Τεχνητή Νοημοσύνη

23η διάλεξη (2023-24)

Ίων Ανδρουτσόπουλος

http://www.aueb.gr/users/ion/

Τι θα ακούσετε σήμερα

- Συνελικτικά νευρωνικά δίκτυα (CNNs).
- Εφαρμογές στην υπολογιστική όραση.
- Προ-εκπαιδευμένα νευρωνικά δίκτυα.
- Επαύξηση δεδομένων (data augmentation).

Convolutions on images

Averaging each pixel with its neighboring values blurs an image:

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

From the blog post "Understanding Convolutional Neural Networks for NLP" of Denny Britz, 2015. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Convolutions on images

Input

-1	**1**	-1	-1	-1	-1
1	**1**	**1**	-1	-1	-1
-1	**1**	-1	-1	-1	-1
-1	-1	-1	**1**	-1	**1**
-1	-1	-1	-1	**1**	-1
-1	-1	-1	**1**	-1	**1**

Kernel (Filter)

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

Feature Map

- **Input: black/white image** with pixel values -1 or +1.
- **Check** if the **input contains any crosses** and report **where**.

Convolutions on images

Input | **Kernel (Filter)** | **Feature Map**

Input:

-1	**1**	-1	-1	-1	-1
1	**1**	**1**	-1	-1	-1
-1	**1**	-1	-1	-1	-1
-1	-1	-1	**1**	-1	**1**
-1	-1	-1	-1	**1**	-1
-1	-1	-1	**1**	-1	**1**

Kernel (Filter):

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

Input:

-1	**1**	-1	-1	-1	-1
1	**1**	**1**	-1	-1	-1
-1	**1**	-1	-1	-1	-1
-1	-1	-1	**1**	-1	**1**
-1	-1	-1	-1	**1**	-1
-1	-1	-1	**1**	-1	**1**

Kernel (Filter):

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

Feature Map:

9

Convolutions on images

Input | **Kernel (Filter)** | **Feature Map**

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

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

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

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

9	

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

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

9	-1

Convolutions on images

Input

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

Kernel (Filter)

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

Feature Map

9	-1	1

- **Let *X*** be the part of the input where we apply the kernel (filter).
- **Let *W*** be the kernel.
- The resulting **feature** of the feature map is: $\boldsymbol{\sum_{i=1}^{3}\sum_{j=1}^{3} W_{i,j}\, X_{i,j}}$
- In practice, we would also use an **activation function** and **bias term**: $f\left(\sum_{i=1}^{3}\sum_{j=1}^{3} W_{i,j}X_{i,j} + b\right)$

Convolutions on images

Input

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

Kernel (Filter)

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

Feature Map

9	-1	1

...

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

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

9	-1	1	-1
-1	-1	-1	-5
1	-1	-1	5
-1	-5	5	-7

- We can think of the resulting **feature map as a new "image"** that indicates the **position(s) of the cross(es)** in the original image.
  - No need to have the crosses at particular parts of the image.
- The new "image" is **4x4 instead of 6x6**, because the **kernel could not slide outside the boundaries** of the original image.

Wide convolutions on images

Input

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

Kernel (Filter)

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

Feature Map

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

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

0

- We can **pad** the surrounding of the image with zeros, to allow the kernel to slide outside the image boundaries.
- We can now obtain a **feature map** with the **same resolution as the input** image (6x6).

Wide convolutions on images

Input

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

Kernel (Filter)

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

Feature Map

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

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

0

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

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

0	-2

Wide convolutions on images

Input

0	0	0	0	0	0	0	0
0	-1	**1**	-1	-1	-1	-1	0
0	**1**	**1**	**1**	-1	-1	-1	0
0	-1	**1**	-1	-1	-1	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	-1	-1	-1	-1	**1**	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	0	0	0	0	0	0	0

Kernel (Filter)

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

Feature Map

0	-2	0			

...

0	0	0	0	0	0	0	0
0	-1	**1**	-1	-1	-1	-1	0
0	**1**	**1**	**1**	-1	-1	-1	0
0	-1	**1**	-1	-1	-1	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	-1	-1	-1	-1	**1**	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	0	0	0	0	0	0	0

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

0	-2	0	-4	-2	-2
-2					

0	0	0	0	0	0	0	0
0	-1	**1**	-1	-1	-1	-1	0
0	**1**	**1**	**1**	-1	-1	-1	0
0	-1	**1**	-1	-1	-1	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	-1	-1	-1	-1	**1**	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	0	0	0	0	0	0	0

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

0	-2	0	-4	-2	-2
-2	9				

Wide convolutions on images

Input

0	0	0	0	0	0	0	0
0	-1	**1**	-1	-1	-1	-1	0
0	**1**	**1**	**1**	-1	-1	-1	0
0	-1	**1**	-1	-1	-1	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	-1	-1	-1	-1	**1**	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	0	0	0	0	0	0	0

Kernel (Filter)

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

Feature Map

0	-2	0	-4	-2	-2
-2	9	-1			

...

