RTG π^3 Compact Course A3

Deep learning for Digital Pathology and Inverse Problems

Daniel Otero Baguer, Peter Maass

Center for Industrial Mathematics (ZeTeM)
University of Bremen

Bremen
22-24.04.2020

Outline

Section 1

Introduction

Sources

- *"Deep Learning for Vision Systems"*, Mohamed Elgendy
  `https://livebook.manning.com/book/deep-learning-for-vision-systems/welcome/v-8`
- *"Convolutional Neural Networks for Visual Recognition"*
  `http://cs231n.stanford.edu/`

Why Deep learning now?

- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes
- Lots of data from the internet
- Imagenet

Why Deep learning now?

- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes
- Lots of data from the internet
- Imagenet

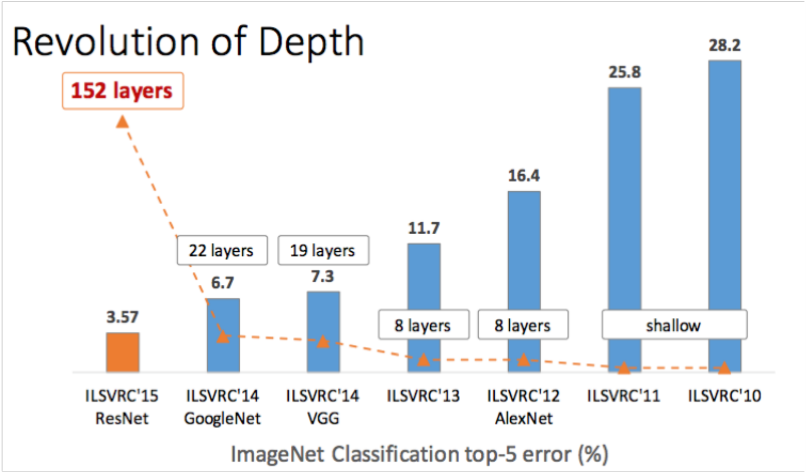
Why Deep learning now?

- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes
- Lots of data from the internet
- Imagenet

Why Deep learning now?

- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes
- Lots of data from the internet
- Imagenet

Why Deep Learning now?



Why Deep Learning now?

- Resources and efforts from large corporations make it **easier**!

Why Deep Learning now?

- Resources and efforts from large corporations make it **easier**!

The learning task

Let's consider a simple regression problem where the observations are:

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)}) \in X \times Y \tag{1}$$

Task: Infer a function f from the training data, which minimizes

$$R(f) = \int_{X \times Y} c(x, y, f(x)) \mathrm{d}P(x, y) \tag{2}$$

where c is a loss function. A common choice is $c(x, y, f(x)) = \frac{1}{2}\|f(x) - y\|^2$.

The learning task

Emprirical Risk

$$R(f) = \frac{1}{m}\sum_{i=1}^{m} c(x^{(i)}, y^{(i)}, f(x^{(i)})) \tag{3}$$

If we allow f to be any function that maps from X to Y, then we can minimize (3) but at the same time be very distant from the minimizer of (2).

$$f(x) = \begin{cases} y^{(i)} & x = x^{(i)} \\ 0 & \mathit{otherwise} \end{cases} \tag{4}$$

The learning task

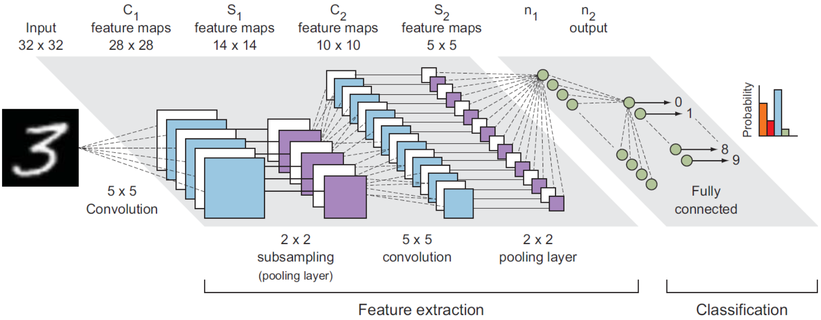
No Free lunch theorem

If we do not make any assumptions on the class of functions where f belongs to, there is no chance to learn anything.

