

Natural Language Processing with Convolutional Neural Networks

2023–24

Ion Androutsopoulos

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

Contents

- Quick background: Convolutional Neural Networks (CNNs) in Computer Vision.
- Image to text generation with CNN encoders and RNN decoders.
- Text processing with CNNs.

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/](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

-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

-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

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

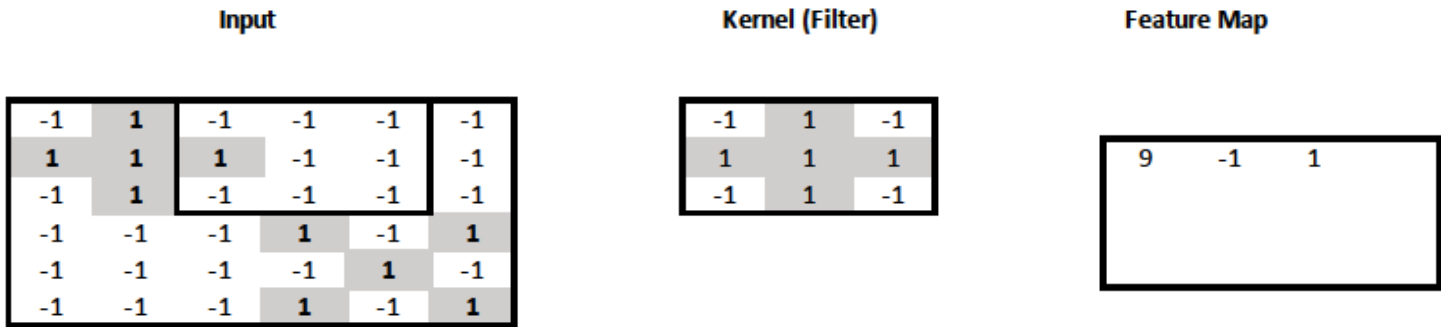
9

-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

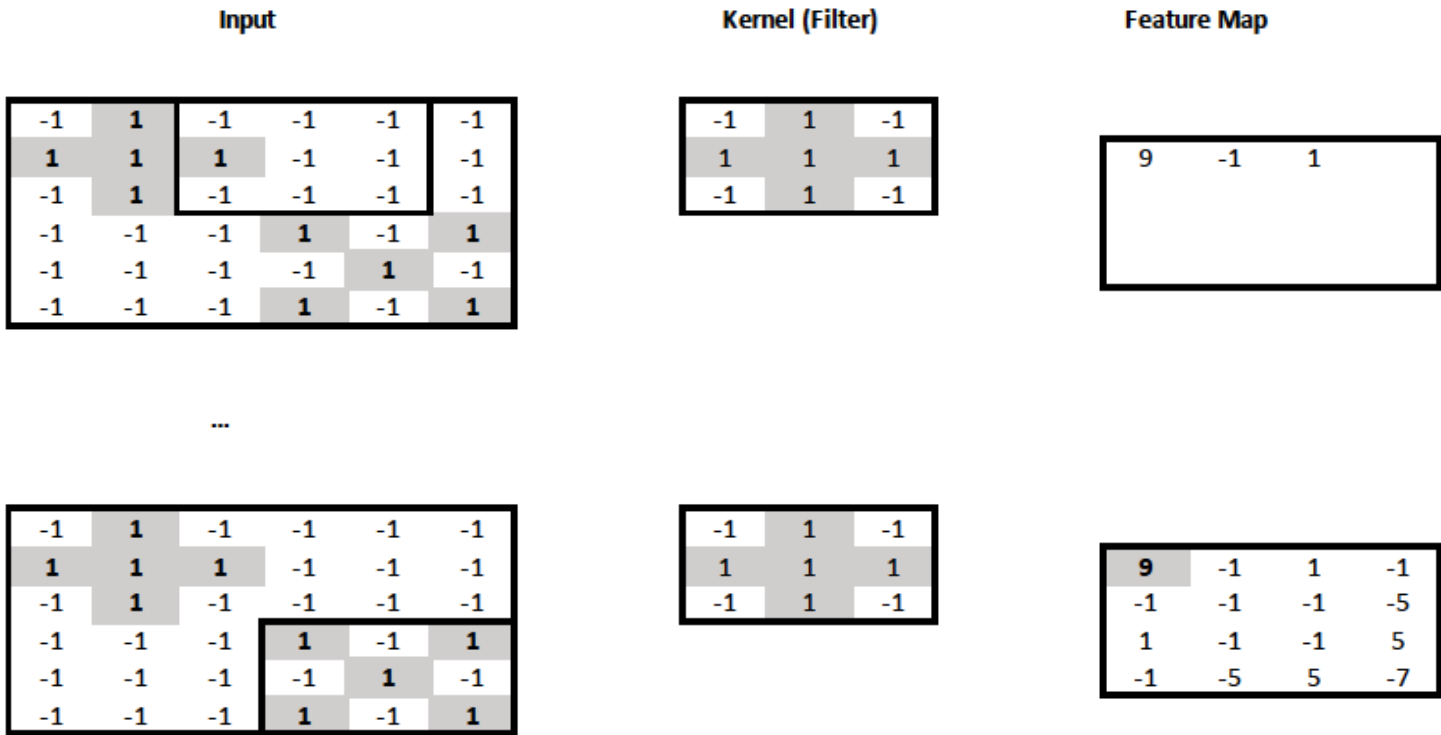
9	-1
---	----

Convolutions on images



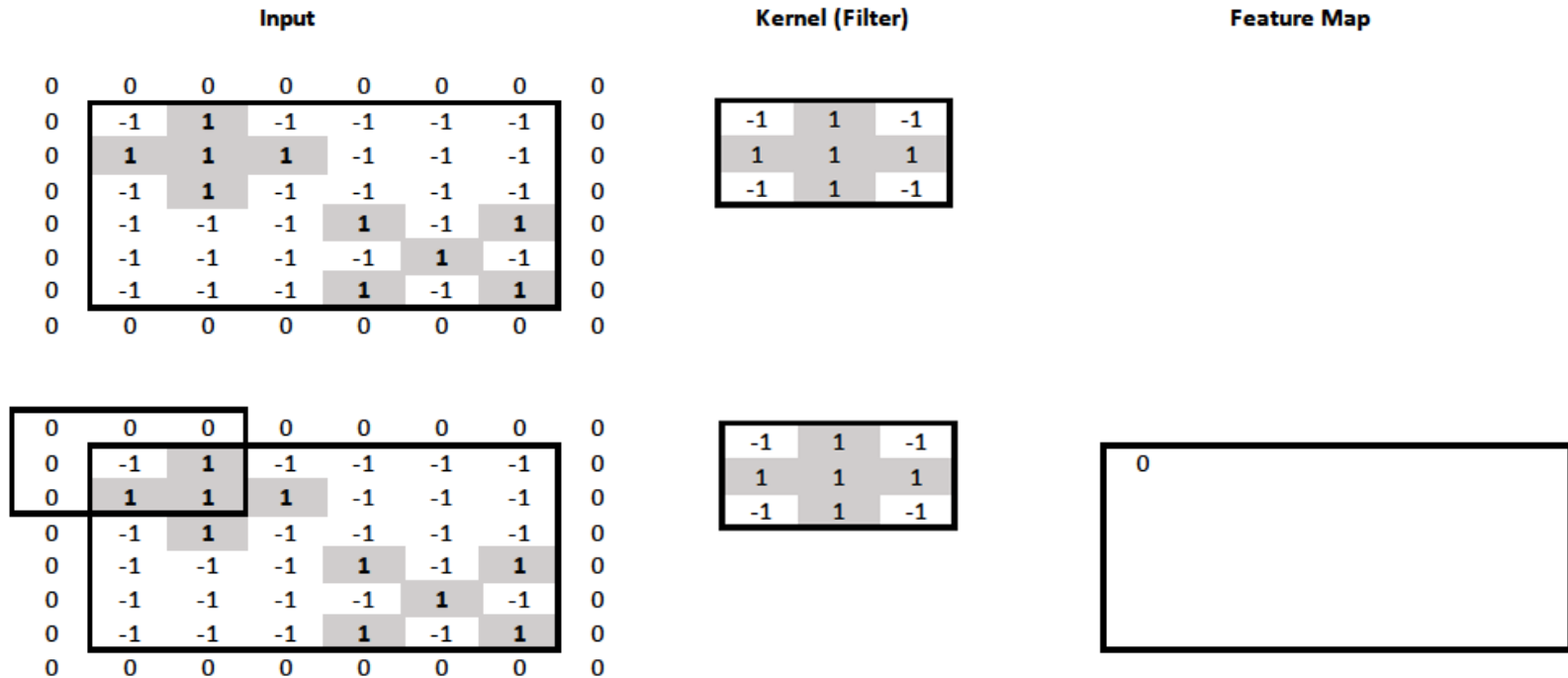
- Let X be the part of the input where we apply the kernel (filter).
- Let W be the kernel.
- The resulting **feature** of the feature map is: $\sum_{i=1}^3 \sum_{j=1}^3 W_{i,j} X_{i,j}$
- In practice, we would also use an **activation function** and **bias term**: $f(\sum_{i=1}^3 \sum_{j=1}^3 W_{i,j}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

