Foundations for Machine Learning

L. Y. Stefanus

TU Dresden, June-July 2018

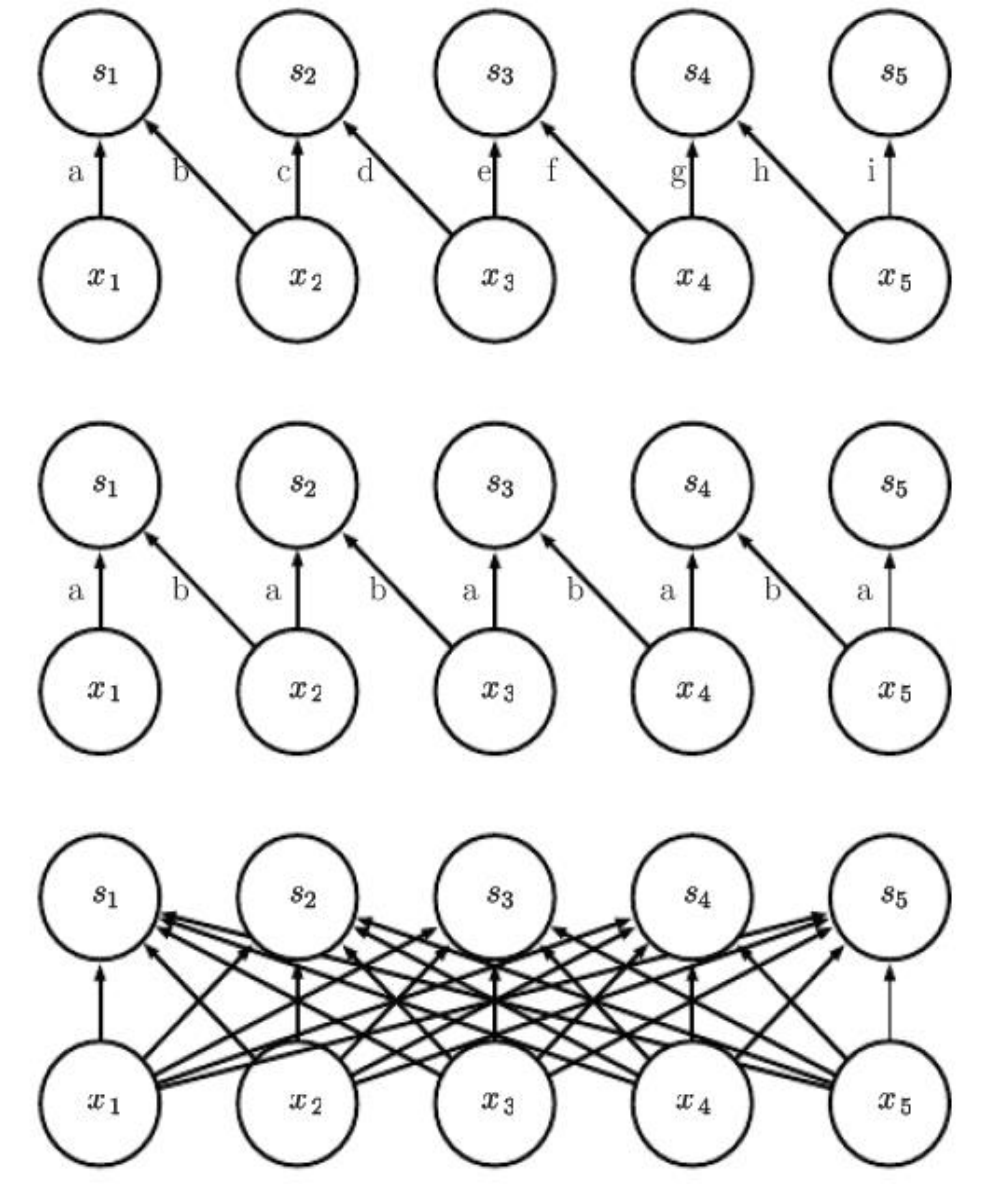
Slides 12

