A tutorial on

Deep Learning in Medical Image Analysis

Ivana Išgum
Bob D. de Vos
Nikolas Lessmann
Jelmer M. Wolterink

University Medical Center Utrecht
The Netherlands

qia.isi.uu.nl

Mads Nielsen
Akshay Pai

University of Copenhagen
Denmark

image.diku.dk

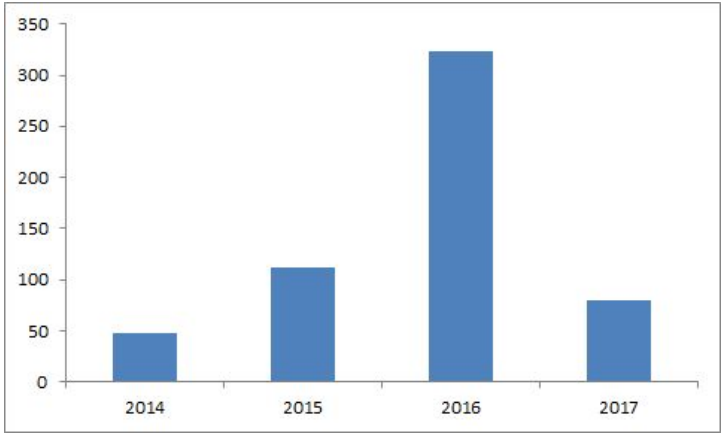
Program

1.	Introduction	Ivana Išgum
2.	Neural networks	Bob de Vos
3.	Convolutional neural networks	Nikolas Lessmann
4.	Generalization	Jelmer Wolterink
5.	Getting started	Akshay Pai
6.	General summary & discussion	Mads Nielsen

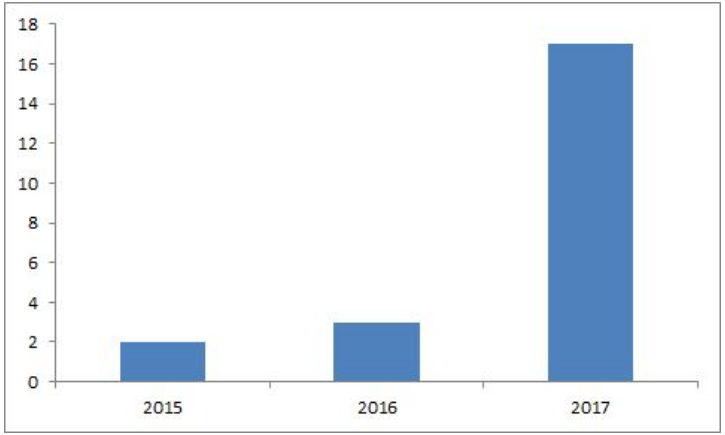
Deep learning?

- Branch of machine learning
- Based on neural networks
- Recently very successful in computer vision tasks
- Increasingly used in analysis of medical images

Pubmed search: "convolutional neural network" OR ConvNet OR "deep learning"



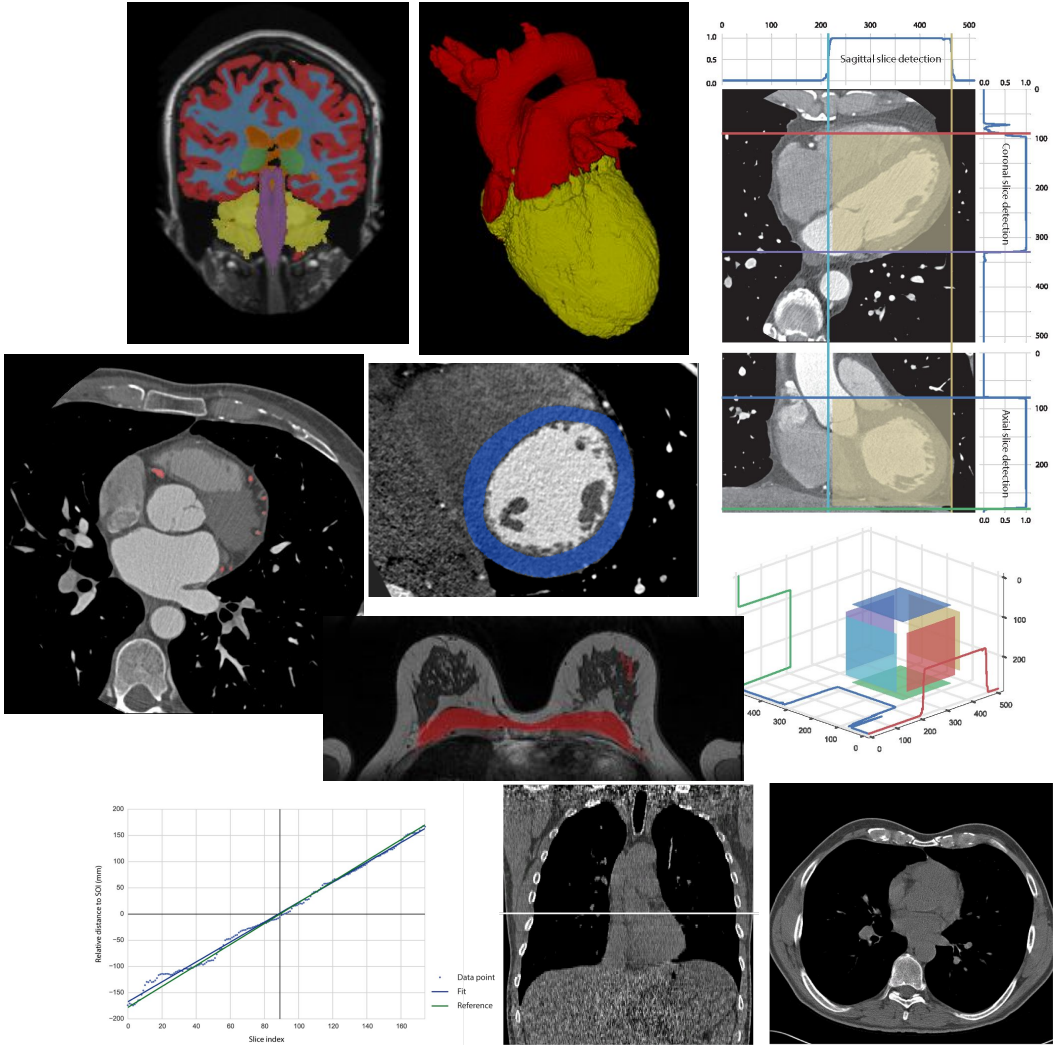
SPIE Medical Imaging: Image Processing conference



Yann LeCun, Yoshua Bengio & Geoffrey Hinton, Nature 521 (2015)
Jürgen Schmidhuber, Neural Networks 61 (2015)
IEEE Trans Med Imaging 35, issue 5 (2016)

Deep learning?

- Application
  - Segmentation
  - Localization
  - Quantification
  - Computer-aided diagnosis
  - .....



P. Moeskops et al., *MICCAI* 2016
J.M. Wolterink et al., *MICCAI* 2016 HVSMR workshop
M. Zreik et al., *ISBI* 2016
B. de Vos et al., *MICCAI* 2016 DLMIA workshop
B. de Vos et al., *IEEE TMI*, 2017

The many flavors of deep learning

- Multilayer perceptrons
- Convolutional neural networks
- Auto-encoders
- Reinforcement learning
- Generative (adversarial) networks
- Recurrent neural networks
- Visual attention networks
- ...

} **FOCUS OF THIS WORKSHOP**

Program

1.	Introduction	Ivana Išgum
2.	Neural networks	Bob de Vos
3.	Convolutional neural networks	Nikolas Lessmann
4.	Generalization	Jelmer Wolterink
5.	Getting started	Akshay Pai
6.	General summary & discussion	Mads Nielsen

Program

1.	Introduction	Ivana Išgum
2.	**Neural networks**	**Bob de Vos**
3.	Convolutional neural networks	Nikolas Lessmann
4.	Generalization	Jelmer Wolterink
5.	Getting started	Akshay Pai
6.	General summary & discussion	Mads Nielsen

2. NEURAL NETWORKS

Bob de Vos

Example: retinal vessel segmentation

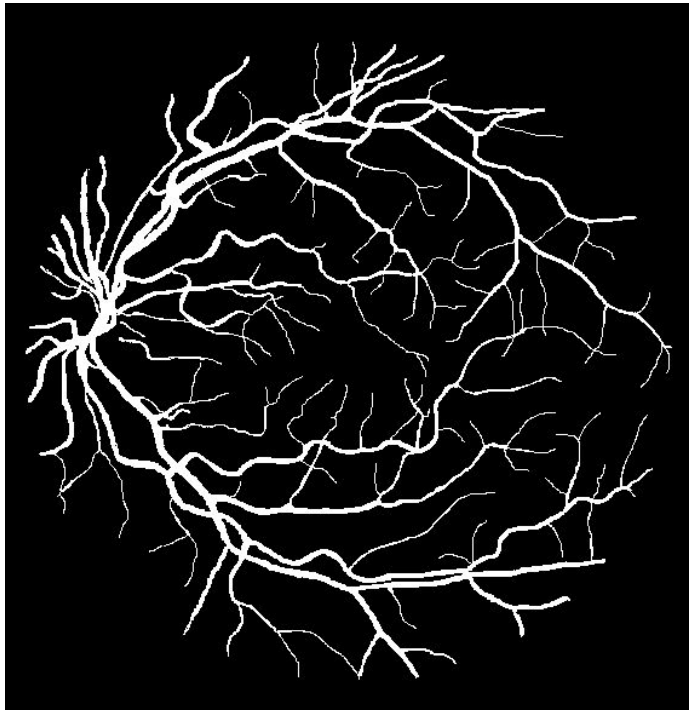
- DRIVE database[1]
  - 2D RGB images
  - 20 training images
  - 20 test images

[1] http://www.isi.uu.nl/Research/Databases/DRIVE/

Example: retinal vessel segmentation

- DRIVE database[1]
  - 2D RGB images
  - 20 training images
  - 20 test images

Supervised pixel classification
Automatically label each pixel as ‘vessel’ or ‘background’ class

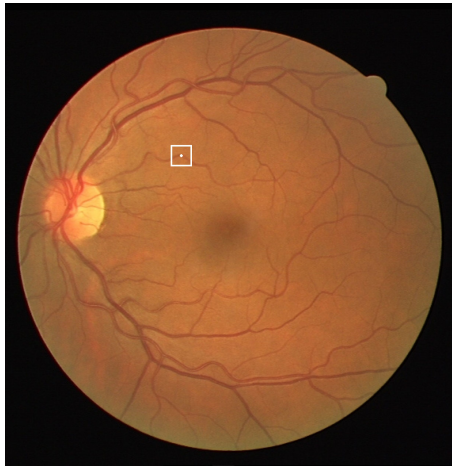


Segmentation of arteries and veins

[1] http://www.isi.uu.nl/Research/Databases/DRIVE/

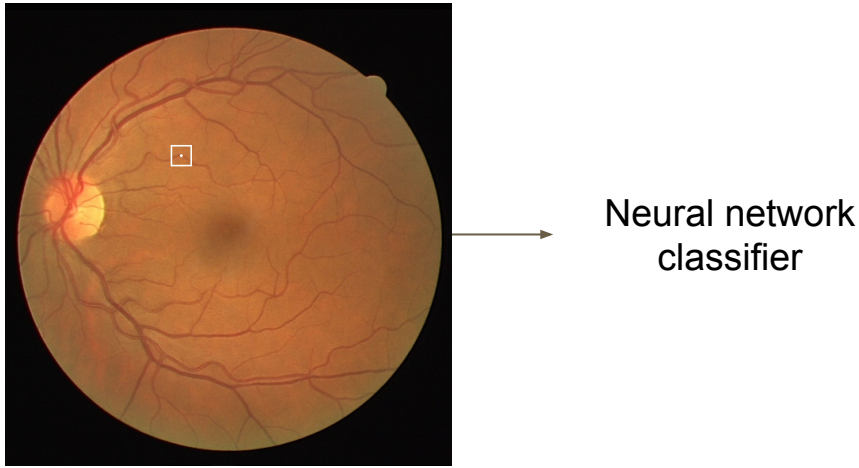
Pixel classification with neural networks

- Single pixel input: the value of a pixel is usually not enough to determine its class
- Patch input: information of the neighborhood around a pixel to describe it



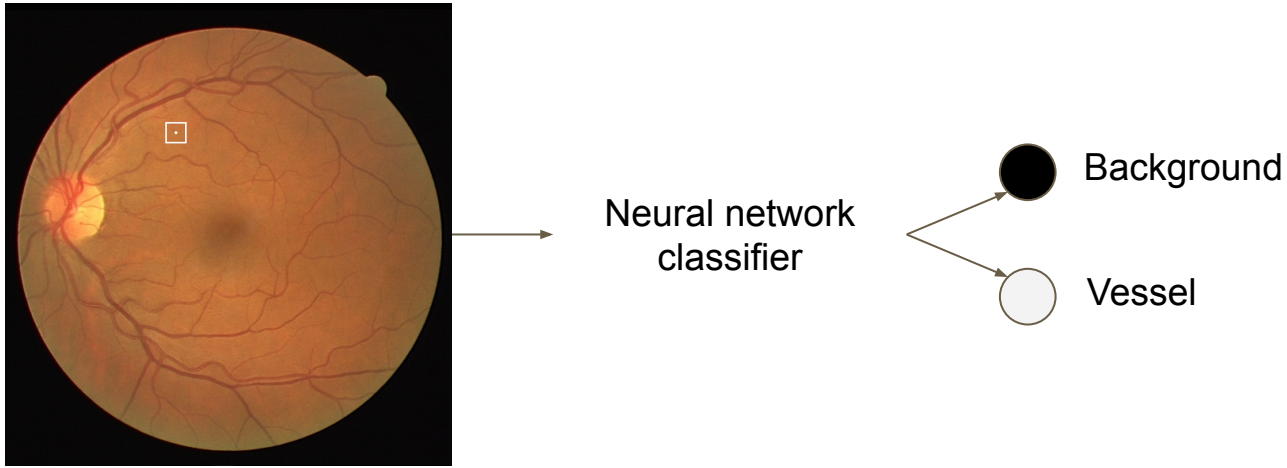
Pixel classification with neural networks

- Single pixel input: the value of a pixel is usually not enough to determine its class
- Patch input: information of the neighborhood around a pixel to describe it

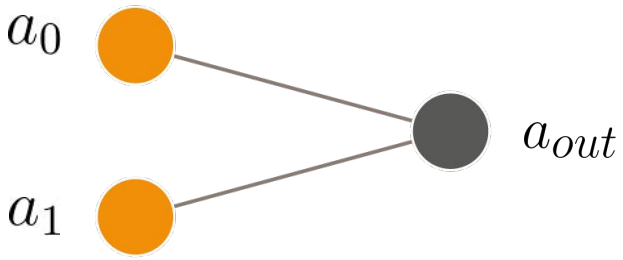


Pixel classification with neural networks

- Single pixel input: the value of a pixel is usually not enough to determine its class
- Patch input: information of the neighborhood around a pixel to describe it



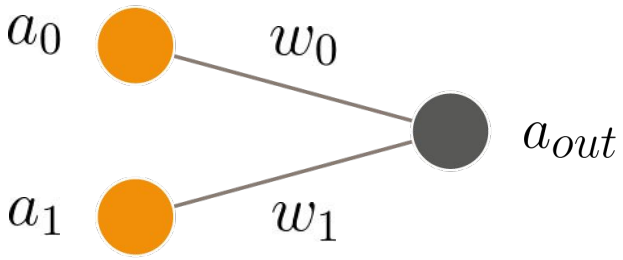
Artificial Neural Networks (ANNs)



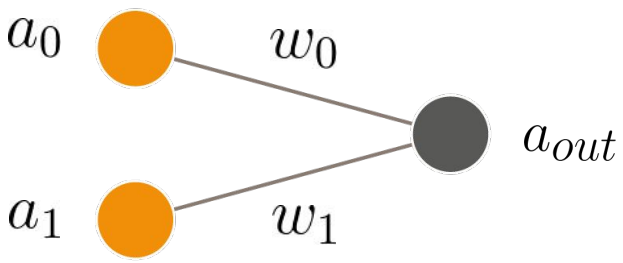
Artificial neural networks are composed of **units**

Artificial Neural Networks (ANNs)

Units are connected by **weights**



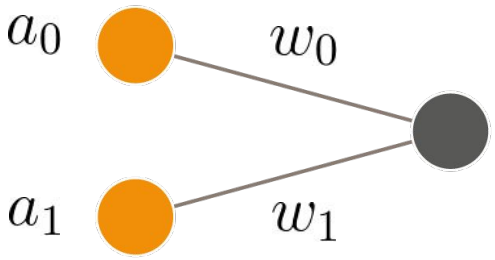
Artificial Neural Networks (ANNs)



The output activation a_{out} of a unit depends on

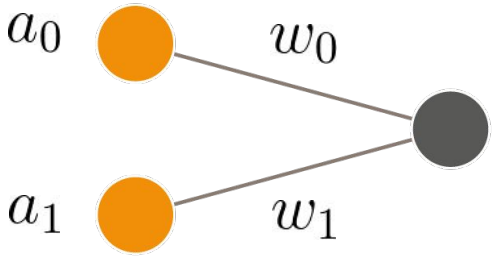
- Incoming activations a_0, a_1
- Weights w_0, w_1

Artificial Neural Networks (ANNs)

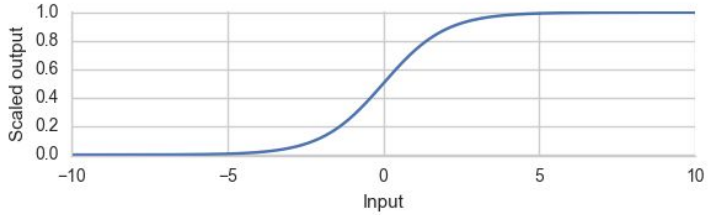


$$a_{out} = f(\sum_i w_i a_i)$$

Artificial Neural Networks (ANNs)

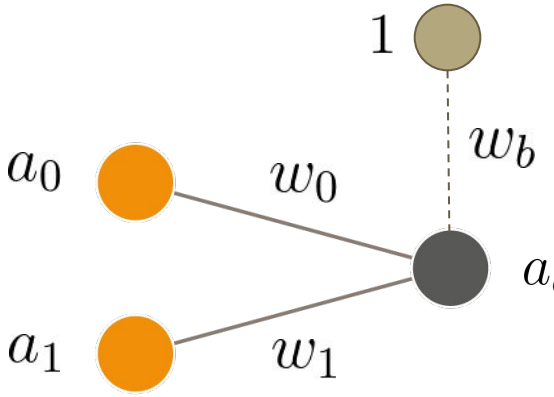


$$a_{out} = f(\sum_i w_i a_i)$$

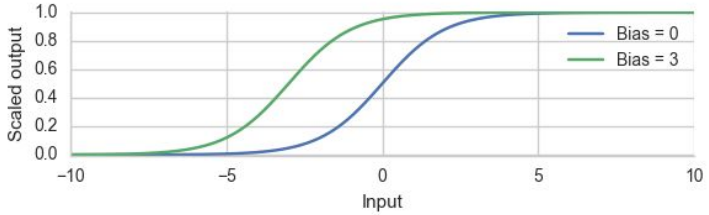


$$f(x) = \frac{1}{1 + e^{-x}}$$

Artificial Neural Networks (ANNs)

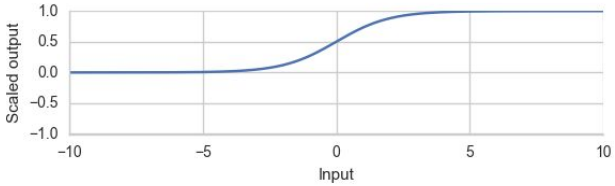


$$a_{out} = f(w_b + \sum_i w_i a_i)$$

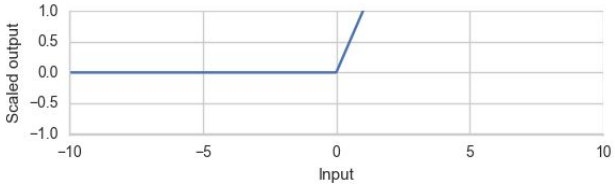


$$f(x) = \frac{1}{1 + e^{-x}}$$

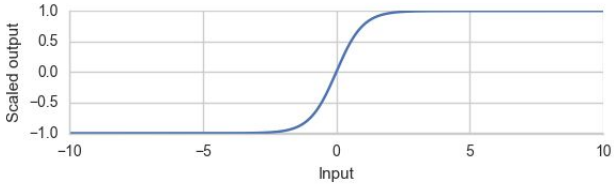
Activation function zoo



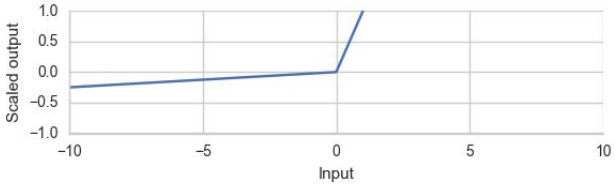
Sigmoid



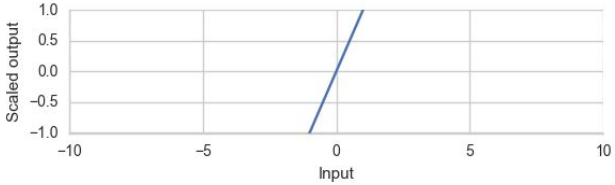
ReLU



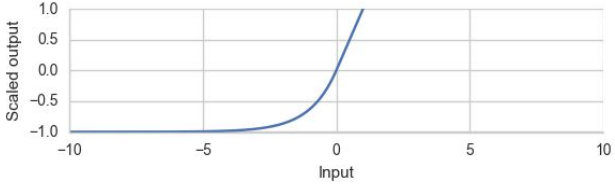
tanh



Leaky ReLU



linear



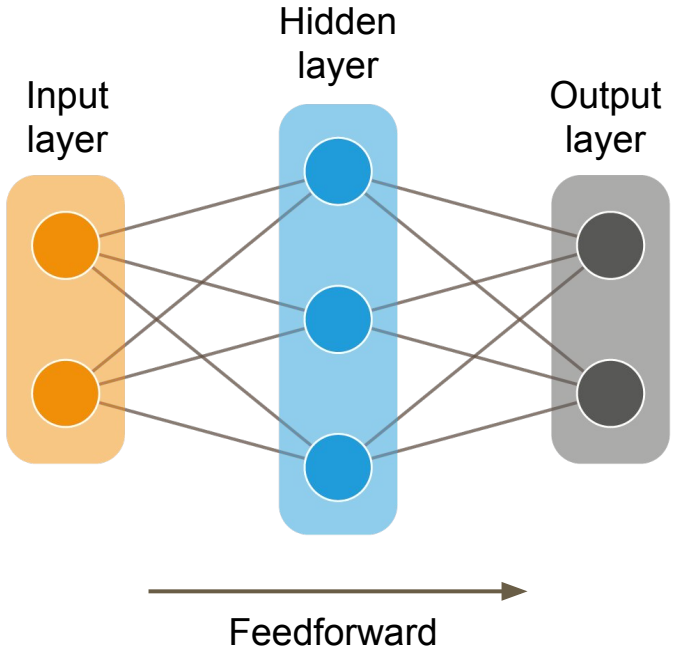
eLU

Multilayer perceptron (MLP)

Feedforward network of artificial neurons

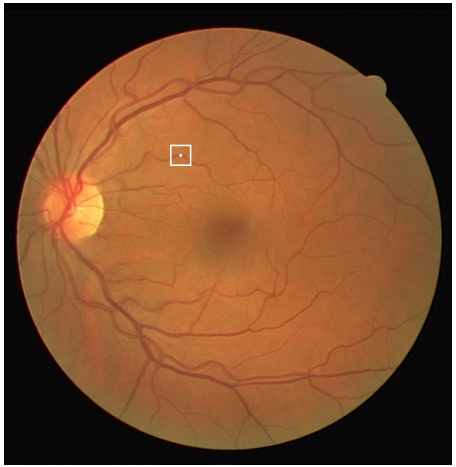
Three types of layers:

- Input layer
  - Contains features for sample
- Hidden layer
  - Combines features
- Output layer
  - Computes output probability for each class



‘Traditional’ machine learning approach

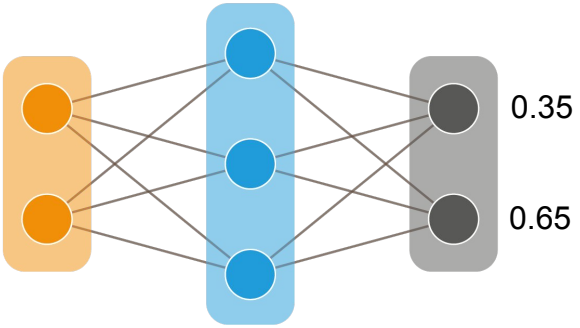
Pixel classification with features computed from the image patch



Input image

Vesselness
Structure Tensor
...