Convolutions on images



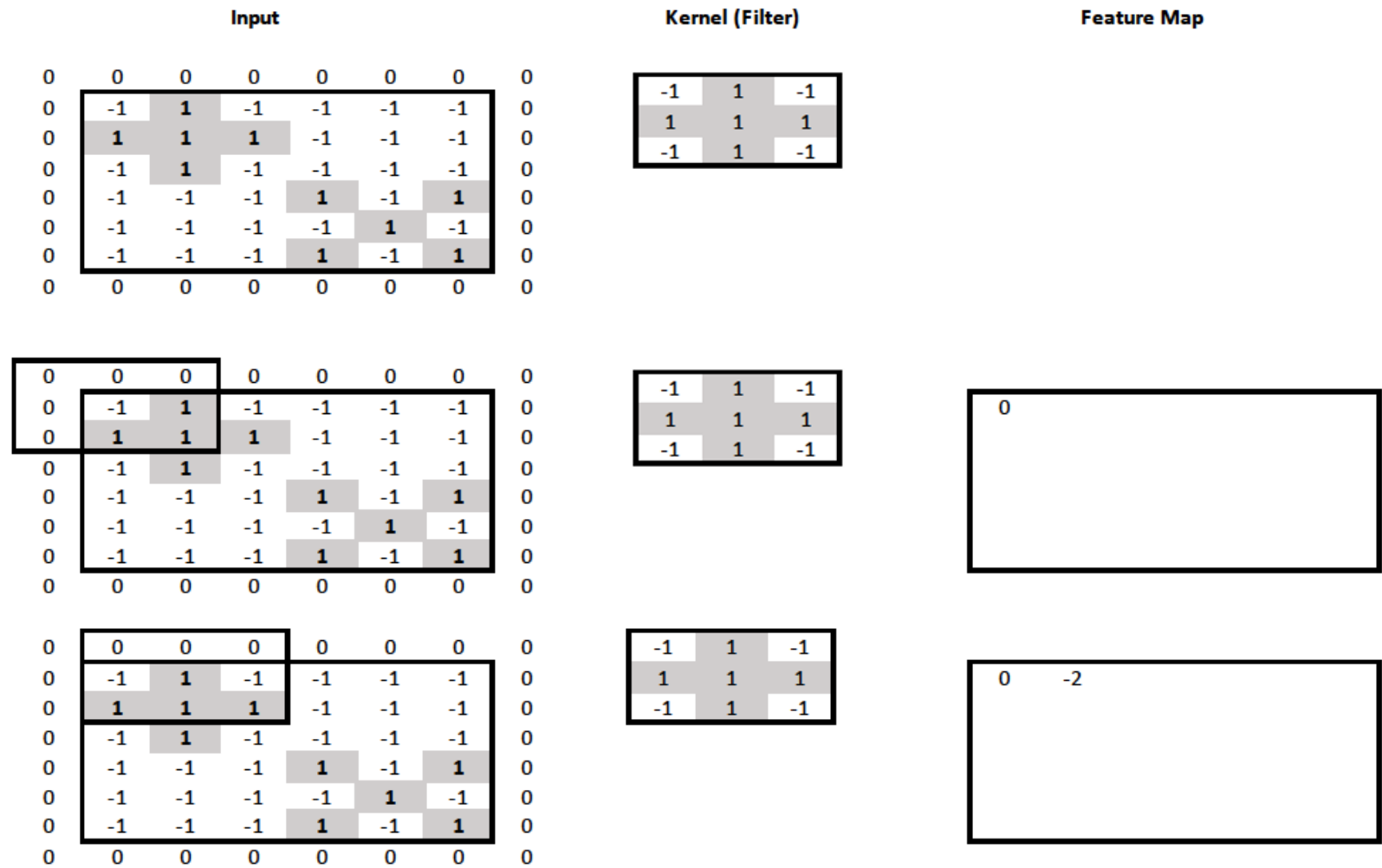
- We can think of the resulting **feature map** as a new **“image”** that indicates the **position(s) of the cross(es)** in the original image.
 - No need to have the crosses at particular parts of the image.
- The new **“image”** is **4x4** instead of **6x6**, because the **kernel could not slide outside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

