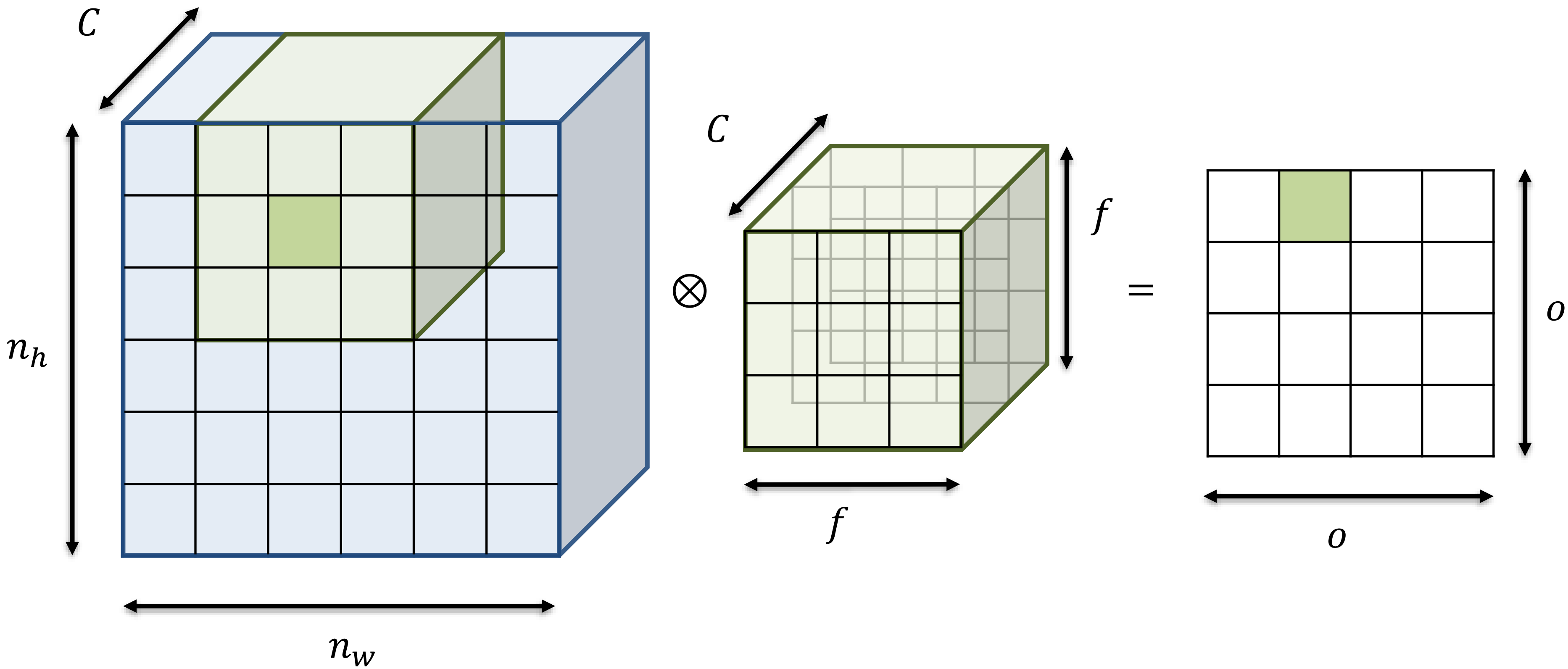
$$O(x,y) = \sum_{i \in \left[-\frac{f}{2}, \frac{f}{2}\right]} \sum_{j \in \left[-\frac{f}{2}, \frac{f}{2}\right]} \sum_{k \in [1,C]} w_{i,j,k} I(x+i, y+j, k)$$

Channels



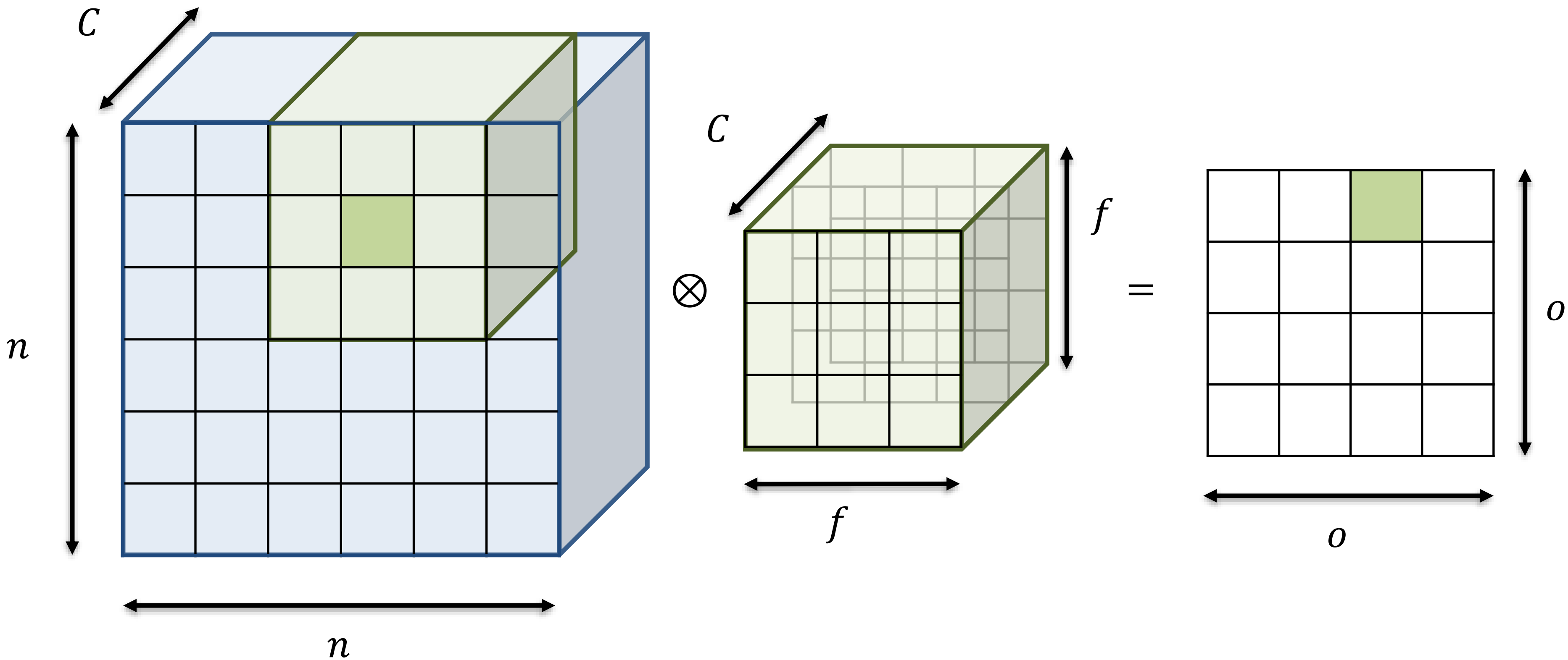
$$O(x,y) = \sum_{i \in \left[-\frac{f}{2},\frac{f}{2}\right]} \sum_{j \in \left[-\frac{f}{2},\frac{f}{2}\right]} \sum_{k \in [1,C]} w_{i,j,k} I(x+i, y+j, k)$$

Channels



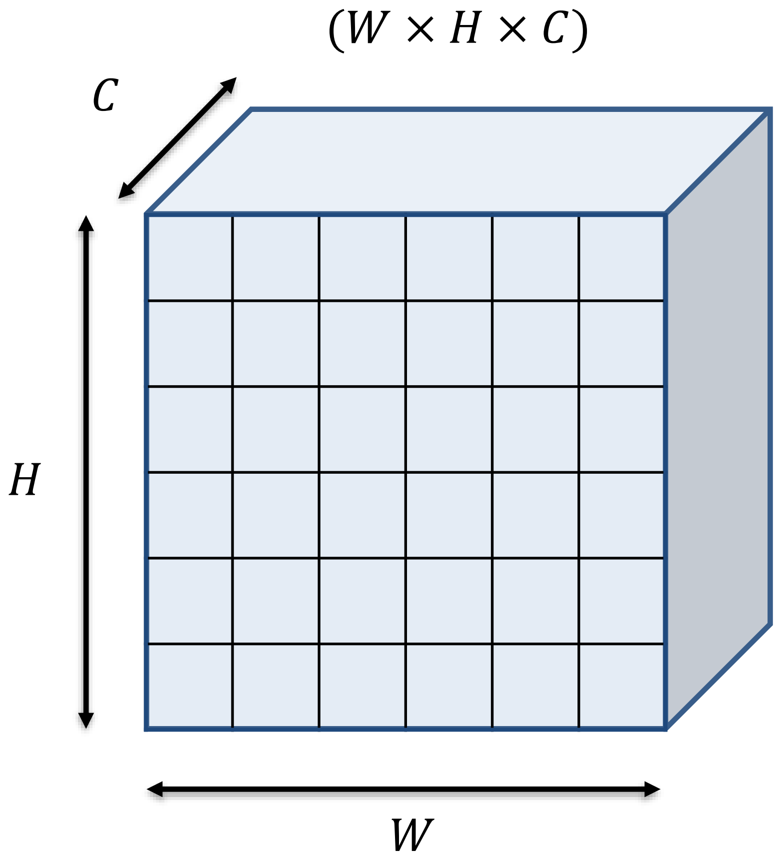
$$O(x,y) = \sum_{i \in \left[-\frac{f}{2},\frac{f}{2}\right]} \sum_{j \in \left[-\frac{f}{2},\frac{f}{2}\right]} \sum_{k \in [1,C]} w_{i,j,k} I(x+i, y+j, k)$$

Channels



$$O(x,y) = \sum_{i \in \left[-\frac{f}{2},\frac{f}{2}\right]} \sum_{j \in \left[-\frac{f}{2},\frac{f}{2}\right]} \sum_{k \in [1,C]} w_{i,j,k} I(x+i, y+j, k)$$

Channels



Ordering of tensor dimensions in Pytorch

For a 2D convolution the input should be in the format: (B, C, H, W), where B is the number of samples / batch size, C is the channels, H and W are height and width.