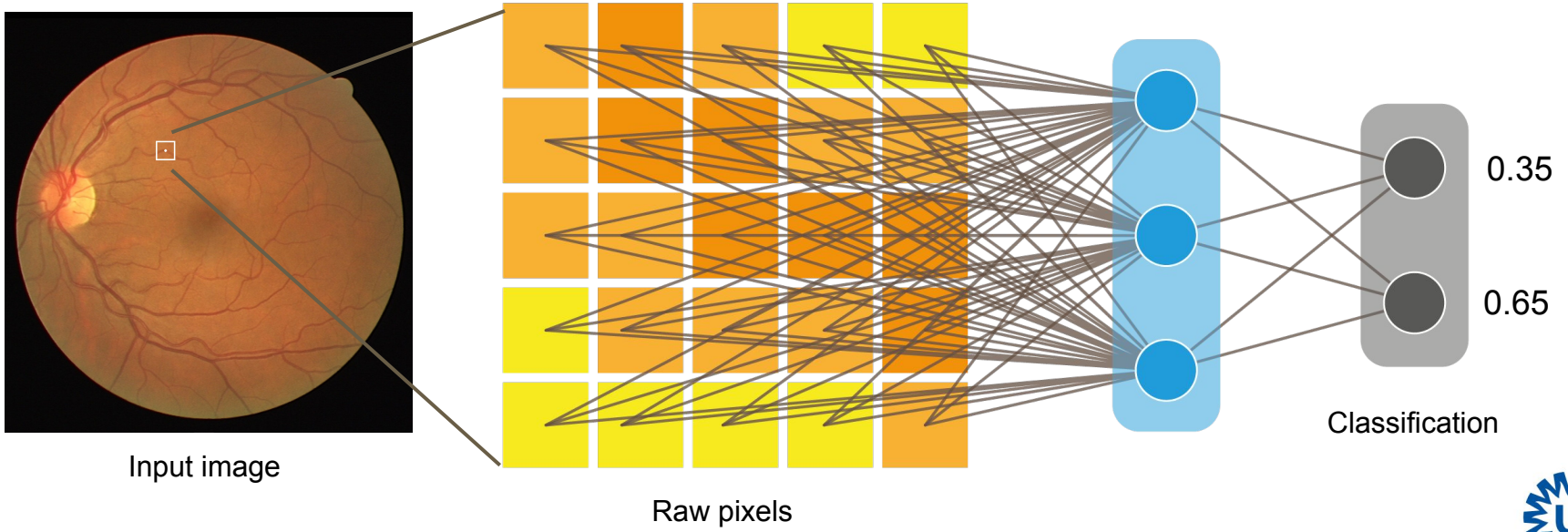
Feature engineering



Classification

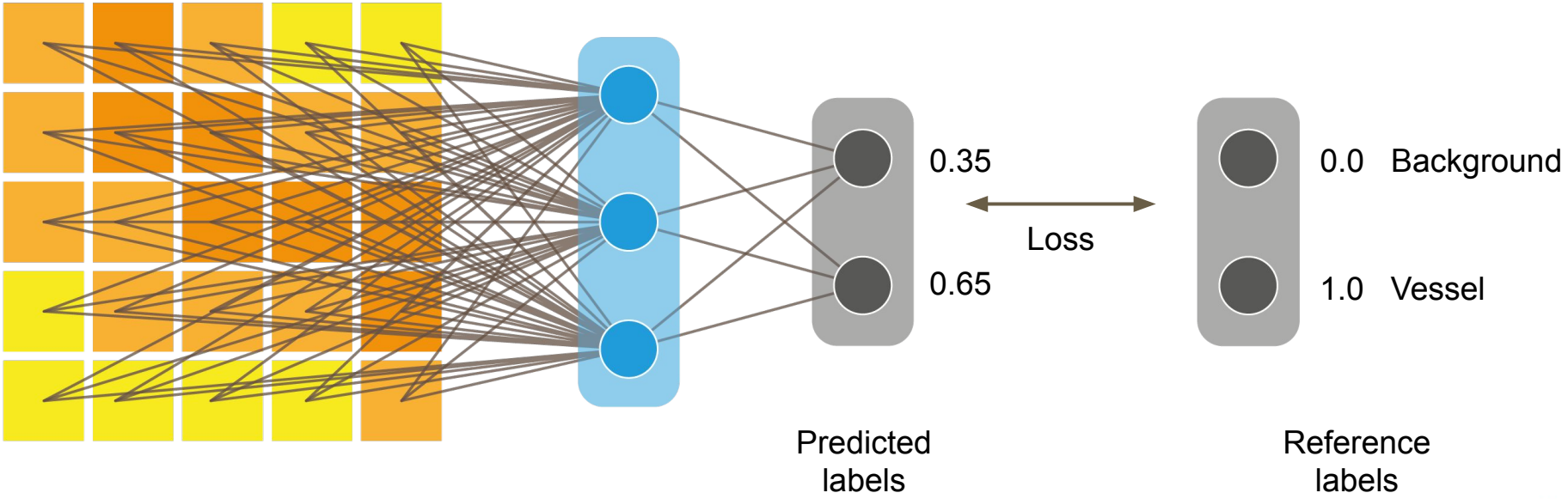
Deep learning approach

Classifier is trained to learn meaningful representations from the **raw pixel values**



How well does the network predict?

Determined by the loss/error between the predicted labels and reference labels



Loss functions

For sample *i*, given prediction p_i and target t_i, define loss L_i

For continuous output

- **Squared error**
  For regression tasks

$$L_i = (t_i - p_i)^2$$

Loss functions

For sample *i*, given prediction p_i and target t_i, define loss L_i

For continuous output

- **Squared error**
  For regression tasks

$$L_i = (t_i - p_i)^2$$

For categorical output

- **Accuracy**
  Used only for evaluation

$$L_i = \mathbb{I}(t_i = \operatorname{argmax}_j p_{i,j})$$

Loss functions

For sample *i*, given prediction p_i and target t_i, define loss L_i

For continuous output

- **Squared error**
  For regression tasks

$$L_i = (t_i - p_i)^2$$

For categorical output

- **Accuracy**
  Used only for evaluation

$$L_i = \mathbb{I}(t_i = \operatorname{argmax}_j p_{i,j})$$

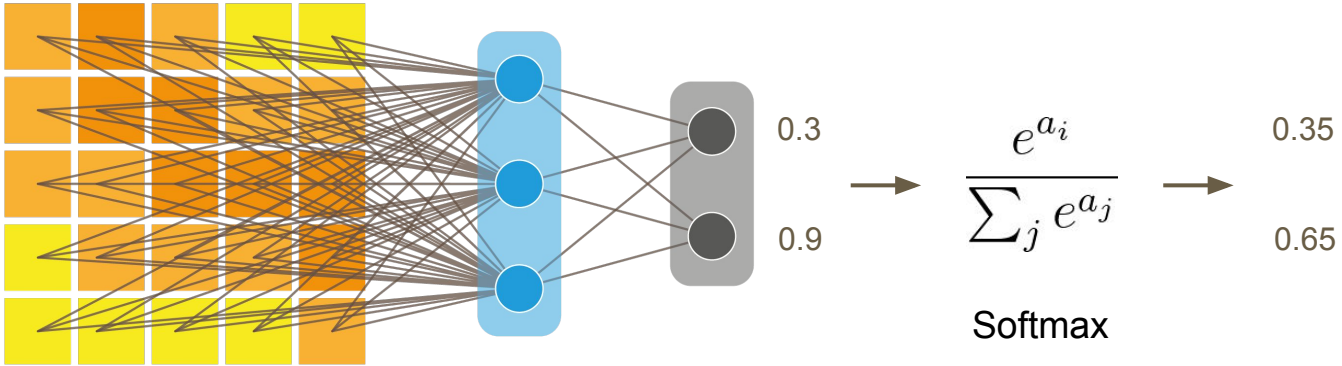
- **Cross-entropy**
  Used during training

$$L_i = -\sum_j t_{i,j} \log p_{i,j}$$

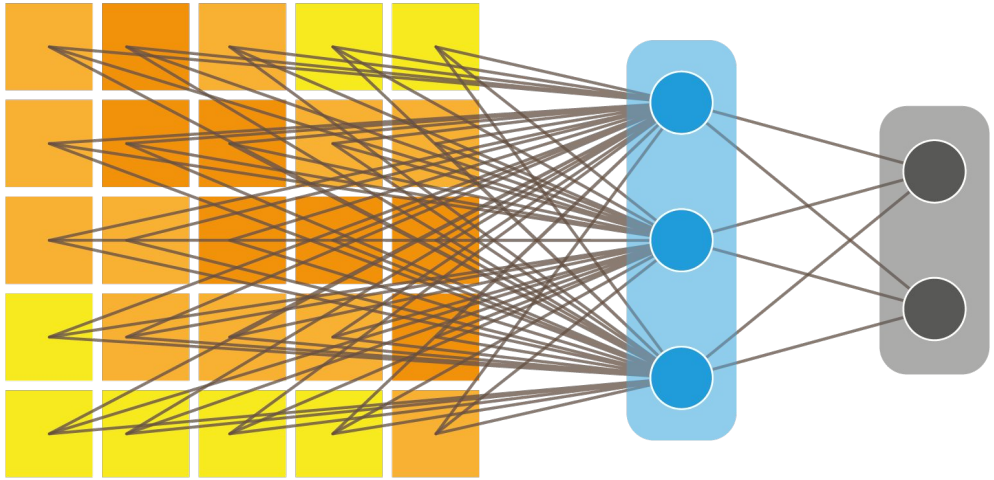
Softmax function

The cross-entropy loss requires that probabilities at the output layer should sum to 1

The softmax function squashes the outputs to real values between 0 and 1 with sum 1

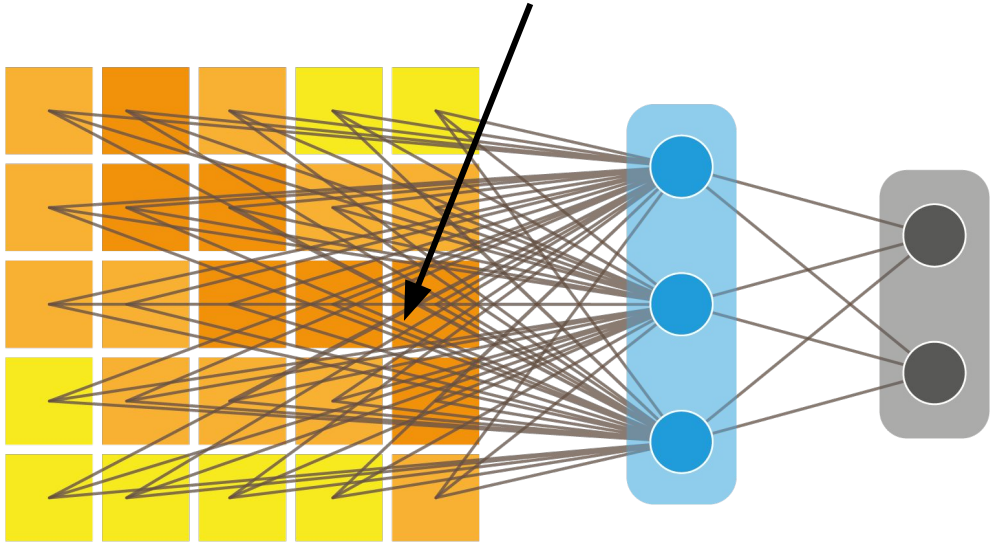


Training a neural network

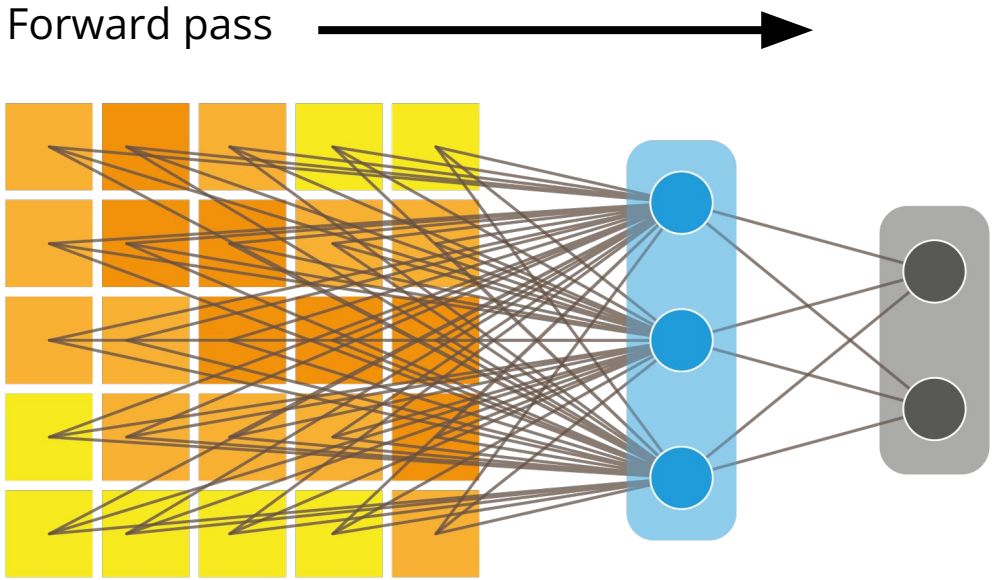


Training a neural network

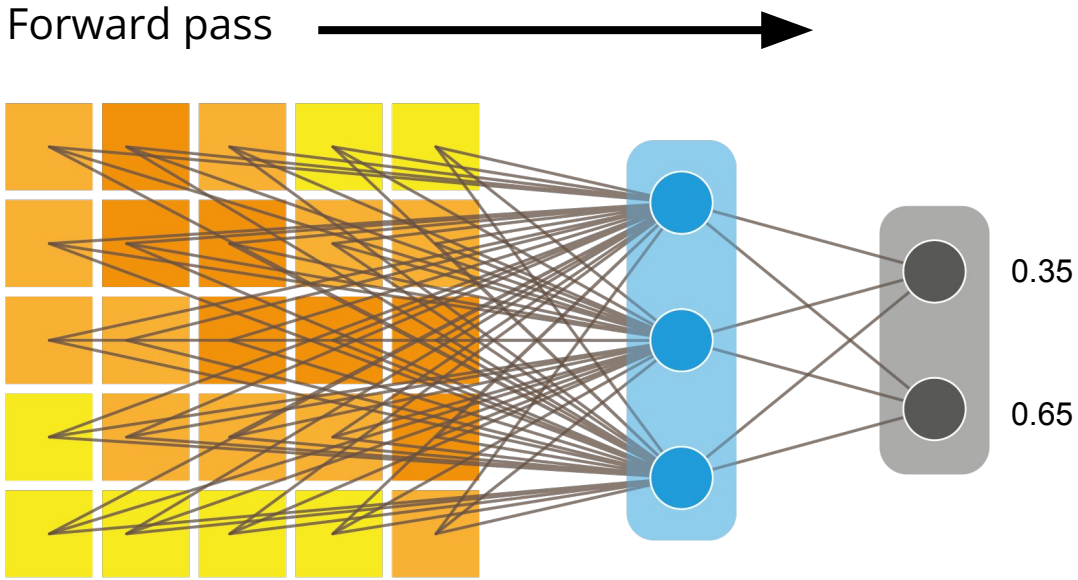
How much should we change this weight to get a better prediction (smaller loss)?



