Chapter 8: Convolutional Networks

Dietrich Klakow
Spoken Language Systems
Saarland University, Germany
Dietrich.Klakow@LSV.Uni-Saarland.De

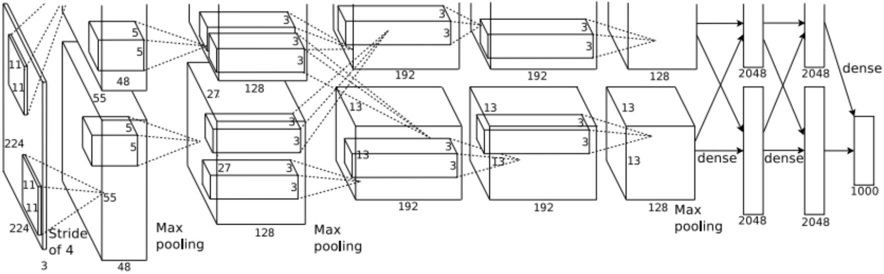
Neural Networks Implementation and Application

UNIVERSITÄT
DES
SAARLANDES

Introduction

Source: http://www.dailymail.co.uk/news/article-564264/The-worlds-tallest-Lego-tower-took-500-000-bricks-build.html

Introduction: AlexNet



- Source: "ImageNet Classification with Deep Convolutional Neural Networks" Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, NIPS 2012
- 650,000 neurons
- 60 million parameters
- Roughly 100 parameters per neuron!

Outline

1. Convolution
2. Motivation for Convolution
3. Pooling
4. Variants of the convolution function
5. Structured outputs and other related issues
6. Example: cifar10 from tensorflow tutorial

Section 1

Convolution

Convolution

- Operates on two functions
- Example signal $s(t)$ that you are hearing depends on
  - Signal produced $x(t)$
  - Impulse response of lecture hall $w(t)$
- $$s(t) = \int x(a)w(t-a)\mathrm{da}$$
- Shorthand notation
  $$s(t) = (x * w)(t)$$
- When $w(t)$ is sometimes called *kernel*
- Convolution is commutative

Convolution

- Discrete convolution (time index t)

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- Two dimensional discrete convolution

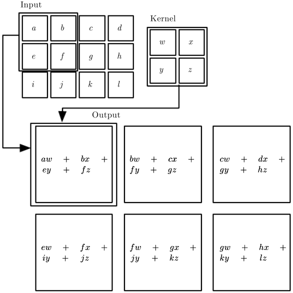
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

- Relation to *cross-correlation*

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

- Cross-correlation is convolution with a flipped kernel
- For this reason the book uses cross-correlation and convolution interchangibly

Example of a 2D convolution



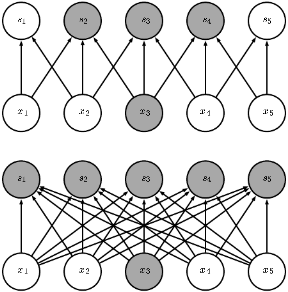
- Strictly speaking this calculates a correlation

Motivation for Convolution

Motivation for Convolution

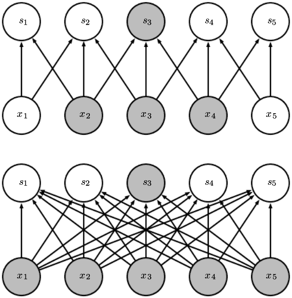
- Parameter sharing in a layer of a neural network
- Provides means to work with inputs of variable size
- Faster computation