Wide convolutions on images



- 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



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

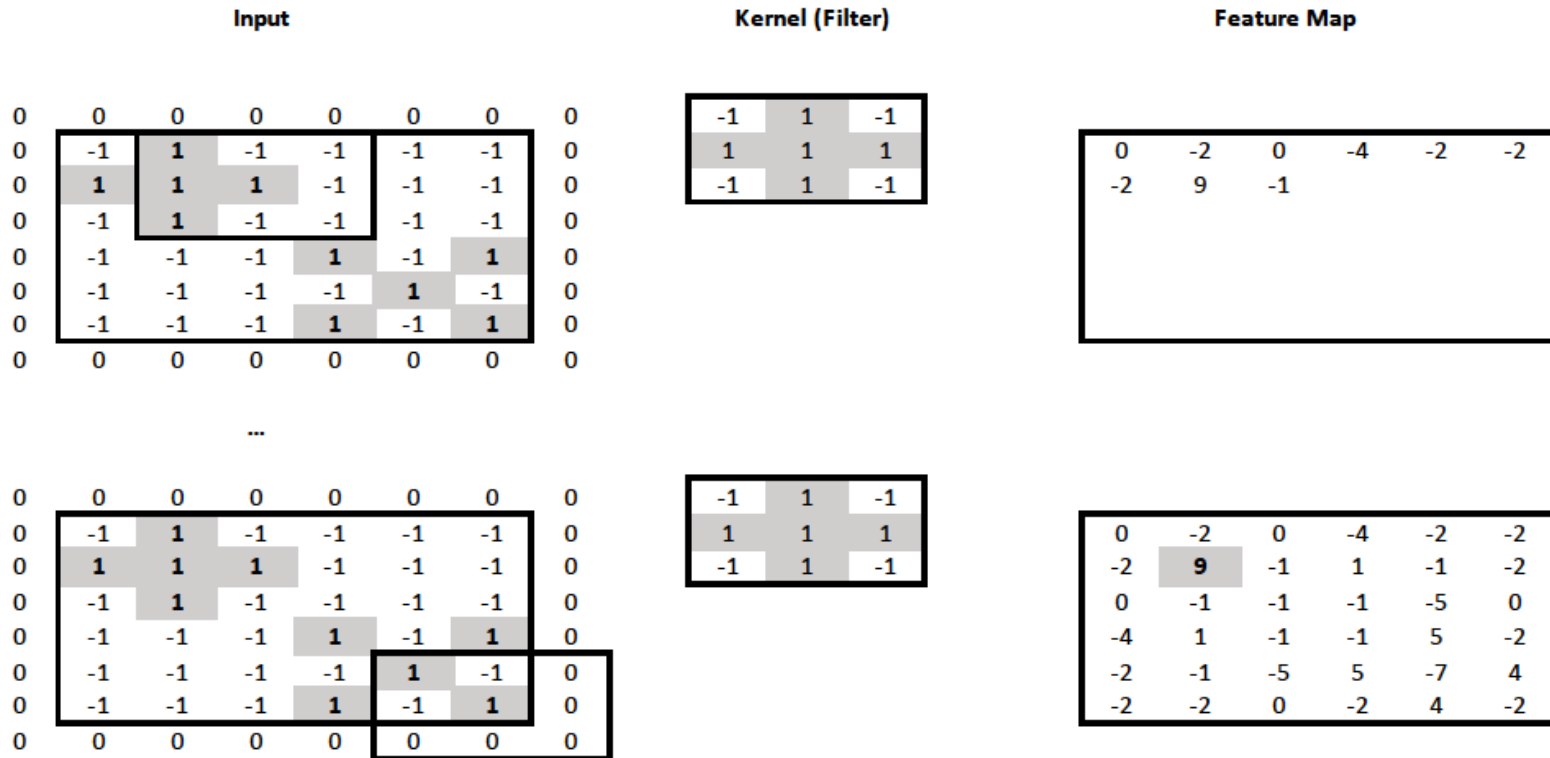
0	-2	0	-4	-2	-2
-2					

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



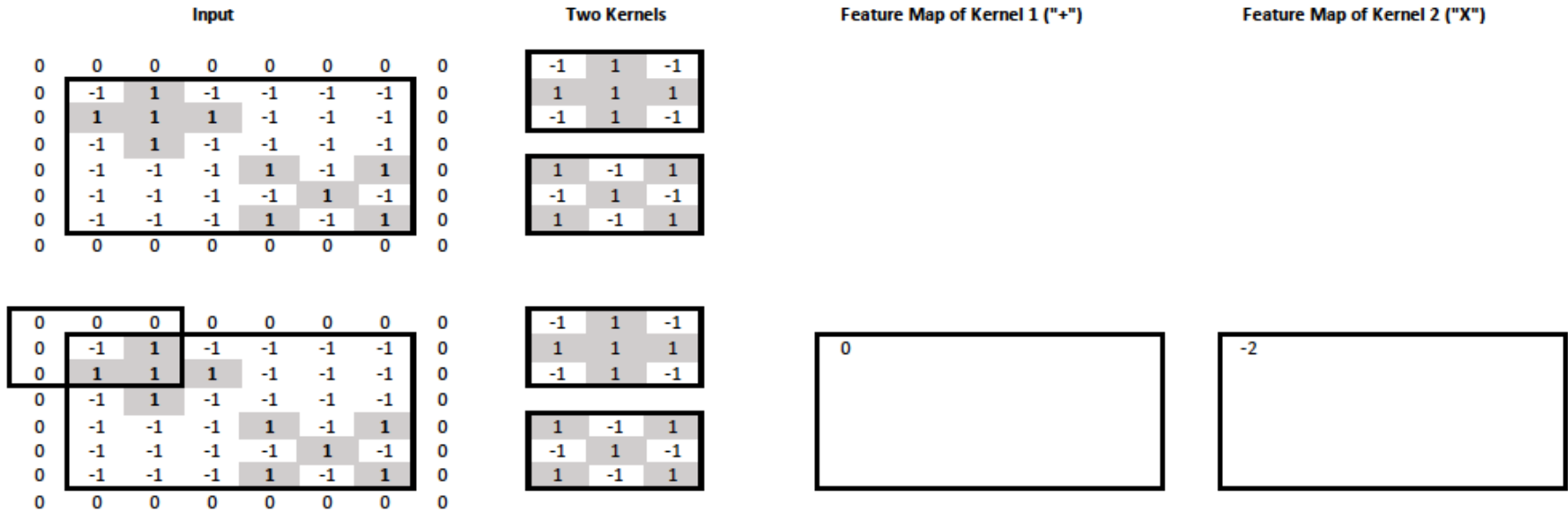
- X : entire input image. F : feature map.
- W : kernel, but with rows and columns numbered $-1, 0, 1$.
- Feature map values: $F_{i,j} = \sum_{k=-1}^1 \sum_{l=-1}^1 W_{k,l} X_{i+k,j+l}$
- In practice: $F_{i,j} = f(\sum_{k=-1}^1 \sum_{l=-1}^1 W_{k,l} X_{i+k,j+l} + b)$

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



- 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 ("+")

0

Feature Map of Kernel 2 ("X")

-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

0	-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

0	-2	0
---	----	---

-2	4	-2
----	---	----

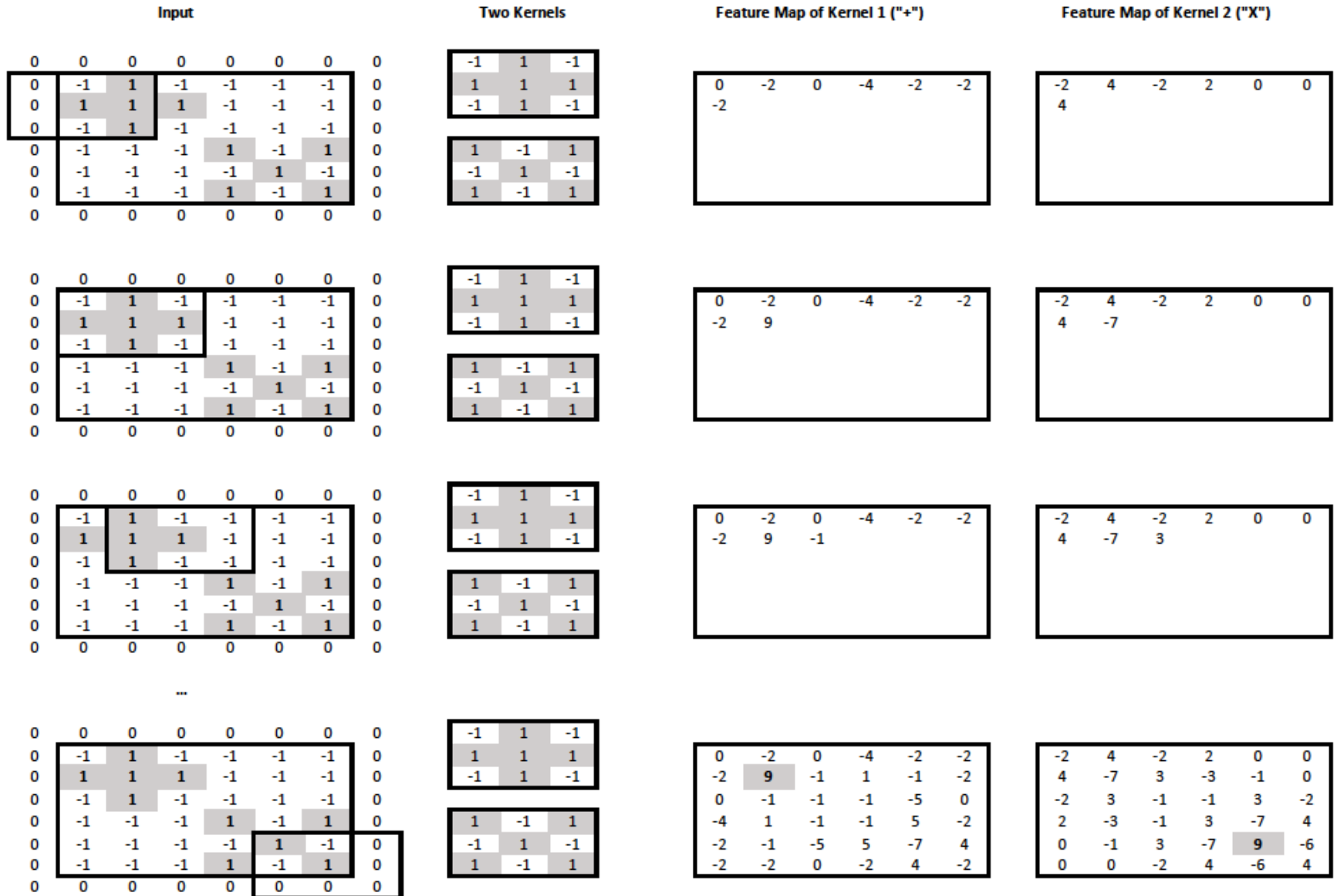
0	0	0	0	0	0	0	0
0	-1	1	-1	-1	-1	-1	0
0	1	1	1	-1	-1	-1	0
0	-1	1	-1	-1	-1	-1	0
0	-1	-1	-1	1	-1	1	0
0	-1	-1	-1	-1	1	-1	0
0	-1	-1	-1	1	-1	1	0
0	0	0	0	0	0	0	0

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

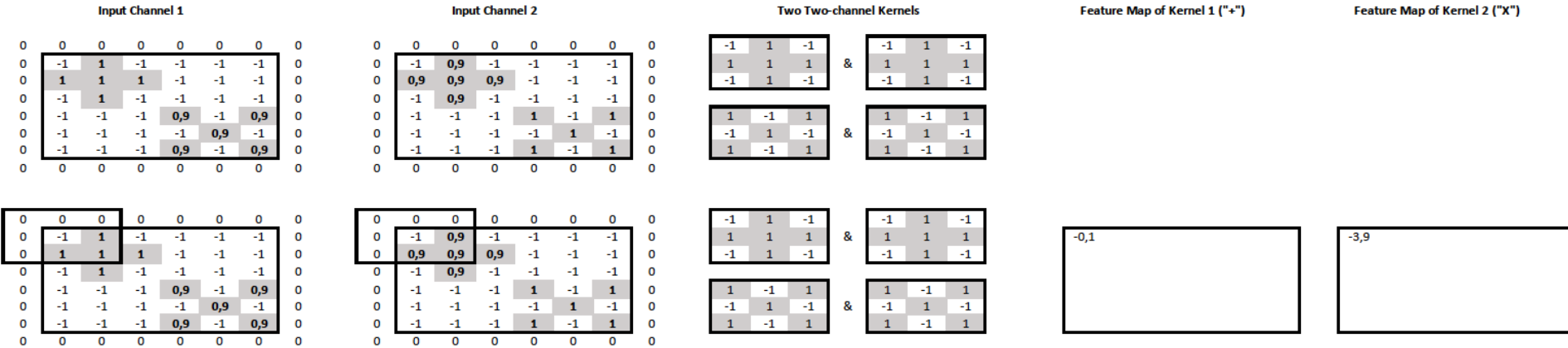
1	-1	1
-1	1	-1
1	-1	1

Two kernels

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



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

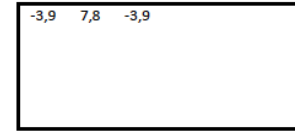
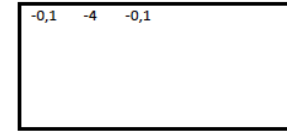
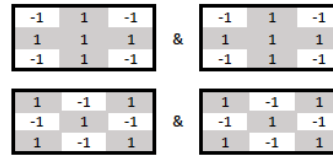
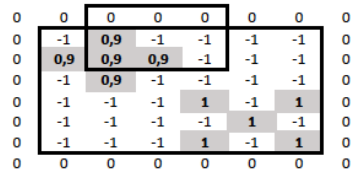
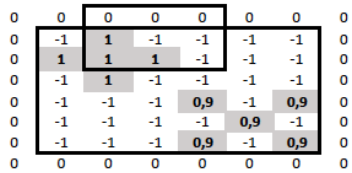
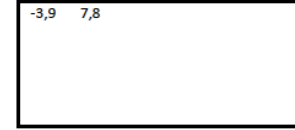
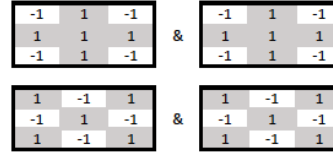
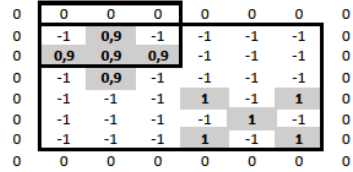
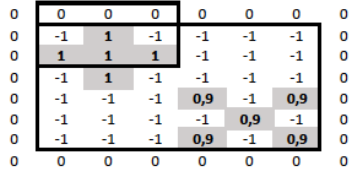
Input Channel 1