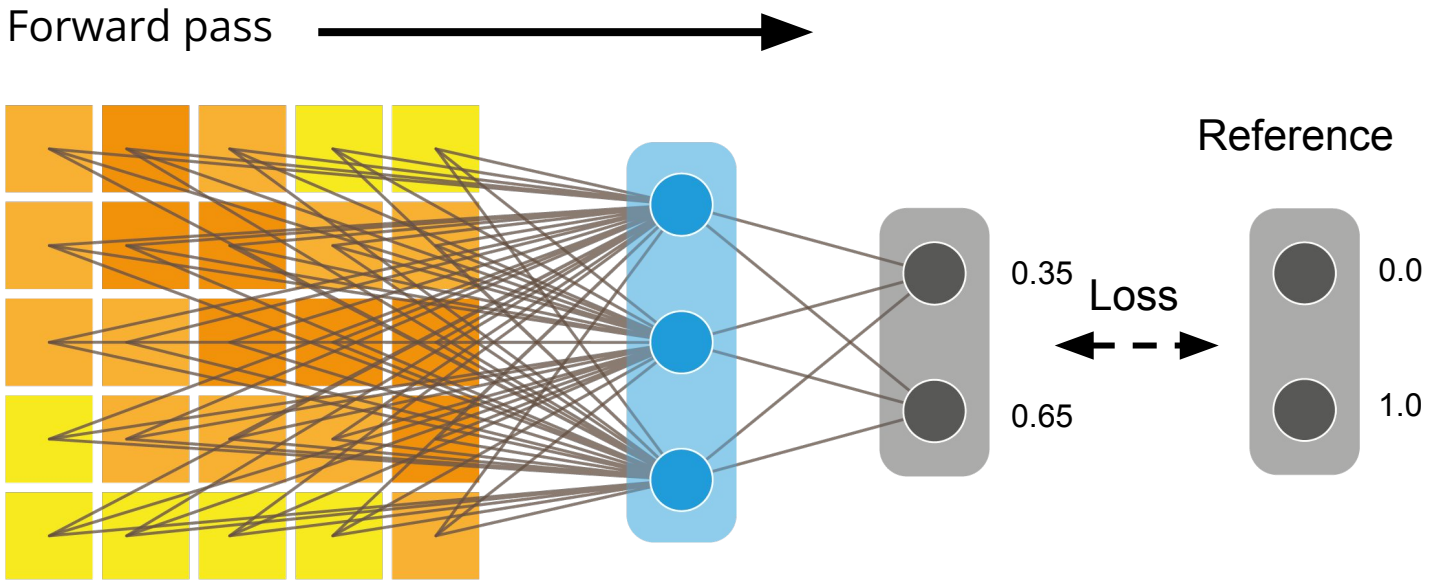
Training a neural network



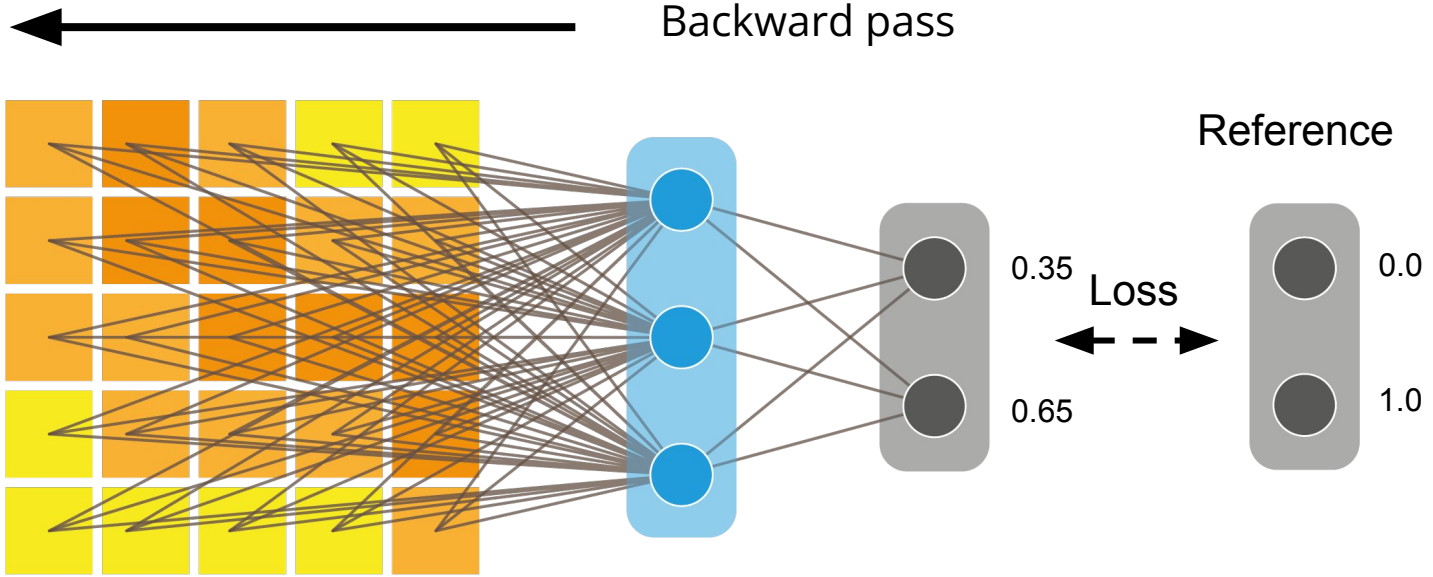
Training a neural network



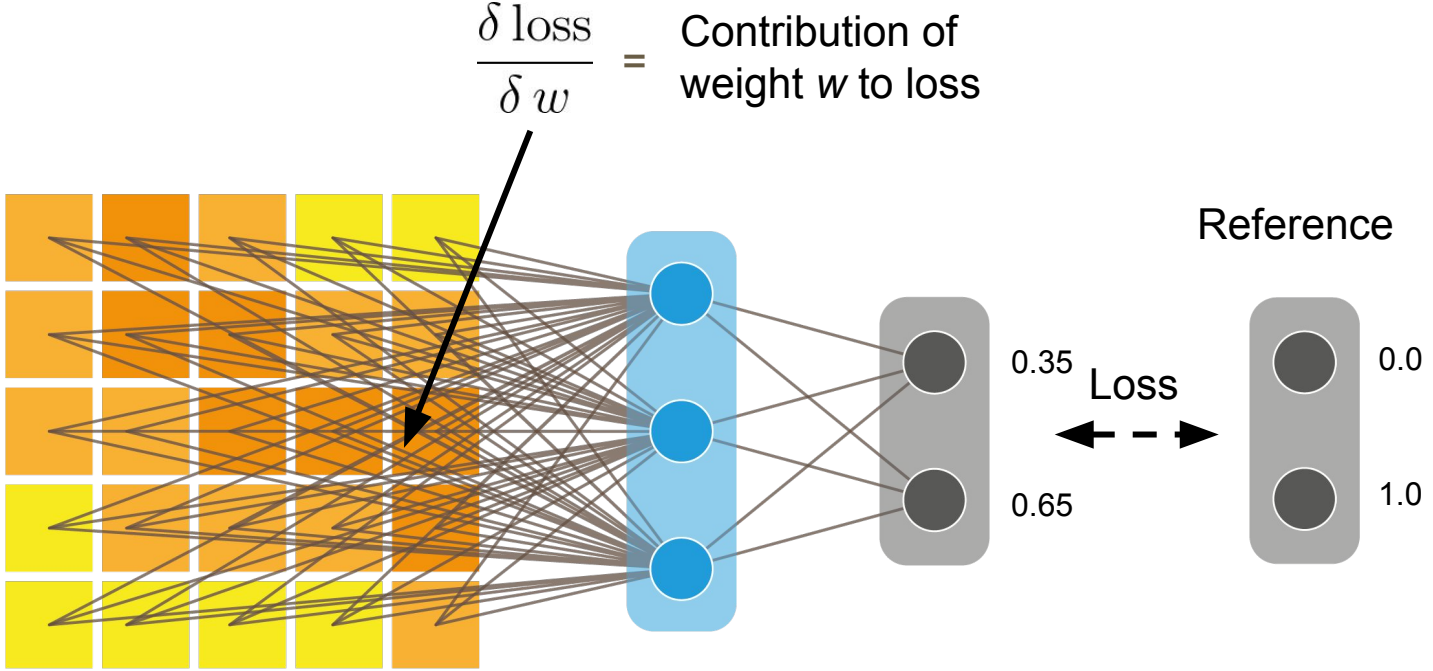
Training a neural network



Training a neural network

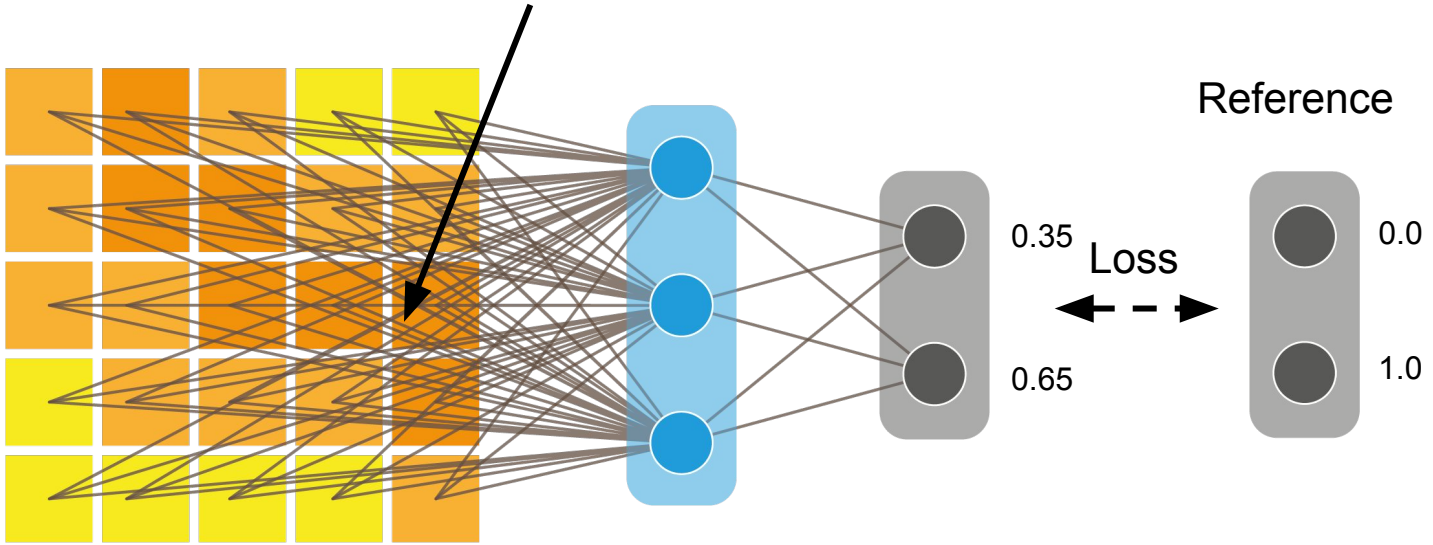


Training a neural network

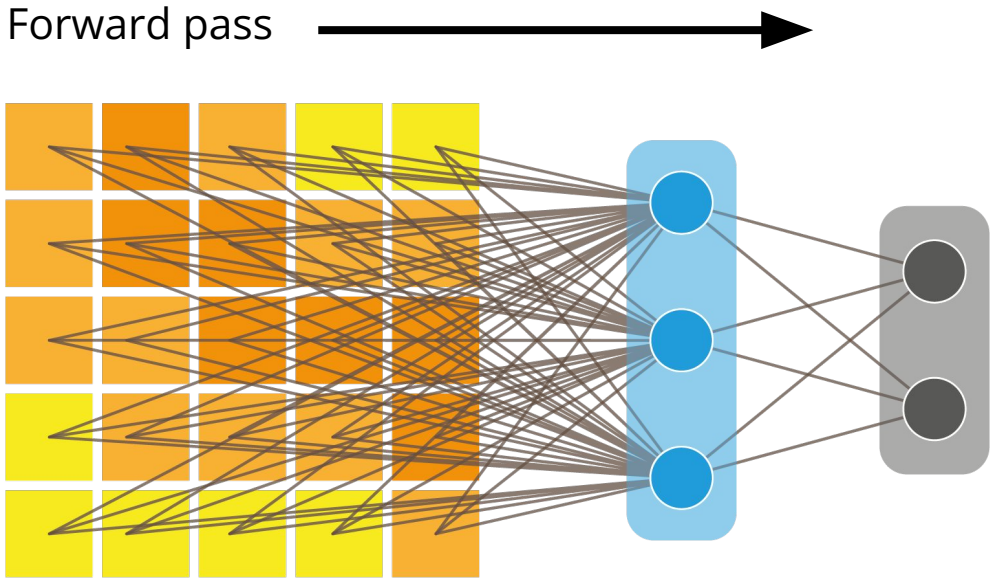


Training a neural network

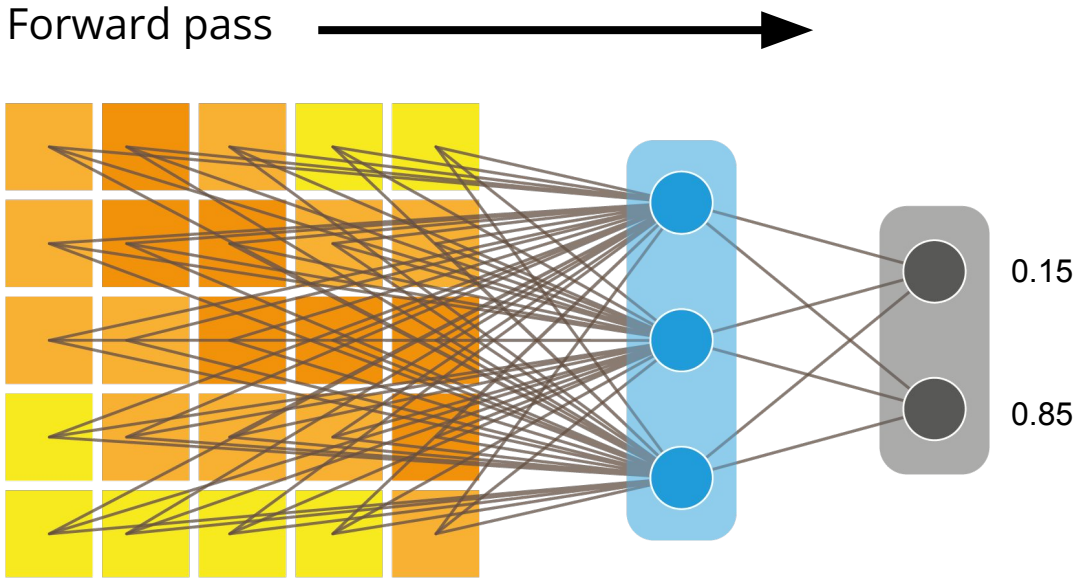
Update rule: $w_{\text{new}} = w_{\text{old}} - \text{LR} \cdot \frac{\delta \, \text{loss}}{\delta \, w_{\text{old}}}$



Training a neural network



Training a neural network



Gradient Descent

Gradient descent is a smart way to move through “weight space”

- All data → Infeasible
- Single sample → Jumpy

Gradient Descent

Gradient descent is a smart way to move through “weight space”

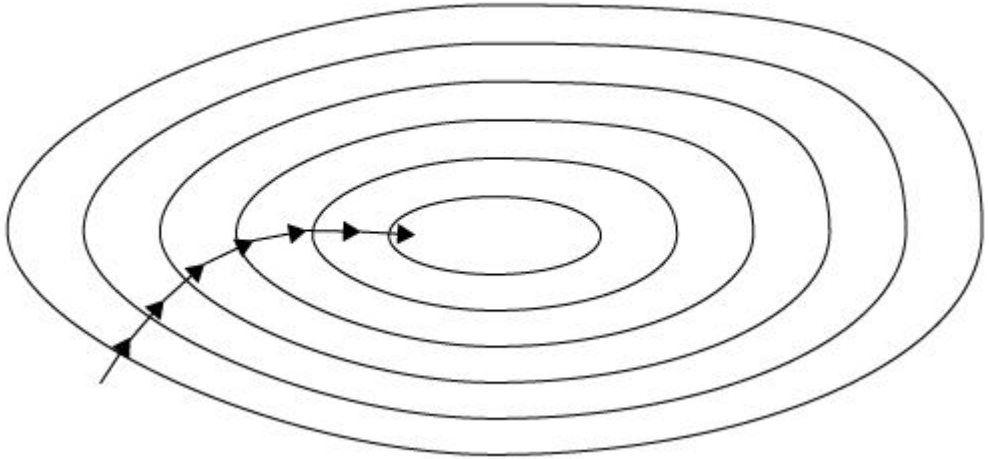
- All data → Infeasible
- Single sample → Jumpy

Minibatch stochastic gradient descent (SGD)

- Compute gradients on a small number of data samples (**minibatch**)
- Update the weights with these imperfect gradients a tiny bit
- Many small updates with good-but-not-perfect gradients move the weights overall in the right direction

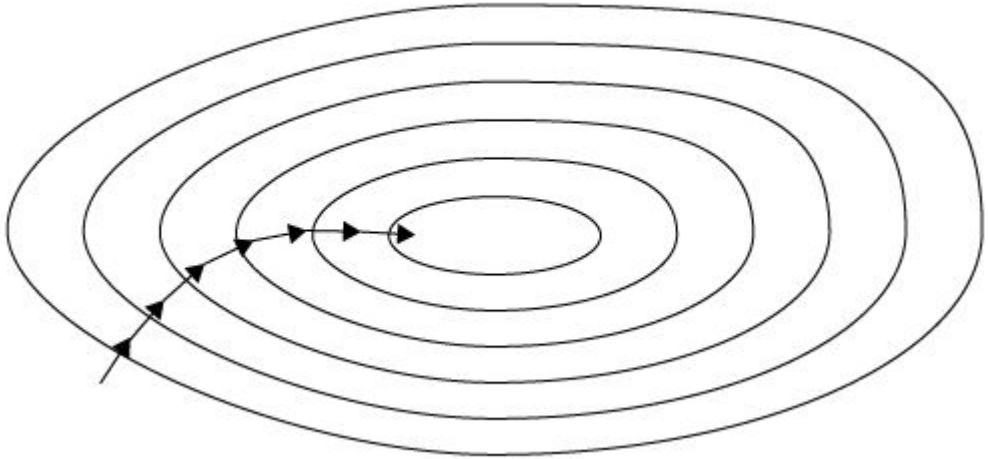
Stochastic Gradient Descent (SGD)

Small steps through “weight space”

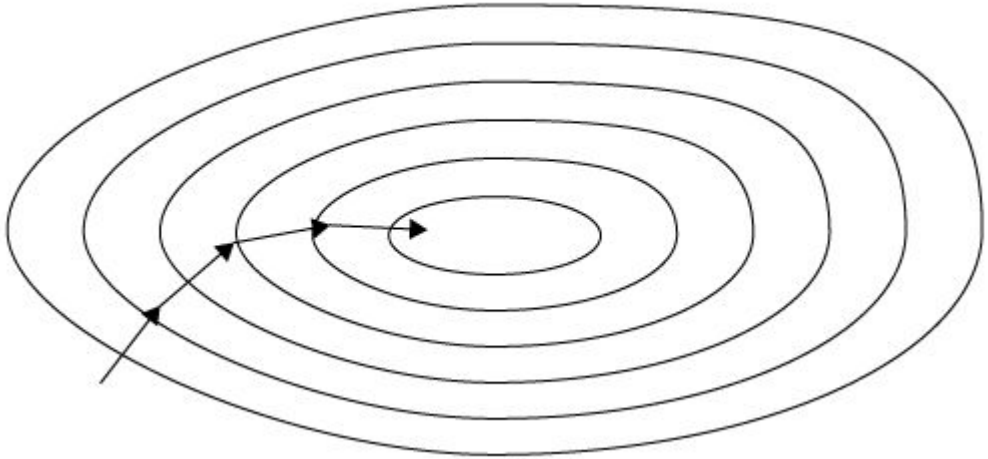


Stochastic Gradient Descent (SGD)

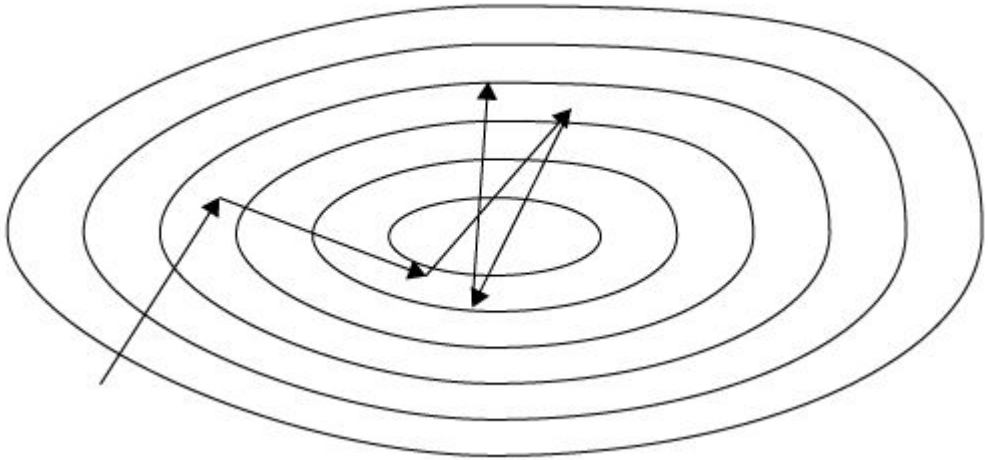
Step size is determined by the learning rate (LR)



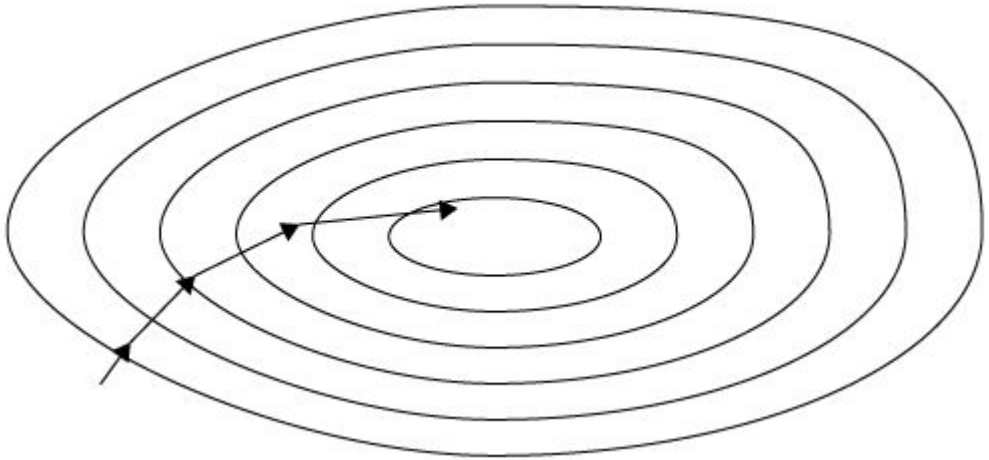
SGD - Increased learning rate



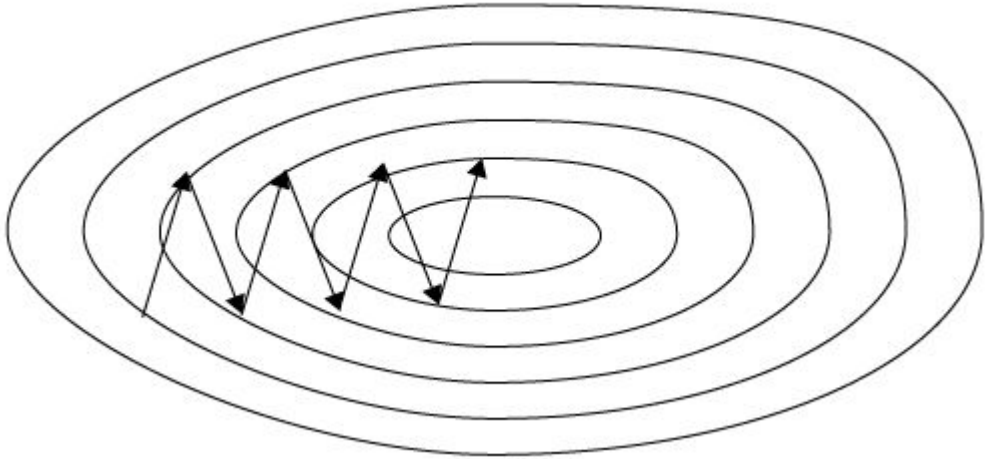
SGD - High learning rate



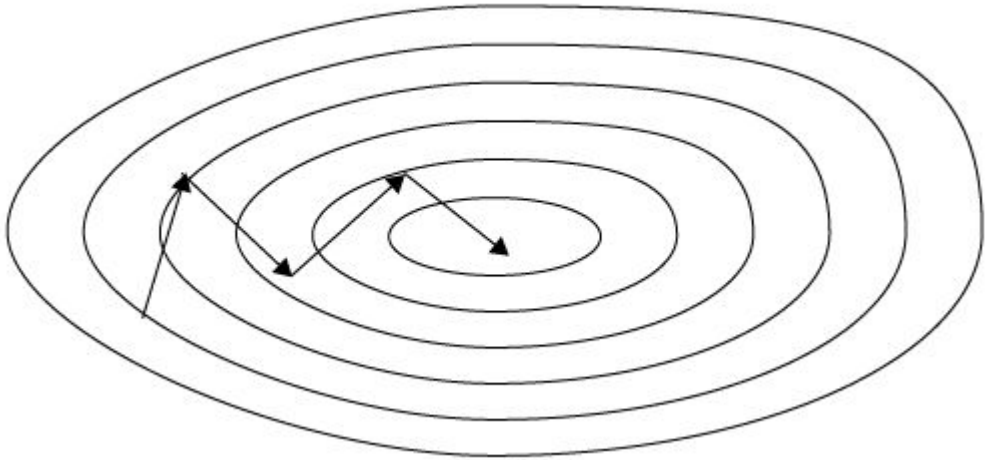
SGD - Momentum



SGD - Oscillations without momentum



SGD - Oscillations with momentum



Adaptive learning rates

Intuitively, the learning rate should decay while approaching the minimum (move faster when far away, move slower when close)

T. Tieleman and G. Hinton, Lecture 6.5 - rmsprop, COURSERA: Neural Networks for Machine Learning (2012)
J. Duchi et al., *Journal of Machine Learning Research* 12,:2121–2159 2011
M.D. Zeiler, arXiv:1212.5701 [cs.LG]

Adaptive learning rates

Intuitively, the learning rate should decay while approaching the minimum
(move faster when far away, move slower when close)

Adaptive learning rates per weight by keeping track of the gradients magnitude for every weight

T. Tieleman and G. Hinton, Lecture 6.5 - rmsprop, COURSERA: Neural Networks for Machine Learning (2012)
J. Duchi et al., *Journal of Machine Learning Research* 12,:2121–2159 2011
M.D. Zeiler, arXiv:1212.5701 [cs.LG]

Adaptive learning rates

Intuitively, the learning rate should decay while approaching the minimum
(move faster when far away, move slower when close)

Adaptive learning rates per weight by keeping track of the gradients magnitude for every weight

- **RMSprop** and **AdaDelta** keep a running average of the gradient magnitude per parameter.
- **AdaGrad** is similar, but has a monotonically decreasing learning rate.