Sparse connectivity viewed from below



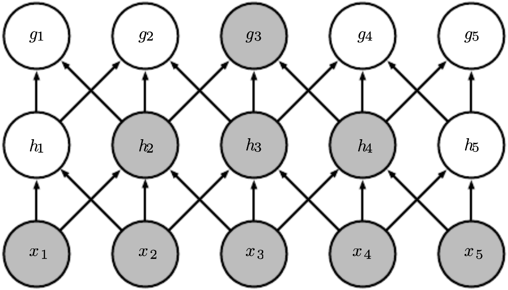
- x_3 influences only s_2, s_3 and s_4

Sparse connectivity viewed from above



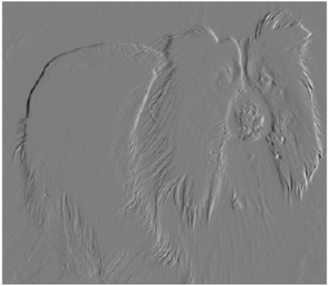
- s_3 is only influenced by x_2, x_3 and x_4
- The neurons that influence s_3 are called the *receptive field* of s_3

Receptive field in a deeper network



- Locally the connections are sparse
- Globally, everything indirectly influences most (or all) of the output

Efficient edge detection



- Take difference between each pixel and its vertical neighbor
- Image is 280x320 pixels
- This edge detection takes 267960 operations
- Applying a fully connected matrix would take 16 billion floating point operations

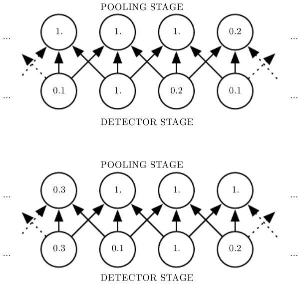
Section 3

Pooling

Pooling

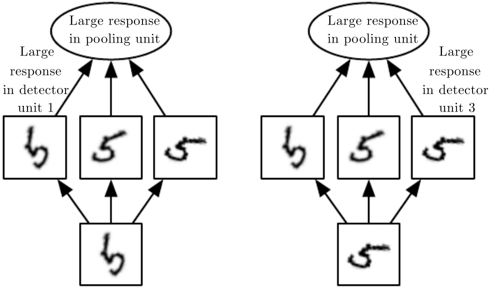
- *Pooling:* replaces the output at a certain location by some summary statistics over neighborhood
- *max pooling:* maximum output within a regular window
- Alternative: average over a rectangular window

Example of max pooling



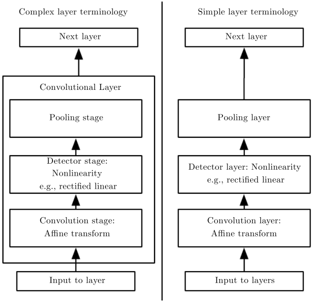
- Input in the two examples above is shifted by a pixel
- Output becomes invariant so small shifts

Example of learned invariances



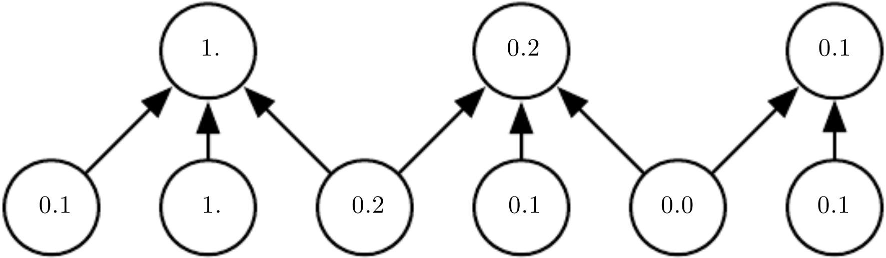
- Orientation of the input does not matter for the output

Typical layout of a convolutional network



- Alternative terminologies exists as to what is a layer

Pooling with down-sampling



- Reduces computational burden for next layer

Section 4

Variants of the convolution function

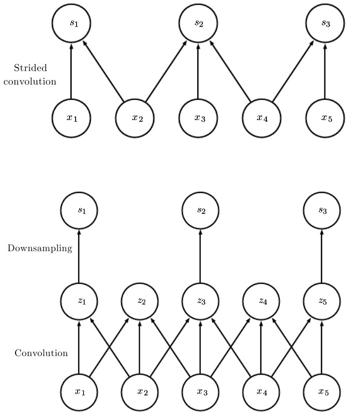
Convolution for multi-channel input (color images)