Input Channel 2

Two Two-channel Kernels

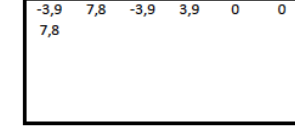
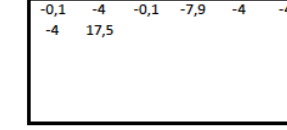
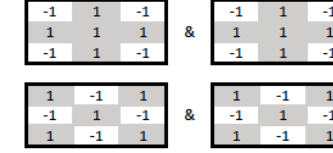
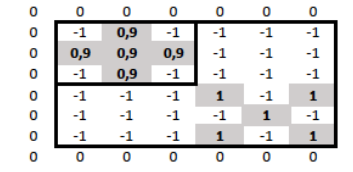
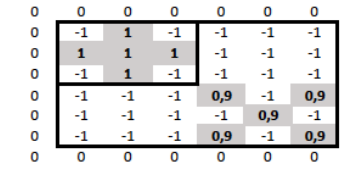
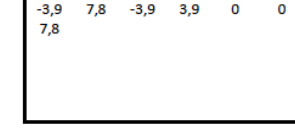
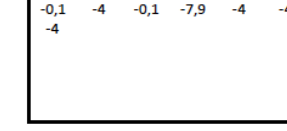
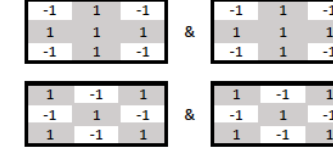
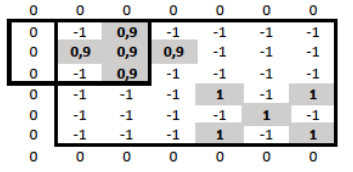
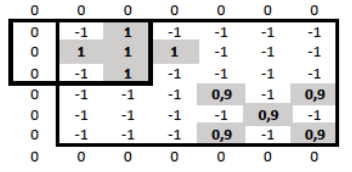
Feature Map of Kernel 1 ("+")

Feature Map of Kernel 2 ("X")

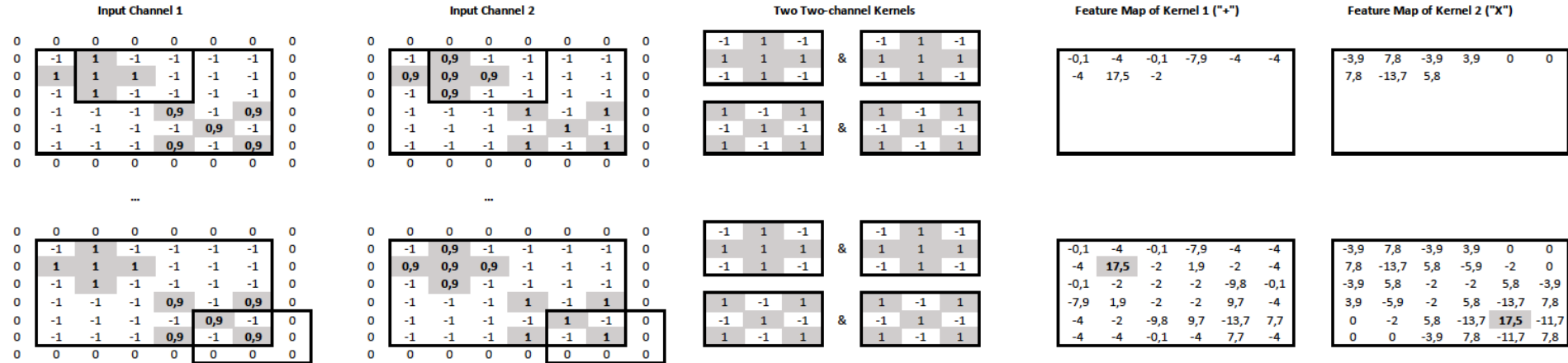


...

...



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 ("+")

-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

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

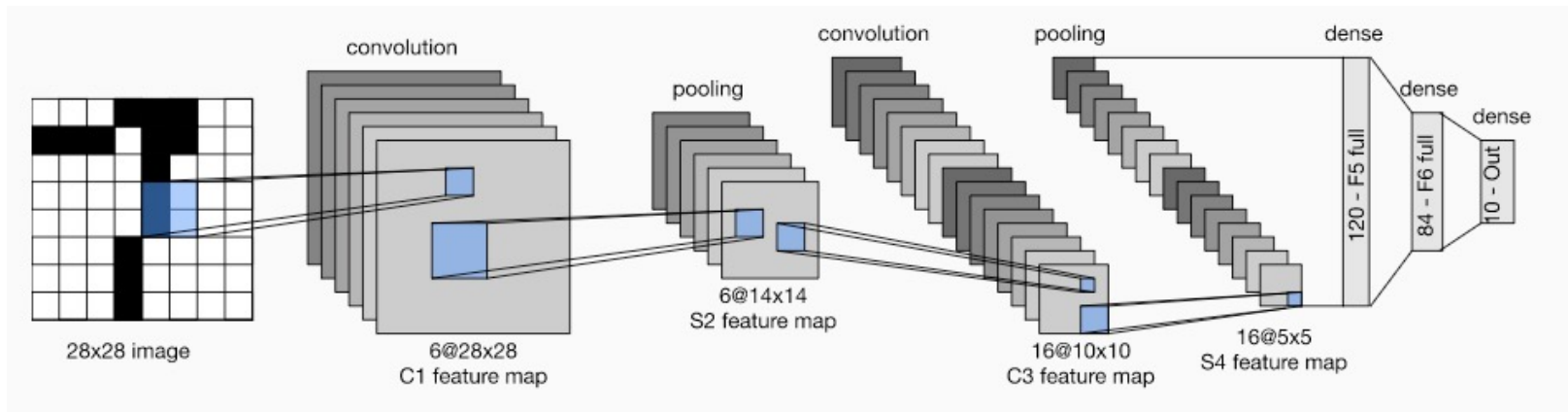
17,5	1,9	-2
1,9	-2	9,7
-2	9,7	7,7

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

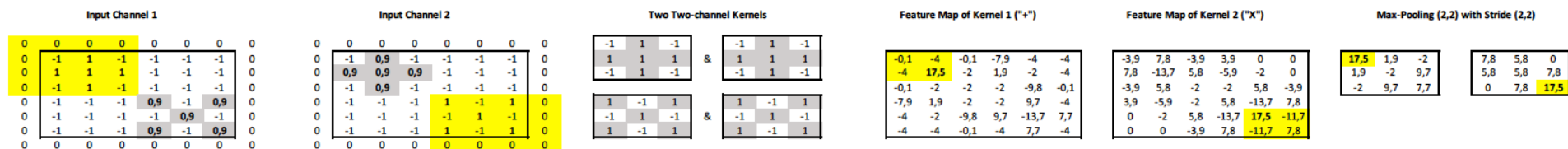
- **Max-pooling** layers are usually placed **between stacked convolutional layers**.

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).

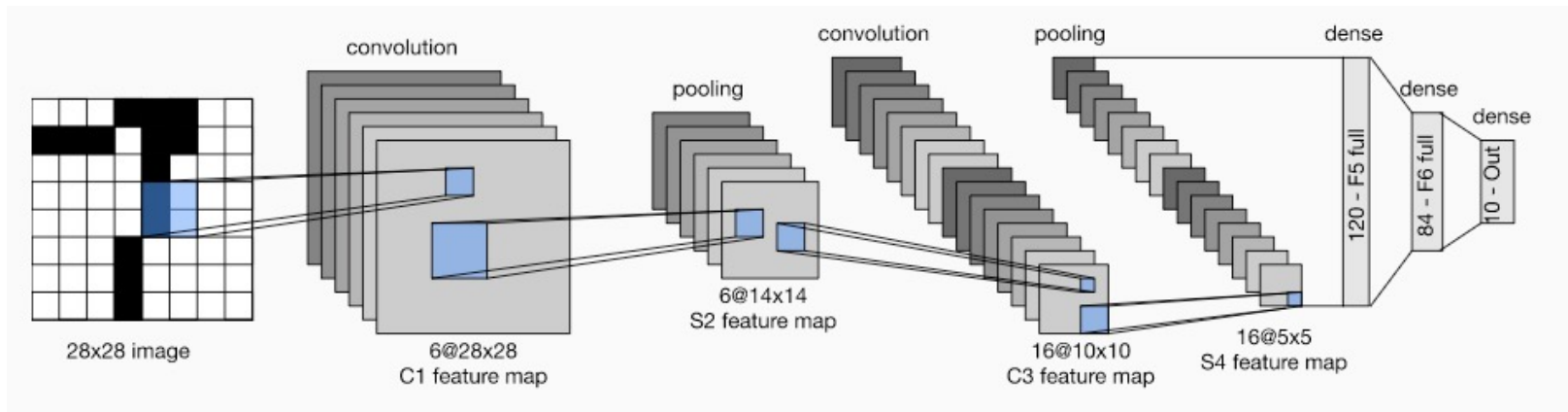


- Max-pooling gradually **reduces the resolution at higher layers, allowing us to use more channels** (for the same total number of trainable parameters/layer).
- It also helps **increase more quickly the receptive field**.