T. Tieleman and G. Hinton, Lecture 6.5 - rmsprop, COURSERA: Neural Networks for Machine Learning (2012)
J. Duchi et al., *Journal of Machine Learning Research* 12,:2121–2159 2011
M.D. Zeiler, arXiv:1212.5701 [cs.LG]

Adam

Momentum stabilizes the **direction** of the updates

Adaptive algorithms (RMSprop) stabilize the **magnitude** of the updates

Adam

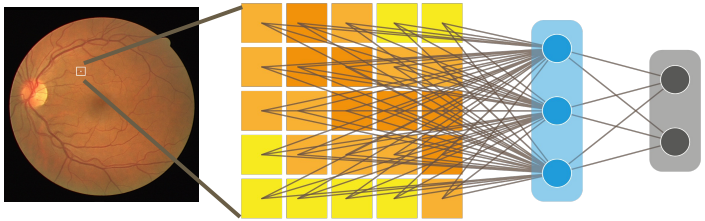
Momentum stabilizes the **direction** of the updates

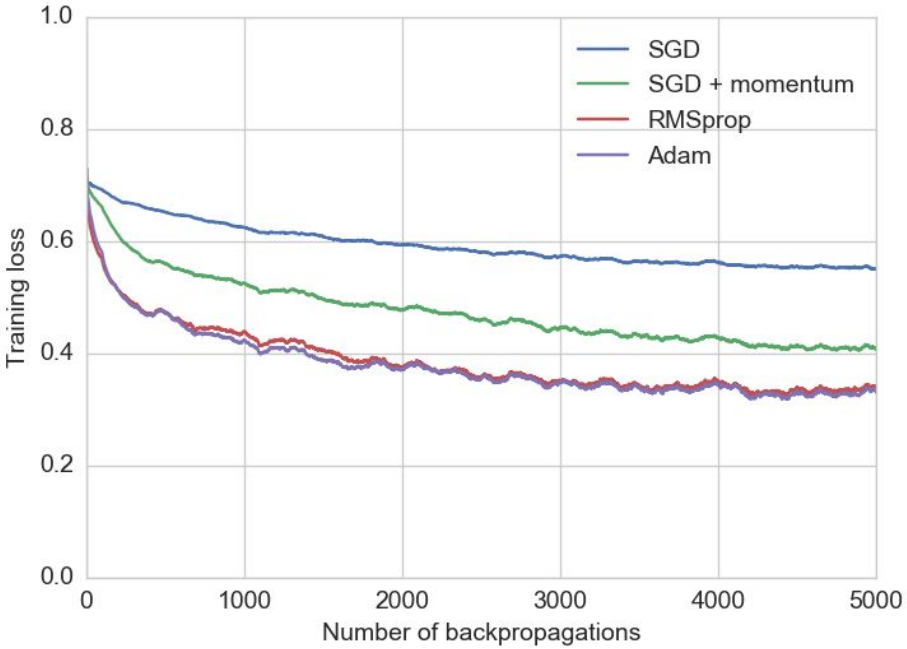
Adaptive algorithms (RMSprop) stabilize the **magnitude** of the updates

Adam stabilizes both, it is like RMSprop with Momentum

Currently a good default choice

Comparing optimizers





Summary

We have explained building blocks of neural networks:

- Units, weights, and activation functions
- Multilayer perceptrons
- Training with a differentiable loss
- Stochastic Gradient Descent (SGD)
- SGD variants

But how can we apply these building blocks to larger images?

Program

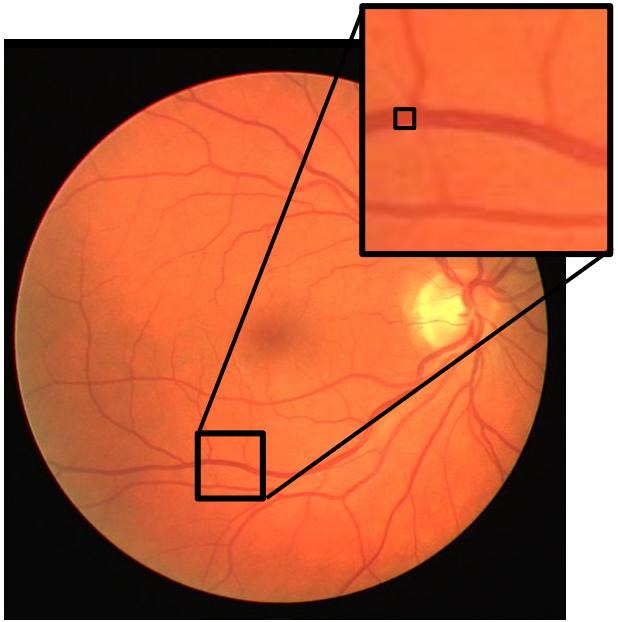
1.	Introduction	Ivana Išgum
2.	Neural networks	Bob de Vos
3.	**Convolutional neural networks**	**Nikolas Lessmann**
4.	Generalization	Jelmer Wolterink
5.	Getting started	Akshay Pai
6.	General summary & discussion	Mads Nielsen

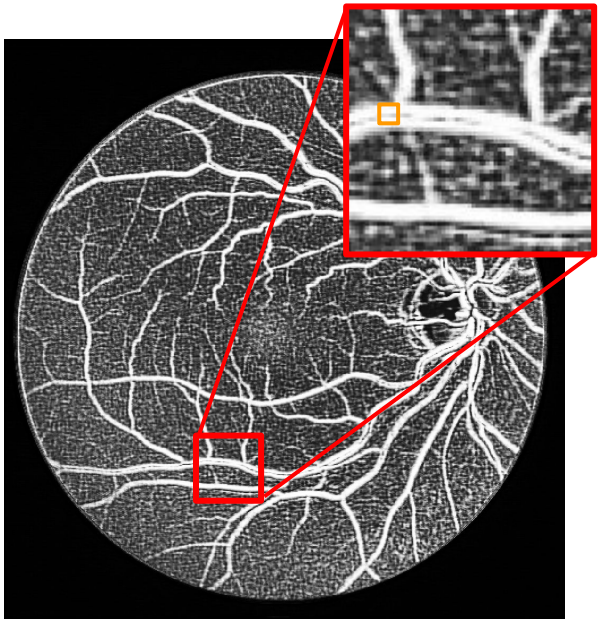
3. CONVOLUTIONAL NEURAL NETWORKS

Nikolas Lessmann

Input patch size

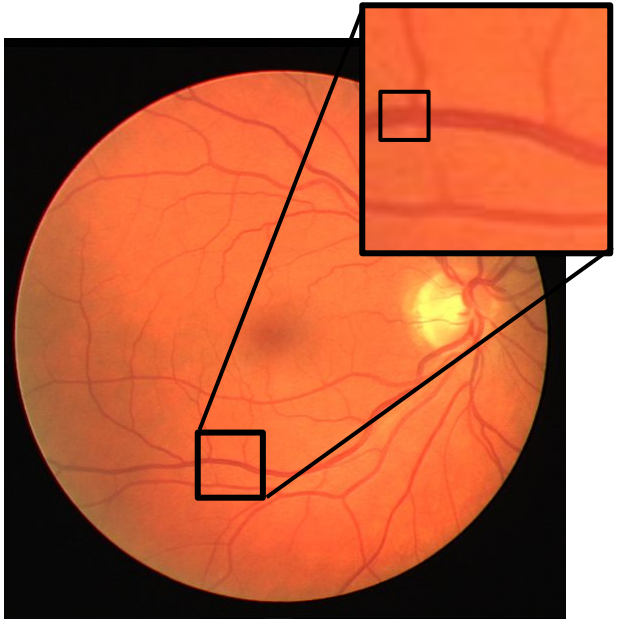
5x5 patches : decisions are based on limited local information

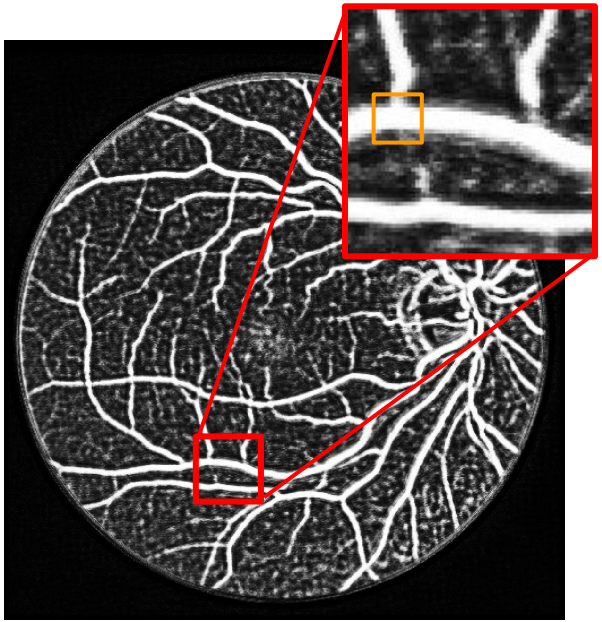




Input patch size

By increasing the patch size to 11x11 pixels, more information from the image is used

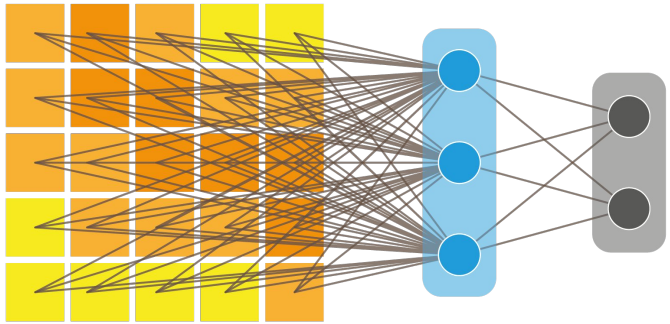




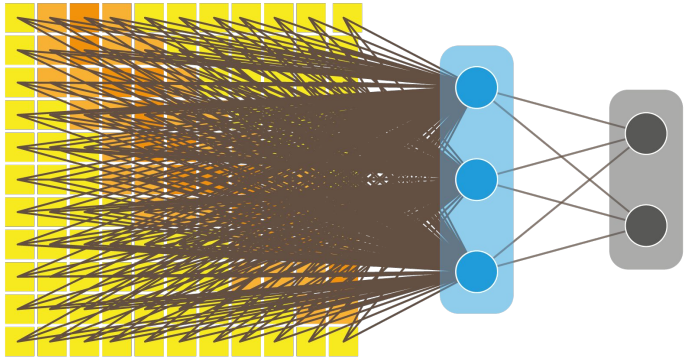
Large input patches are problematic

Using each voxel as an input feature rapidly increases the number of weights

More training data required to generalize well



5x5 input patch

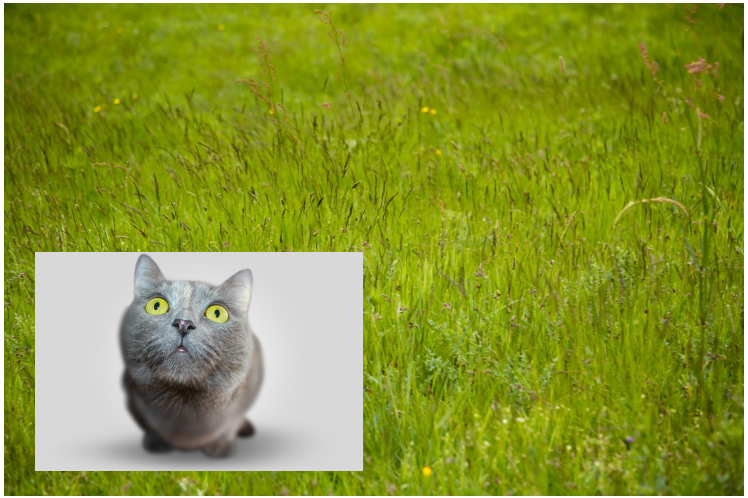


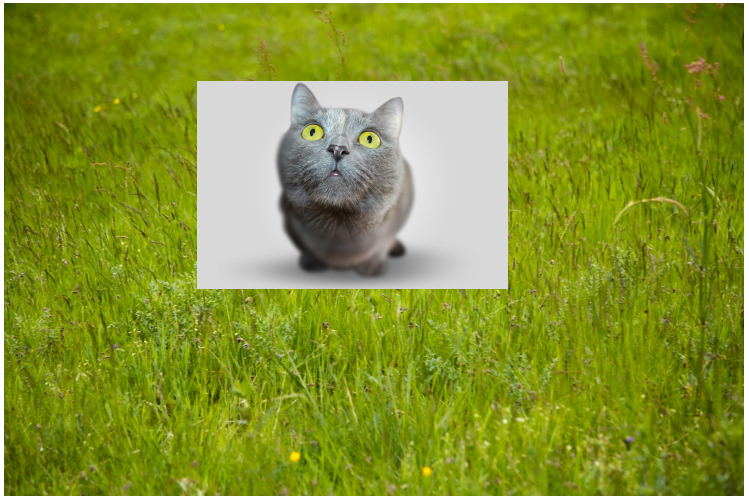
11x11 input patch

Spatial invariance

In many tasks, structures and objects can appear at various places in the image

⇒ very difficult to learn for MLPs





From low-level to high-level features

Similar low-level features can be present at different places in the image

MLPs receive all information at once, but it would be smarter to first combine spatially close information

Image values

↓

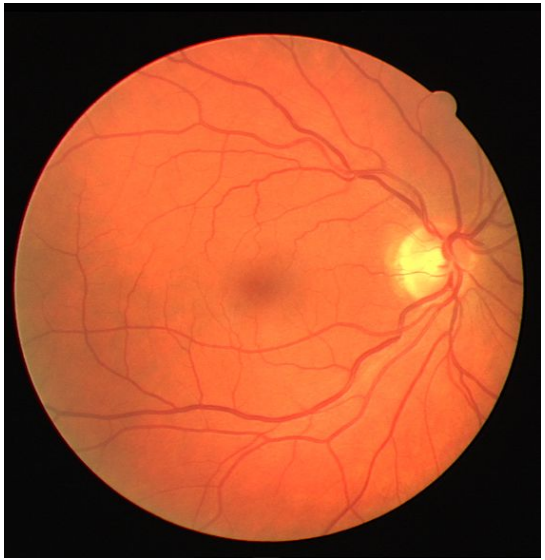
Gradients

↓

Simple shapes

↓

...

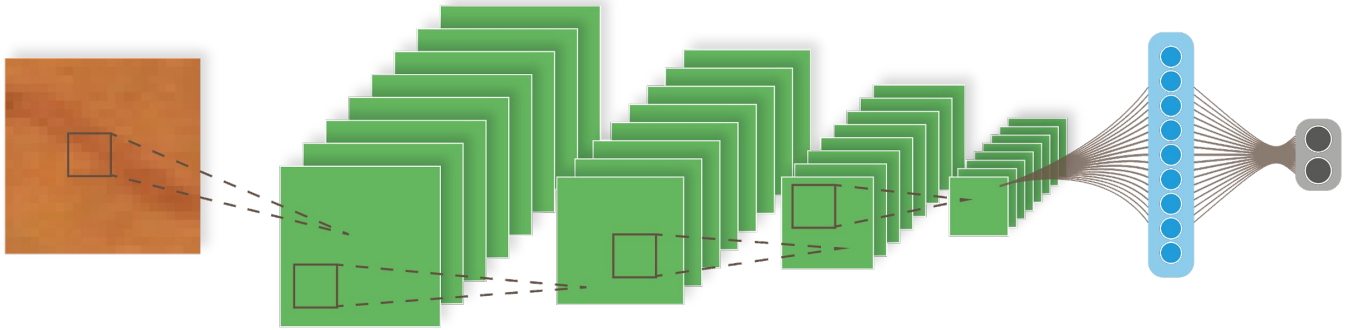


Convolutional neural networks

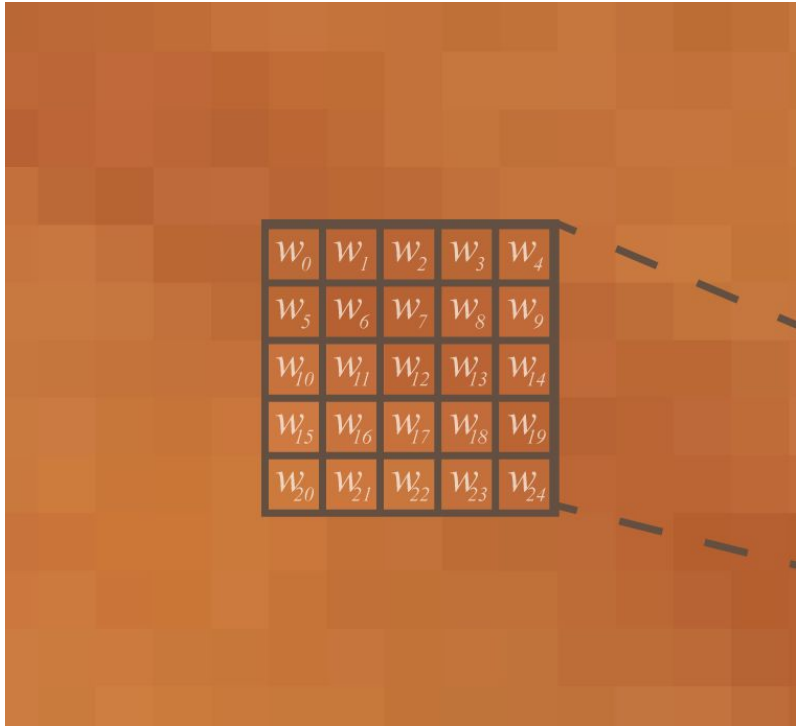
A convolutional layer convolves an image with multiple smaller kernels

The kernels are not predefined, but learned

Stacking convolutional layers leads to more complex features



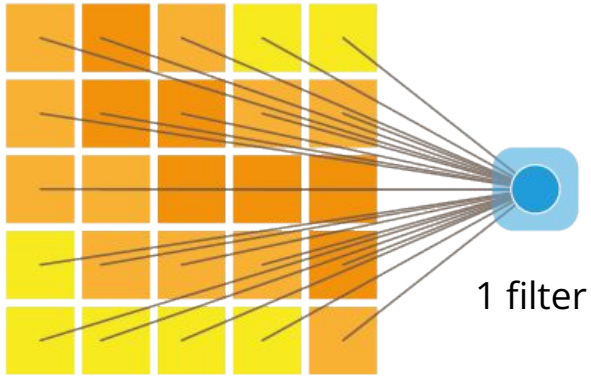
Convolutional neural networks



Similar to the multilayer perceptron (MLP)

Values in the filter kernel = weights

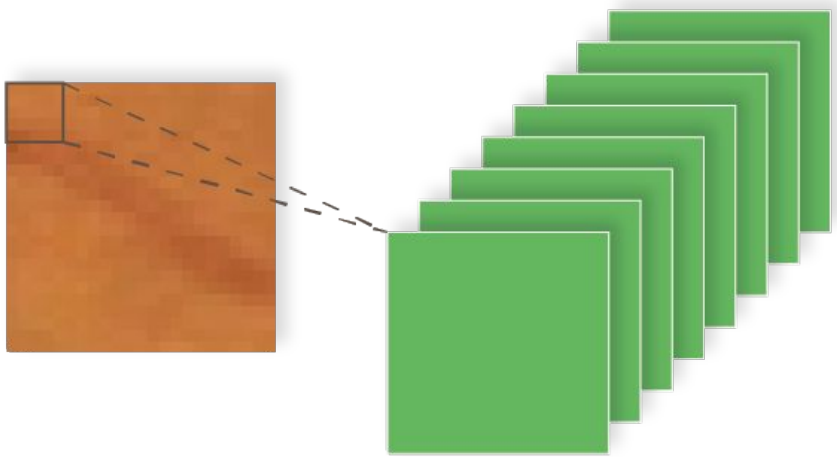
Few weights due to local receptive field



Convolutional neural networks

The filter kernel moves over the image → feature map

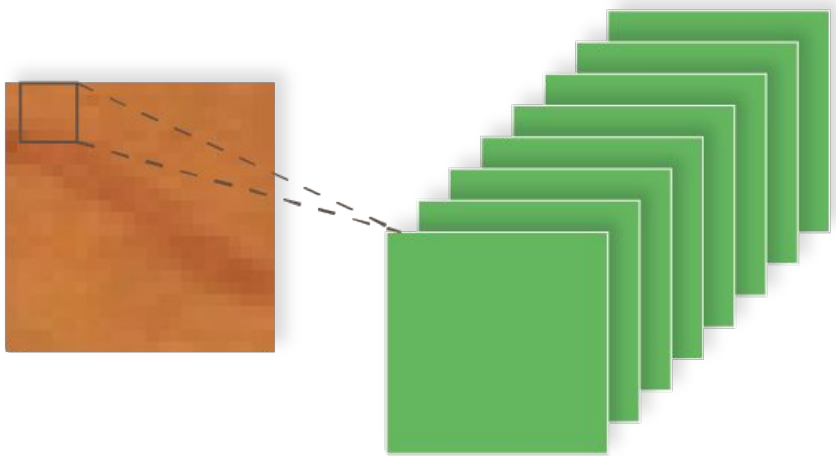
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

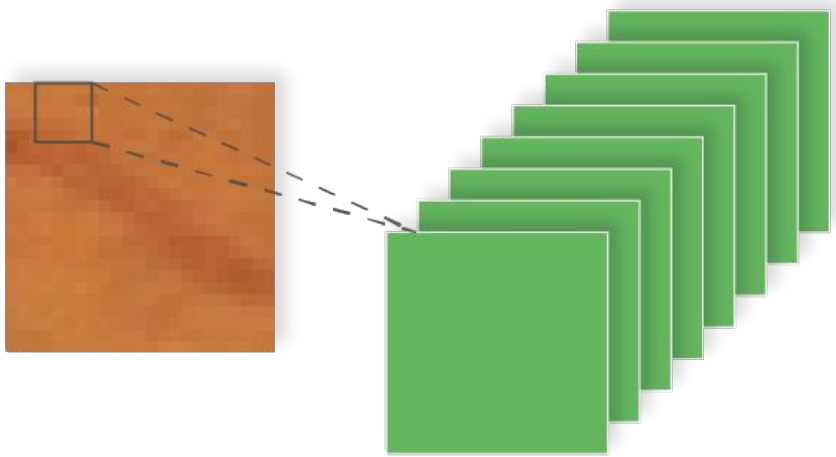
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