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d

For an 1D convolution, input should be formatted as (B, C, L)

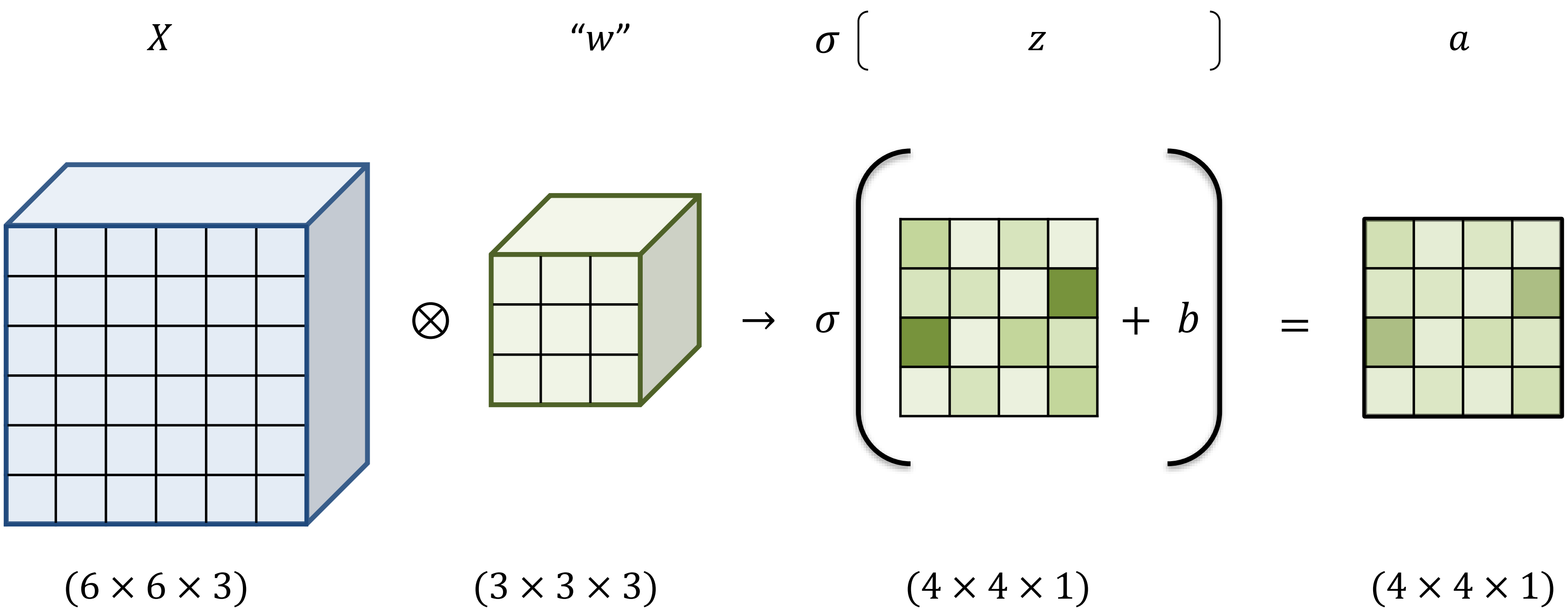
https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html#torch.nn.Conv1d

BUILDING A CNN

Features and Feature Maps

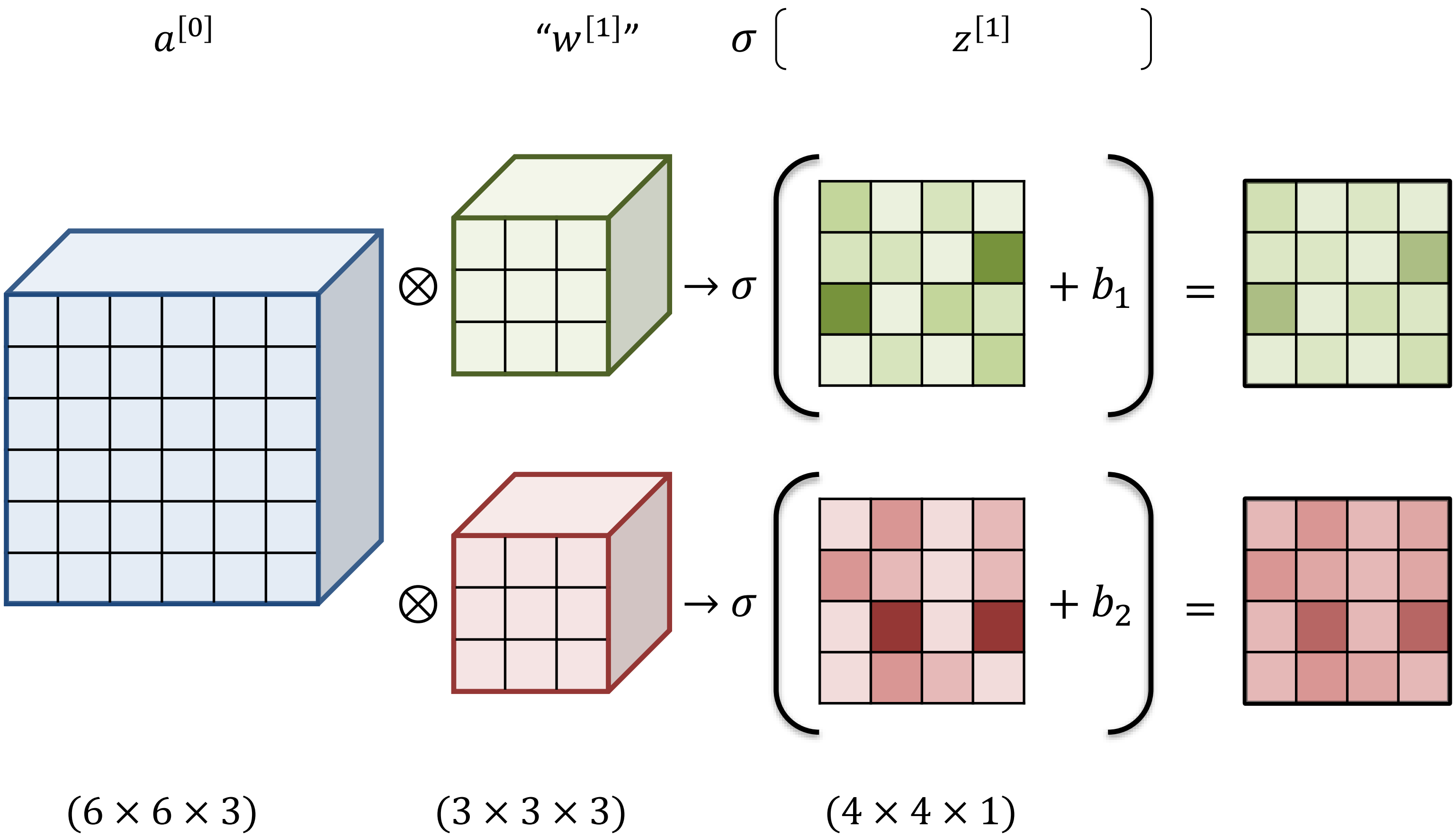
All neurons in the first hidden layer detect exactly the same feature at different locations in the input image. (with one single filter)

This produces a feature map for the particular feature (filter) – that we can pass through an activation function to produce an activation map



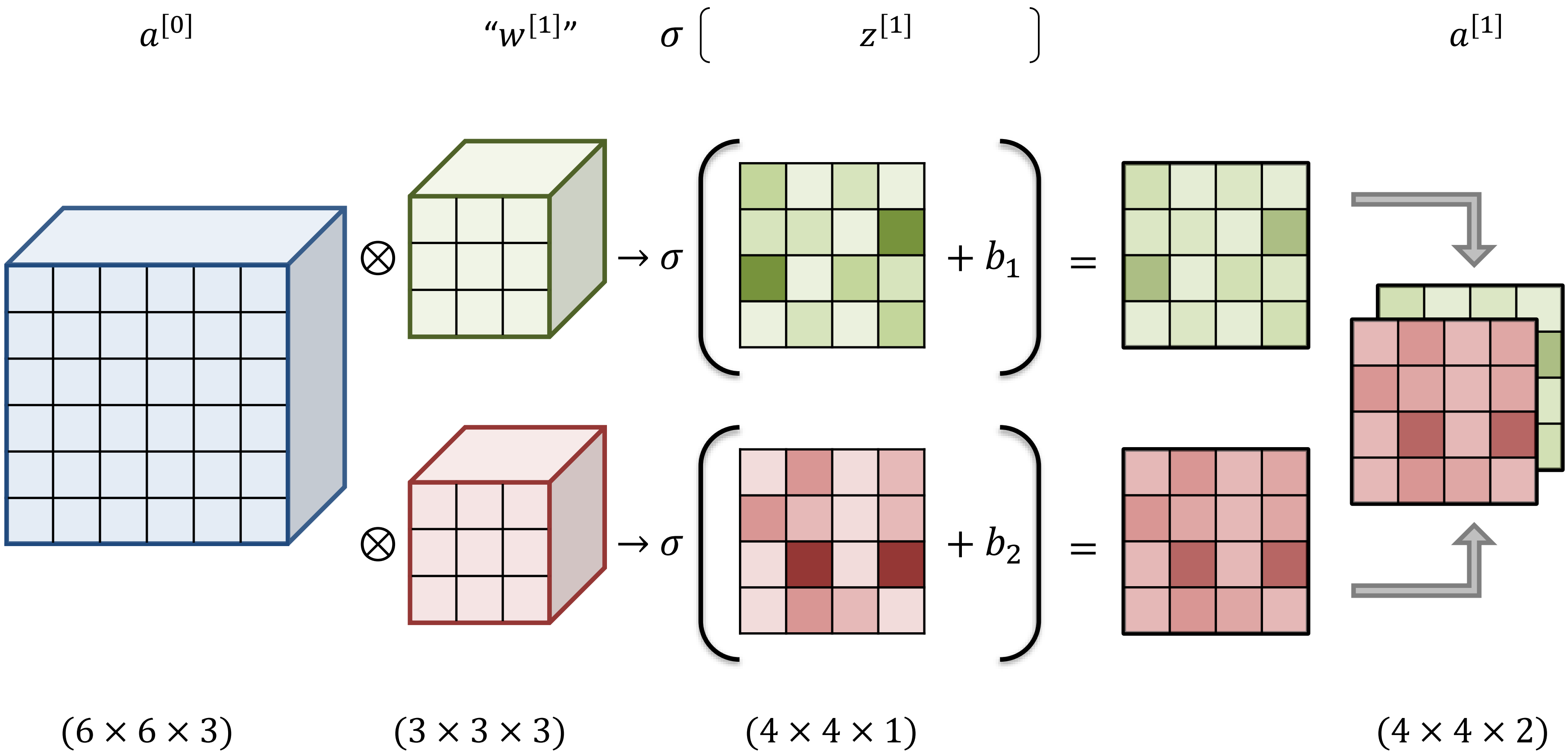
Multiple Filters

A different filter (set of weights) defines a different feature. We want to extract a number of different features from the images, each one giving rise to a separate activation map