- 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”).

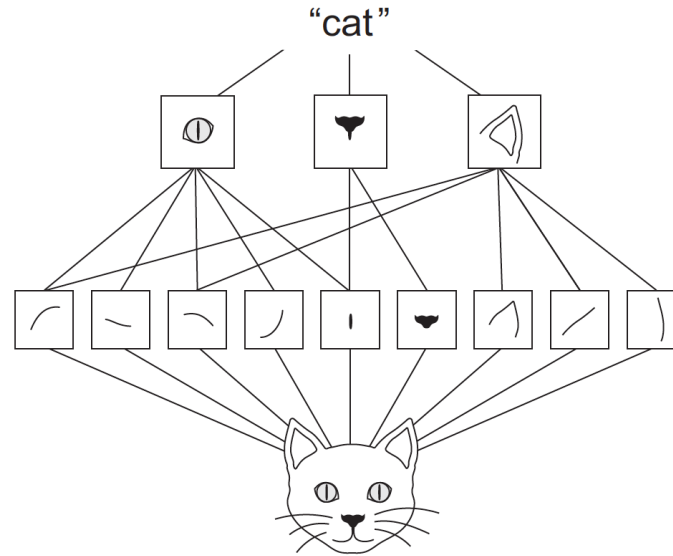
Stacking convolution, pooling, dense layers



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?



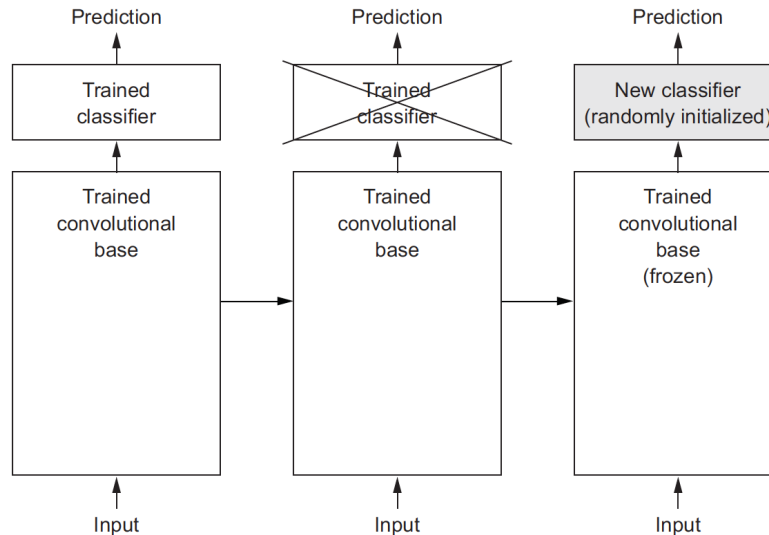
- The kernels of **lower layers** tend to detect **low-level features (e.g., edges of different directions)**. The kernels of **higher layers** tend to detect **higher-level features (e.g., eyes, ears)**.
- **Pre-trained kernels of lower levels** can be useful in many different tasks.

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

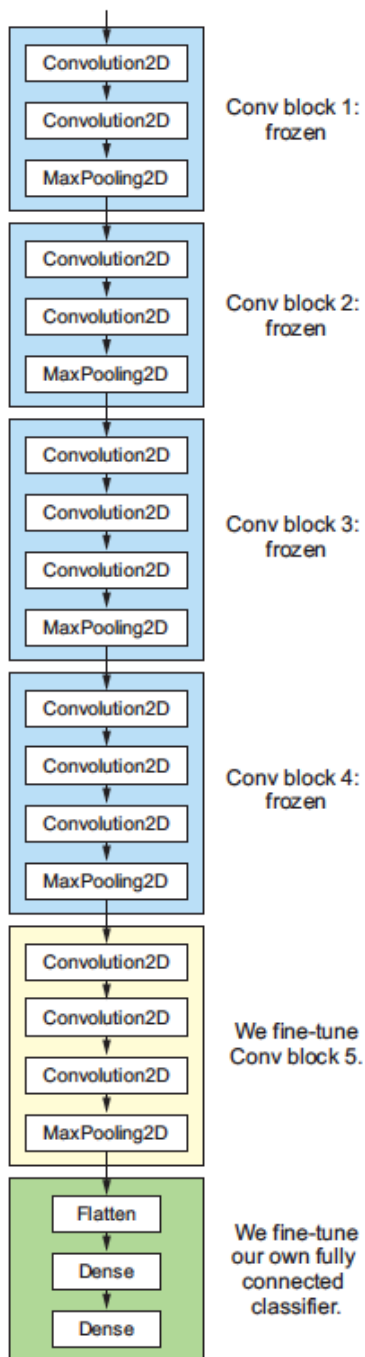


Figure from the recommended book “**Deep Learning with Python**” by F. Chollet, Manning Publications, 1st edition. Also covers Keras. <https://www.manning.com/books/deep-learning-with-python>
<https://www.manning.com/books/deep-learning-with-python-second-edition>

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

Data augmentation

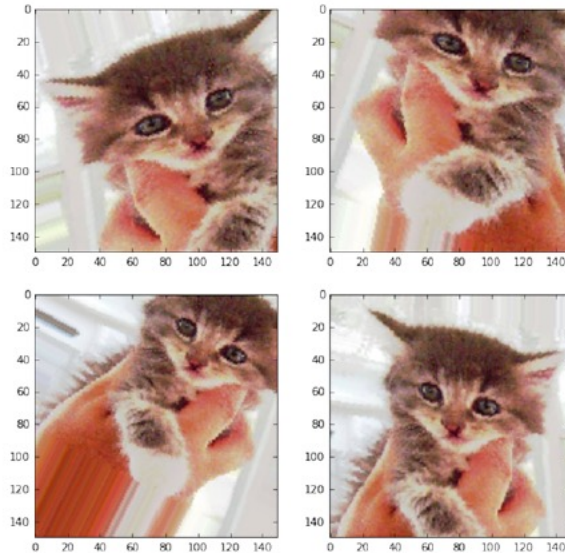


Figure 5.11 Generation of cat pictures via random data augmentation

- 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.
- How do we do **data augmentation for NLP**?

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>

Image captioning

Optional study

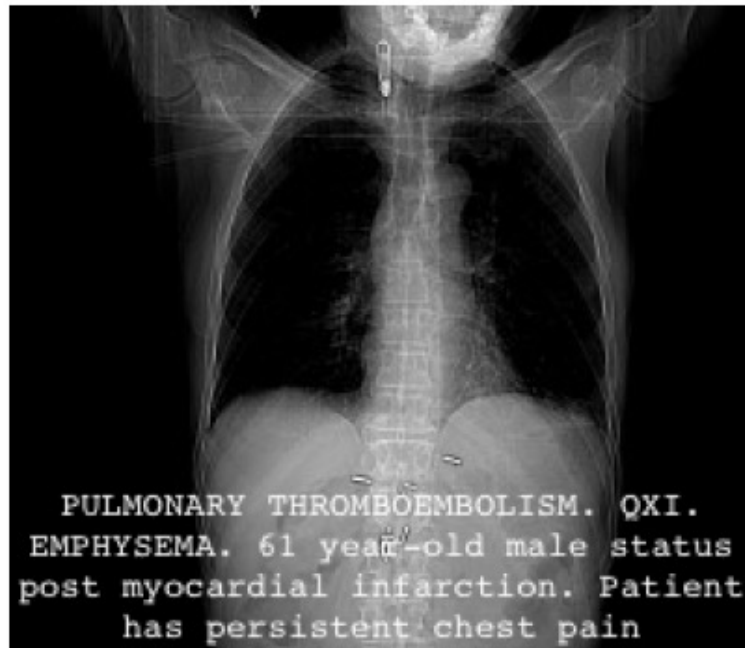


A blue and yellow train traveling down train tracks.

(a) General

Possible applications:

- Image retrieval via captions.
- Eyesight problems.
- Drafting medical reports.