0	0	0	0	0	0	0	0
0	-1	**1**	-1	-1	-1	-1	0
0	**1**	**1**	**1**	-1	-1	-1	0
0	-1	**1**	-1	-1	-1	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	-1	-1	-1	-1	**1**	-1	0
0	-1	-1	-1	**1**	-1	**1**	0
0	0	0	0	0	0	0	0

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

0	-2	0	-4	-2	-2
-2	**9**	-1	1	-1	-2
0	-1	-1	-1	-5	0
-4	1	-1	-1	5	-2
-2	-1	-5	5	-7	4
-2	-2	0	-2	4	-2

- ***X*: entire input** image. ***F*: feature map**.
- ***W*: kernel**, but with rows and columns numbered $-1, 0, 1$.
- **Feature map values**: $F_{i,j} = \sum_{k=-1}^{1} \sum_{l=-1}^{1} W_{k,l} X_{i+k,j+l}$
- In practice: $F_{i,j} = f(\sum_{k=-1}^{1} \sum_{l=-1}^{1} W_{k,l} X_{i+k,j+l} + b)$

Convolution or cross-correlation?

Optional study

- **Cross-correlation**: $F_{i,j} = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} W_{k,l} \, X_{i+k,j+l}$
- **Convolution**: $F_{i,j} = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} W_{k,l} \, X_{i-k,j-l} = W * X$
- We are **actually computing cross-correlations**, not convolutions.
  - The **cross-correlations** we compute are **equal to convolutions with the kernel (or the image) flipped** both vertically and horizontally.
    - Convolution is like cross-correlation, but flips one of the two signals. We don't flip the kernel inside the cross-correlation, which is equivalent to giving the kernel already flipped to the convolution; the convolution will flip the kernel once more, ending up using the kernel without flipping.
  - So we actually compute **convolutions with flipped kernels** or **cross-correlations with the original kernels**.
  - The **example kernels were symmetric**, so no difference.
  - **In CNNs** (Convolutional Neural Networks), the **kernels are learned**, so **we don't care** if they are flipped in the "convolutions" we compute.
  - So we usually say **CNNs "compute convolutions"**, though we actually use the formulae of cross-correlations.

Two kernels

Input

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

Two Kernels

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

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

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

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

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

Feature Map of Kernel 1 ("+")

0

Feature Map of Kernel 2 ("X")

-2

- We now want to **check the input image for crosses and "X"s**.
- We use **two kernels**, one for crosses, one for "X"s.

Two kernels

Input

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

Two Kernels

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

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

Feature Map of Kernel 1 ("+")

Feature Map of Kernel 2 ("X")

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

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

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

Feature map of Kernel 1: 0

Feature map of Kernel 2: -2

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

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

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

Feature map of Kernel 1: 0 -2

Feature map of Kernel 2: -2 4

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

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

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

Feature map of Kernel 1: 0 -2 0

Feature map of Kernel 2: -2 4 -2

Two kernels

We can **think of the two feature maps as two "channels" of the new image**, one for "+" info, one for "X" info.

Input

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

Two Kernels

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

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

Feature Map of Kernel 1 ("+")

0	-2	0	-4	-2	-2
-2					

Feature Map of Kernel 2 ("X")

-2	4	-2	2	0	0
4					

Input

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

Two Kernels

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

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

Feature Map of Kernel 1 ("+")

0	-2	0	-4	-2	-2
-2	9				

Feature Map of Kernel 2 ("X")

-2	4	-2	2	0	0
4	-7				

Input

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

Two Kernels

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

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

Feature Map of Kernel 1 ("+")

0	-2	0	-4	-2	-2
-2	9	-1			

Feature Map of Kernel 2 ("X")

-2	4	-2	2	0	0
4	-7	3			

...

Input

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

Two Kernels

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

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

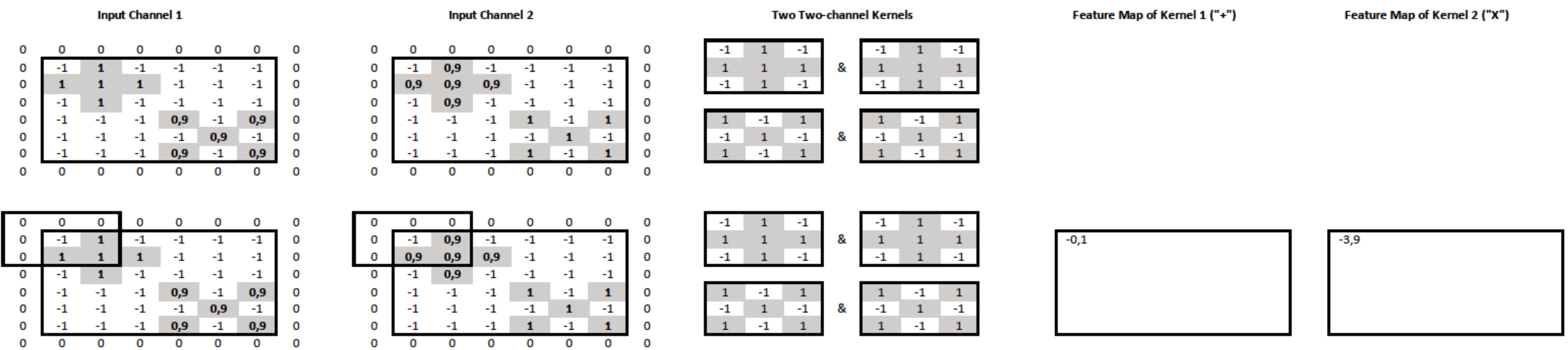
Feature Map of Kernel 1 ("+")