A Convolutional Layer

Detecting multiple features (with different features) and stacking the activation maps constitutes one layer of a CNN



A Convolutional Layer

Input size: $\alpha^{[l-1]} \rightarrow n_W^{[l-1]} \times n_H^{[l-1]} \times n_c^{[l-1]}$

Output size: $\alpha^{[l]} \rightarrow n_W^{[l]} \times n_H^{[l]} \times n_c^{[l]}$

$$n_{W/H}^{[l]} = \frac{n_{W/H}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{l}} + 1$$

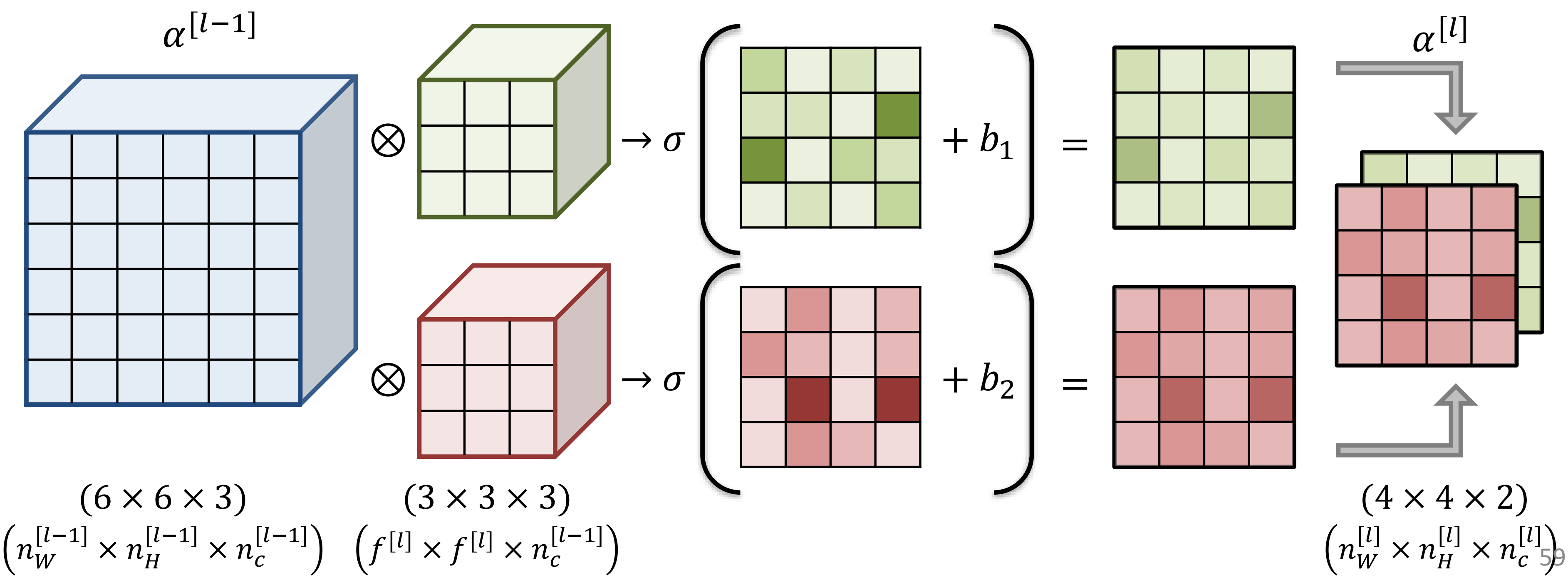
Filter size: $f^{[l]}$

Padding: $p^{[l]}$

Stride: $s^{[l]}$

Number of filters: $n_c^{[l]}$

Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

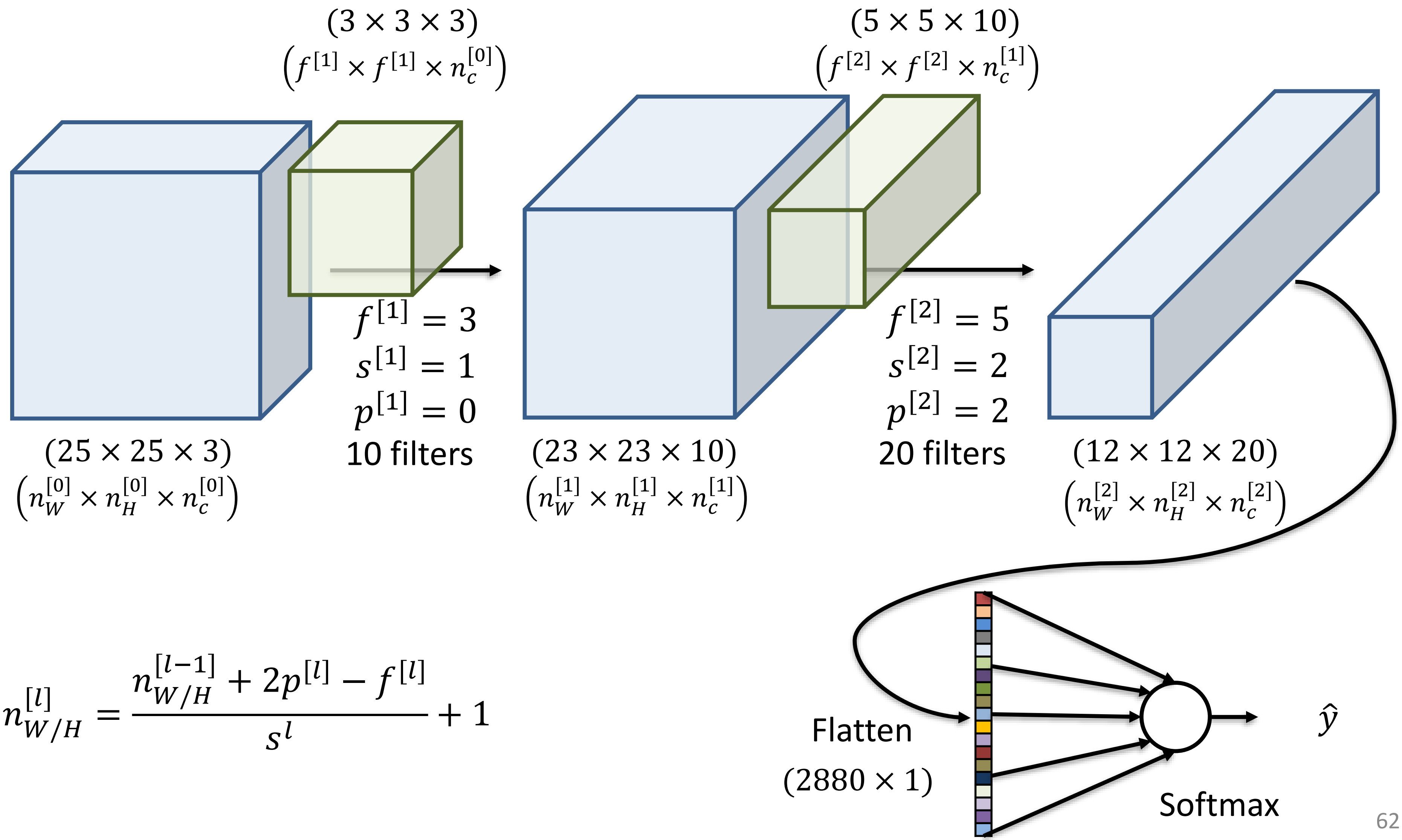


For Example

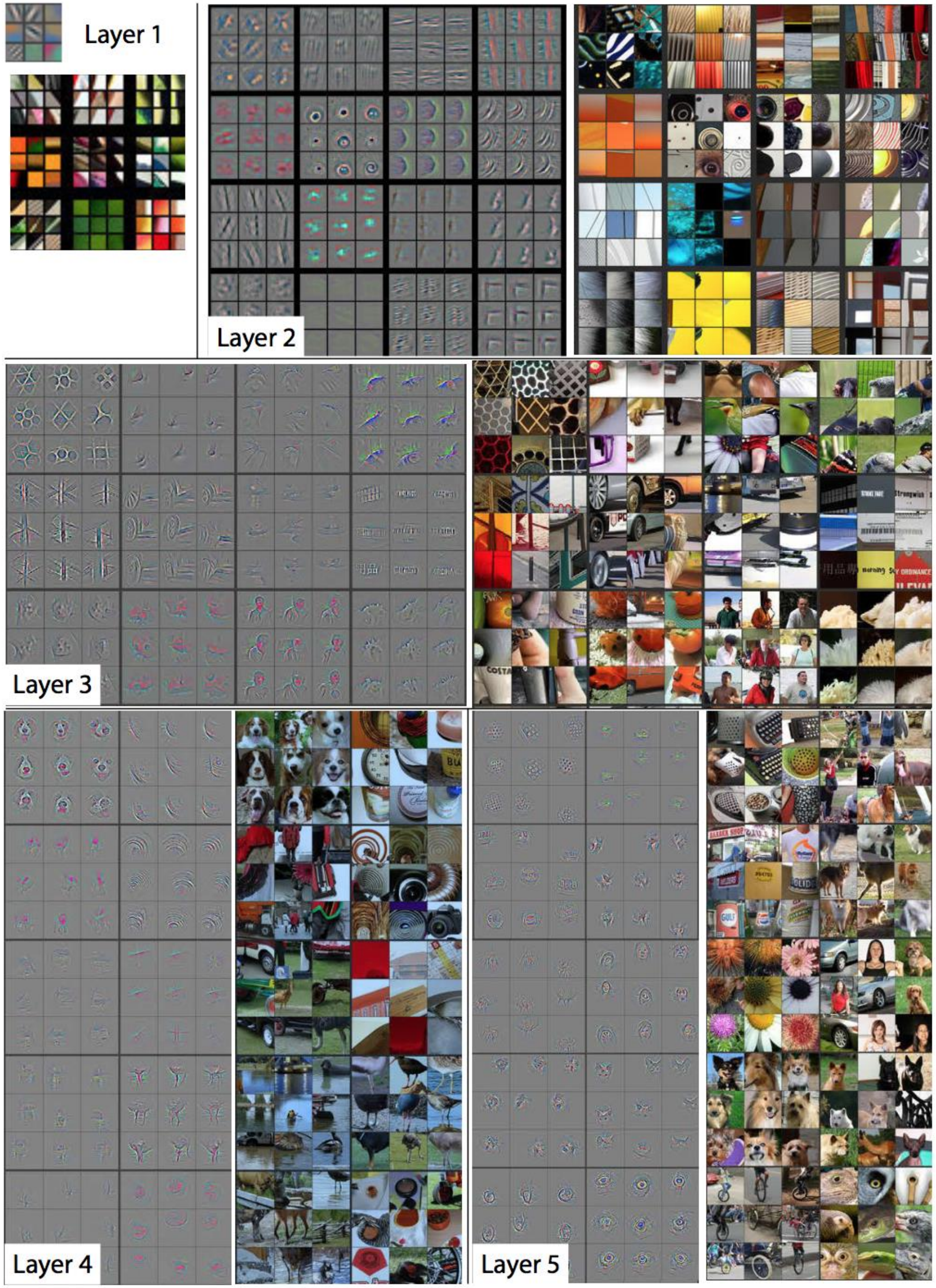
If you have 10 filters that are 5x5x3, and the input has 3 channels (640x480x3), how many parameters does that layer have?

Answer 760

Building a CNN

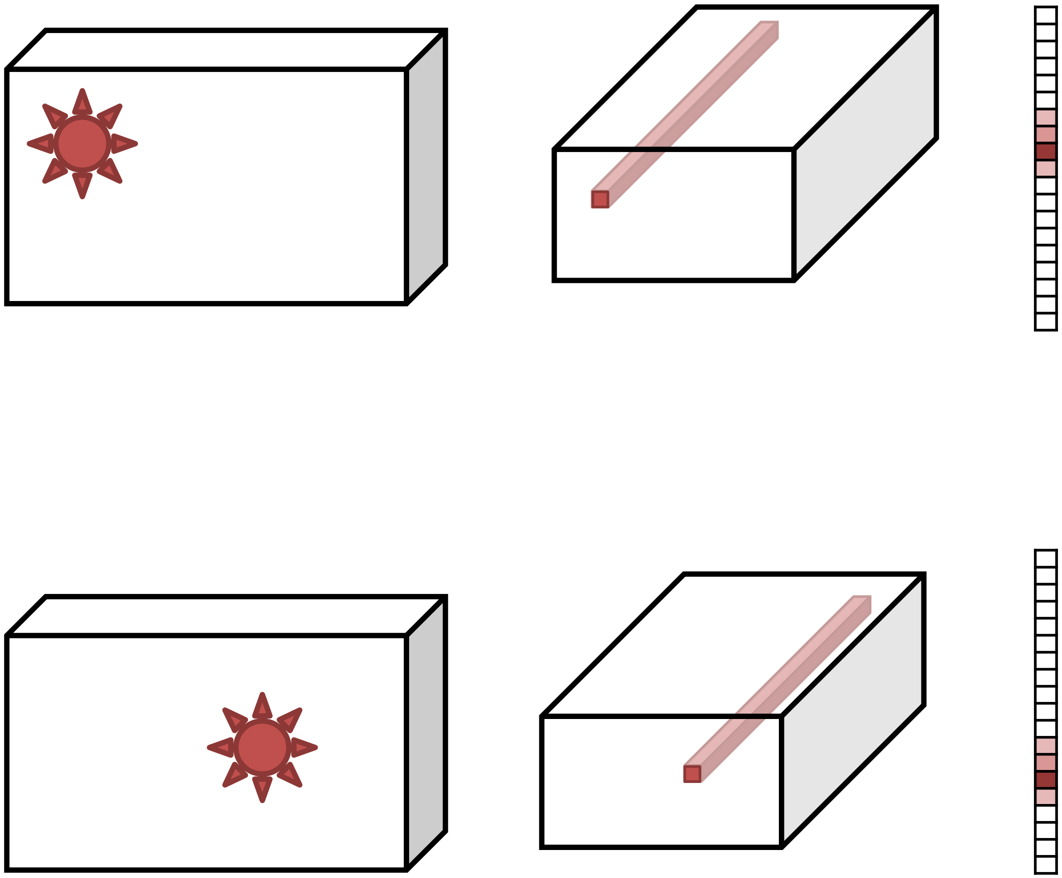