- Let $V_{l,j,k}$ be a color image where l denotes the color channel and j and k the position in the image
-
$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n}$$
- Above formula assumes C/Python indexing in the arrays
- *stride:* skip some of the outputs
- Convolution with stride

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n}$$

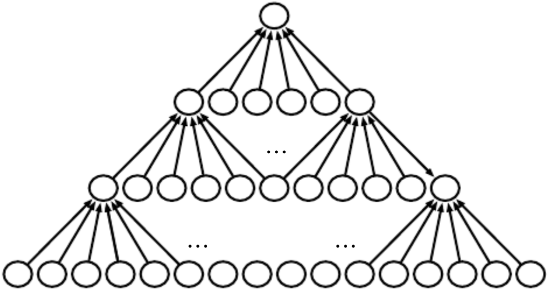
- s is the stride
- Indices i and l are indexing color
- j, k, m, n: space indices

Convolution with stride



- Convolution with stride:
  equivalent to convolution with down-sampling
- Issue: s_1 and s_3 are influenced only by two inputs

Pooling and zero padding



- Pooling results in an undesired reduction in the number of nodes

- Zeros (black circles) are added to keep layer size fixed

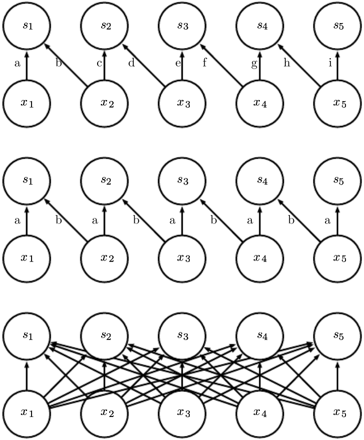
Unshared convolution/local connections

- $$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,j,k,l,m,n}$$

Use when:

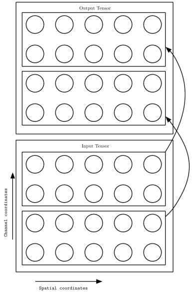
- all interactions are local
- different regions of image behave differently
- Indices i and l are indexing color
- j, k, m, n: space indices

Comparison of full weights and convolution



- Top: locally connected
- Center: convolution
- Bottom: fully connected

Tiled convolution

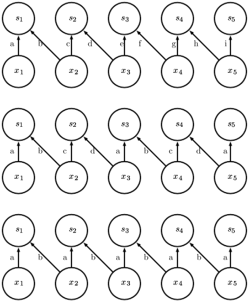


- Compromise between locally connected and convolution (see next slide)
- No connections between different channels (e.g. colors)
- Formal definition

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n,j\%t+1,k\%t+1}$$

- % is the modulo operation

Tiled convolution



- Top: general locally connected network
- Center: t=4
- Bottom: t=2

Minimize conv. nets with stride

- Suppose we want to minimize $J(\mathrm{V}, \mathrm{K})$
- Suppose during back propagation we receive tensor

$$G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(\mathrm{V}, \mathrm{K})$$

- Gradient to update K

$$g(\mathrm{G}, \mathrm{V}, s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} J(\mathrm{V}, \mathrm{K}) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k,(n-1)\times s+l}$$

- To backpropagate further to the beginning of the network

$$h(\mathrm{K}, \mathrm{V}, s)_{i,j,k} = \frac{\partial}{\partial V_{i,j,k}} J(\mathrm{V}, \mathrm{K}) = \sum_{\substack{m,n s.t. \\ (l-1)\times s+m=j}} \sum_{\substack{n,p s.t. \\ (n-1)\times s+p=k}} \sum_{q} K_{q,i,m,p} G_{q,l,n}$$

- Note: forget about the details! Not relevant for exam in this generality.