0	-2	0	-4	-2	-2
-2	9	-1	1	-1	-2
0	-1	-1	-1	-5	0
-4	1	-1	-1	5	-2
-2	-1	-5	5	-7	4
-2	-2	0	-2	4	-2

Feature Map of Kernel 2 ("X")

-2	4	-2	2	0	0
4	-7	3	-3	-1	0
-2	3	-1	-1	3	-2
2	-3	-1	3	-7	4
0	-1	3	-7	9	-6
0	0	-2	4	-6	4

Two input channels too

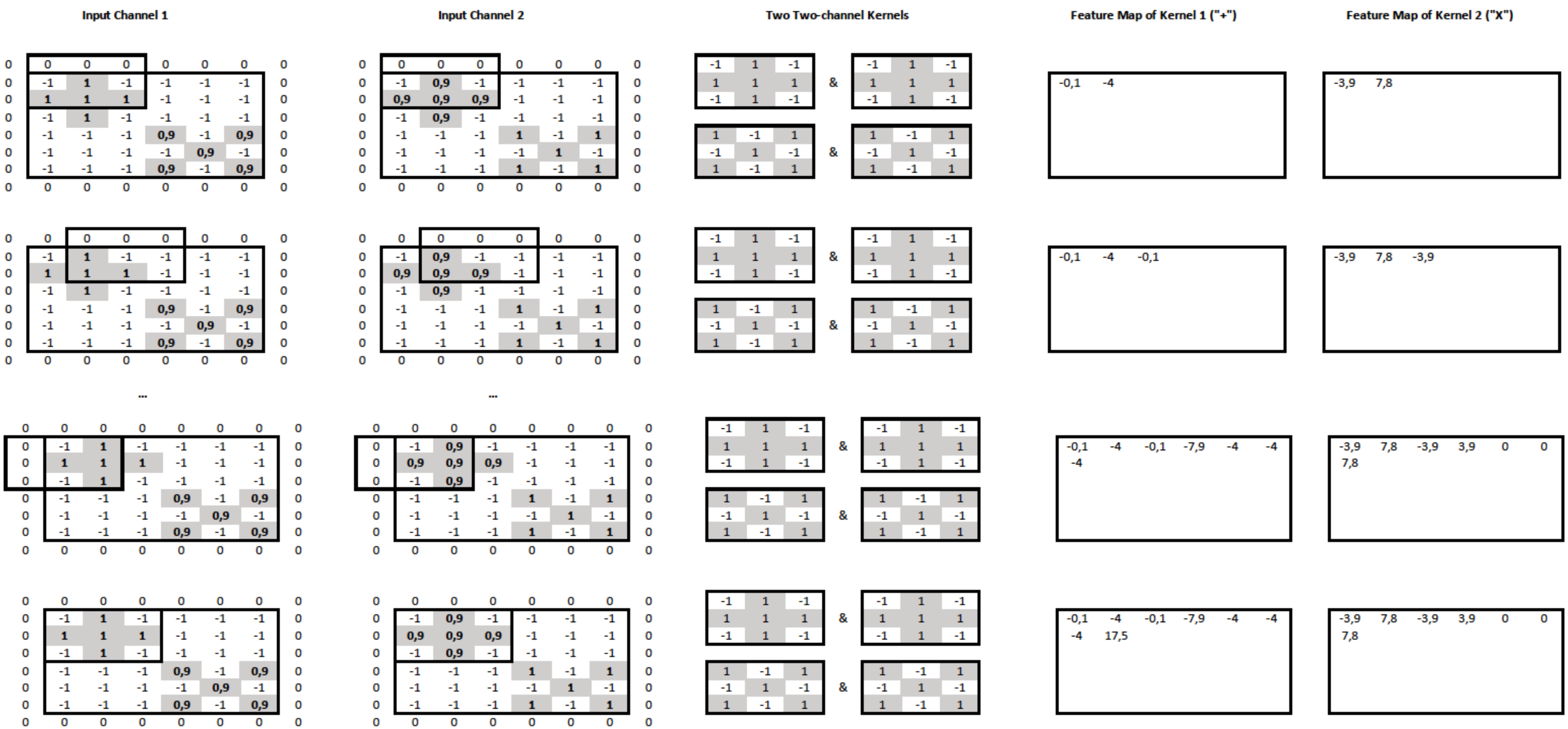


- The **input image** now also has **two channels** (e.g., from grayscale and depth cameras). **Each kernel** now operates on **both input channels**.
  - It has **two slices**, one per input channel ($c = 1, c = 2$).
- We have **two kernels**, so the **output** also has **two channels**.
- At the output feature map of kernel $W^{(m)}$, the value at cell (i, j) is:

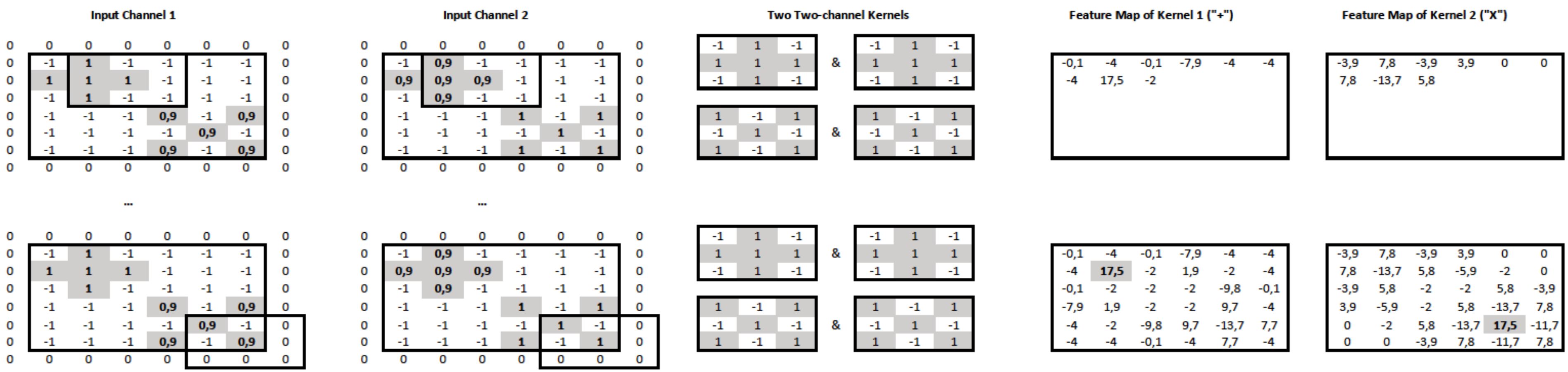
$$F_{i,j,m} = \sum_{k=-1}^{1} \sum_{l=-1}^{1} \sum_{c=1}^{2} W_{k,l,c}^{(m)} X_{i+k,j+l,c}$$

- In practice, we would also have an activation function and bias term.

Two input channels too



Two input channels too



- We now have a mechanism, a "**convolutional layer**", that maps an **input image of any number of channels** to a new **output "image" of any number of channels** (feature maps).
  - The **kernels** will have **as many slices as the input channels**.
  - The **number of kernels** will be **equal to the number of output channels**.
- We can **stack** multiple **convolutional layers**.
  - Each one will operate on the "image" produced by the previous layer.
  - **All kernels** will be randomly initialized and **learned via backpropagation**.

Max-pooling

Feature Map of Kernel 1 ("+") | **Feature Map of Kernel 2 ("X")** | **Max-Pooling (2,2) with Stride (2,2)**

-0,1	-4	-0,1	-7,9	-4	-4
-4	**17,5**	-2	1,9	-2	-4
-0,1	-2	-2	-2	-9,8	-0,1
-7,9	1,9	-2	-2	9,7	-4
-4	-2	-9,8	9,7	-13,7	7,7
-4	-4	-0,1	-4	7,7	-4

-3,9	7,8	-3,9	3,9	0	0
7,8	-13,7	5,8	-5,9	-2	0
-3,9	5,8	-2	-2	5,8	-3,9
3,9	-5,9	-2	5,8	-13,7	7,8
0	-2	5,8	-13,7	**17,5**	-11,7
0	0	-3,9	7,8	-11,7	7,8

17,5		

7,8		

-0,1	-4	-0,1	-7,9	-4	-4
-4	**17,5**	-2	1,9	-2	-4
-0,1	-2	-2	-2	-9,8	-0,1
-7,9	1,9	-2	-2	9,7	-4
-4	-2	-9,8	9,7	-13,7	7,7
-4	-4	-0,1	-4	7,7	-4

-3,9	7,8	-3,9	3,9	0	0
7,8	-13,7	5,8	-5,9	-2	0
-3,9	5,8	-2	-2	5,8	-3,9
3,9	-5,9	-2	5,8	-13,7	7,8
0	-2	5,8	-13,7	**17,5**	-11,7
0	0	-3,9	7,8	-11,7	7,8