$$n_{W/H}^{[l]} = \frac{n_{W/H}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{l}} + 1$$



Visualizing and Understanding Convolutional Neural Networks (Zeiler & Fergus 2014)

Achieving Viewpoint Invariance



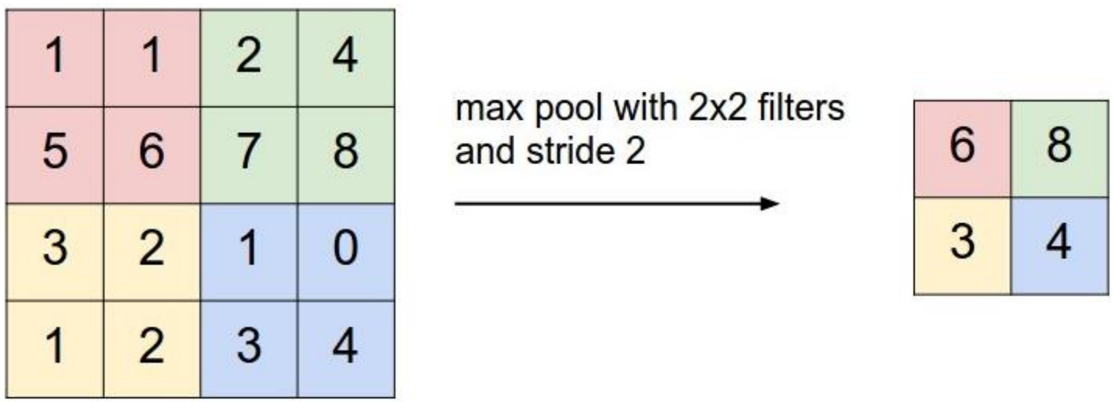
Replicating features achieves **equivariance**, not invariance. You can detect the **same thing** in **different places** (equivariance in the activations, invariance in the weights).

To achieve **viewpoint (translation) invariance** in the final activations we need to **pool features**.

The more we keep pooling, the more we lose precise positions.

Pooling

A **pooling layer** summarises a region of neurons from the previous layer. Max pooling for example is the same like asking if a particular feature has been detected anywhere in the receptive field



Hyperparameters
Filter size: f
Stride: s
No padding
NO learnable parameters!