Section 2

Convolutional Neural Networks

Convolutional Neural Network

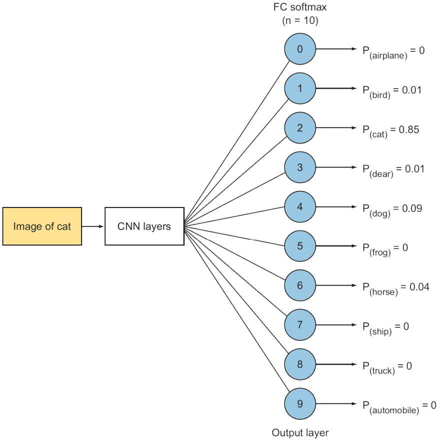


Layer types

- Input

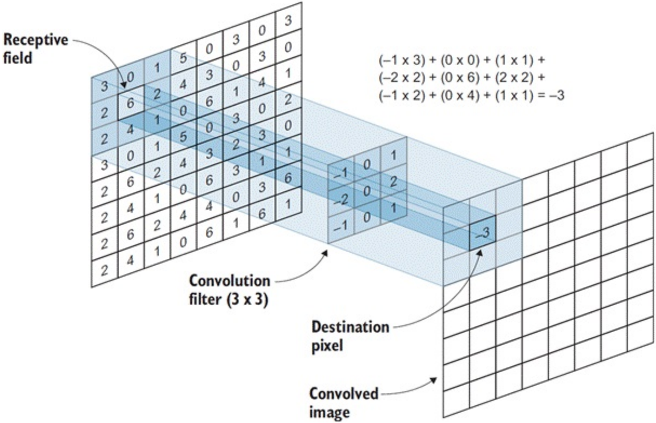
Layer types

- Output



Layer types

- Convolutions



Layer types

- Convolutions

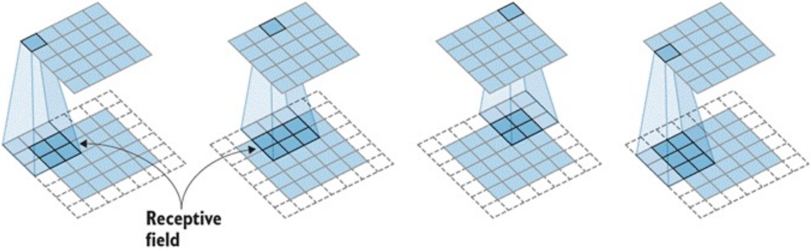


Figure: Receptive field

Layer types

- Convolutions

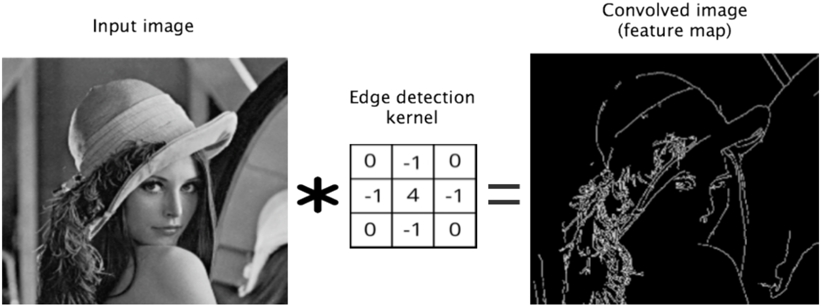


Figure: Convolutional layers are inspired on standard image filtering

Layer types

- Pooling

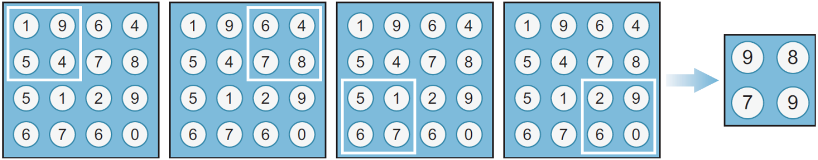
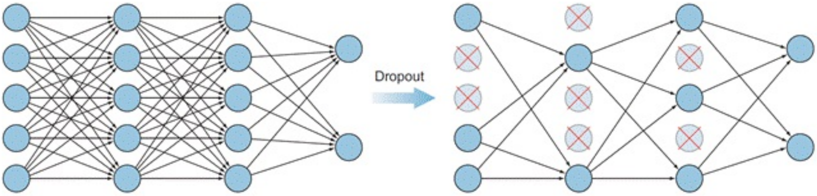


Figure: Max pooling example

Layer types

- Dropout



Tensors

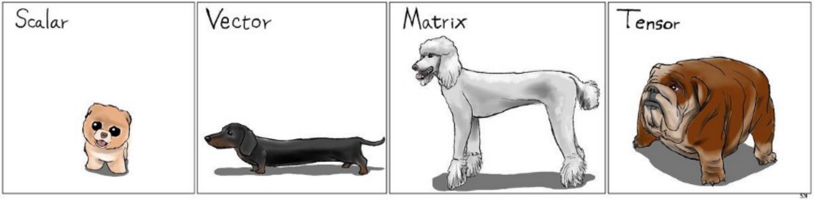


Figure: Tensors [1]

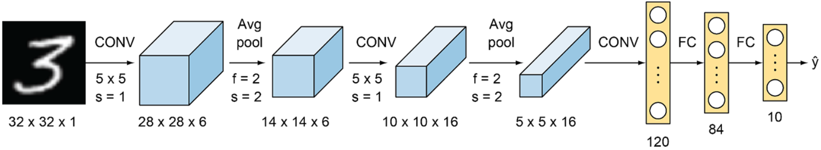
[1] Image source: https://towardsdatascience.com/understanding-pytorch-with-an-example-a-step-by-step-tutoria