17,5	1,9	

7,8	5,8	

-0,1	-4	-0,1	-7,9	-4	-4
-4	**17,5**	-2	1,9	-2	-4
-0,1	-2	-2	-2	-9,8	-0,1
-7,9	1,9	-2	-2	9,7	-4
-4	-2	-9,8	9,7	-13,7	7,7
-4	-4	-0,1	-4	7,7	-4

-3,9	7,8	-3,9	3,9	0	0
7,8	-13,7	5,8	-5,9	-2	0
-3,9	5,8	-2	-2	5,8	-3,9
3,9	-5,9	-2	5,8	-13,7	7,8
0	-2	5,8	-13,7	**17,5**	-11,7
0	0	-3,9	7,8	-11,7	7,8

17,5	1,9	-2

7,8	5,8	0

- We keep the **max value of each window**, separately from each channel.
- The **stride** determines **how much the window shifts** vertically & horizontally.

Max-pooling

Feature Map of Kernel 1 ("+")

-0,1	-4	-0,1	-7,9	-4	-4
-4	**17,5**	-2	1,9	-2	-4
-0,1	-2	-2	-2	-9,8	-0,1
-7,9	1,9	-2	-2	9,7	-4
-4	-2	-9,8	9,7	-13,7	7,7
-4	-4	-0,1	-4	7,7	-4

Feature Map of Kernel 2 ("X")

-3,9	7,8	-3,9	3,9	0	0
7,8	-13,7	5,8	-5,9	-2	0
-3,9	5,8	-2	-2	5,8	-3,9
3,9	-5,9	-2	5,8	-13,7	7,8
0	-2	5,8	-13,7	**17,5**	-11,7
0	0	-3,9	7,8	-11,7	7,8

Max-Pooling (2,2) with Stride (2,2)

17,5	1,9	-2
1,9		

7,8	5,8	0
5,8		

...

-0,1	-4	-0,1	-7,9	-4	-4
-4	**17,5**	-2	1,9	-2	-4
-0,1	-2	-2	-2	-9,8	-0,1
-7,9	1,9	-2	-2	9,7	-4
-4	-2	-9,8	9,7	-13,7	7,7
-4	-4	-0,1	-4	7,7	-4

...

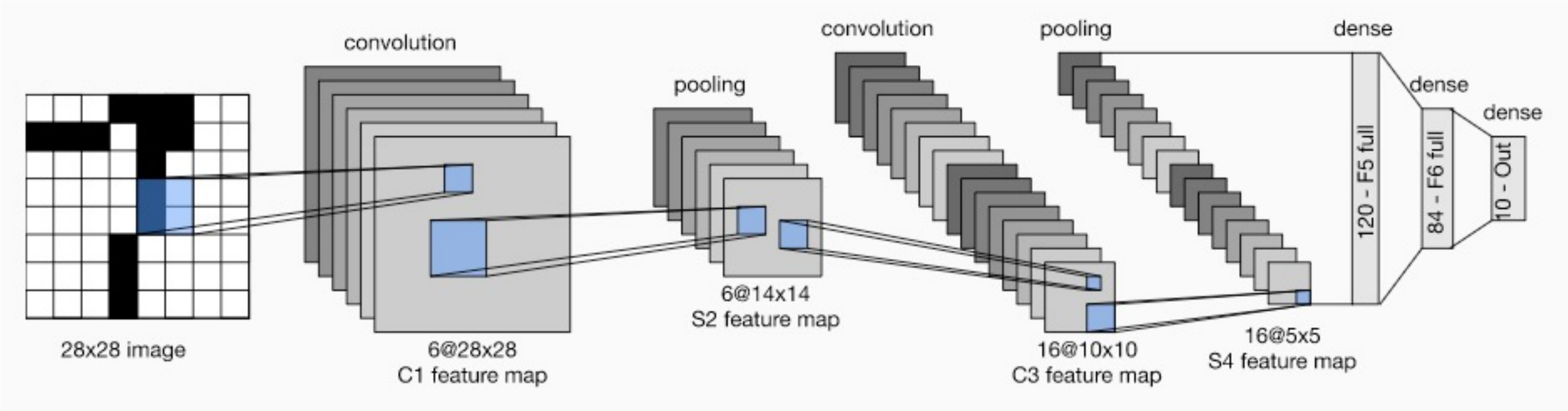
-3,9	7,8	-3,9	3,9	0	0
7,8	-13,7	5,8	-5,9	-2	0
-3,9	5,8	-2	-2	5,8	-3,9
3,9	-5,9	-2	5,8	-13,7	7,8
0	-2	5,8	-13,7	**17,5**	-11,7
0	0	-3,9	7,8	-11,7	7,8

17,5	1,9	-2
1,9	-2	9,7
-2	9,7	7,7

7,8	5,8	0
5,8	5,8	7,8
0	7,8	**17,5**

- **Max-pooling** layers are usually placed **between stacked convolutional layers**.

Stacking convolution, pooling, dense layers



LeNet architecture as illustrated in *Dive into Deep Learning* by Zhang et al. (https://d2l.ai/chapter_convolutional-neural-networks/lenet.html).