PULMONARY THROMBOEMBOLISM. OXI.
EMPHYSEMA. 61 year-old male status
post myocardial infarction. Patient
has persistent chest pain

(b) Biomedical

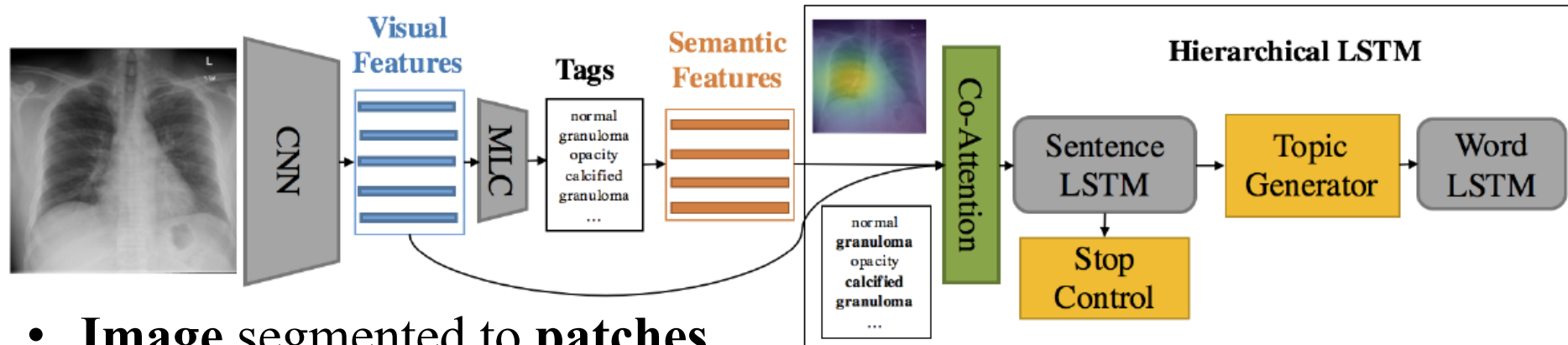
Figure 1: Example of a caption produced by the model of Vinyals et al. (2017) for a non-biomedical image (1a) and an example of a PEIR Radiology image with its associated caption (1b).

From I. Pavlopoulos, V. Kougia, I. Androutsopoulos, “A Survey on Biomedical Image Captioning”.

<https://www.aclweb.org/anthology/W19-1803/>

Biomedical image to text generation

Optional
study



- **Image** segmented to **patches**.
- **CNN** converts each **patch** to a **vector**, producing “**visual features**”.
- **MLP** (“**MLC**”) **predicts tags** (classes) given the visual features.
- The **word embeddings** of the **tags** are “**semantic features**”.
- **Sentence-level LSTM** produces **sentence embeddings** (“**topics**”).
 - A **stop control** (classifier) decides when to **stop producing sentences**.
 - At each time-step, **attention** over **visual** and **semantic features**.
- For each sentence embedding, **word-level LSTM** produces **words**.

Convolutions on text

Let's **pretend** that we know what the **dimensions** of the word **embeddings represent**, and that the dimensions are **binary**.

2

Words	Embeddings			
	Subject	Positive	Stress	Quantity
I	1	0	0	0
like	0	1	0	0
this	0	0	0	0
movie	0	0	0	0
very	0	0	1	0
much	0	0	0	1
!	0	0	1	0

Filter for “I like”, “we admire”...			
1	0	0	0
0	1	0	0

Filter for “very much”, “so much”...			
0	0	1	0
0	0	0	1

Convolutions on text

Let's **pretend** that we know what the **dimensions** of the word **embeddings represent**, and that the dimensions are **binary**.

Words	Embeddings			
	Subject	Positive	Stress	Quantity
I	1	0	0	0
like	0	1	0	0
this	0	0	0	0
movie	0	0	0	0
very	0	0	1	0
much	0	0	0	1
!	0	0	1	0

0

Filter for “I like”, “we admire”...

1	0	0	0
0	1	0	0

Filter for “very much”, “so much”...

0	0	1	0
0	0	0	1

Convolutions on text

Words	Embeddings			
	Subject	Positive	Stress	Quantity
I	1	0	0	0
like	0	1	0	0
this	0	0	0	0
movie	0	0	0	0
very	0	0	1	0
much	0	0	0	1
!	0	0	1	0

2
0
0
0
0
0
0

Filter for “I like”, “we admire”...			
1	0	0	0
0	1	0	0

Filter for “very much”, “so much”...			
0	0	1	0
0	0	0	1

Convolutions on text

Words	Embeddings				
	Subject	Positive	Stress	Quantity	
I	1	0	0	0	2 0 0 0 0 0 0 0 0 0
like	0	1	0	0	
this	0	0	0	0	
movie	0	0	0	0	
very	0	0	1	0	
much	0	0	0	1	
!	0	0	1	0	

Filter for “I like”, “we admire”...			
1	0	0	0
0	1	0	0

Filter for “very much”, “so much”...			
0	0	1	0
0	0	0	1

Convolutions on text

Words	Embeddings			
	Subject	Positive	Stress	Quantity
I	1	0	0	0
like	0	1	0	0
this	0	0	0	0
movie	0	0	0	0
very	0	0	1	0
much	0	0	0	1
!	0	0	1	0

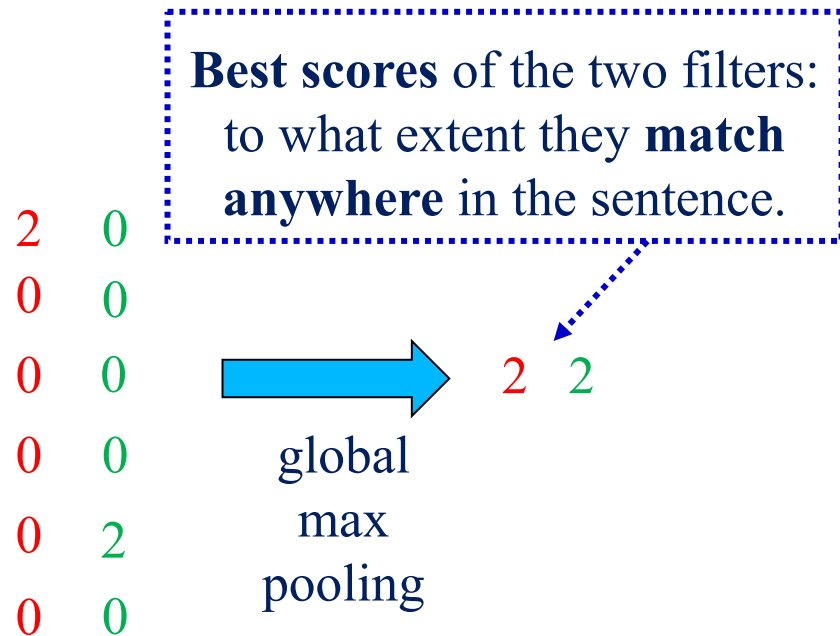
2	0
0	0
0	0
0	0
0	0
0	2
0	

Filter for “I like”, “we admire”...			
1	0	0	0
0	1	0	0

Filter for “very much”, “so much”...			
0	0	1	0
0	0	0	1

Convolutions on text

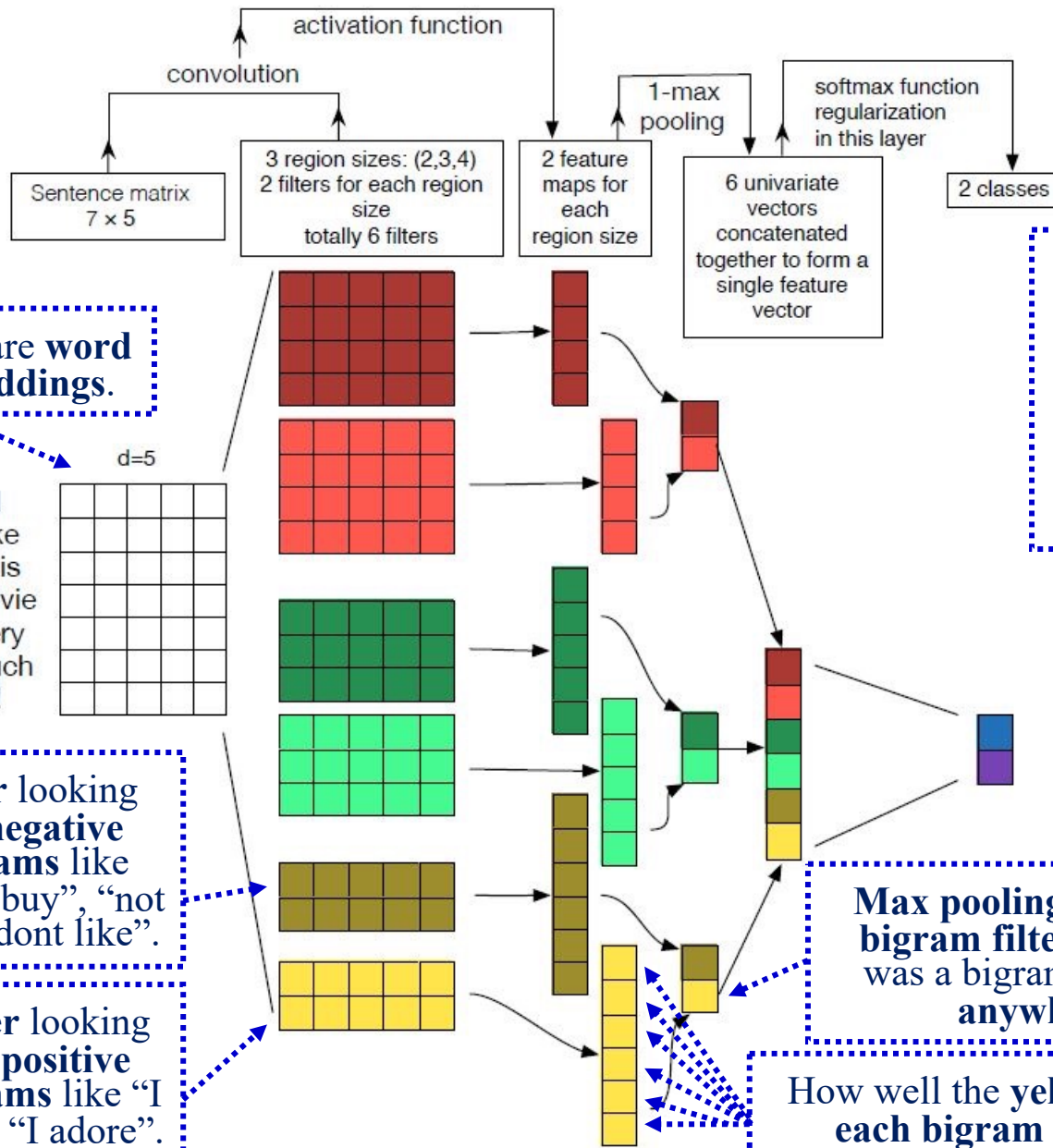
Words	Embeddings			
	Subject	Positive	Stress	Quantity
I	1	0	0	0
like	0	1	0	0
this	0	0	0	0
movie	0	0	0	0
very	0	0	1	0
much	0	0	0	1
!	0	0	1	0