Section 3

Architectures

LeNet

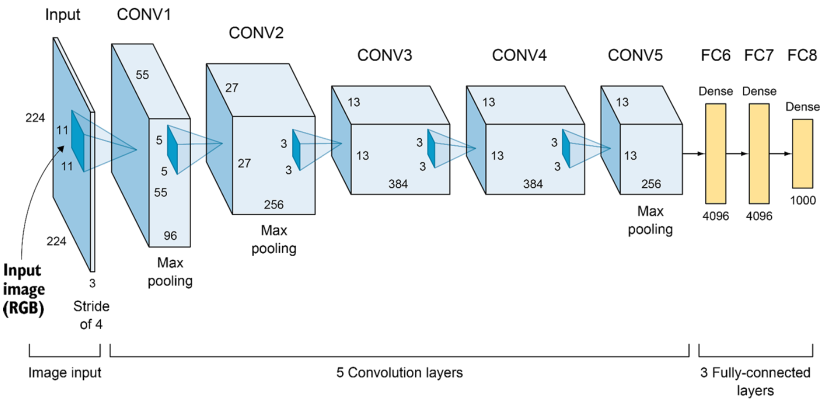
- Introduced in 1998, by LeCun et al. in their paper "Gradient-Based Learning Applied to Document Recognition"
- 5 layers: 3 conv + 2 fully-connected
- 61706 parameters



AlexNet

- Winner of the ILSVRC image classification competition in 2012.
- Introduced by Alex Krizhevsky, Geoffrey Hinton and Ilya Sutskever in their paper“ImageNet Classification with Deep Convolutional Neural Networks”
- 8 layers: 5 conv + 3 fully-connected
- 60 million parameters

AlexNet


Input
CONV1
CONV2
CONV3
CONV4
CONV5
FC6
FC7
FC8
224
11
11
224
3
55
5
5
55
96
27
3
27
3
256
13
3
13
3
384
13
3
13
3
384
13
13
256
Dense
Dense
Dense
4096
4096
1000
Input image (RGB)
Stride of 4
Max pooling
Max pooling
Max pooling
Image input
5 Convolution layers
3 Fully-connected layers

How deep can we go?