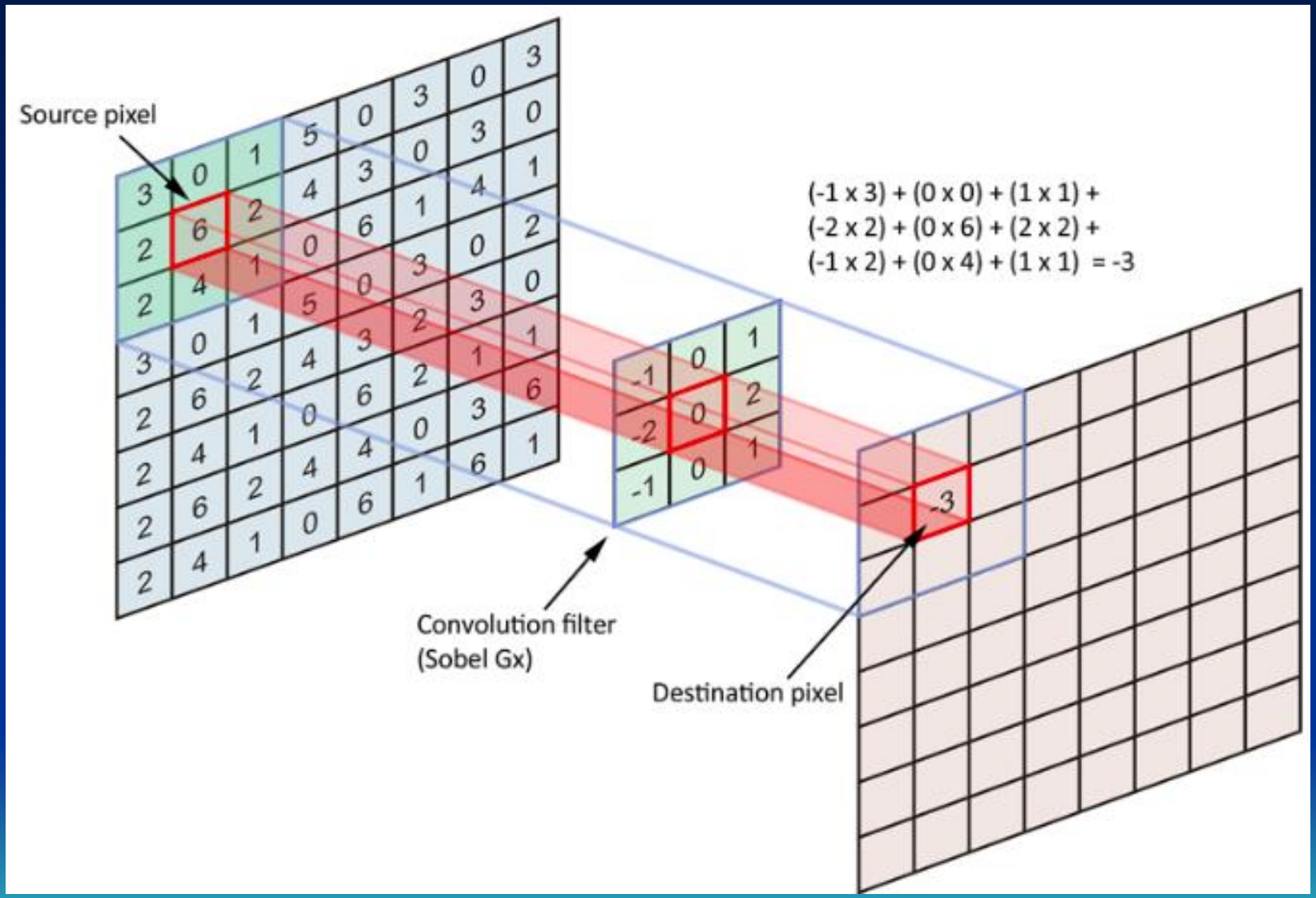
Reference

- Ian Goodfellow and Yoshua Bengio and Aaron Courville. DEEP LEARNING. MIT Press, 2016.