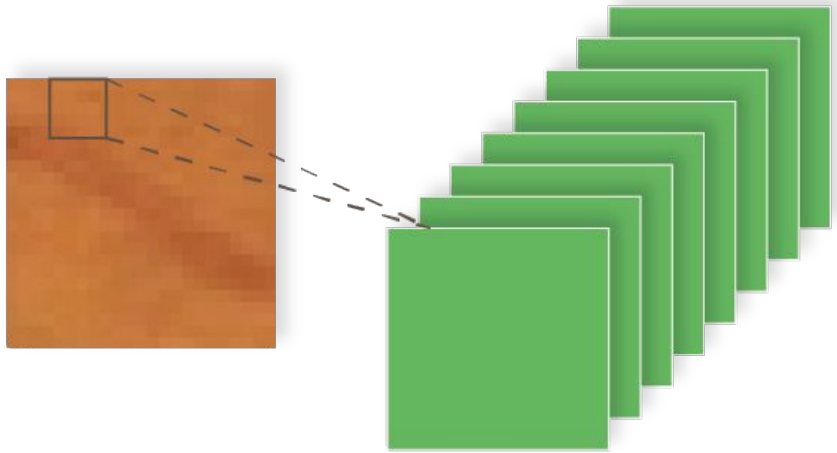
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

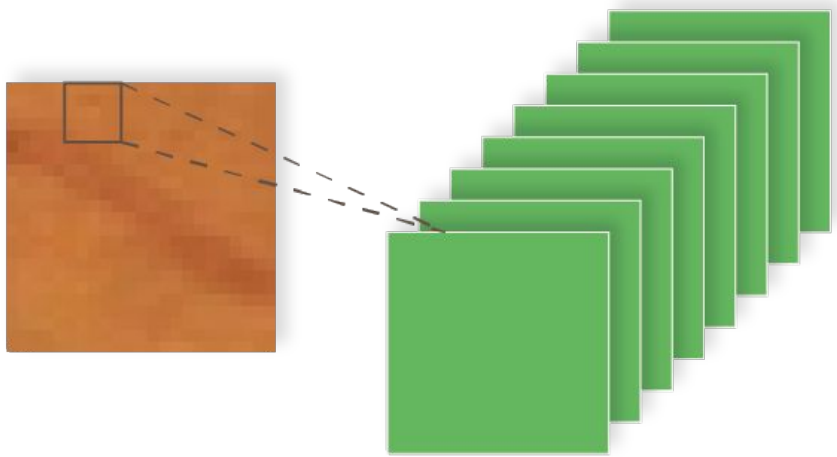
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

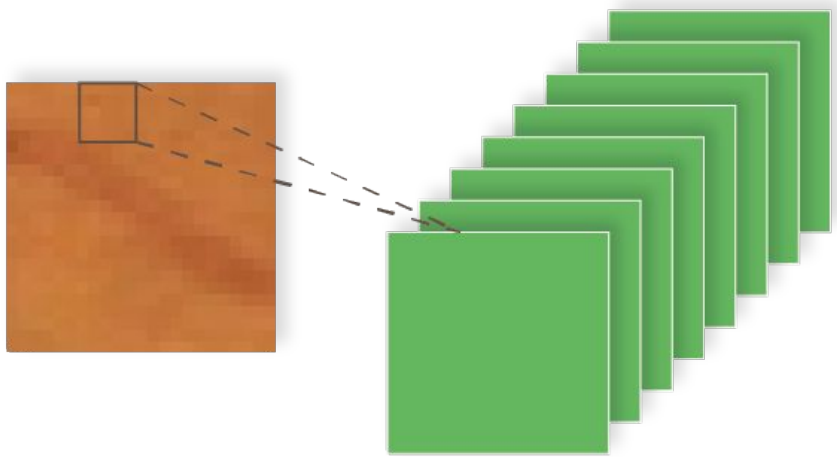
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

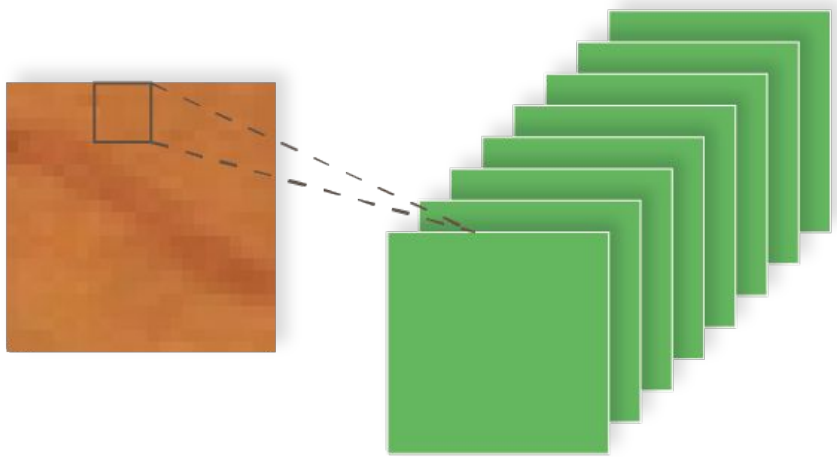
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

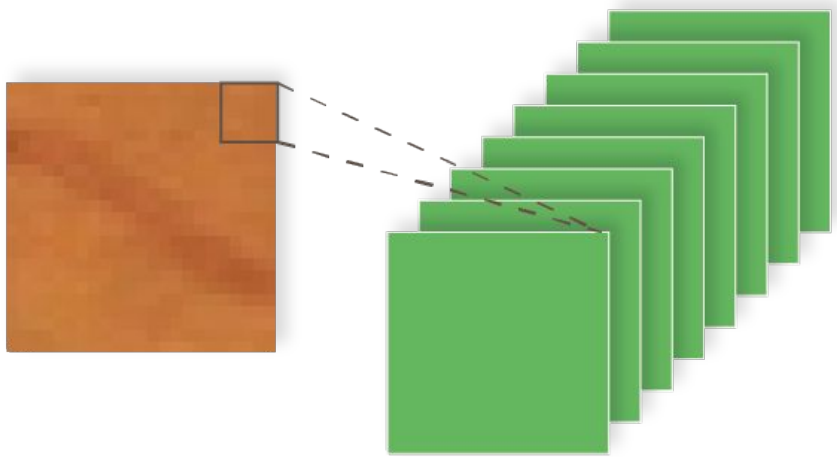
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

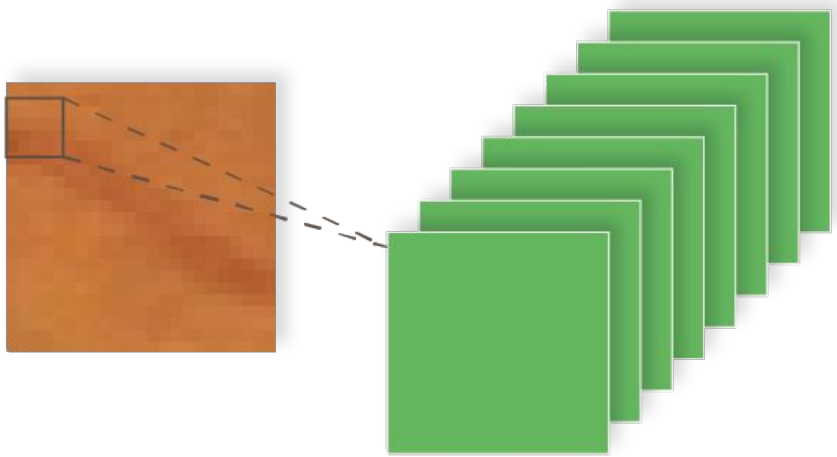
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

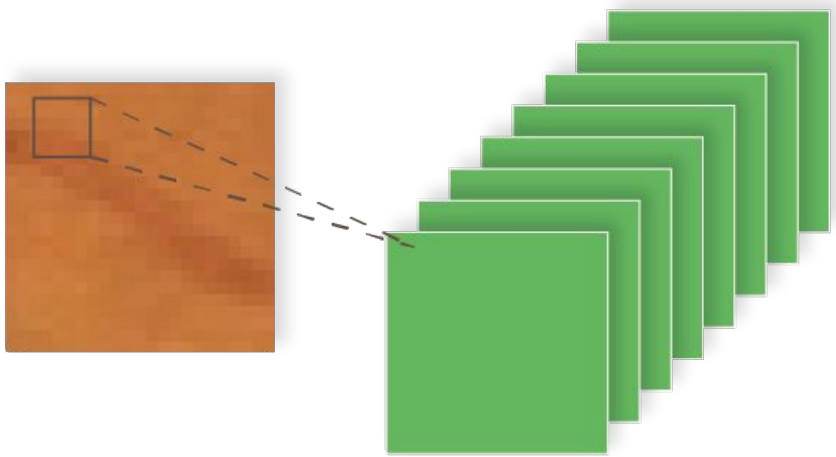
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

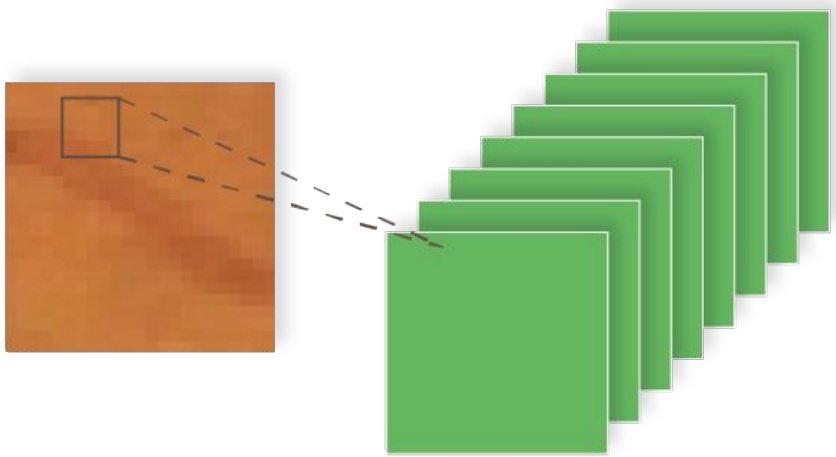
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

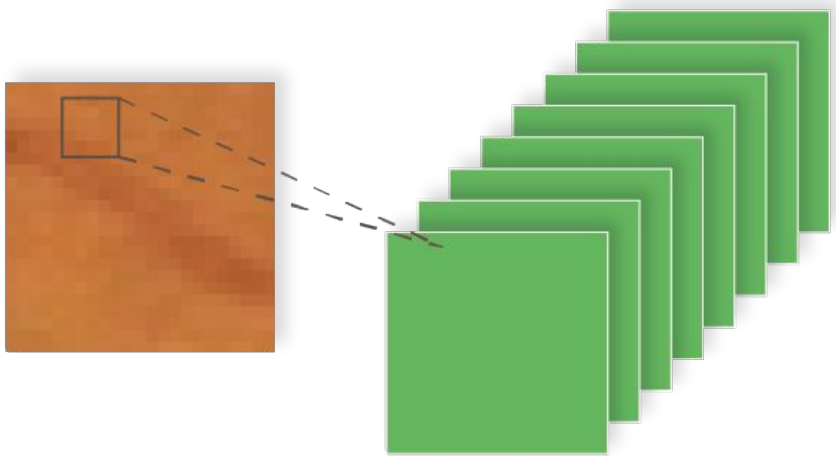
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

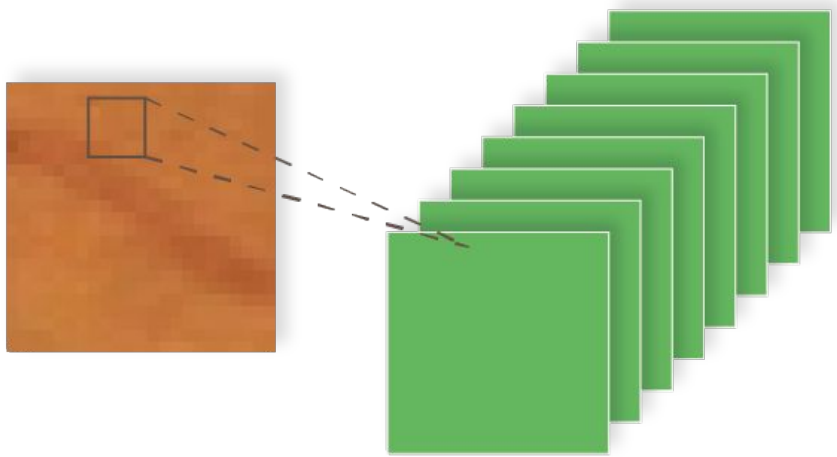
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

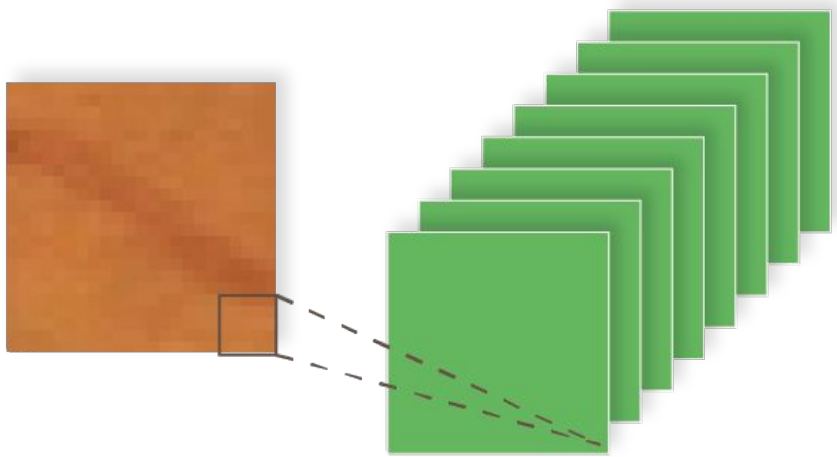
The step size is called **stride** (usually stride = 1)



Convolutional neural networks

The filter kernel moves over the image → feature map

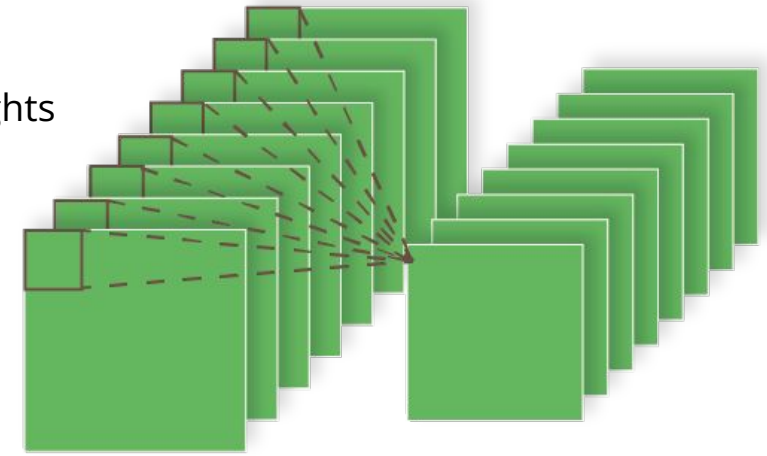
The step size is called **stride** (usually stride = 1)



Convolutions across multiple channels

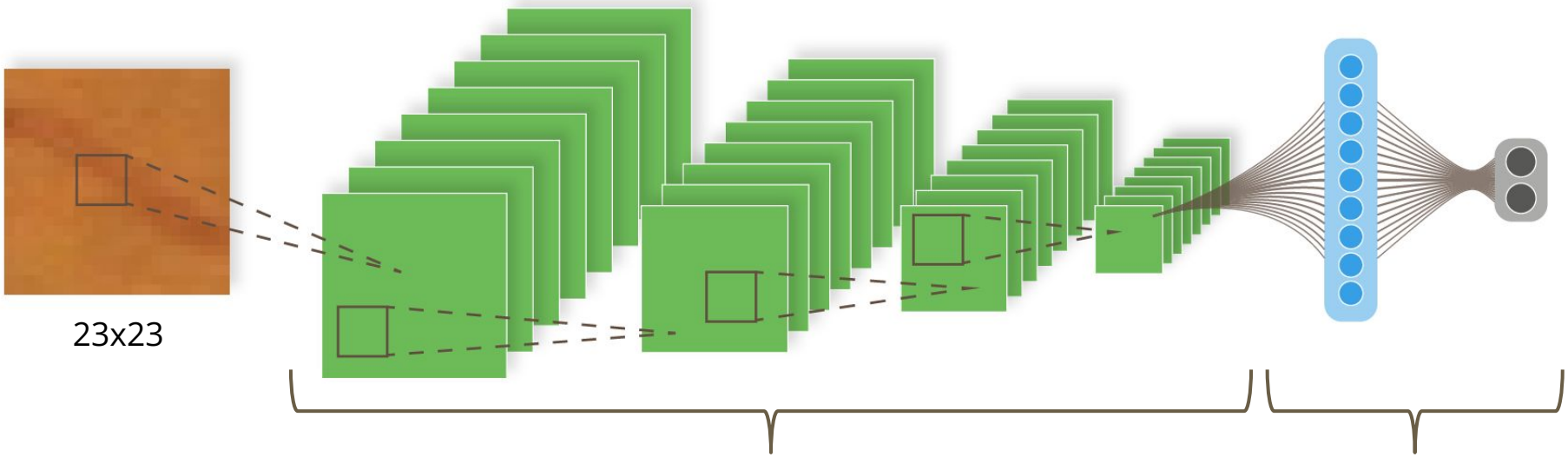
Input channels (features, RGB, ...) are combined

8 x 5x5 weights

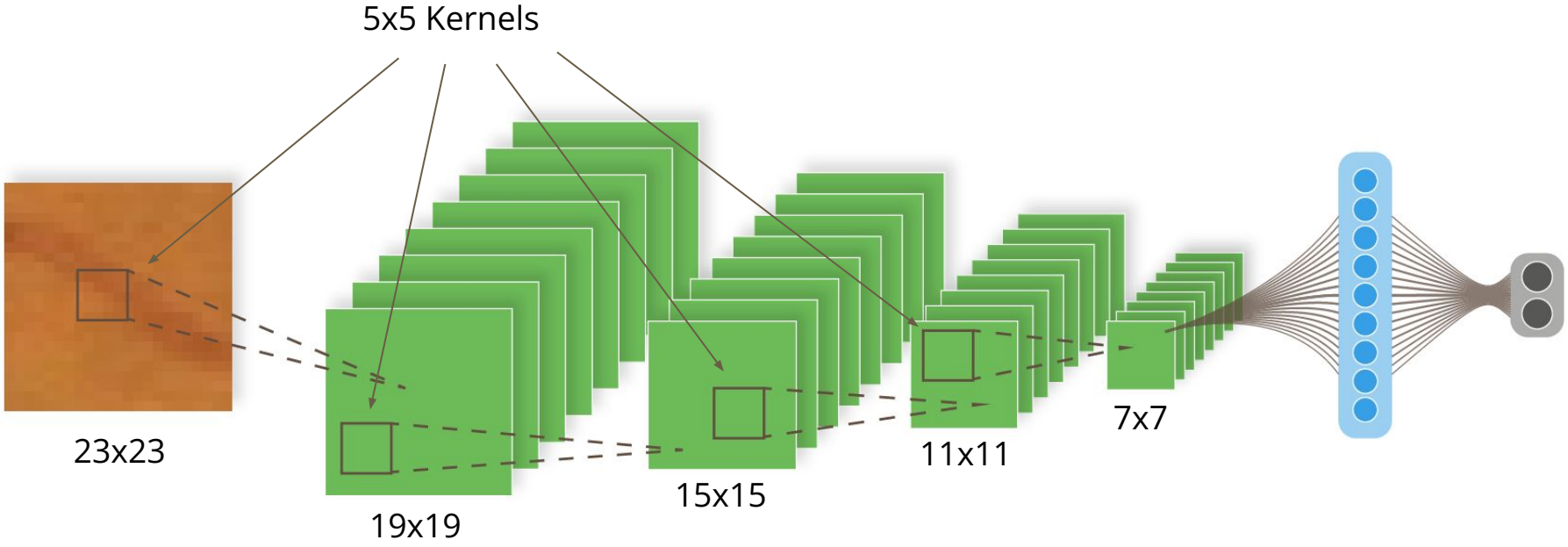


N input channels M output channels

Combining convolutions and MLPs

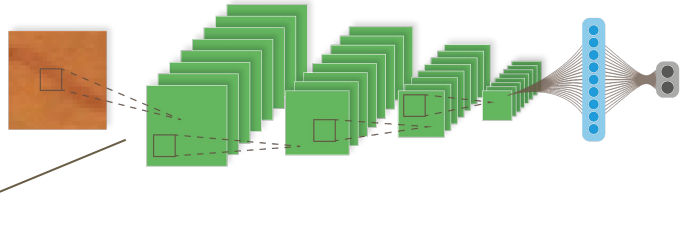


Valid convolutions / border handling

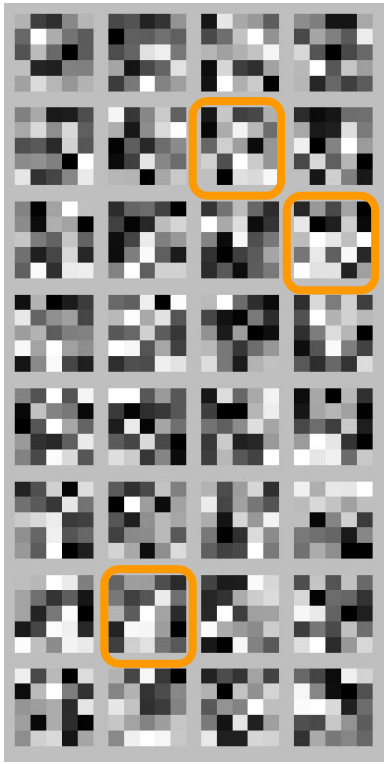


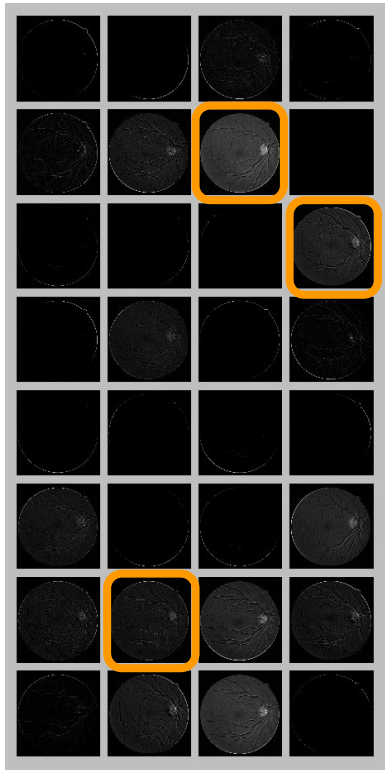
Lost 2 pixels on the left
and 2 pixels on the right

Filter kernels & feature maps

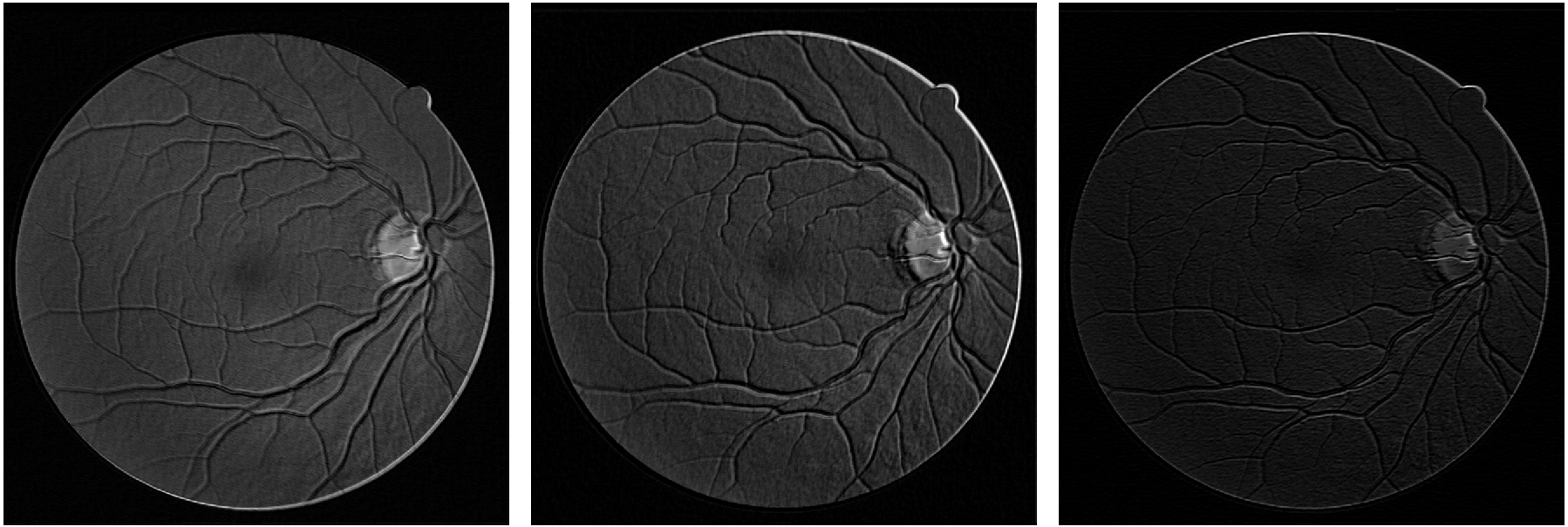


⇒ what did it learn?