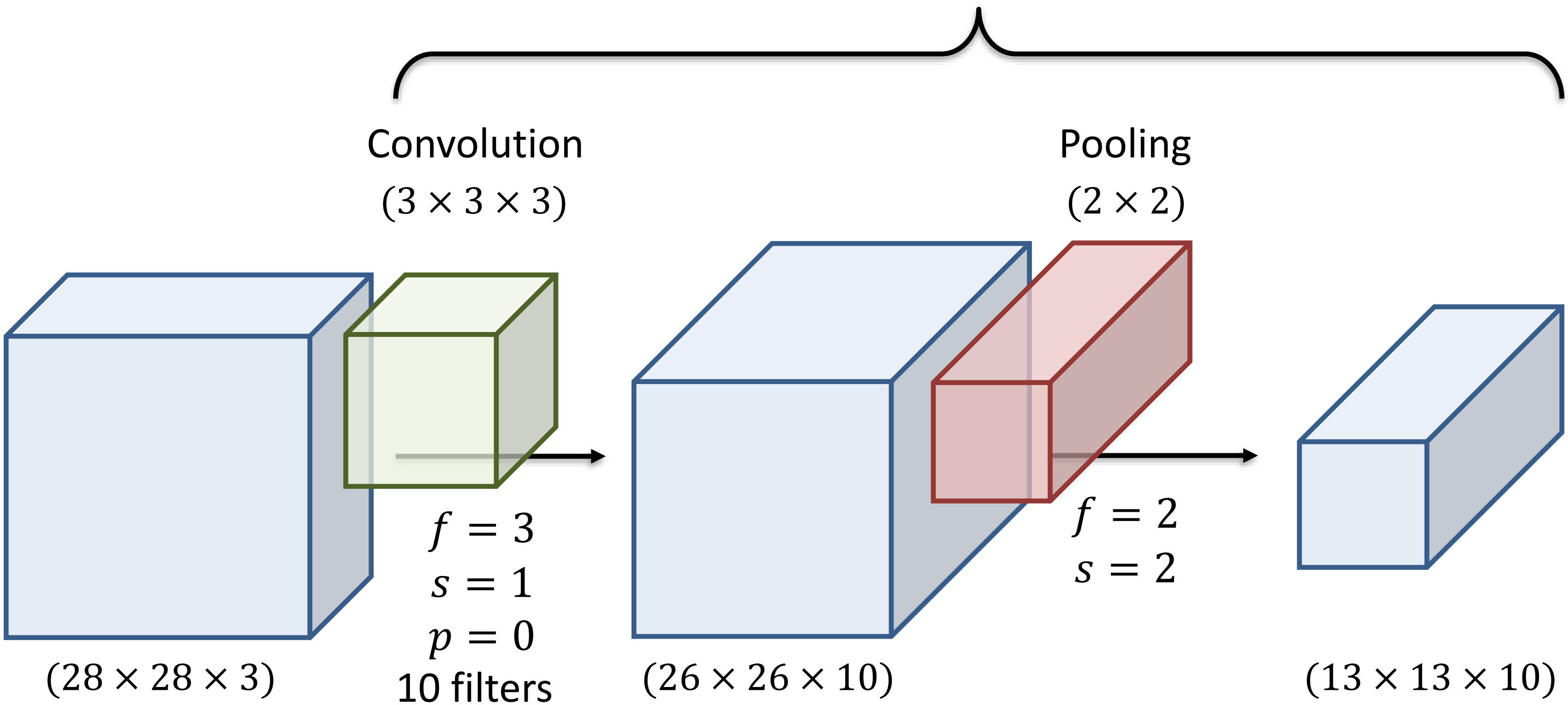
$$n_{W/H}^{[out]} = \frac{n_{W/H}^{[in]} - f}{s} + 1$$

$$n_C^{[out]} = n_C^{[in]}$$

Image credit: http://neuralnetworksanddeeplearning.com/index.html

Pooling

Can be reported as a single Layer, since pooling has no learnable parameters



Pooling

Pooling achieves a small amount of translational invariance. After several levels of pooling, we lose information about the precise positions of things.

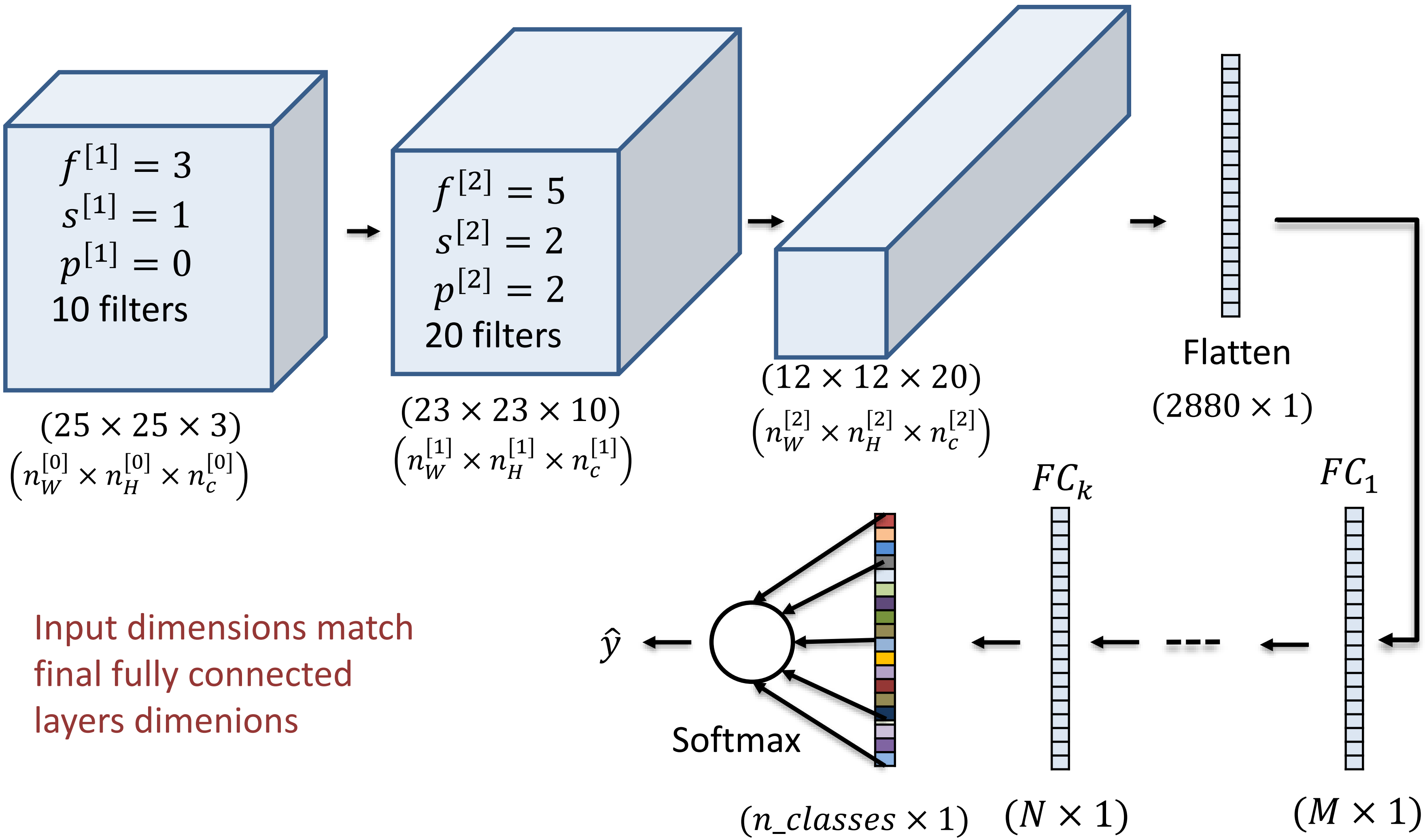
Max pooling

- Asks what is the best detection of a particular feature in the region
- Backpropagation: all gradient flows through the winner

Average pooling

- Calculates the average response to a feature
- Backpropagation: equal parts through all the units