- Max-pooling gradually **reduces the resolution at higher layers**, **allowing us to use more channels** (for the same total number of trainable parameters).
- It also helps **increase more quickly the receptive field**.

Input Channel 1

0	0	0	0	0	0	0	0
0	-1	1	-1	-1	-1	-1	0
0	1	1	1	-1	-1	-1	0
0	-1	1	-1	-1	-1	-1	0
0	-1	-1	-1	0,9	-1	0,9	0
0	-1	-1	-1	-1	0,9	-1	0
0	-1	-1	-1	0,9	-1	0,9	0
0	0	0	0	0	0	0	0

Input Channel 2

0	0	0	0	0	0	0	0
0	-1	0,9	-1	-1	-1	-1	0
0	0,9	0,9	0,9	-1	-1	-1	0
0	-1	0,9	-1	-1	-1	-1	0
0	-1	-1	-1	1	-1	1	0
0	-1	-1	-1	-1	1	-1	0
0	-1	-1	-1	1	-1	1	0
0	0	0	0	0	0	0	0

Two Two-channel Kernels

-1	1	-1	&	-1	1	-1
1	1	1		1	1	1
-1	1	-1		-1	1	-1

1	-1	1	&	1	-1	1
-1	1	-1		-1	1	-1
1	-1	1		1	-1	1

Feature Map of Kernel 1 ("+")

-0,1	-4	-0,1	-7,9	-4	-4
-4	17,5	-2	1,9	-2	-4
-0,1	-2	-2	-2	-9,8	-0,1
-7,9	1,9	-2	-2	9,7	-4
-4	-2	-9,8	9,7	-13,7	7,7
-4	-4	-0,1	-4	7,7	-4

Feature Map of Kernel 2 ("X")

-3,9	7,8	-3,9	3,9	0	0
7,8	-13,7	5,8	-5,9	-2	0
-3,9	5,8	-2	-2	5,8	-3,9
3,9	-5,9	-2	5,8	-13,7	7,8
0	-2	5,8	-13,7	17,5	-11,7
0	0	-3,9	7,8	-11,7	7,8

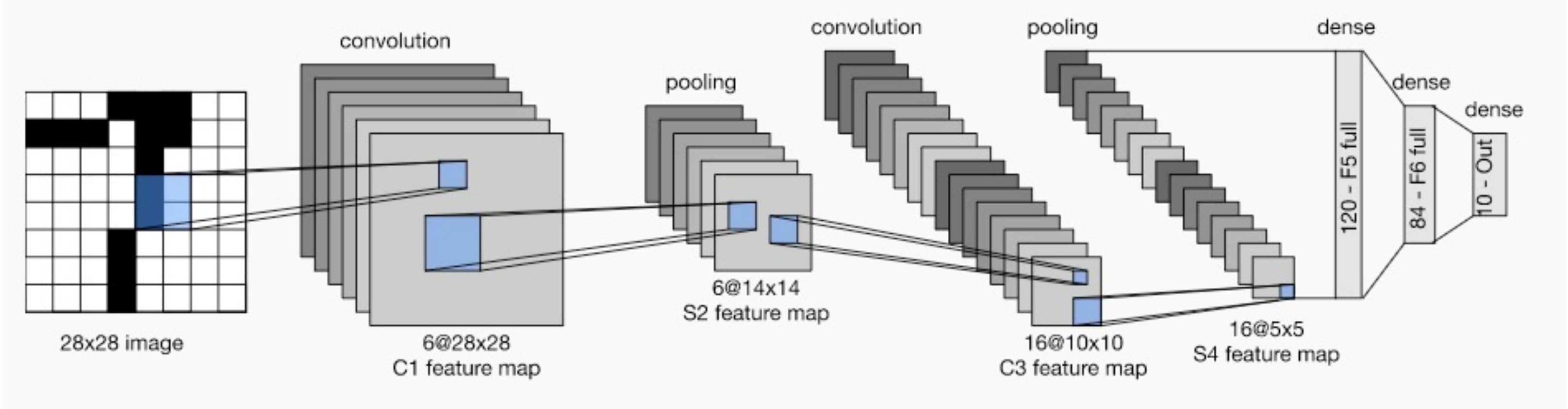
Max-Pooling (2,2) with Stride (2,2)

17,5	1,9	-2
1,9	-2	9,7
-2	9,7	7,7

7,8	5,8	0
5,8	5,8	7,8
0	7,8	17,5

- **Each feature of the max-pooled feature maps** is derived from (is "looking at") **4 features of the pre-pooled** feature maps, and **16 features of the input**.
- **By stacking** convolution and pooling layers, we can get **features that are increasingly aware of larger parts of the input** (larger "**receptive field**").