First layer feature maps

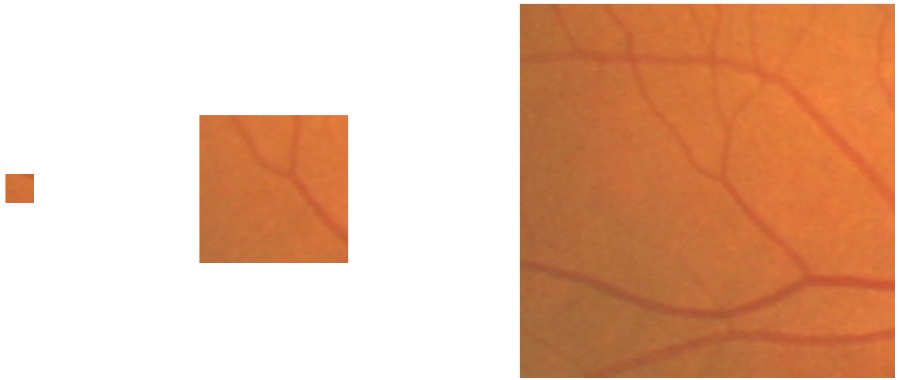


Receptive fields

Strategies for large receptive fields

Receptive field = region of the image analyzed by the network when labeling one pixel

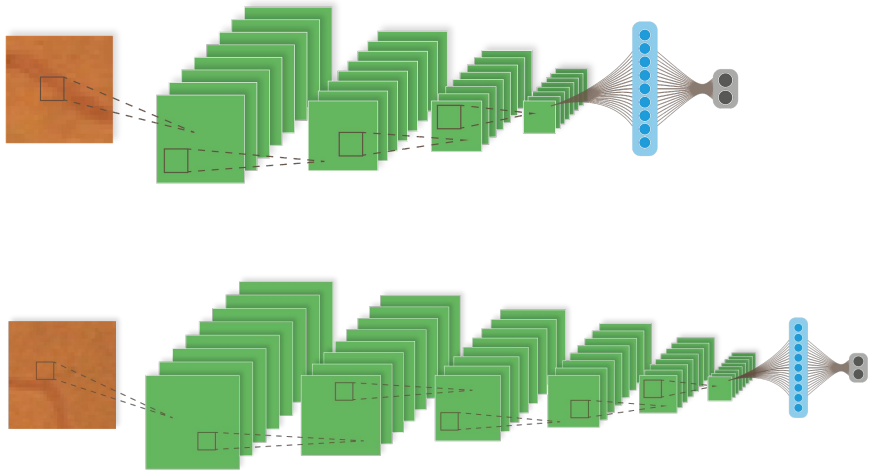
We usually want to use as much information as possible to label a pixel



11x11 51x51 151x151

Deep convolutional neural networks

- Larger receptive field
- More possibilities to combine features (higher expressiveness)
- Very deep networks can be hard to train
- More likely to overfit

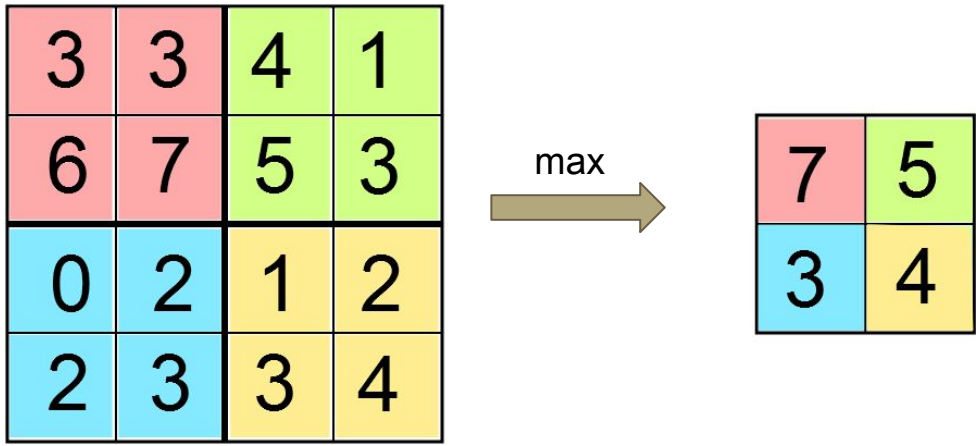


Modern architectures predominantly use small 3x3 kernels → even deeper networks

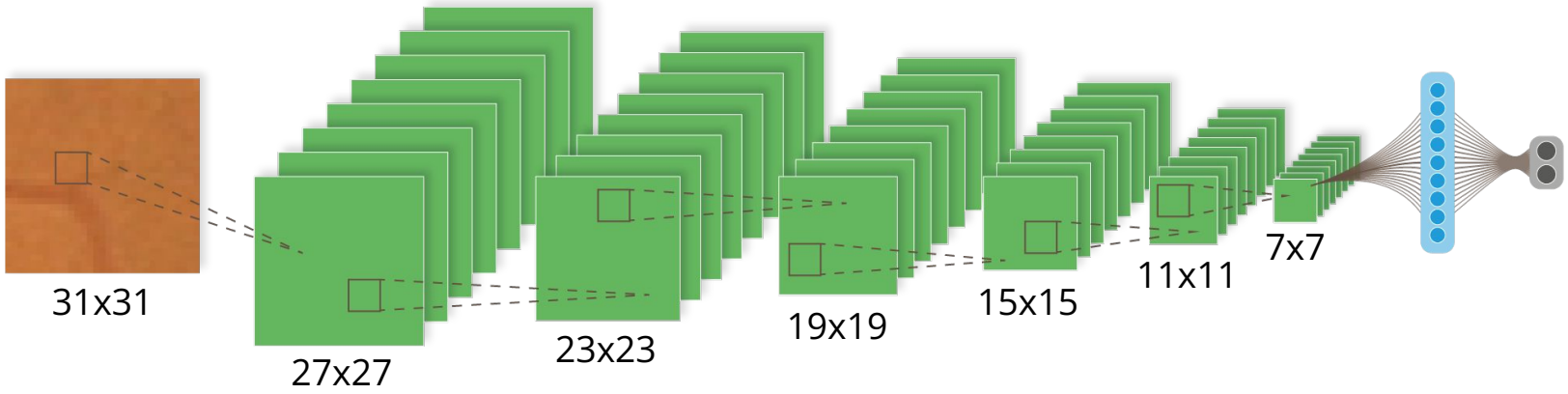
Pooling

Statistics over neighboring features to reduce the size of the feature maps

- Separate the image into non-overlapping subimages (e.g. 2x2)
- Select the maximum / average / ... in each subimage

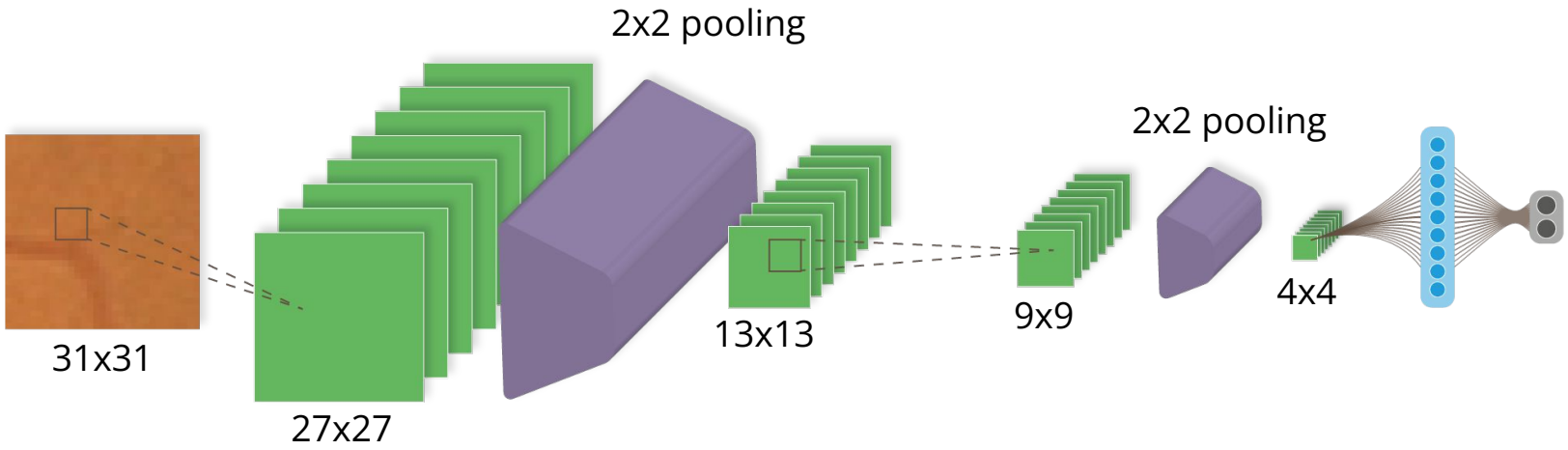


Pooling



Trainable convolutional parameters: **8200**

Pooling



Trainable convolutional parameters: **1800**

Translational invariance

Pooling provides a form of translation invariance

⇒ Small shifts in the input can lead to the same output

This is **very useful** when trying to answer **‘what’ questions**

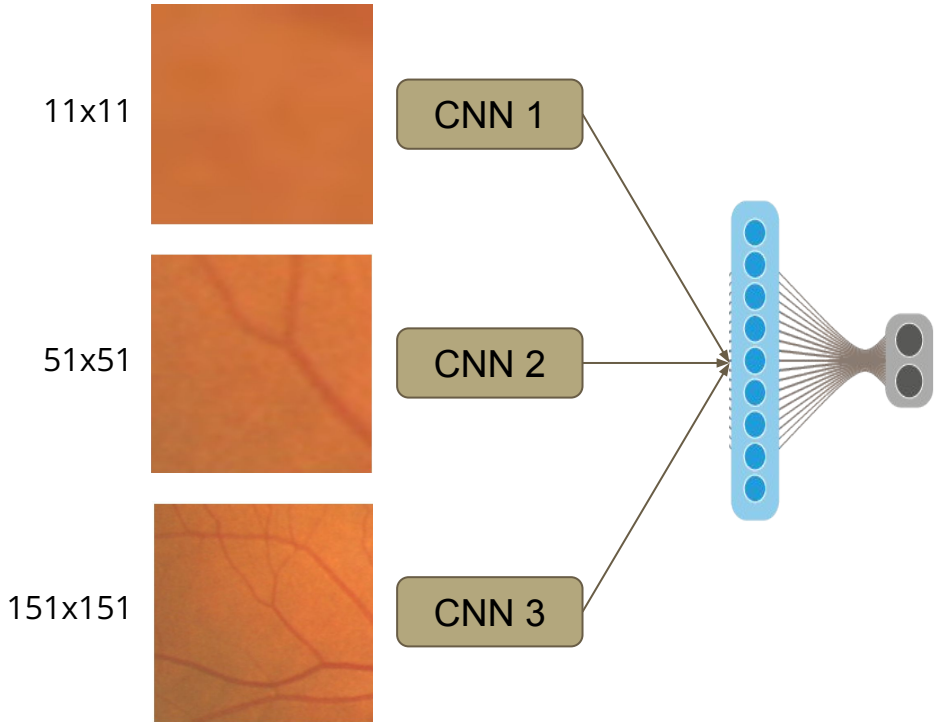
- Is there a cat in this image?

This is **less useful** when trying to answer **‘where’ questions**

- Is this pixel just in or just outside a retinal vessel?

Neighboring patches can get identical feature maps → poor boundaries

Multi-scale patches



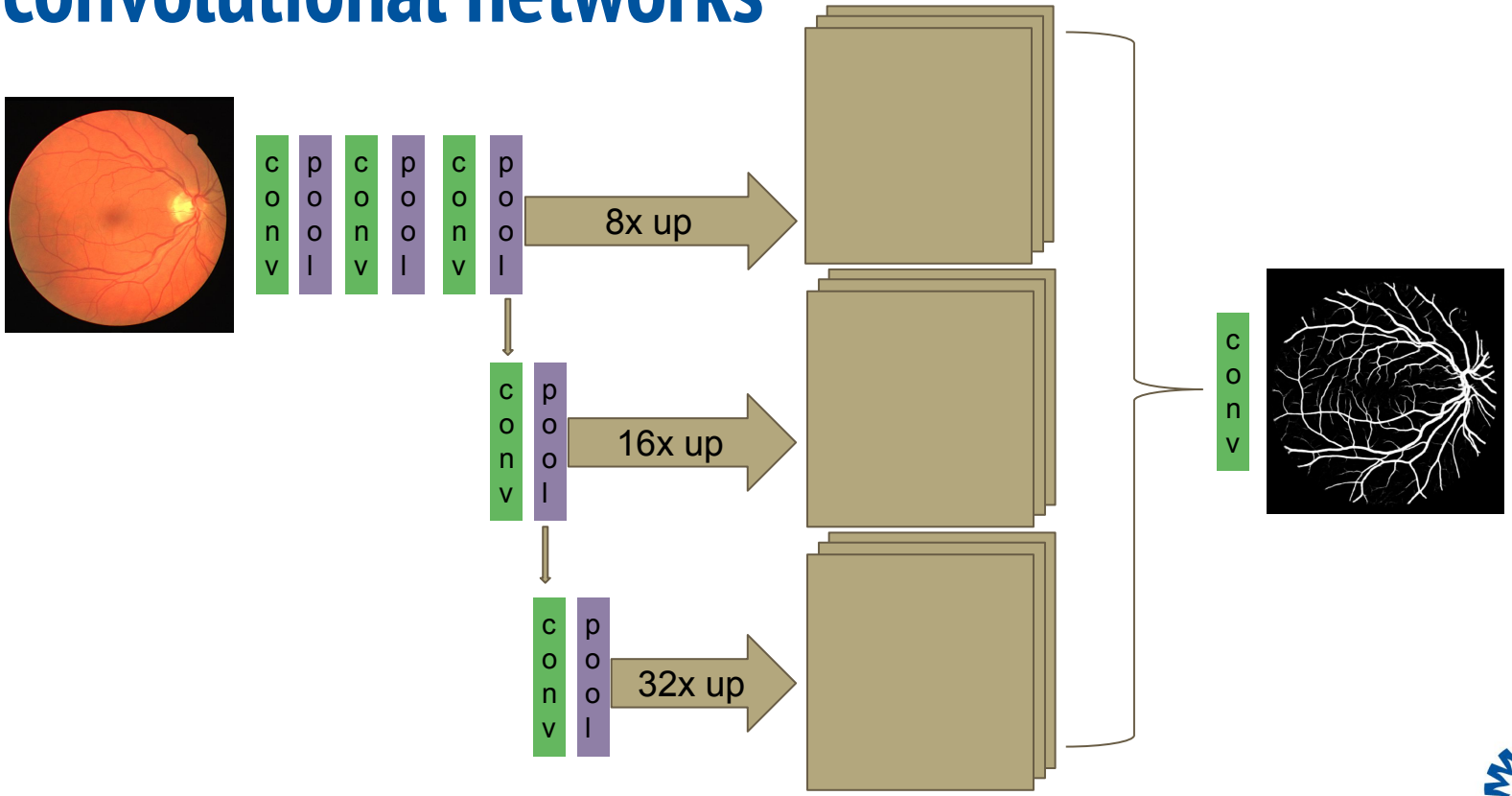
Different subnetworks extract features at different scales

The top network is forced to focus on the local details (exact boundaries)

The bottom networks analyze the spatial context

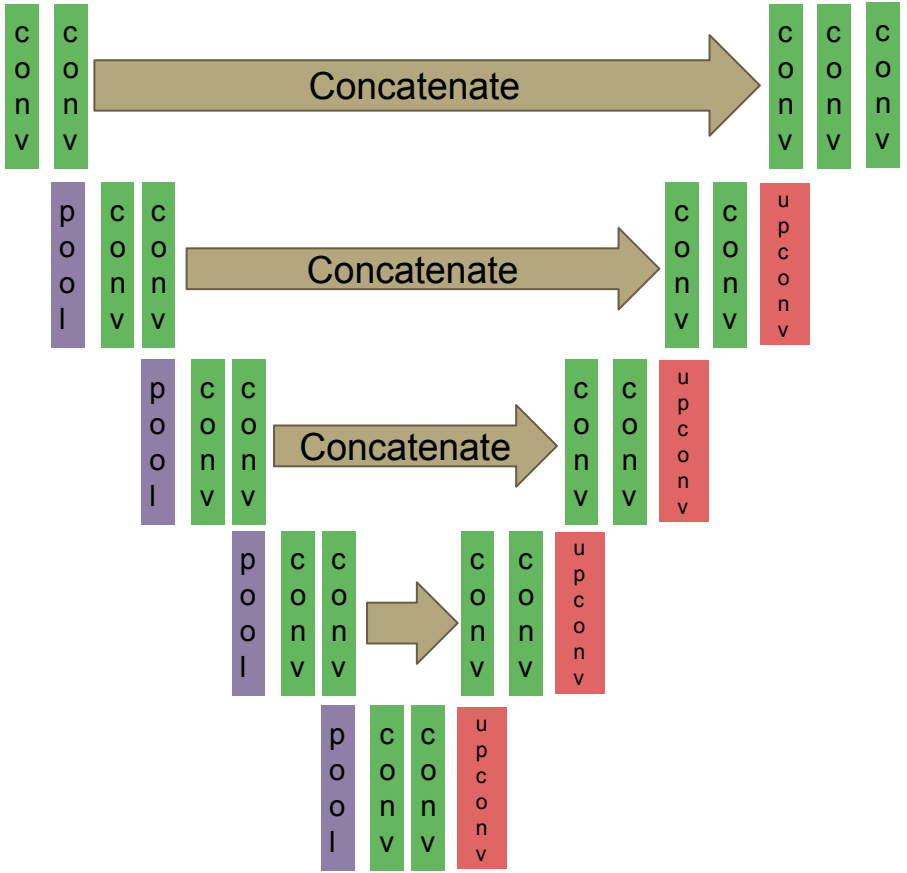
P. Moeskops et al.,, IEEE TMI 35 (5), 2016

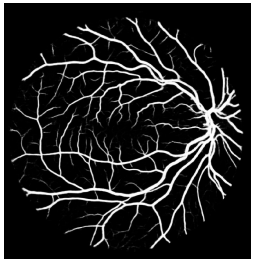
Fully convolutional networks



J. Long et al. *CVPR*. 2015.

U-Net



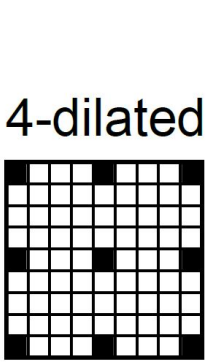


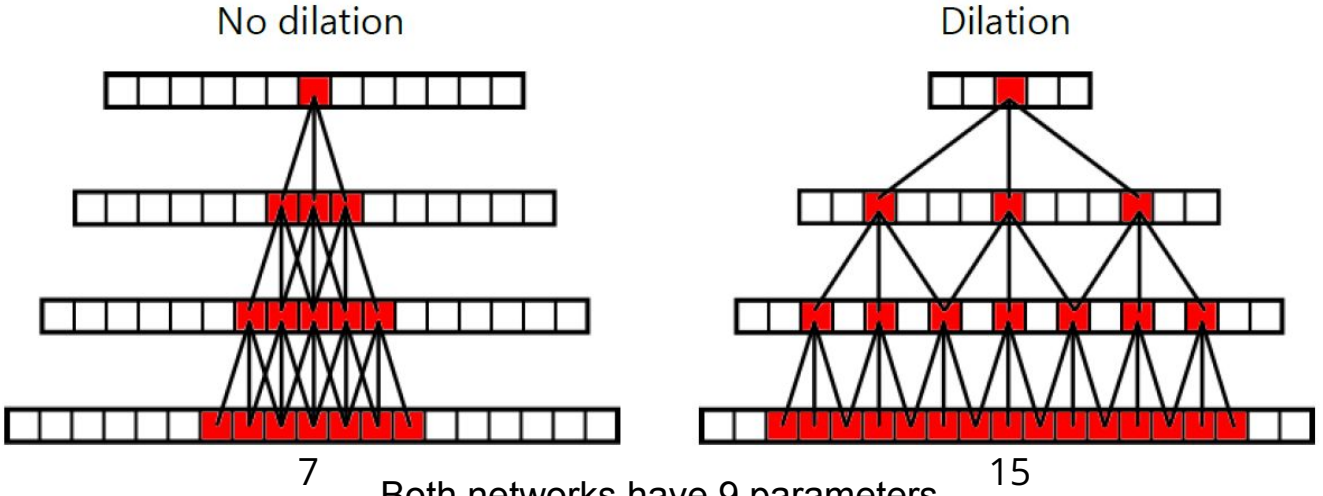
O. Ronneberger et al., *MICCAI* 2015

Dilated convolutions

Add spacing between kernel elements to make kernels wider

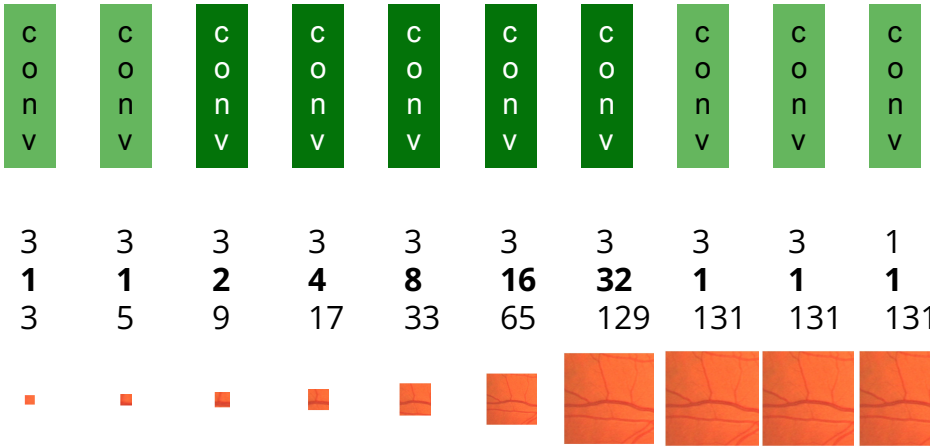
Stacked kernels with increasing dilation → rapidly increasing receptive field

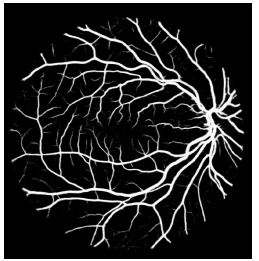




Y. Fisher & V. Koltun, ICLR 2016
J.M. Wolterink et al., *MICCAI* 2016 HVSMR workshop

Dilated convolutional neural network





Kernel width	3	3	3	3	3	3	3	3	3	1
Dilation	**1**	**1**	**2**	**4**	**8**	**16**	**32**	**1**	**1**	**1**
Receptive field	3	5	9	17	33	65	129	131	131	131

Summary

We looked at

- Convolutional layers (more general / invariant)
- Stacked convolutional layers allow more complex analysis (deep networks)
- Ways to build convolutional neural networks with large receptive fields

How do we ensure that the networks we build learn what we want them to learn?