Filter for “I like”, “we admire”...			
1	0	0	0
0	1	0	0

Filter for “very much”, “so much”...			
0	0	1	0
0	0	0	1

Convolutional Neural Networks



From “A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification”, Zhang et al., 2015.
<http://arxiv.org/abs/1510.03820>

The **numbers in each filter** are learned by **backpropagation**. The **embeddings** can also be learned during backpropagation.

How well the **yellow bigram filter** matched each bigram of the sentence (scores).

Convolutions on text – closer to reality

	Embeddings			
Words	d_1	d_2	d_3	d_4
I	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
like	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
this	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
movie	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
very	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$
much	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$
!	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	$x_{7,4}$

$$\text{ReLU}(wx + b)$$

$$x^T = \langle x_{1,1}, x_{1,2}, x_{1,3}, \dots, x_{2,3}, x_{2,4} \rangle$$

A bigram filter			
$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	$w_{1,4}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$

$$w = \langle w_{1,1}, w_{1,2}, w_{1,3}, \dots, w_{2,3}, w_{2,4} \rangle$$

$$b$$

Convolutions on text – closer to reality

	Embeddings			
Words	d_1	d_2	d_3	d_4
I	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
like	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
this	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
movie	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
very	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$
much	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$
!	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	$x_{7,4}$

$$\text{ReLU}(wx + b)$$

$$x^T = \langle x_{2,1}, x_{2,2}, x_{2,3}, \dots, x_{3,3}, x_{3,4} \rangle$$

A bigram filter			
$w_{1,1}$	$w_{1,2}$	$w_{1,3}$	$w_{1,4}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$	$w_{2,4}$

$$w = \langle w_{1,1}, w_{1,2}, w_{1,3}, \dots, w_{2,3}, w_{2,4} \rangle$$

b

Now applying **three** bigram filters

	Embeddings			
Words	d_1	d_2	d_3	d_4
I	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
like	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
this	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
movie	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
very	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$
much	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$
!	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	$x_{7,4}$

$$h_2 = \text{ReLU}(Wx + b) \in \mathbb{R}^{3 \times 1}$$

$$x^T = \langle x_{2,1}, x_{2,2}, \dots, x_{3,3}, x_{3,4} \rangle \in \mathbb{R}^{1 \times 8}$$

$$W = \begin{bmatrix} w_{1,1,1} & w_{1,1,2} & w_{1,1,3} & \dots & w_{1,2,3} & w_{1,2,4} \\ w_{2,1,1} & w_{2,1,2} & w_{2,1,3} & \dots & w_{2,2,3} & w_{2,2,4} \\ w_{3,1,1} & w_{3,1,2} & w_{3,1,3} & \dots & w_{3,2,3} & w_{3,2,4} \end{bmatrix} \in \mathbb{R}^{3 \times 8}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}$$

Applying 3 bigram filters

	Embeddings			
Words	d_1	d_2	d_3	d_4
I	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
like	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
this	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
movie	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
very	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$
much	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$
!	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	$x_{7,4}$

$$h_2 = \langle h_{2,1}, h_{2,2}, h_{2,3} \rangle^T \in \mathbb{R}^{3 \times 1}$$

$$W = \begin{bmatrix} w_{1,1,1} & w_{1,1,2} & w_{1,1,3} & \dots & w_{1,2,3} & w_{1,2,4} \\ w_{2,1,1} & w_{2,1,2} & w_{2,1,3} & \dots & w_{2,2,3} & w_{2,2,4} \\ w_{3,1,1} & w_{3,1,2} & w_{3,1,3} & \dots & w_{3,2,3} & w_{3,2,4} \end{bmatrix} \in \mathbb{R}^{3 \times 8} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}$$

Applying 3 bigram filters

$$h^{max} = \langle \max(h_{*,1}), \max(h_{*,2}), \max(h_{*,3}) \rangle^T$$

Words	Embeddings			
	d_1	d_2	d_3	d_4
I	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$
like	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$
this	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$
movie	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$
very	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$
much	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$
!	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	$x_{7,4}$



global max
pooling

Feature vector
sent to a
classifier,
regressor, etc.

$$h_1 = \langle h_{1,1}, h_{1,2}, h_{1,3} \rangle^T$$

$$h_2 = \langle h_{2,1}, h_{2,2}, h_{2,3} \rangle^T$$

$$h_3 = \langle h_{3,1}, h_{3,2}, h_{3,3} \rangle^T$$

$$h_4 = \langle h_{4,1}, h_{4,2}, h_{4,3} \rangle^T$$

...

$$h_7 = \langle h_{7,1}, h_{7,2}, h_{7,3} \rangle^T$$

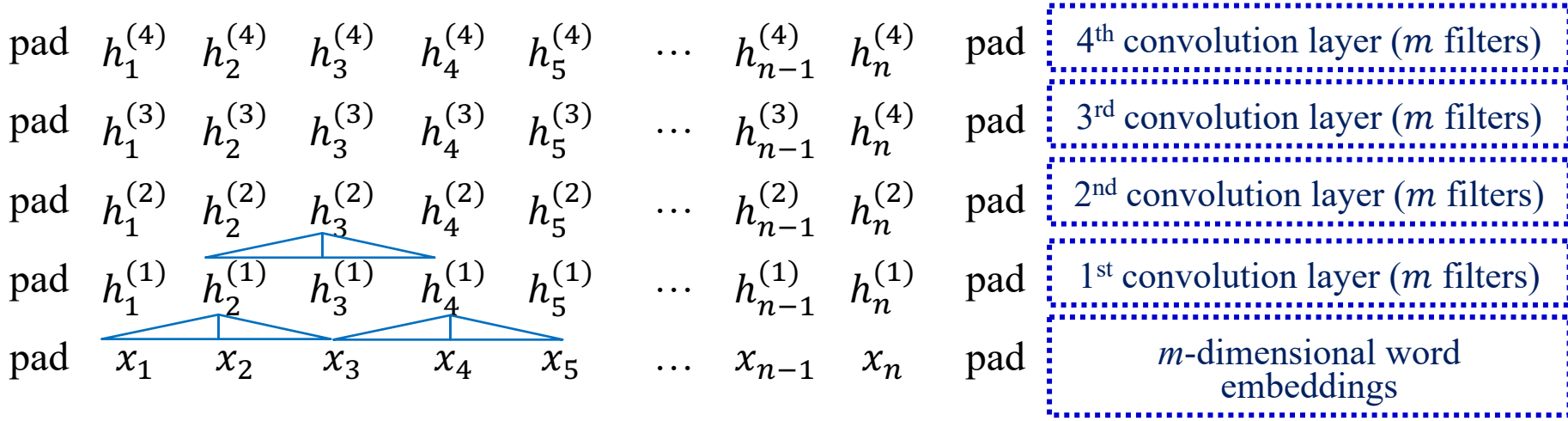
$$W = \begin{bmatrix} w_{1,1,1} & w_{1,1,2} & w_{1,1,3} & \dots & w_{1,2,3} & w_{1,2,4} \\ w_{2,1,1} & w_{2,1,2} & w_{2,1,3} & \dots & w_{2,2,3} & w_{2,2,4} \\ w_{3,1,1} & w_{3,1,2} & w_{3,1,3} & \dots & w_{3,2,3} & w_{3,2,4} \end{bmatrix} \in \mathbb{R}^{3 \times 8} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}$$

Stacked CNNs for classification/regression

$$h^{max} = \left\langle \max(h_{*,1}^{(4)}), \max(h_{*,2}^{(4)}), \dots, \max(h_{*,m}^{(4)}) \right\rangle^T \in \mathbb{R}^{1 \times m}$$

↑ global max pooling

Feature vector sent to a document classifier or regressor (e.g., MLP).