- Very deep networks are able to represent very complex functions
- The network can learn features at many different levels of abstraction

Vanishing gradients

- By the chain rule, the derivatives of each layer are multiplied down the network
- Gradient decreases exponentially as we propagate down to the initial layers
- First hidden layers are learning much slower than later hidden layers

How deep can we go?

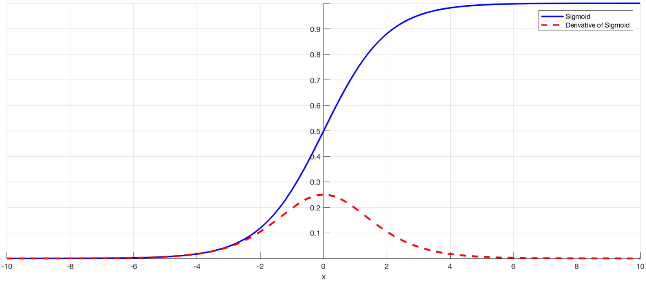


Figure: Sigmoid activation

How deep can we go?

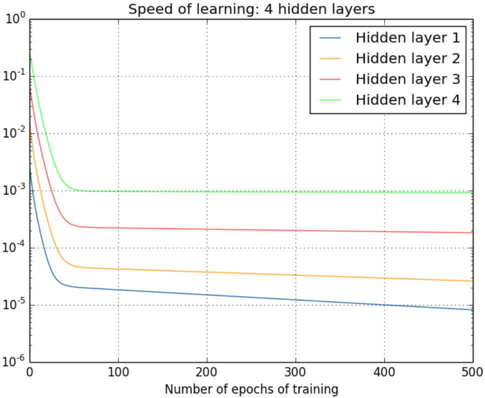


Figure: Vanishing gradient effect. First layers train much slower.

How deep can we go?

Solutions:

- Use ReLu activations
- Normalization layers
- Residual Networks

How deep can we go?

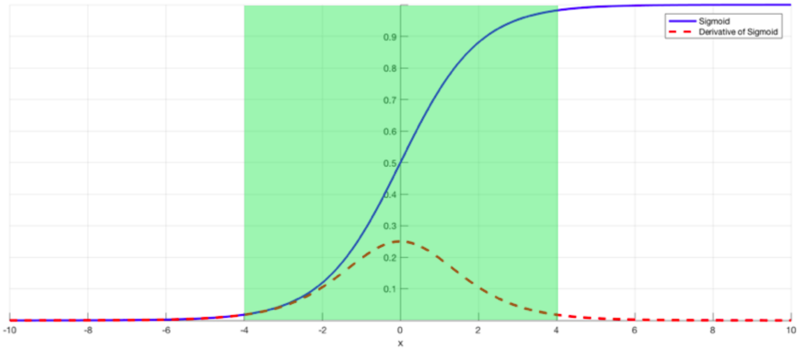


Figure: Sigmoid with restricted inputs

ResNet

- Introduces shortcut (skip-connection) that allows the gradient to directly back-propagate to earlier layers
- Allows the layer to learn an identity function (will perform at least as good as the previous layer)