Program

1.	Introduction	Ivana Išgum
2.	Neural networks	Bob de Vos
3.	Convolutional neural networks	Nikolas Lessmann
4.	**Generalization**	**Jelmer Wolterink**
5.	Getting started	Akshay Pai
6.	General summary & discussion	Mads Nielsen

4. GENERALIZATION

Jelmer Wolterink

Generalization

Two stages when using (convolutional) neural networks

1. **Training** with a dedicated training set
2. **Testing** or inference with a new and unseen test set

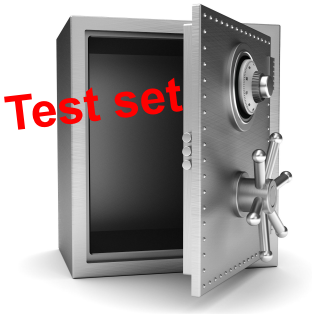
Goals

- Low error L_{train} on the training set
- Low error L_{test} on the test set
- Low generalization error: $L_{test} - L_{train}$

Experimental setup

Separate the data set into three sets

- **Training set**: use to optimize the network's parameters
- **Validation set**: use to monitor and optimize the training process
- **Test set**: use only for the final evaluation



Important in medical imaging datasets
Separate sets at patient level, not at voxel or image level
to prevent data leakage

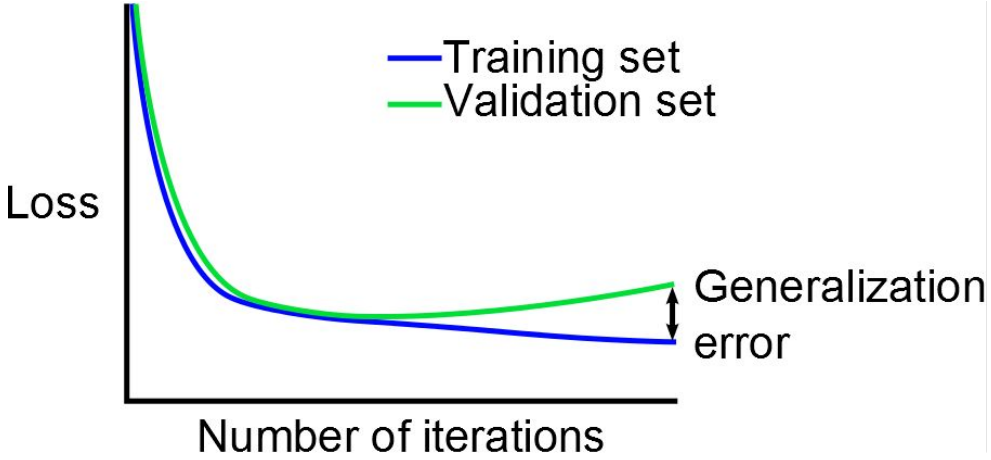
Overfitting

Reasons

- Too little data
- Too many parameters

Solutions

- Increase amount of data
- Reduce number of parameters
- Regularization techniques

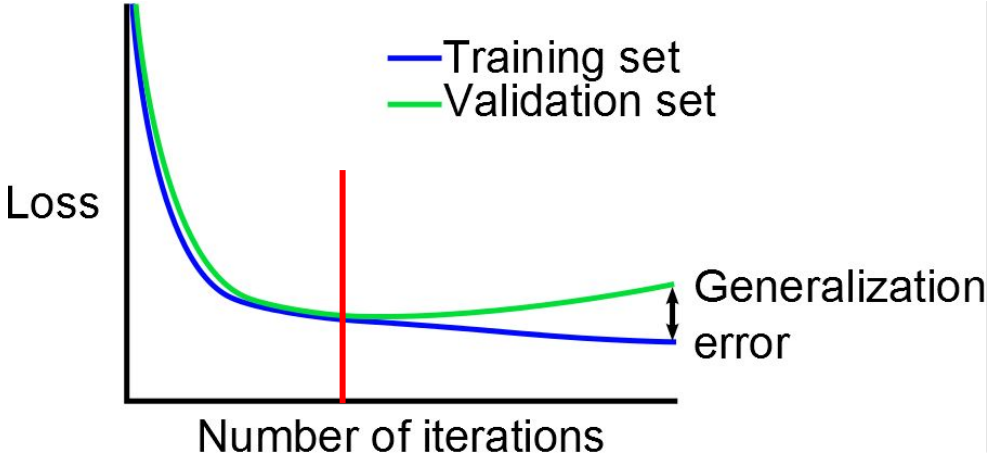


Early stopping

Monitor performance on training set and validation set

Stop at the point where validation loss is minimal

The network has not started overfitting on the training data and will likely generalize to test data



Ensemble of neural networks

Train multiple different neural networks

- Same network, different training data (bagging)
- Different network, same training set
- Same network, same training set, different time points
- ....

Each network will overfit in its own way

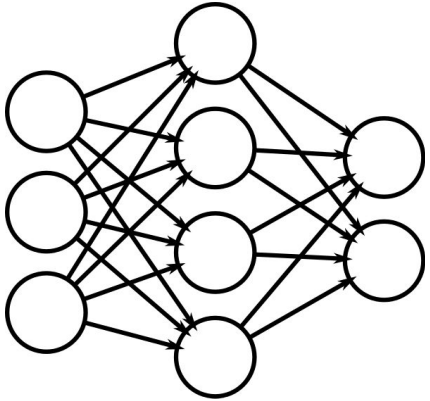
Averaging the network outputs leads to a reduced generalization error

Disadvantage: computationally inefficient training + testing

Dropout

Train one network, but in each minibatch iteration

- ‘Dropout’/temporarily remove each unit with a probability *p*
- Perform forward and backward pass over remaining units



During testing

- Do not dropout any units, but use the full network
- Multiply all outgoing weights by *p* to approximate average of a large number of networks

Individual networks cannot rely on arbitrary pieces of information in the training data

N. Srivastava et al. *Journal of Machine Learning Research* 15.1 (2014): 1929-1958.

Weight decay

Large weights lead to strong responses to noise in the training set

Small weights lead to simpler and more generalizable representations

Minimize the L2-norm of weights in the loss function

$$L = L_0 + \lambda \|\mathbf{w}\|_2$$

Could also be L1-norm, leading to sparse weight vectors → Feature importance

Parameter initialization

If all network weights are initialized to the same value

- The updates to all weights in one layer will always be the same

To overcome this, weights should be randomly initialized

- Too small → the weight updates through subsequent layers shrink too much
- Too large → the weight updates through subsequent layers explode

Solution: Glorot initialization

- Normalize weight values by the number of incoming and outcoming weights

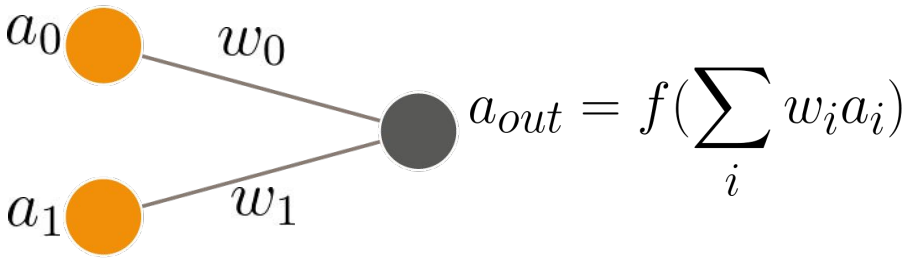
Glorot & Bengio.,*AISTATS* 249-256, 2010

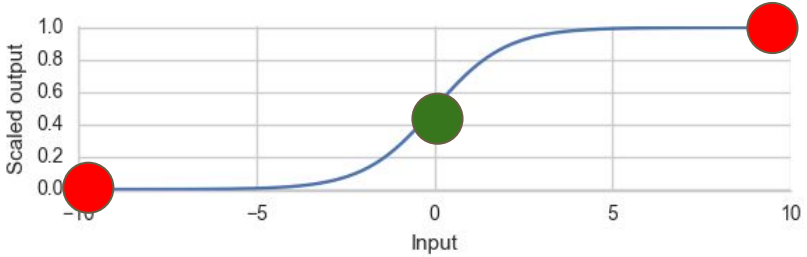
Accelerating training

Neural networks don't train well with poor gradients

Very low or high input values to the nonlinearity can lead to zero gradients

$f(\sum_i w_i a_i)$ has the strongest gradient when $\sum_i w_i a_i$ is around 0





Batch normalization

Scale inputs to nonlinearities to zero-mean unit-variance

Batch normalization transform

$$x = \sum_i w_i a_i$$

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B} + \epsilon}$$

The inputs to a unit *x* are normalized using mean and variance in a minibatch

$$y = \gamma \hat{x} + \beta$$

Shift and scale the new value using learned parameters

$$a_{out} = f(y)$$

Batch normalization can greatly accelerate training and has a regularizing effect

S. Ioffe & C. Szegedy. *ICML 2015*

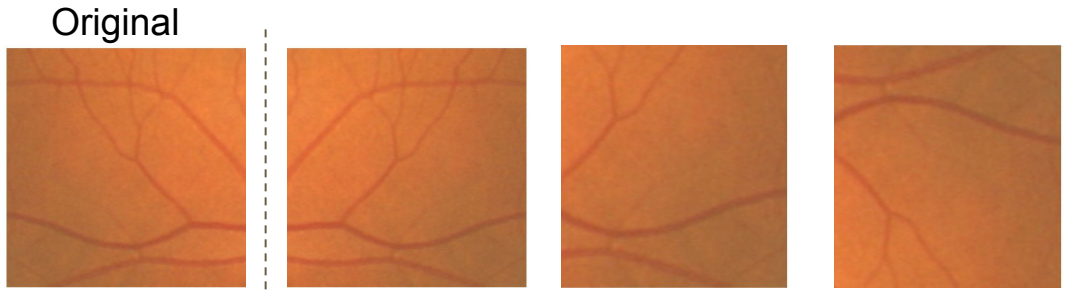
Data augmentation

Increasing the amount of training data is an obvious way to avoid overfitting

Obtaining more training data can be very time-consuming (e.g. ~100 hrs/brain MRI)

The available data can be augmented to increase the training set

- Rotations
- Mirroring
- Cropping/scaling
- Non-linear deformations
- .....



Not all augmentation strategies always make sense for every medical imaging task

Pre-training / fine-tuning / transfer learning

If training data is limited, we could use a pre-trained network

1. First, train a network on different data
2. Then, two possibilities
   a. Fine-tune network with problem-specific data
   b. Extract features from the network and train a new classifier

Pre-training could be used to fine-tune on completely unrelated data

- Natural images → dermatological image classification[1]
- Natural images → pulmonary nodules in CT[2]

[1] Esteva et al. *Nature* 2017
[2] Ciompi et al., *MedIA* 26(1):195-202, 2015

Reducing the number of parameters

Convolutional neural networks by default regularize through parameter sharing

Other possibilities to reduce the number of parameters in a large network include

- Dilated convolutions (discussed by Nikolas)[1]
- Multi-task convolutional neural networks[2]
- Share parameter values in different layers[3]
- Train separable kernels[4]
- ...

1. Yu et al., ICLR 2016
2. Moeskops et al., MICCAI 2016
3. Wolterink et al. MedIa 2016
4. Zheng et a. MICCAI 2015

Summary

To prevent overfitting, and minimize the generalization error

- Regularization techniques
  - Early stopping
  - Ensemble
  - Dropout
  - Weight decay
  - Batch normalization
- Increase training data
  - Data augmentation
  - Pre-training + fine-tuning
- Reduce the number of parameters

How do we actually implement all this?

Program

1.	Introduction	Ivana Išgum
2.	Neural networks	Bob de Vos
3.	Convolutional neural networks	Nikolas Lessmann
4.	Generalization	Jelmer Wolterink
5.	**Getting started**	**Akshay Pai**
6.	General summary & discussion	Mads Nielsen

Deep learning in medical images - Practical insights

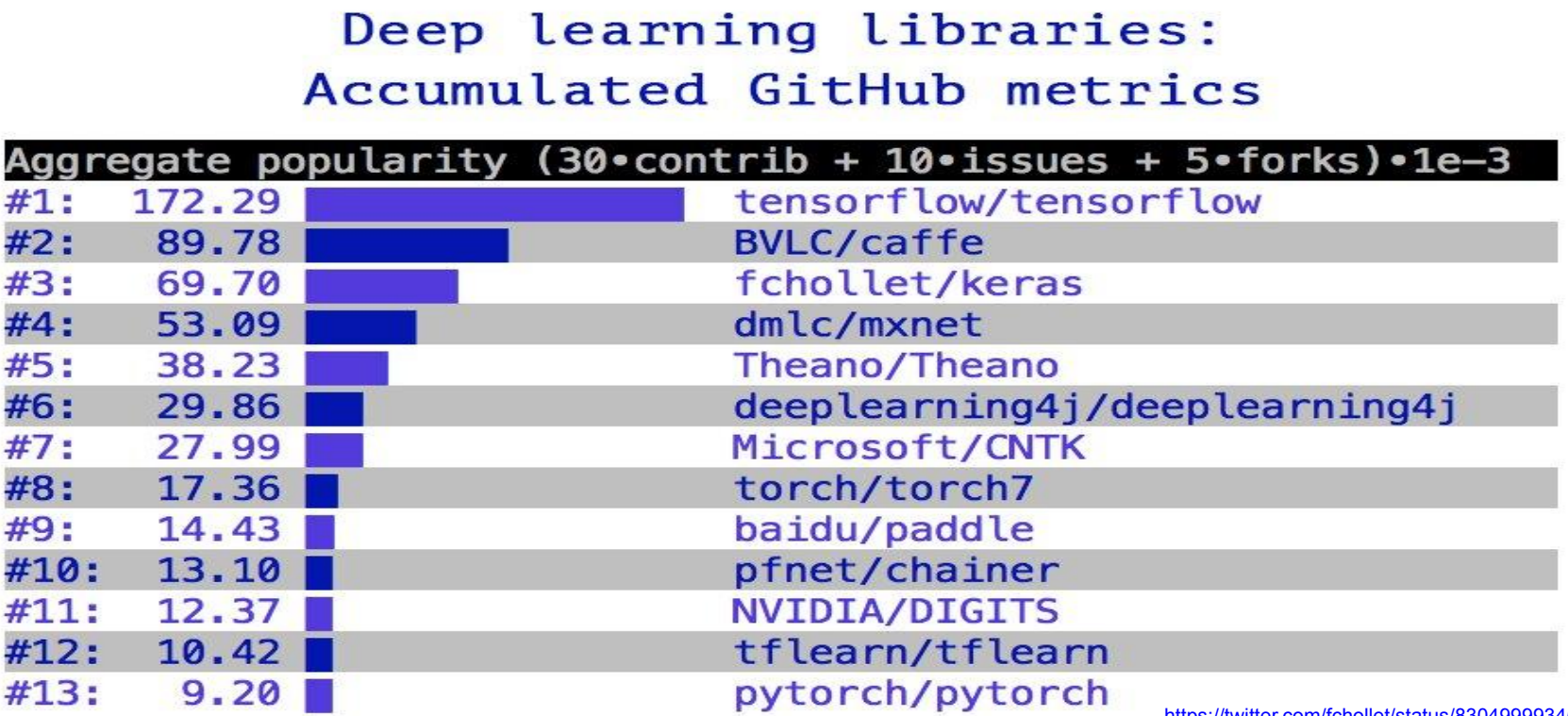
A Pai[1,2], Yuan-Ching Teng[1], Joseph Blair[1], M Nielsen[1,2]

[1]Image Group, DIKU, University of Copenhagen
[2]Biomediq A/S, Copenhagen, Denmark

Contents

- Overview of frameworks.
- Practices and pitfalls
- How to get started - Keras
- Future work.
  - Examples - Unsupervised learning.
  - Generative adversarial networks.

Popularity



https://twitter.com/fchollet/status/830499993450450944

Theano

- Compiles symbolic graphs to construct mathematical relations (not just NN)
- Calls C/C++ to do the heavy-lifting works
- Compute the gradient automatically
- Numerous open-source deep-libraries have been built on top of Theano, including Keras and Lasagne
- Windows compatible

High level wrappers (Keras, Lasagne) available	Error messages can be unhelpful
Computational graph is nice abstraction	Long compile time
Fast in running time	Tricky to load pretrained models
High flexibility in building the models	Multi-GPU support is not mature yet
Easy configuration and speed up in single GPU	

TensorFlow

- “Google version” Theano
- Use C/C++ as the engine
- Supports server side and mobile devices
- Supports Multi-GPU and distributed environment

Faster compile times than Theano	Slower than other frameworks for now
TensorBoard for visualization	Computational graph is pure Python, therefore slow
Multi-GPU usage (regards multi-GPUs as a big one)	Configuration is harder than Theano
High level wrapper available (Keras, skflow, etc)	
Provides both low-level and high-level functionalities	

Keras

- High level wrapper provides cleaner APIs for neural network
- Use either TensorFlow or Theano to perform low-level tensor operations
- TensorFlow/Theano users can define customized low-level operations
- TensorFlow is going to include Keras into its core soon

High level APIs make prototyping faster	Different backends may have different results
Easy to use for beginners, customizable for experienced users	

Caffe

- Caffe is a machine-vision library that ported Matlab's implementation of fast convolutional nets to C and C++
- Caffe is not intended for other deep-learning applications such as text, sound or time series data.
- Caffe already powers academic research projects, startup prototypes, and even large-scale industrial applications in vision, speech, and multimedia.

Good for feedforward networks and image processing	Cumbersome for big networks
Good for fine tuning existing networks	Not extensible
Useful python interface	

A Pai, Deep learning in medical image analysis workshop, 2017

Torch

- Torch is a scientific computing framework wide an API written in Lua and an underlying C/CUDA implementation.
- It has excellent support for machine learning algorithms that puts GPUs first.
- A Python API for Torch, known as Pytorch, was open-sourced by Facebook in January 2017
- PyTorch offers dynamic computation graphs, which let you process variable-length inputs and outputs

Benchmarks

For some benchmarks of popular frameworks refer to the following papers:

https://arxiv.org/abs/1511.06435

http://autumnai.com/deep-learning-benchmarks

http://dlbench.comp.hkbu.edu.hk/#intro
(released in Jan, 2017. Various hardware settings and architectures are tested)

Practicalities

Data normalization

- Normalization could make the training faster. For instance, using whitening.
- Remember to apply same normalization in training and test samples.
  - Using mean and standard deviation derived from the training set.

Model training suggestions

- Overfit the training data first to ensure the pipeline is correct.
  - DO NOT use the test set when optimizing the model hyperparameters.
- Use early stopping to stop training if the validation accuracy goes bad.
- Add weight decay to overcome plateaus.
- Can replace ReLU with trainable variation such as LeakyReLU, PReLU.
  - Cons for LeakyReLU: additional hyperparameter to tune.
- Do not be worried if validation accuracy is higher than training accuracy initially, it will eventually even out.

A Pai, Deep learning in medical image analysis workshop, 2017

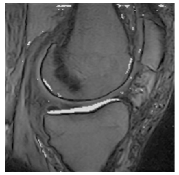
Common practices and pitfalls

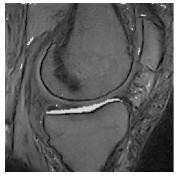
- Stochastic gradient descent works the most consistent across all problems.
- AdaGrad is suited for more shallow networks.
- Adam, RMS Prop work very well for certain problems.
- Different versions of library (e.g. CUDA, cuDNN) may have different results.
- Besides the code, it is recommended to wrap entire testing environment (e.g. Docker container, Vagrant, Amazon machine Images). I.e., Fix all the libraries for consistent comparison of performances.