Section 5

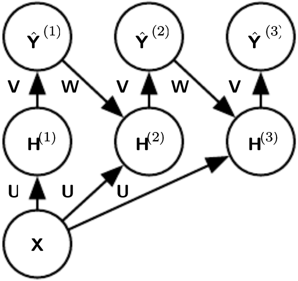
Structured outputs and other related issues

Structured outputs

Source: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/29517/versions/5/screenshot.jpg

In image segmentation an image X produces an output $\hat{\mathrm{Y}}$ for each pixel

Structured outputs



- X is the matrix representing the input image
- $\hat{Y}^{(i)}$ is the matrix with labels for each input pixel
- Refining output for image segmentation iteratively
- Simple recurrent network
- Captures interaction between neighboring pixels

Data types

- Single channel
  - 1-D: audio waveform
  - 2-D: grey level image
  - 3-D: CT scan
- Multi-channel
  - 1-D: Movement of skeleton (one channel per joint)
  - 2-D: Color image
  - 3-D: Color video

Efficient convolution algorithms

Assume kernel is w pixels wide
Kernel is d dimensional
$\rightarrow O(w^d)$ operations

- Fourier transform
  - Transform kernel and image to frequency space: $O(d \times w \log w)$ operations
  - Multiply in frequency space: $O(d \times w)$ operations
  - Transform back to position space: $O(d \times w \log w)$ operations
- Separable kernels
  - Kernel can be written as an out product of vectors
  - That means that the kernel can be applied in each dimension separately
  - Requires $O(d \times w)$ operations
  - Note: only some kernels are separable

Gabor functions

- Let $I(x, y)$ be an image
- and $w(x, y)$ a filter
- Filtered image is given by

$$S(x_f, y_f) = \sum_{x,y} w(x, y) I(x - x_f, y - y_f)$$

- Gabor function

$$w(x, y) = \alpha \exp(-\beta_x x'^2 - \beta_y y'^2) \cos(fx' + \phi)$$

- where

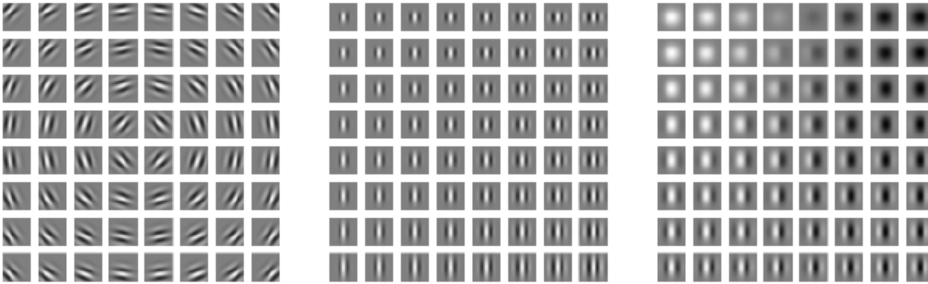
$$x' = (x - x_0) \cos(\tau) + (y - y_0) \sin(\tau)$$

- and

$$y' = -(x - x_0) \sin(\tau) + (y - y_0) \cos(\tau)$$

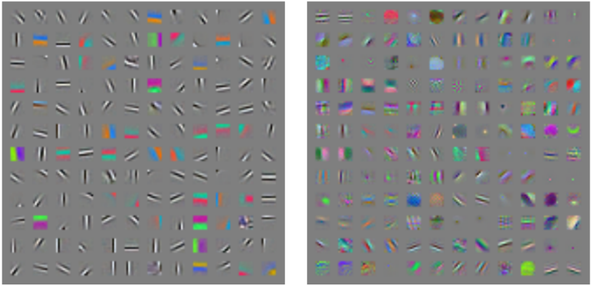
- $\beta_x . \beta_y, f, x_0, y_0, \phi, \tau$ are parameters