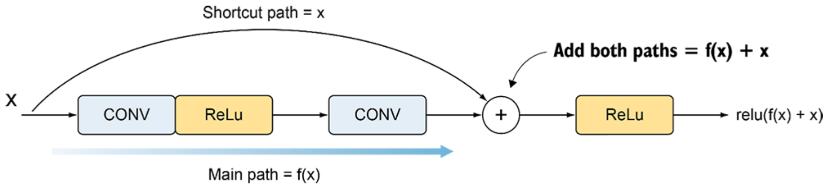


Figure: ResNet's skip connection

ResNet

- From 18 to 152 layers
- With 152 layers: $\approx$ 60 million parameters

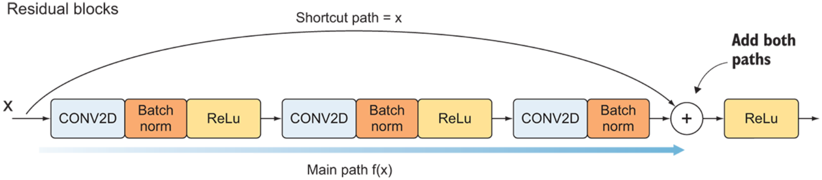


Figure: Residual block

ResNet

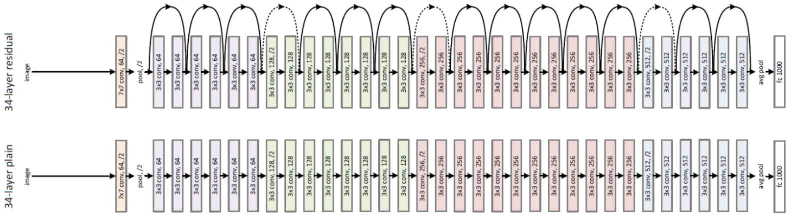


Figure: ResNet

Section 4

Data augmentation

Data augmentation

Main idea: Generate additional images based on existing ones

- Random flip
- Random erasing
- Random sized crop
- Random perspective
- Color jitter

Figure: Data augmentation

Data augmentation

Insights

- The epoch size does not change
- We get different randomly transformed samples every epoch (dynamic dataset)
- Helps to avoid overfitting

Advise

- Applying heavy augmentations unnecessarily can result in poor accuracy
- Do not use data augmentation in validation or test set

Data augmentation

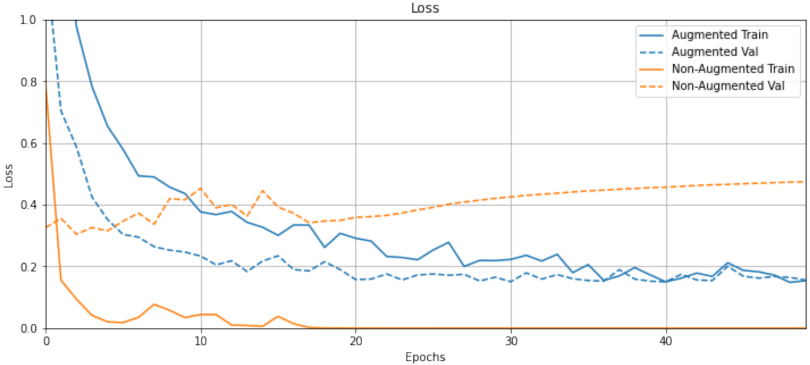


Figure: Non-augmented model vs augmented model [2].

[2] `https://www.tensorflow.org/tutorials/images/data_augmentation`

Section 5

Transfer learning

Transfer learning

Common phenomena on CNNs trained on natural images:

Figure: First layer learn features similar to Gabor filters and color blobs. Each of the 96 filters shown here is of size $[11 \times 11 \times 3]$,

Claim: Such a layer is not specific to a particular dataset or task

Transfer learning

Main idea: Use a pre-trained network

Use cases

- Fixed feature extractor: Take a pre-trained network and train only the last fully-connected layer
- Fine-tuning: Fine-tune the weights of the pre-trained network by continuing the back-propagation
  - Use small learning rate
  - Keep some of the earlier layers fixed (due to overfitting concerns)

When and how?

	Small dataset	Large dataset
Similar	Fixed feature extractor + Linear Classifier	Fine-tuning
Very different	Keep early layers fixed + Linear classifier	Pre-trained as initialization

Practical advice

- Constraints from pre-trained models
- Small learning rate for fine-tuning (avoid distorting pre-trained parameters too quickly or too much)