Convolutional Neural Networks (CNN)
part 2

Exercise:
Which network is convolutional?




Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel
-3

Convolution and Pooling as Prior Knowledge

- Recall the concept of a prior knowledge.
- Prior knowledge can take the form of a probability distribution over the parameters of a hypothesis that encodes our beliefs, before we have seen any data.
- Prior knowledge can be considered weak or strong depending on how concentrated the probability density in the prior knowledge is.
- A weak prior knowledge is a prior distribution with high entropy, such as a Gaussian distribution with high variance. Such a prior knowledge allows the data to move the parameters more or less freely.

Convolution and Pooling as Prior Knowledge

- A strong prior knowledge has very low entropy, such as a Gaussian distribution with low variance. Such a prior knowledge plays a more active role in determining where the parameters end up as a result of learning.
- An infinitely strong prior knowledge places zero probability on some parameters and says that these parameter values are completely forbidden, regardless of how much support the data give to those values.

Convolution and Pooling as Prior Knowledge

- We can imagine a CNN as being similar to a fully connected net, but with an infinitely strong prior knowledge over its weights.
- This infinitely strong prior knowledge says that the weights for one hidden unit must be identical to the weights of its neighbor but shifted in space. It also says that the weights must be zero, except for in the small, spatially contiguous receptive field assigned to that hidden unit.
- Overall, we can think of the use of convolution as introducing an infinitely strong prior probability distribution over the parameters of a layer. This prior knowledge says that the function the layer should learn contains only local interactions and is equivariant to translation.

Convolution and Pooling as Prior Knowledge

- Likewise, the use of pooling is an infinitely strong prior knowledge that each unit should be invariant to small translations.
- Of course, implementing a CNN as a fully connected neural net with an infinitely strong prior knowledge would be extremely wasteful computationally. But thinking of a CNN as a fully connected neural net with an infinitely strong prior knowledge can give us some insights into how CNNs work.

Convolution and Pooling as Prior Knowledge

- One key insight is that convolution and pooling can cause underfitting.
- Like any prior knowledge, convolution and pooling are only useful when the assumptions made by the prior knowledge are reasonably accurate.
- If a task relies on preserving precise spatial information, then using pooling on all features can increase the training error.
- When a task involves incorporating information from very distant locations in the input, then the prior knowledge imposed by convolution may be inappropriate.

Structured Outputs

- CNNs can be used to output a high-dimensional structured object, rather than just predicting a class label for a classification task or a real value for a regression task.
- Typically this object is a tensor, emitted by a standard convolutional layer. For example, the model might emit a tensor S, where $S_{i,j,k}$ is the probability that pixel (j, k) of the input to the network belongs to class i.
- This allows the model to label every pixel in an image and draw precise masks that follow the outlines of individual objects.

- Once a prediction for each pixel is made, various methods can be used to further process these predictions to obtain a segmentation of the image into regions.
- The general idea is to assume that large groups of contiguous pixels tend to be associated with the same label.

Data Types

- The data used with a convolutional neural network usually consist of several channels, each channel being the observation of a different quantity at some point in space or time.
- The following table shows some examples of data types with different dimensionalities and number of channels.

1-D Data Type

Single channel

Audio waveform: The axis we convolve over corresponds totime. We discretize time and measure the amplitude of the waveform once per time step.

Multichannel

Skeleton animation data: Animations of 3-D computer-rendered characters are generated by altering the pose of a “skeleton” over time. At each point in time, the pose is described by a specification of the angles of each of the joints in the skeleton. Each channel in the data represents the angle about one axis of one joint.