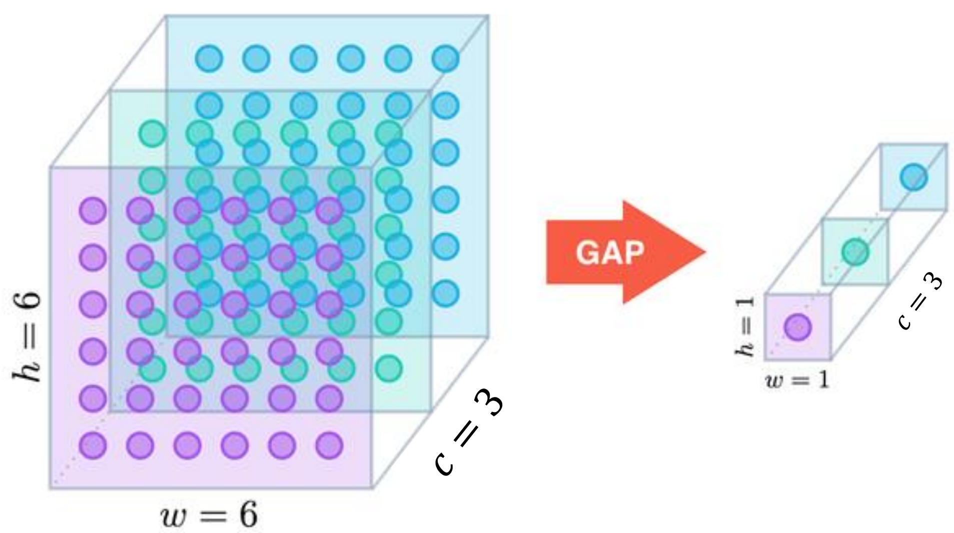
Building a CNN



Input dimensions match final fully connected layers dimenions

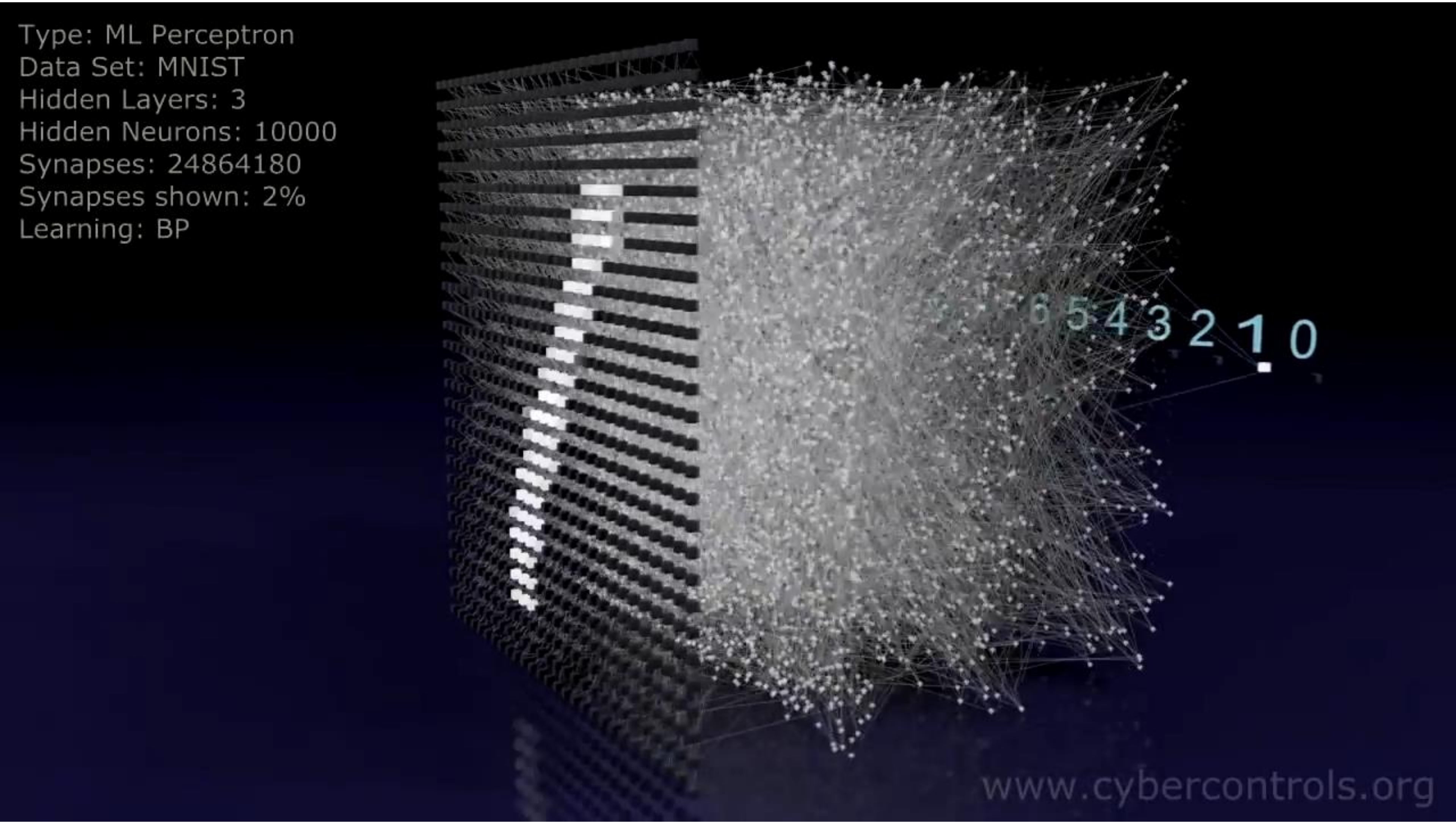
Global Average Pooling

Global Average Pooling was designed to replace fully connected layers. The idea is to generate one feature map for each category of the classification task in the last convolutional layer. Then take the average of each feature map, and the resulting vector is fed directly into the softmax layer. It enforces correspondence between feature maps and categories



Lin et al, "Network In Network", 2014 https://arxiv.org/pdf/1312.4400.pdf

A CNN compared to an MLP



Credit: Denis Dmitriev, https://www.cybercontrols.org/, 2016

Example Network - LeNet-5 (1998)

PROC. OF THE IEEE, NOVEMBER 1998 1

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

PROC. OF THE IEEE, NOVEMBER 1998 7

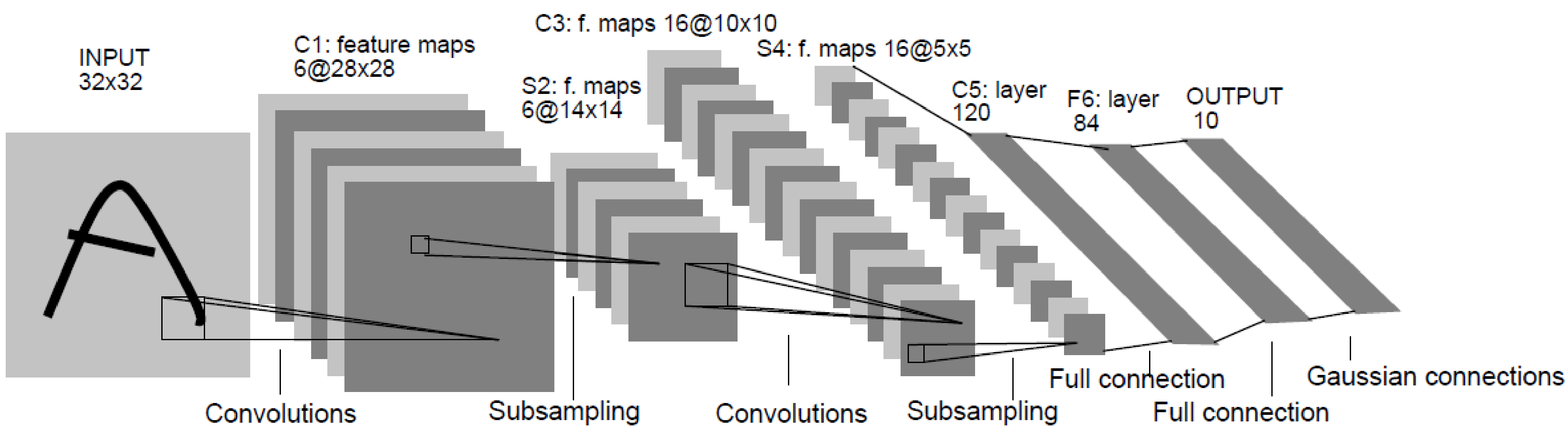
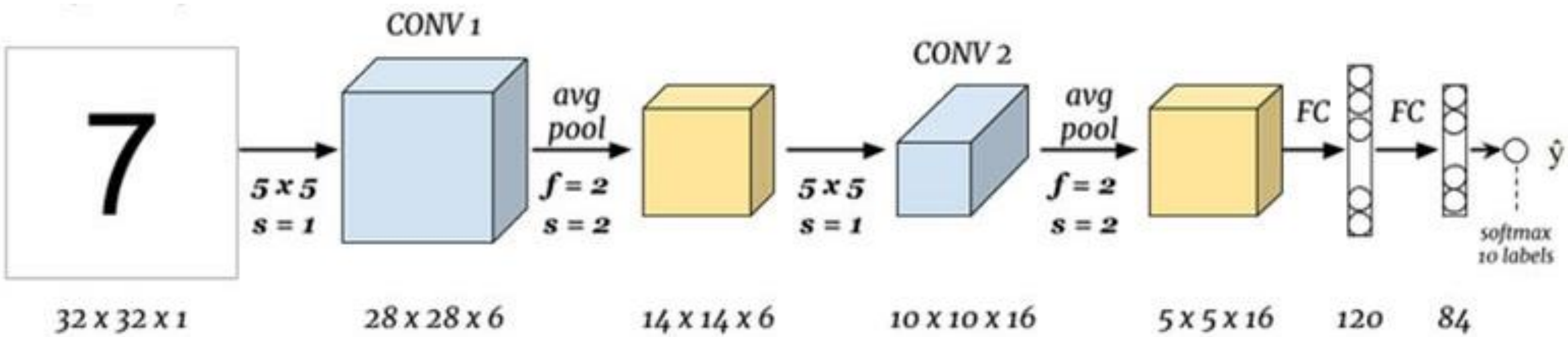


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Example Network - LeNet-5 (1998)



LeNet-5 mistakes

82 Errors
In 10,000
test images

Fig. 8. The 82 test patterns misclassified by LeNet-5. Below each image is displayed the correct answers (left) and the network answer (right). These errors are mostly caused either by genuinely ambiguous patterns, or by digits written in a style that are under-represented in the training set.

DATA AUGMENTATION

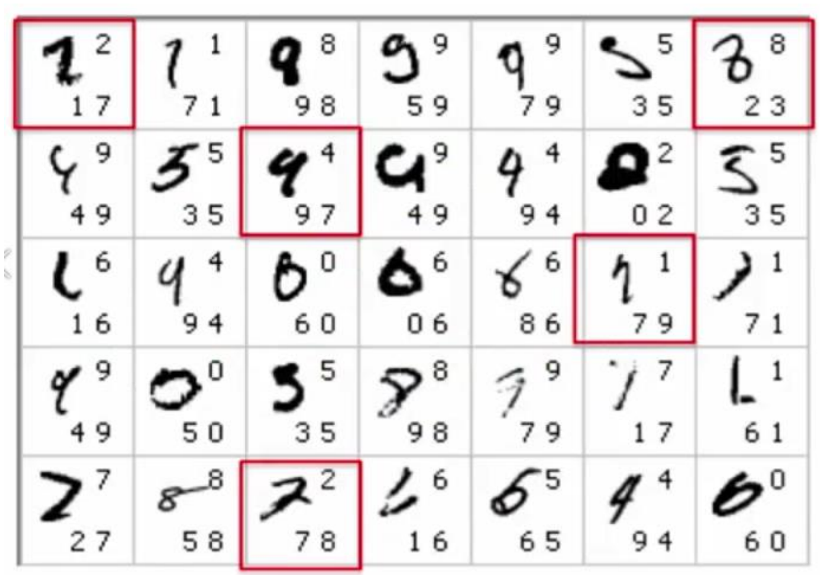
Data Augmentation

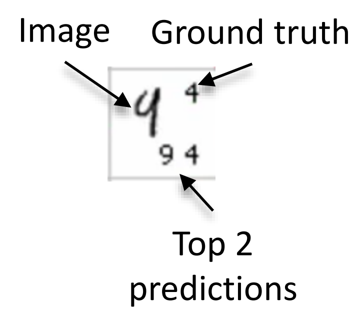
Data augmentation is another way to introduce our prior knowledge.

Instead of designing the network and hyperparameters (specifying to some degree how to solve the problem), we can instead use a more flexible model and design more data.