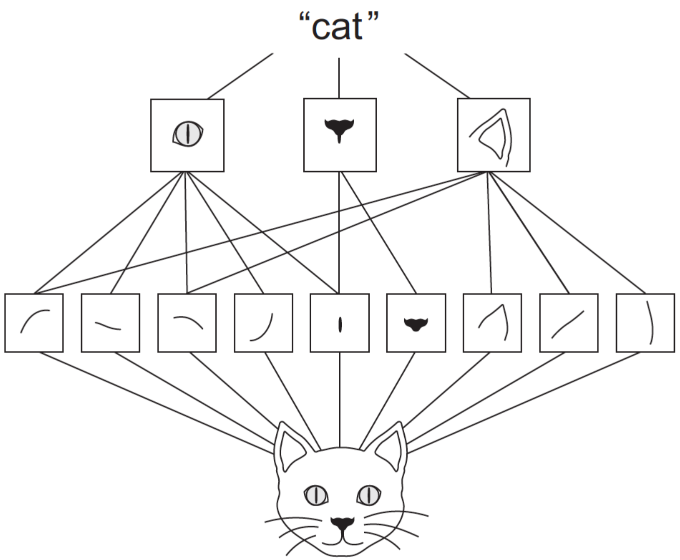
Stacking convolution, pooling, dense layers



LeNet architecture as illustrated in *Dive into Deep Learning* by Zhang et al. (https://d2l.ai/chapter_convolutional-neural-networks/lenet.html).

- **The features of the top feature maps** are concatenated to a **single vector** and **passed to a dense** (fully connected) **layer** or an **MLP** (with hidden layers).
  - To **recognize the digit** (0-9) in an image, the dense layer (or output layer of the MLP) would have **10 neurons with softmax**, and we would use **cross-entropy** loss.
  - To output the **coordinates of the eyes** in images (or video frames) of faces, the **dense layer** (or output layer of the MLP) could have **4 neurons** (x1, y1, x2, y2) with no activation function, and we could use the **mean squared error** as loss. (But better, more advanced models can be used...)
  - The **training examples** would be digit or face **images** (or video frames) **annotated with the correct responses** (digits or coordinates of the eyes).
- In practice we would also include **dropout** layers and **residuals**.

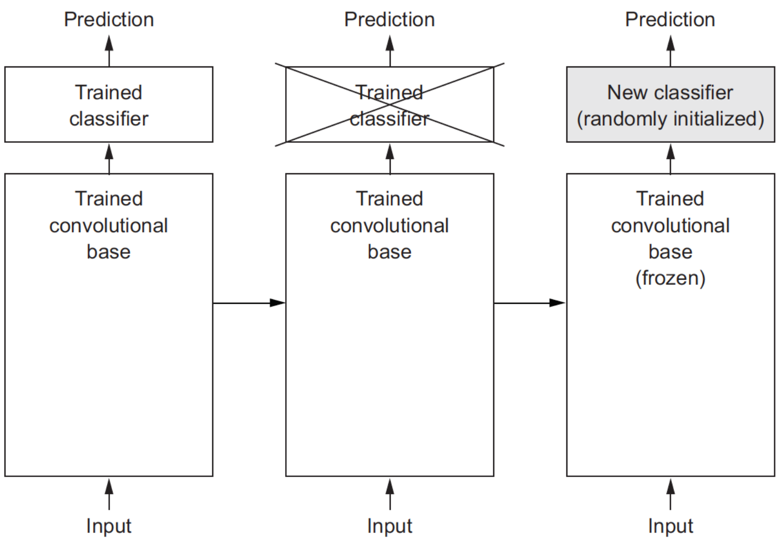
What do the layers learn?



- The kernels of **lower layers** tend to detect **low-level features (e.g., edges of different directions)**. The kernels of **higher layers** tend to detect **higher-level features (e.g., eyes, ears)**.
- **Pre-trained kernels of lower levels** can be useful in many different tasks.

Figure from the recommended book **"Deep Learning with Python"** by F. Chollet, Manning Publications, 1st edition. Also covers Keras. Optionally consult Chapter 5 (Deep Learning for Computer Vision) for ways to visualize what CNN layers learn.
https://www.manning.com/books/deep-learning-with-python
https://www.manning.com/books/deep-learning-with-python-second-edition

Re-using pretrained layers



- In practice, we start with a **CNN pre-trained on a very large dataset**.
  - Often **ImageNet**, 1.4 million images, 1,000 classes (e.g., dogs, cats).
- We **replace the top layers** with a **task-specific classification/regression layer**.
  - We **train the task-specific layer on task-specific data**, keeping the **pre-trained convolutional layers frozen** (no weight updates in the frozen layers).
  - We may then **gradually unfreeze some of the convolutional layers too** (weight updates in both the task-specific layers and the unfrozen convolutional layers).