Gabor functions for a variety of parameters settings



- Gabor functions probe texture on different directions with different “wave length”

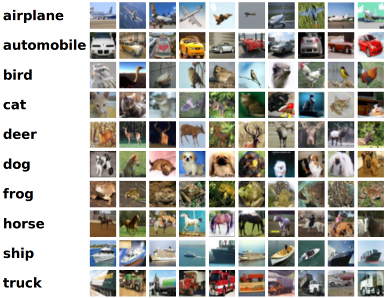
Features learned in first layer of CNN



- Resembles Gabor functions

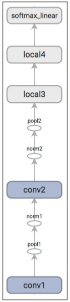
Example: cifar10 from tensorflow tutorial

Cifar 10: the data



- 60000 32x32 color images
- 10 classes,
- 6000 images per class
- 50000 training images
- 10000 test images.

Architecture of the model



2-D convolution given 4-D input

```
tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None, data_format=None, name=None)
```

Given:

- an input tensor of shape
  `[batch, in_height, in_width, in_channels]`
- A filter / kernel tensor of shape
  `[filter_height, filter_width, in_channels, out_channels]`

In detail, with the default NHWC format,

```
output[b, i, j, k] =
    sum_di, dj, q input[b, strides[1] * i + di,
        strides[2] * j + dj, q] * filter[di, dj, q, k]
```

Max pooling

`tf.nn.max_pool_with_argmax(input, ksize, strides, padding, Targmax=None, name=None)`

Performs max pooling on the input and outputs both max values and indices.

The indices in argmax are flattened, so that a maximum value at position [b, y, x, c] becomes flattened index ((b * height + y) * width + x) * channels + c.

Local Response Normalization

`tf.nn.local_response_normalization(input, depth_radius=None, bias=None, alpha=None, beta=None, name=None)`

- The 4-D input tensor is treated as a 3-D array of 1-D vectors (along the last dimension)
- each vector is normalized independently
- Within a given vector, each component is divided by the weighted, squared sum of inputs within depth_radius.

In detail:

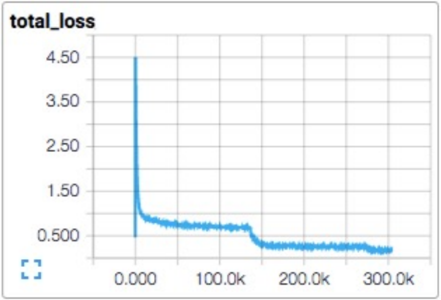
```
sqr_sum[a, b, c, d] =
   sum(input[a, b, c, d-depth_radius : d+depth_radius+1 ]**2)
output = input / (bias + alpha * sqr_sum) ** beta
```

Purpose: “lateral inhibition” that is an excited neuron reduces the neighbors in importance

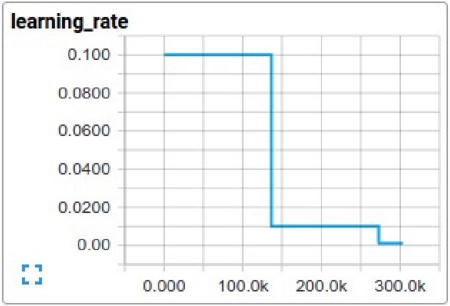
Example: cifar10

Walk through cifar10.py starting line 204

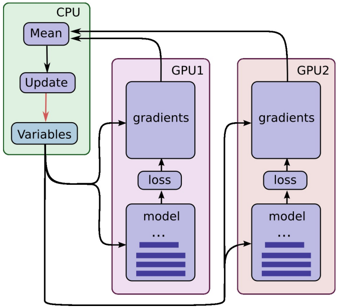
Total loss



Learning rate



Parallelism



- Calculate gradients for different batches on different GPUs
- Assumes that all GPUs need the same time for any batch

Summary

- Convolutional networks root in image processing
- Together with pooling they are good to cover invariances
- Surprisingly they work for language as well