It allows much more flexibility to the system to figure out how to do things

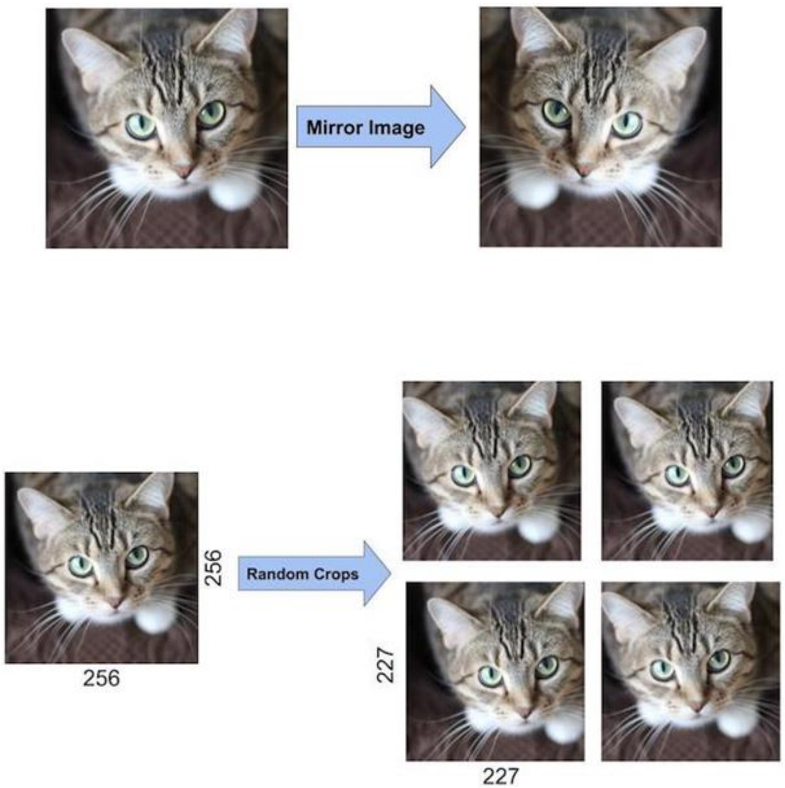
~25 Errors
In 10,000 test images

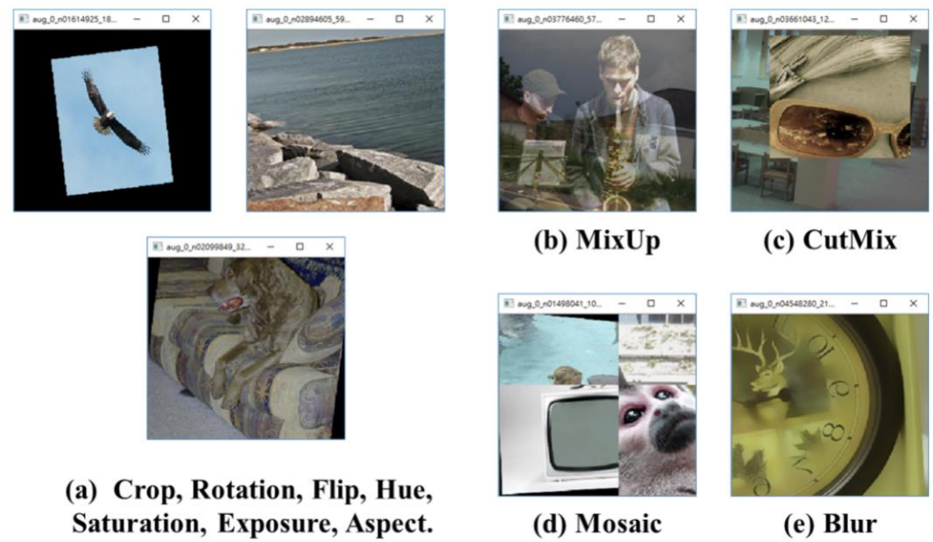




D.C. Ciresan, et al. "Deep, big, simple neural nets for handwritten digit recognition" 2010

Augmentation in SoA Models

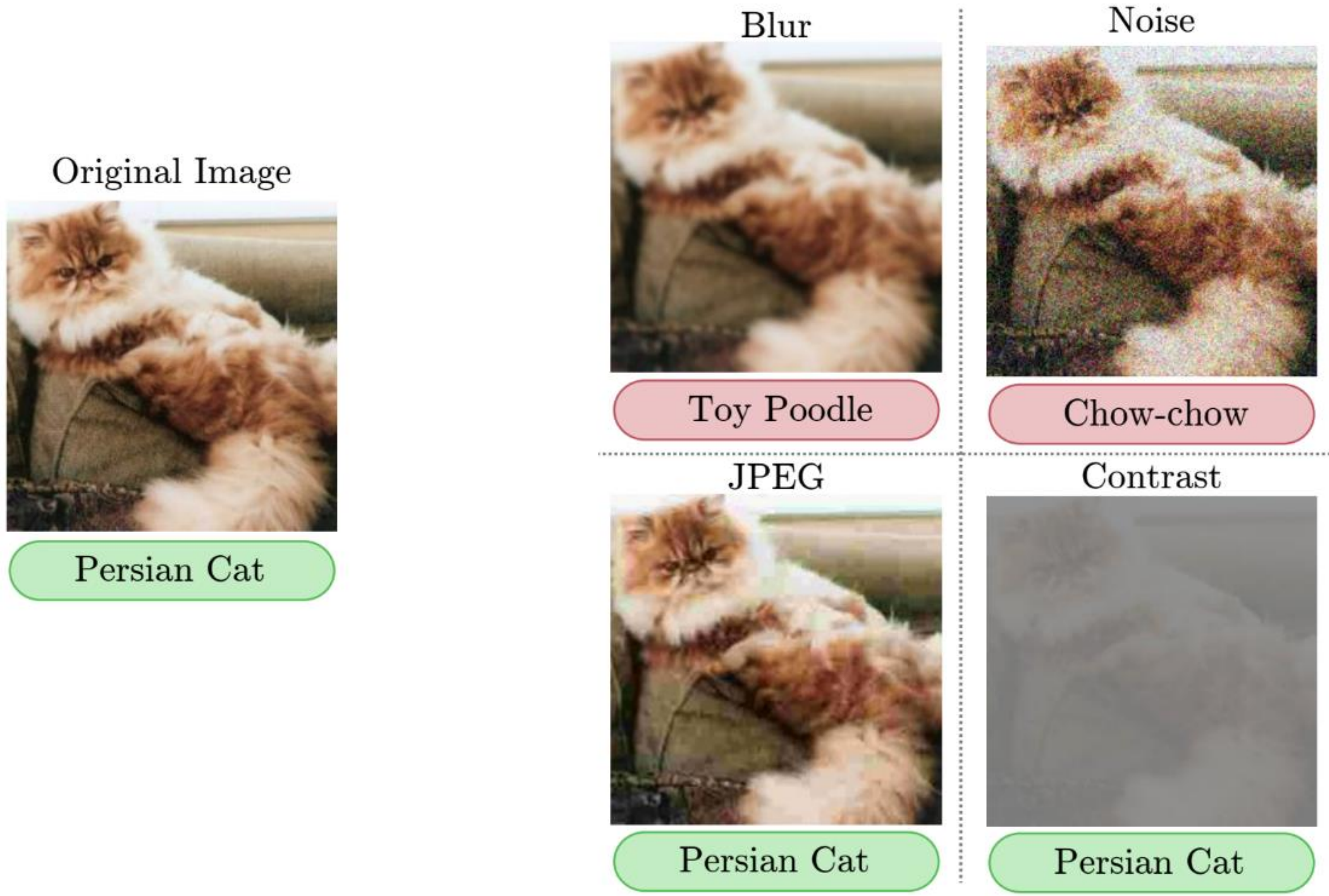




Alex Krizhevsky et al "ImageNet classification with deep convolutional neural networks" 2012

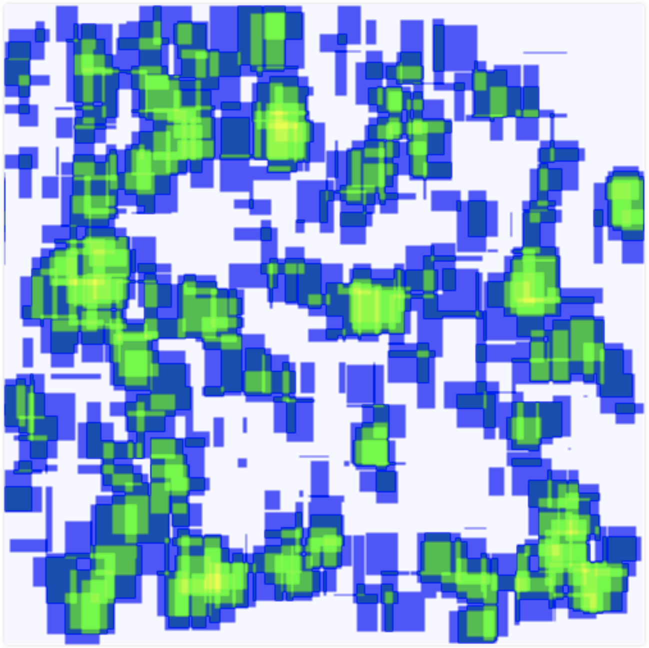
A. Bochkovskiy, et al "Yolov4: Optimal speed and accuracy of object detection" 2020