Figure from the recommended book **"Deep Learning with Python"** by F. Chollet, Manning Publications, 1st edition. Also covers Keras. https://www.manning.com/books/deep-learning-with-python https://www.manning.com/books/deep-learning-with-python-second-edition

Re-using pretrained layers

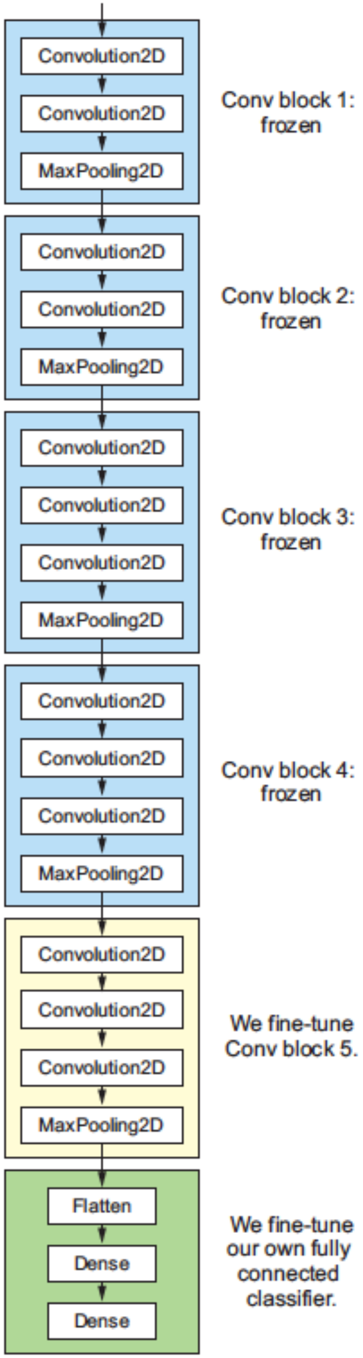


Figure 5.19 Fine-tuning the last convolutional block of the VGG16 network

Figure from the recommended book **"Deep Learning with Python"** by F. Chollet, Manning Publications, 1st edition. Also covers Keras. https://www.manning.com/books/deep-learning-with-python
https://www.manning.com/books/deep-learning-with-python-second-edition

Data augmentation

Figure 5.11 Generation of cat pictures via random data augmentation

- We can **increase the number of task-specific training examples** by adding artificial training examples.
  - For example, we can **rotate, squeeze, flip** etc. the task-specific **training images**.
  - **Big improvements** usually.

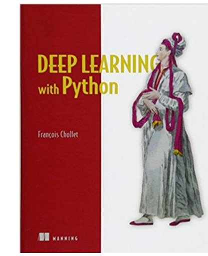
Figure from the recommended book **“Deep Learning with Python”** by F. Chollet, Manning Publications, 1st edition. Also covers data augmentation in Keras.
https://www.manning.com/books/deep-learning-with-python
https://www.manning.com/books/deep-learning-with-python-second-edition

NLP with CNNs and Transformers

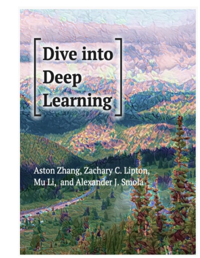
- **CNNs** can also be **applied to texts**.
  - Viewed as **1D images**. Each **"pixel" is a word**. The **channels of the input** 1D image are the **dimensions of the word embeddings**.
  - **Faster than RNNs**, but usually **worse results**.
- **Pre-trained layers** recently led to big improvements NLP.
  - Mostly using **Transformers**, a type of neural nets not covered here. Used in **BERT**, **ChatGPT**, …
- **More information** on CNNs for text and Transformers in the **Human-Computer Interaction course** and the MSc courses "Natural Language Processing" and "Text Analytics" (slides/videos available on AUEB's e-class/MS Stream).
- **Transformers** are starting to be used in **Computer Vision** too.

Recommended reading

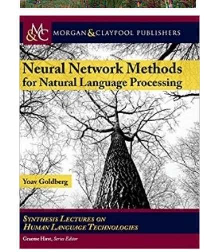
- F. Chollet, *Deep Learning in Python*, Manning Publications, 1st edition, 2017, Chapter 5.
    - The 1st edition is freely available, suffices for this course: https://www.manning.com/books/deep-learning-with-python
    - 2nd edition also available, requires payment, recommended: https://www.manning.com/books/deep-learning-with-python-second-edition



- A. Zhang et al., *Dive into Deep Learning*, Chapter 6.
    - Freely available at: https://d2l.ai/



- Y. Goldberg, *Neural Network Models for Natural Language Processing*, Morgan & Claypool Publishers, 2017.
    - Chapter 13 discusses applying CNNs to text.



- See also the recommended reading/resources of lecture 20.

Βιβλιογραφία

- Russel & Norvig (4η έκδοση): ενότητες 21.3, 25.4, μόνο όσα αναφέρουν οι διαφάνειες.
  - Όσοι ενδιαφέρονται μπορούν να μελετήσουν προαιρετικά και τις υπόλοιπες ενότητες αυτών των κεφαλαίων.
- Βλαχάβας κ.ά: ενότητα 19.9.1.
  - Όσοι ενδιαφέρονται μπορούν να μελετήσουν προαιρετικά ολόκληρο το κεφάλαιο 19.