2-D Data Type

Single channel

Audio data that has been preprocessed with a Fourier transform:

The audio waveform is transformed into a 2-D tensor with different rows corresponding to different frequencies and different columns corresponding to different points in time. Using convolution across the frequency axis makes the model equivariant to frequency.

Multichannel

Color image data:

One channel contains the red pixels, one the green pixels, and one the blue pixels. The convolution kernel moves over both the horizontal and the vertical axes of the image, conferring translation equivariance in both directions.

3-D Data Type

Single channel

Volumetric data:

A common source of this kind of data is medical imaging technology, such as CT scans.

Multichannel

Color video data:

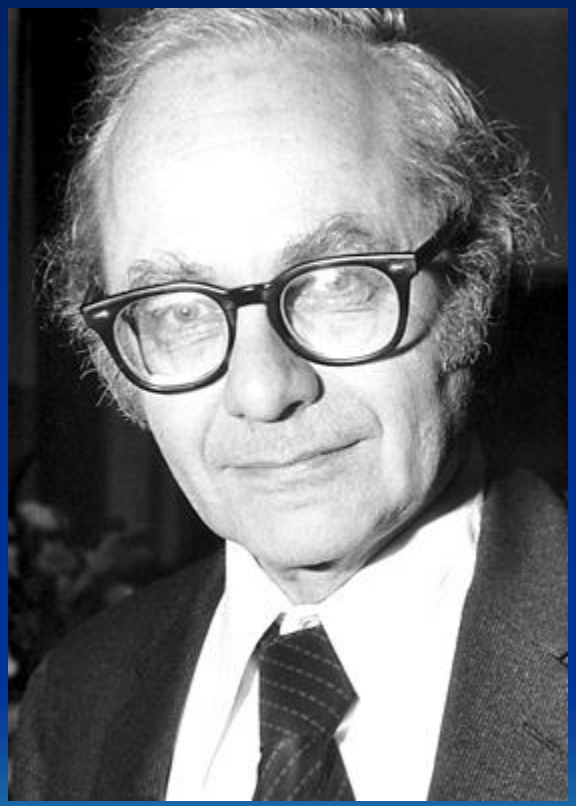
One axis corresponds to time, one to the height of the video frame, and one to the width of the video frame.

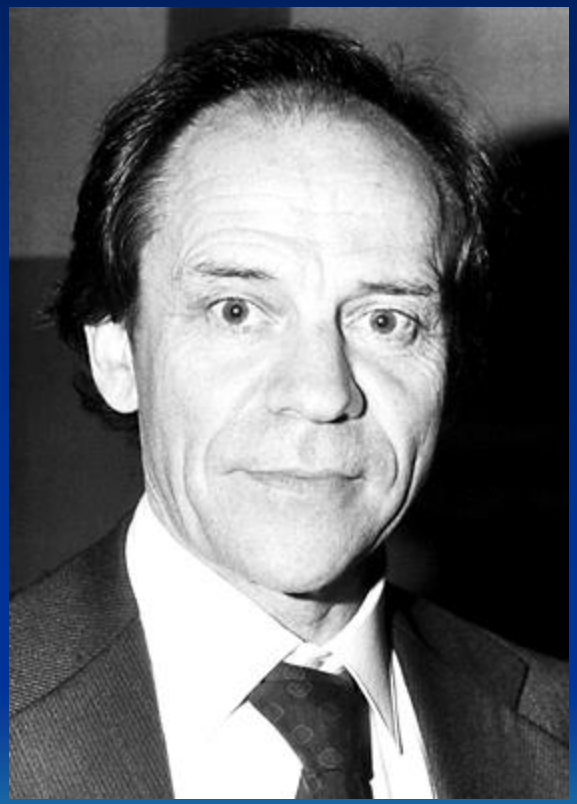
- One advantage to convolutional neural networks is that they can also process inputs with varying spatial extents.
- These kinds of input simply cannot be represented by traditional, matrix multiplication-based neural networks.
- For example, consider a collection of images in which each image has a different width and height. It is unclear how to model such inputs with a weight matrix of fixed size. Convolution is straightforward to apply; the kernel is simply applied a different number of times depending on the size of the input, and the output of the convolution operation scales accordingly.