The importance of augmentation

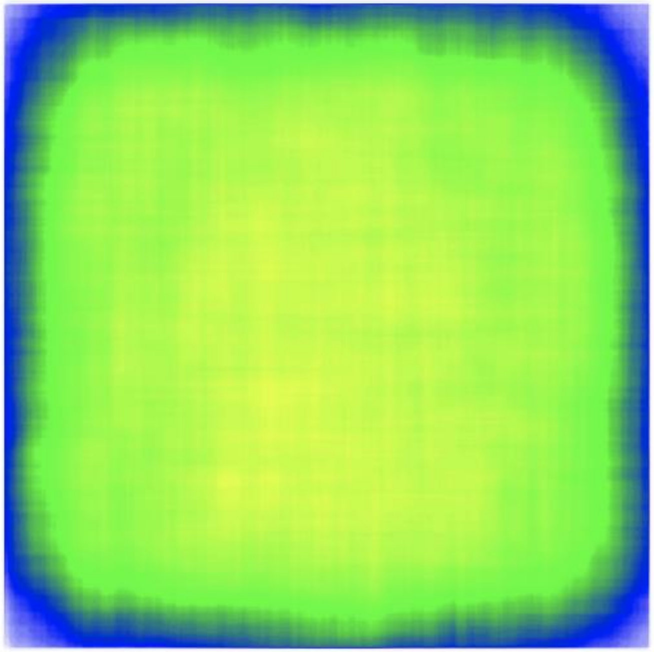


S. Dodge, "Understanding How Image Quality Affects Deep Neural Networks", 2016

The importance of augmentation



Platelets spatial distribution



After a few flip and rotates

Photometric or Geometric Distortion

gaussian noise

elastic transform

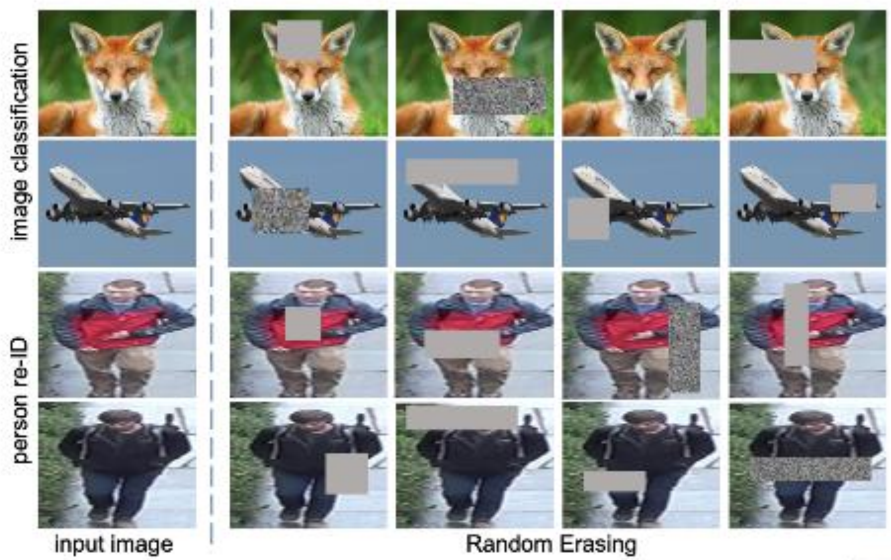
random brightness
and contrast

random gamma

Contrast limited adaptive
histogram equalisation

blur

Image Occlusion



Random erase: replaces regions of the image with random values
https://arxiv.org/pdf/1708.04896.pdf

Hide and Seek: Divide the image into a grid of SxS patches. Hide each patch with some probability (p_hide)
https://arxiv.org/pdf/1704.04232.pdf

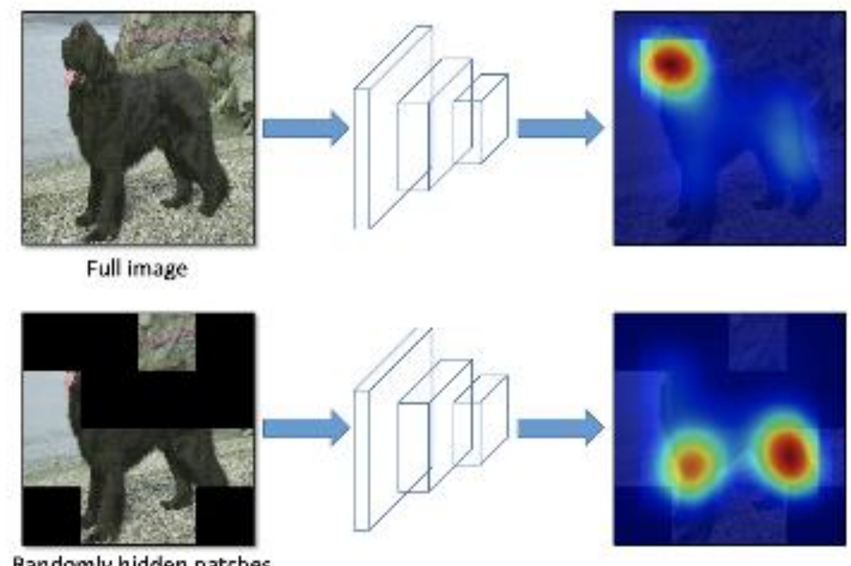
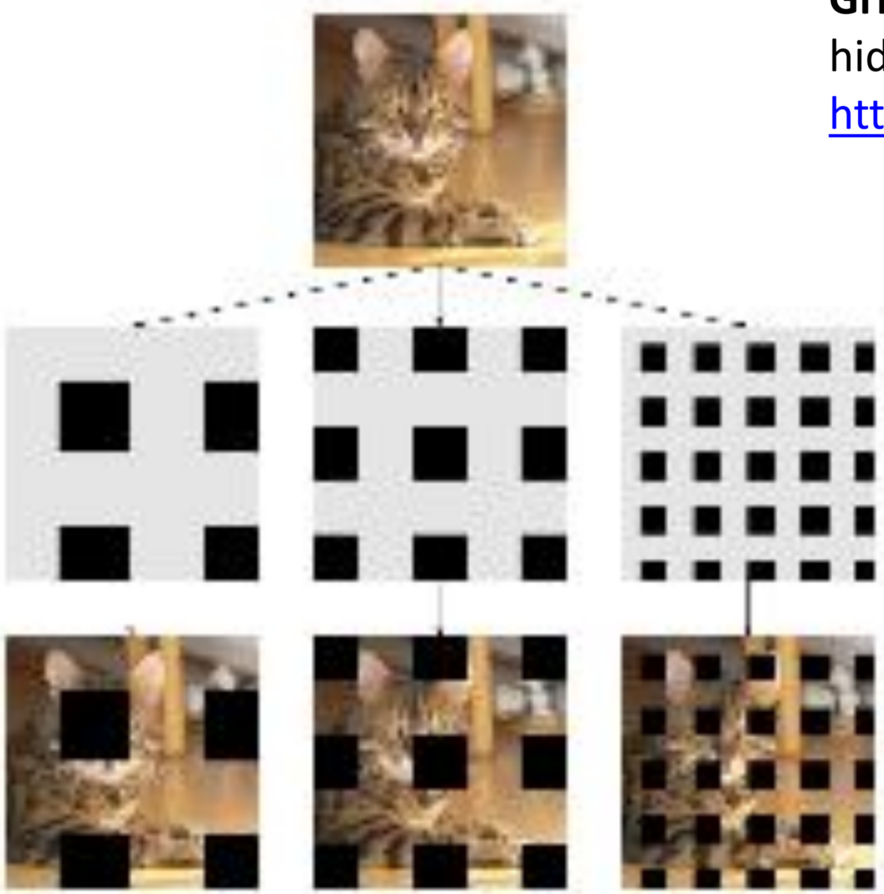


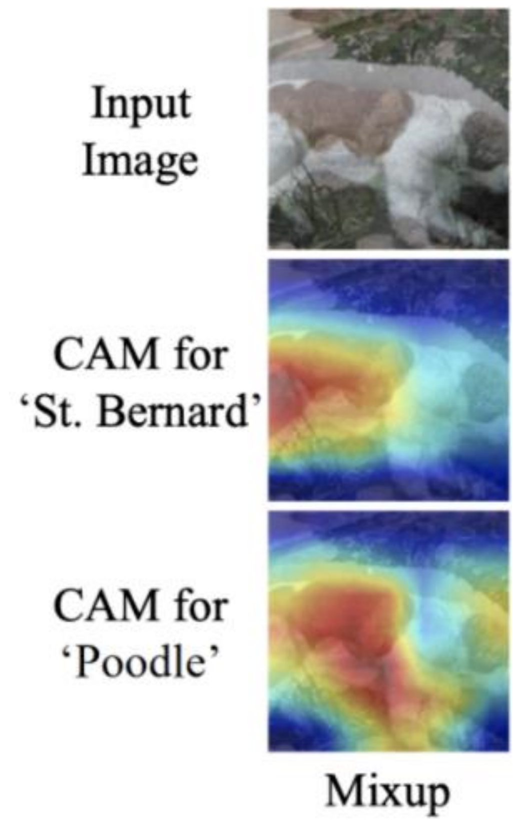
Image Occlusion



Grid Mask: Regions of the image are hidden in a grid like fashion
https://arxiv.org/pdf/2001.04086.pdf

Image / Labels mixing

Mosaic data augmentation: combines 4 training images into one in certain ratios



MixUp: Convex overlaying of image pairs and their labels
https://arxiv.org/pdf/1905.04899.pdf

Domain Specific Augmentation

Images

- Geometric transformations –randomly flip, crop, rotate or translate images
- Color space transformations – change RGB color channels, intensify any color
- Kernel filters – sharpen or blur an image
- Random Erasing – delete a part of the initial image
- Mixing images –mix images with one another

Text

- Word/sentence shuffling
- Word replacement – replace words with synonyms
- Syntax-tree manipulation – paraphrase the sentence to be grammatically correct using the same words

Audio

- Noise injection
- Shifting
- Changing the speed of the tape

Transforms Library in PyTorch

Transforms library is the augmentation part of the torchvision package that consists of popular datasets, model architectures, and common image transformations for Computer Vision tasks.

Additionally, there is the **torchvision.transforms.functional** module. It has various functional transforms that give fine-grained control over the transformations.

Have a look at the documentation here:
https://pytorch.org/vision/stable/transforms.html

Custom Libraries

Augmentor

- https://augmentor.readthedocs.io/en/master/userguide/install.html
- Allows to pick a probability parameter for every transformation operation that controls how often the operation is applied. Augmenting pipeline that chains together a number of operations that are applied stochastically.

Albumentations

- https://albumentations.ai/docs/getting_started/installation/
- Optimized for maximum speed and performance, many image transformation operations, integration with PyTorch and Keras

ImgAug

- https://imgaug.readthedocs.io/en/latest/
- Easily execute augmentations on multiple CPU cores.

Autoaugment

- https://github.com/barisozmen/deepaugment
- Autoaugment algorithm (Google 2018) designed to search for the best augmentation policies.

Kornia Augmentation

- https://kornia.readthedocs.io/en/latest/augmentation.html
- Derivable computer vision operations, that can be performed in the GPU