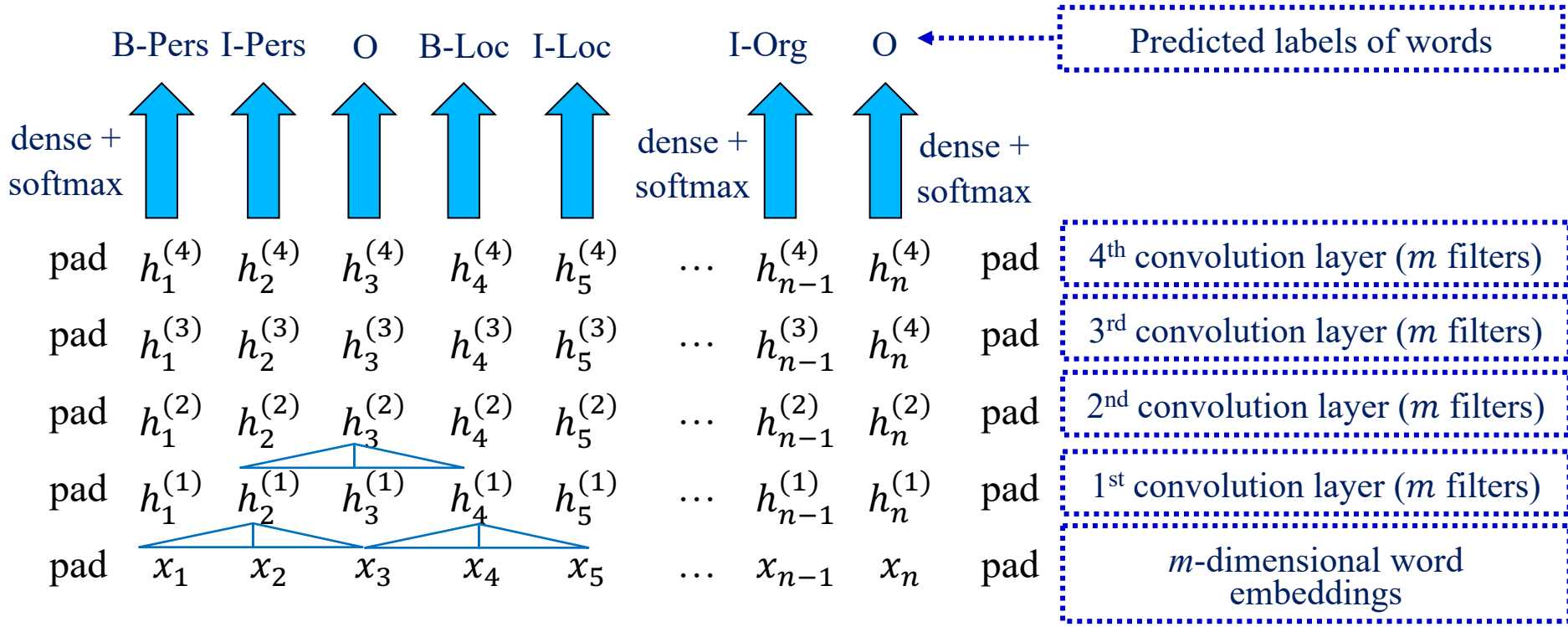


$$h_i^{(1)} = \text{ReLU}(W^{(1)} [x_{i-1}; x_i; x_{i+1}] + b^{(1)}) + x_i \in \mathbb{R}^{m \times 1}$$

$$h_i^{(j)} = \text{ReLU}(W^{(j)} [h_{i-1}^{(j-1)}; h_i^{(j-1)}; h_{i+1}^{(j-1)}] + b^{(j)}) + h_i^{(j-1)} \in \mathbb{R}^{m \times 1}$$

Residual (shortcut) connection, needed when stacking many CNNs (or RNNs).

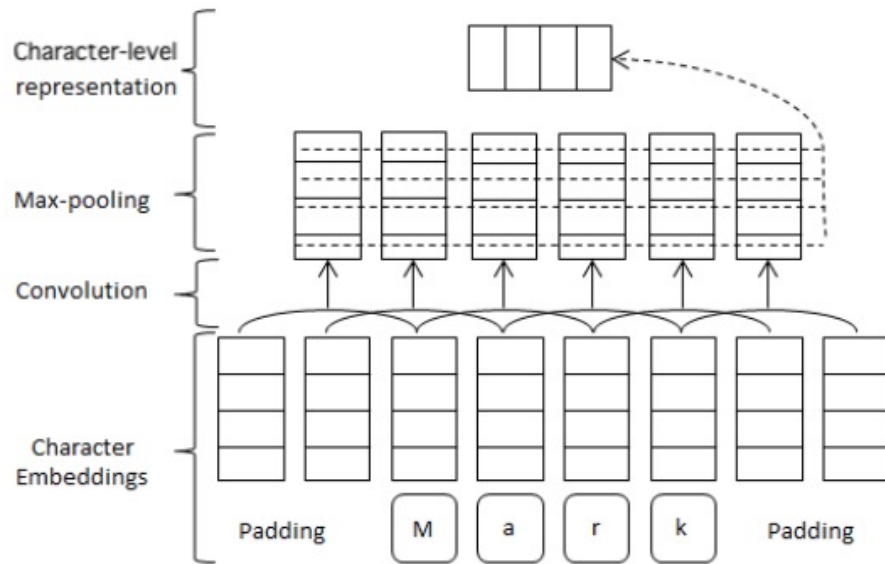
Stacked CNNs for token classification



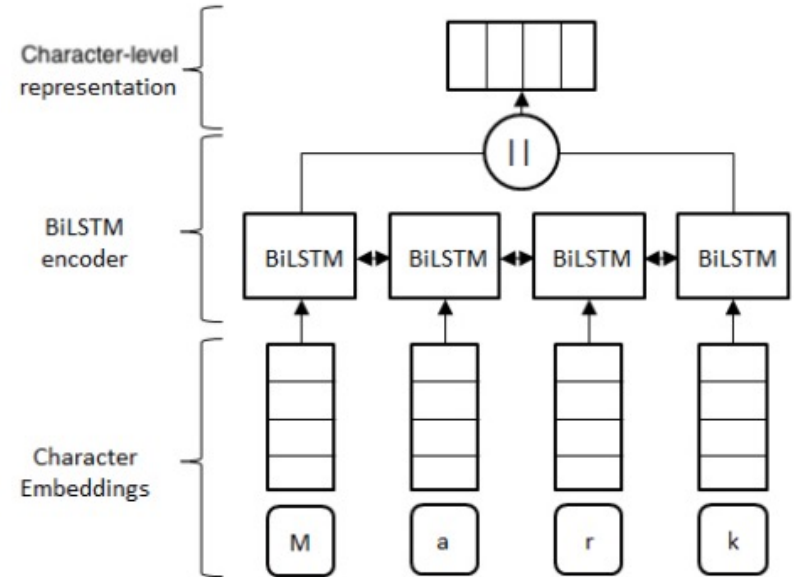
$$h_i^{(1)} = \text{ReLU}(W^{(1)}[x_{i-1}; x_i; x_{i+1}] + b^{(1)}) + x_i \in \mathbb{R}^{m \times 1}$$

$$h_i^{(j)} = \text{ReLU}(W^{(j)}[h_{i-1}^{(j-1)}; h_i^{(j-1)}; h_{i+1}^{(j-1)}] + b^{(j)}) + h_i^{(j-1)} \in \mathbb{R}^{m \times 1}$$

CNNs/RNNs that produce word embeddings from character embeddings



(CNN-based character-level word representation)



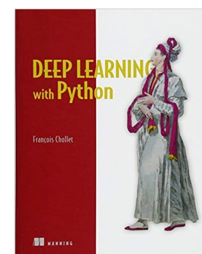
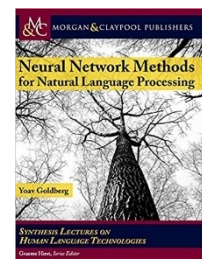
(LSTM-based character-level word representation)

Figure 2: Character-level word representations. This figure is also adapted from Reimers and Gurevych (2017a).

Z. Zhai, D.Q. Nguyen and K. Verspoor, “Comparing CNN and LSTM Character-Level Embeddings in BiLSTM-CRF Models for Chemical and Disease Named Entity Recognition”. 9th Int. Workshop on Health Text Mining and Information Analysis, Brussels, Belgium, 2018. <http://aclweb.org/anthology/W18-5605>

Recommended reading

- Y. Goldberg, *Neural Network Models for Natural Language Processing*, Morgan & Claypool Publishers, 2017.
 - Mostly Chapter 13.
- Jurafsky and Martin's, *Speech and Language Processing* is being revised (3rd edition) to include DL methods.
 - <http://web.stanford.edu/~jurafsky/slp3/>
- F. Chollet, *Deep Learning in Python*, 1st edition, Manning Publications, 2017.
 - 1st edition freely available (and sufficient for this course): <https://www.manning.com/books/deep-learning-with-python>
 - See Chapter 6 for CNNs in Computer Vision.
 - 2nd edition (2022) now available, requires payment. Highly recommended.



Recommended reading – continued

- A. Zhang et al., *Dive into Deep Learning*.
 - Freely available at: <https://d2l.ai/>
 - See Chapter 6 for CNNs.
- See also the recommended reading and resources of the previous parts (NLP with MLPs, RNNs) of this course.