- Note that the use of convolution for processing variably sized inputs makes sense only for inputs that have variable size because they contain varying amounts of observation of the same kind of thing, for example, different lengths of recordings over time, different widths of observations over space, and so forth.
- Convolution does not make sense if the input has variable size because it can include different kinds of observations.

The Neuroscientific Basis for CNN

- Although CNNs have been guided by many fields, some of the key design principles of CNNs were drawn from neuroscience. The history of CNNs begins with neuroscientific experiments long before the relevant computational models were developed.
- Neurophysiologists David Hubel and Torsten Wiesel collaborated for several years to determine many of the most basic facts about how the mammalian vision system works (Hubel and Wiesel, 1959, 1962, 1968). Their accomplishments were recognized with a Nobel prize in 1981.





David Hubel and Torsten Wiesel

- Their findings that have had the greatest influence on deep machine learning were based on recording the activity of individual neurons in cats.
- They observed how neurons in the cat's brain responded to images projected in precise locations on a screen in front of the cat.
- Their great discovery was that neurons in the early visual system responded most strongly to very specific patterns of light, such as precisely oriented bars, but responded hardly at all to other patterns.

- For simplicity, we focus on a part of the brain called V1, also known as the primary visual cortex. V1 is the first area of the brain that begins to perform significantly advanced processing of visual input.
- In this simplified view, images are formed by light arriving in the eye and stimulating the retina, the light-sensitive tissue in the back of the eye. The neurons in the retina perform some simple preprocessing of the image but do not substantially alter the way it is represented. The image then passes through the optic nerve and a brain region called the LGN (lateral geniculate nucleus). The main role of both anatomical regions is to carry the signal from the eye to V1, which is located at the back of the head.

A CNN layer is designed to capture three properties of V1:

1. V1 is arranged in a spatial map. It actually has a two-dimensional structure, mirroring the structure of the image in the retina. CNNs capture this property by having their features defined in terms of two-dimensional maps.
2. V1 contains many simple cells. A simple cell's activity can to some extent be characterized by a linear function of the image in a small, spatially localized receptive field. The detector units of a convolutional network are designed to emulate these properties of simple cells.

3. V1 also contains many complex cells. These cells respond to features that are similar to those detected by simple cells, but complex cells are invariant to small shifts in the position of the feature. This inspires the pooling units of CNNs. Complex cells are also invariant to some changes in lighting that cannot be captured simply by pooling over spatial locations. These invariances have inspired some of the cross-channel pooling strategies in CNNs.

- It is worth mentioning that neuroscience has told us relatively little about how to train CNNs.
- CNN structures with parameter sharing across multiple spatial locations date back to early connectionist models of vision (Marr and Poggio, 1976), but these models did not use the modern back-propagation algorithm and SGD (stochastic gradient descent).
- Modern CNN was developed by Yann LeCun *et al.* around 1989.

Implementation of CNN

- If we select the programming language Python for implementing CNN, popular libraries are:
  - Theano: a Python library for defining mathematical functions (operating over vectors and matrices), and computing the gradients of these functions. Building deep learning models fundamentally involves optimizing complicated loss functions using stochastic gradient descent (SGD). This is where Theano comes in handy.

Implementation of CNN

- Keras: a Python library that provides highly powerful and abstract building blocks to build deep learning networks. The building blocks are built using Theano as well as TensorFlow (which is an alternative to Theano). Keras supports both CPU and GPU computation and is a great tool for quickly prototyping learning systems.

Auf Wiedersehen!

Go and learn deep …
deep learning ☺