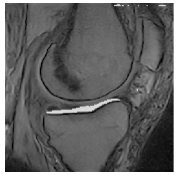
A Pai, Deep learning in medical image analysis workshop, 2017

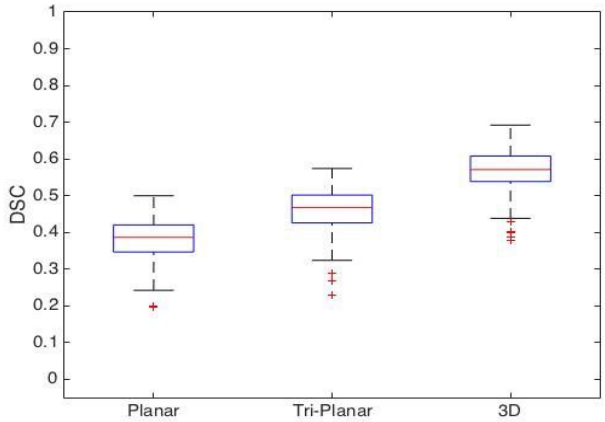
Using different inputs[1]

- Planar
- Triplanar (2.5D)
- 3D

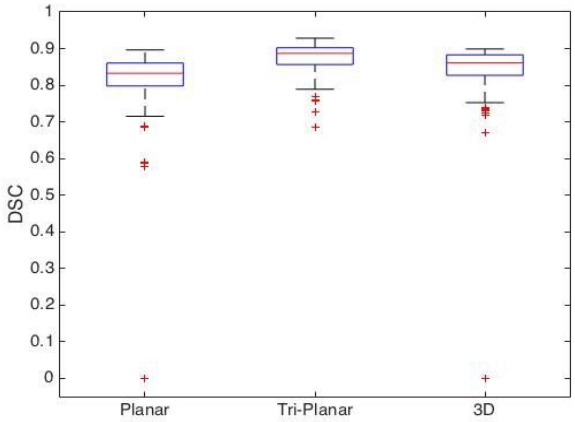

planar


triplanar


3D



Without post-processing



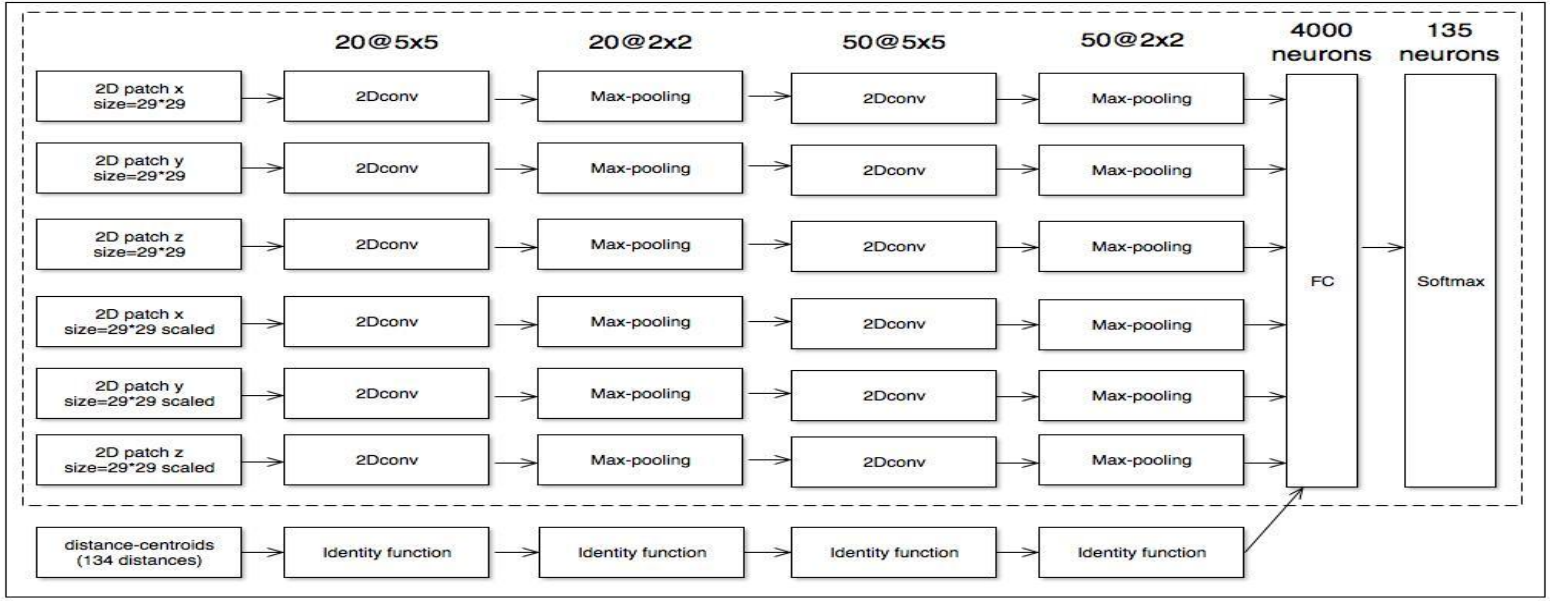
With post-processing

1. *J, Blair, Convolutional neural networks for knee cartilage segmentation, master thesis, university of copenhagen, 2016*

Example work

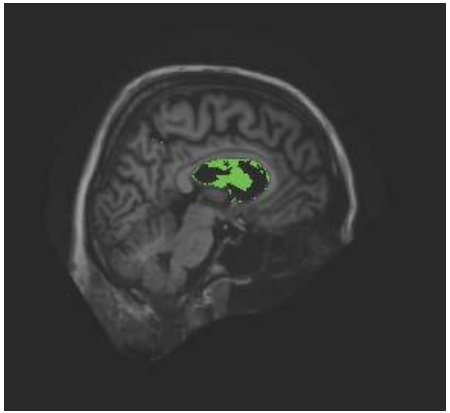
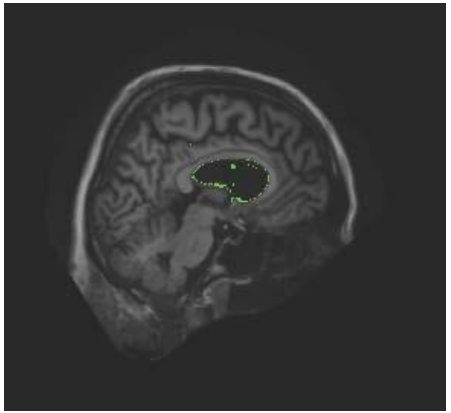
- Whole brain segmentation
- Dataset released by MICCAI 2012 Multi-Label challenge
- Segment brain scan into 134 regions
- Voxel-wise classifier using convolutional neural network (CNN)
- Triplanar CNN with 2 scales of patch size
- Add distance to centroids as external feature for the network

Architecture



Example code to define above architecture in Keras
https://goo.gl/rNty03

Results[1]

Architecture	Triplanar CNN	Triplanar CNN + centroids
Difference to ground truth		

- *Using spatial priors help in organized data segmentation problems.*
- *May not help in problems where the location of the object is random.*

1. *Pai et al, Characterization of errors in deep-learning based image registration, Deep learning in medical image analysis, 1st edition*

Keras installation

Install CUDA
https://developer.nvidia.com/cuda-downloads

Install cuDNN
https://developer.nvidia.com/cudnn

Install Virtual environment

```
$ pip install virtualenv
```

Create an environment and install packages in it

```
$ virtualenv kerasenv
$ source kerasenv/bin/activate
(kerasenv)$ pip install keras
(kerasenv)$ pip install tensorflow-gpu
```

All Set!!

Code examples - Simple network

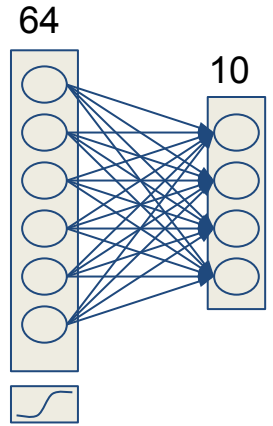
```
os.environ["KERAS_BACKEND"] = "tensorflow"
from keras.models import Sequential
model = Sequential()

from keras.layers.cores import Dense, Activation
model.add(Dense(output_dim=64, input_dim=100,
                init="glorot_uniform"))
model.add(Activation("relu"))
model.add(Dense(output_dim=10, init="glorot_uniform"))
model.add(Activation("softmax"))
```

Or Theano

Can be a graphical model too

Fully connected layer

Initialization of weights



Code example: compile model

```
from keras.optimizers import SGD

model.compile(loss='categorical_crossentropy',optimizer=SGD(lr=0.01,

momentum=0.9, nesterov=True))
```

Or, Adam, Adagrad, RMSprop

Mean squared error,
kullback leibler...

Code examples

Training

model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)

Number of iterations

Minibatch size

Evaluation

objective_score = model.evaluate(X_test, Y_test,batch_size=32)

Testing

classes = model.predict_classes(X_test, batch_size=32)

Code examples: ConvNet

```
model = Sequential()
model.add(Convolution2D(20, 5,5, dim_ordering='tf', border_mode='valid',
                        input_shape=(29,29,3)))
model.add(PReLU())
model.add(MaxPooling2D(pool_size=(2, 2), dim_ordering="tf"))
model.add(Convolution2D(50, 3,3, dim_ordering='tf', border_mode='valid'))
model.add(PReLU())
model.add(MaxPooling2D(pool_size=(2, 2), dim_ordering="tf"))
model.add(Flatten())
model.add(Dense(2000))
model.add(PReLU())
model.add(Dense(135))
model.add(Activation('softmax'))
```

Tensor-flow backend

Hardware

- Currently, GPU-support requires graphic cards supporting CUDA, such as
  - GTX 1080
  - K40
  - Titan X
  - K80

- CPU's may also be used to run training and testing of CNNs. However, the execution takes a significantly longer time.

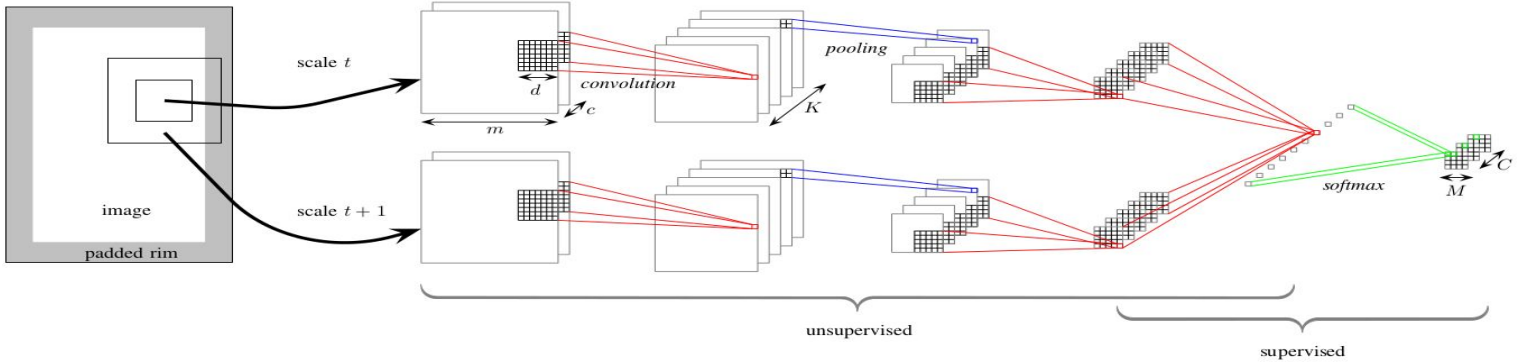
- http://www.geforce.com/hardware/technology/cuda/supported-gpus

Future work

- Training data with manual annotations are expensive to obtain.
- Unsupervised and semi-supervised learning.
- Unsupervised learning (and semi-supervised) use generative models.
  - Boltzman machines.
  - Restricted Boltzman machines.
  - Deep belief networks.
  - Generative adversarial networks.
  - Generative stochastic netoworks.
  - Variational autoencoders.
  - Auto-regressive networks.

Using unsupervised learning[1]

- input: mammographic patches at multiple scales
- network: two parts:
  - unsupervised (stacked auto-encoders)
  - supervised (softmax regression)
- output: texture risk score
  - for each patch → averaging → texture score per image



1. *Kallenburg et al, Unsupervised deep learning applied to breast density segmentation and mammographic risk scoring, IEEE TMI, 2016*

Generative Adversarial Networks

- Formula's p(.) are intractable to compute in bayesian generative models.
- Approximate the distribution of the training data by playing a zero sum game.
- Two players D, and G compete against each other. 'G' tricks 'D' by producing samples that 'D' cannot distinguish.
- Natural fit to unsupervised learning. For synthetic data generation.
- Semi-supervised learning?
- Take advantage of large unlabeled data.
- Train G and D on unlabeled data. Replace the last layer of D, and continue training with small amount of labeled data.
- 'G' can act as a regularizer of 'D'.

Summary

- Torch is the fastest in terms of execution speed.
- Tensor-flow the slowest in terms of execution speed.
- Keras is a convenient API wrapping tensor-flow and theano.
- Practical:
  - Normalize test data the same way as the training data.
  - Ensure the libraries (cuDNN, CUDA) are same when comparisons between networks are made.

Program

1.	Introduction	Ivana Išgum
2.	Neural networks	Bob de Vos
3.	Convolutional neural networks	Nikolas Lessmann
4.	Generalization	Jelmer Wolterink
5.	Getting started	Akshay Pai
6.	**General summary & discussion**	**Mads Nielsen**

Deep learning medical images

What's special?

Feature based ML image analysis

- Gaussian derivatives
- Local binary patterns
- SIFT
- Histogram of Gradients

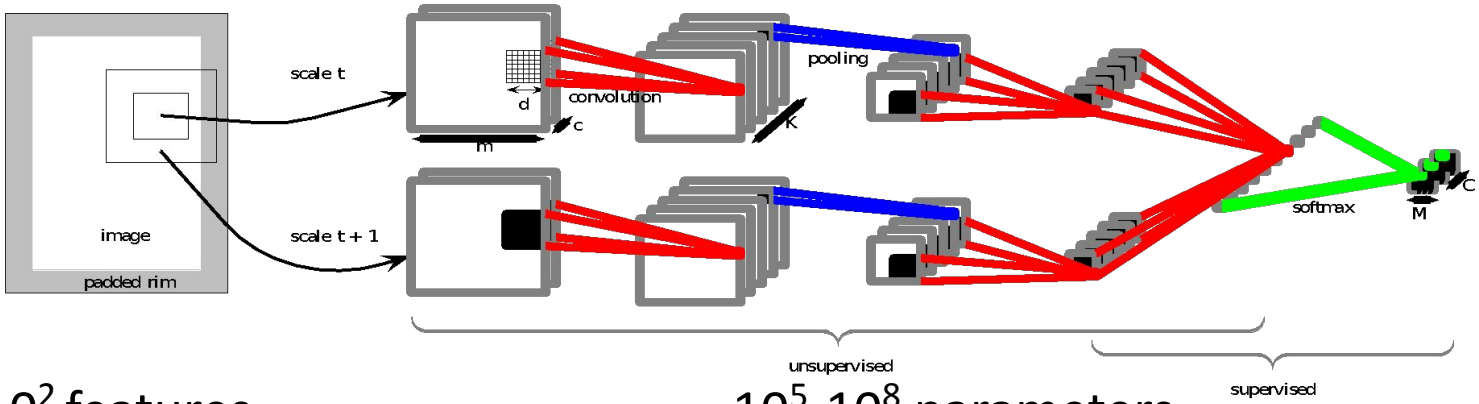
} SVM
Random forest
KNN

} Pixel classification

- 10^2 features
- Invariance

10^3 parameters

Deep learning



- 10^2 features

10^5-10^8 parameters

A Pai, Deep learning in medical image analysis workshop, 2017

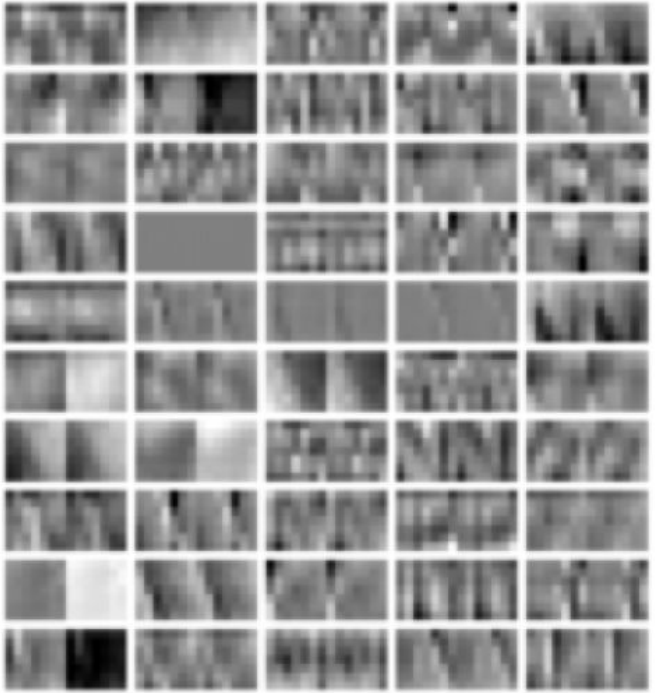
Pros and cons

Pros:

- Higher learning capacity
- Adapt to images
- Linear feature manifold

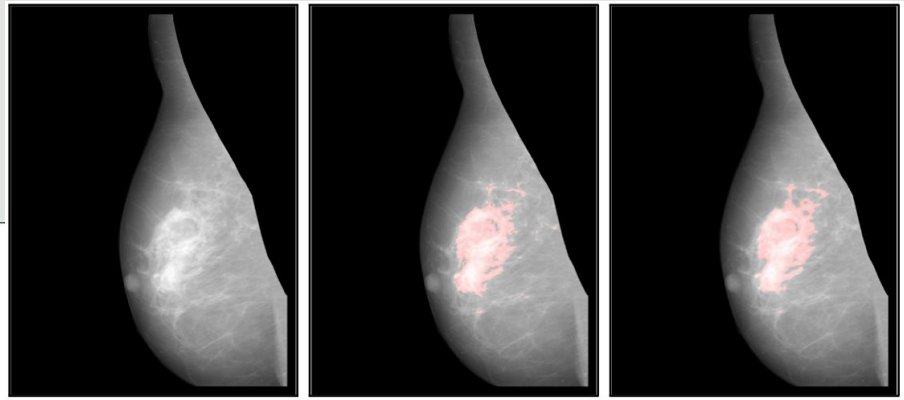
Cons:

- Overfit
- Invariance must be learned
- Needs more data



A Pai, Deep learning in medical image analysis workshop, 2017

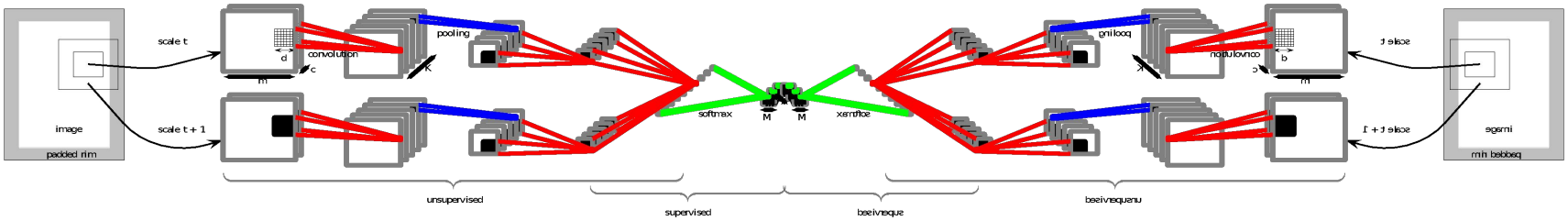
Overfit



Regularization:

- Sparsity
- Early stopping and smart initialization

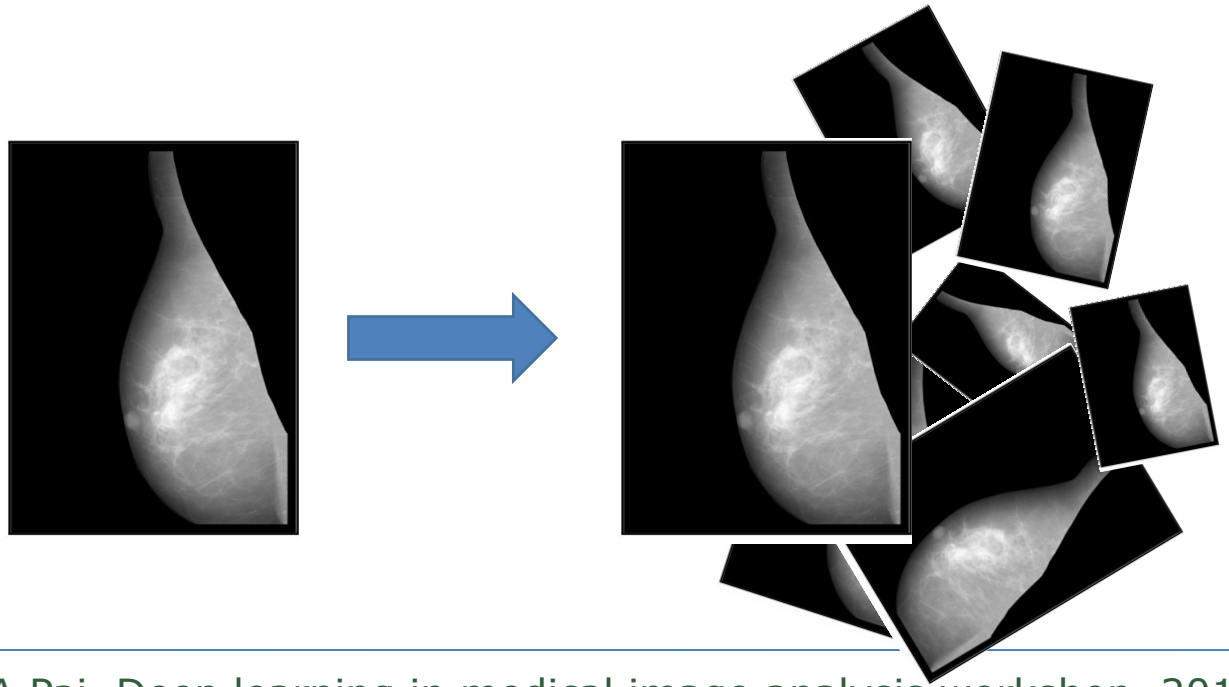
Autoencoders can learn features from unlabelled data



A Pai, Deep learning in medical image analysis workshop, 2017

Learning invariance

Induce invariance in data: scaling, rotation, intensity transforms



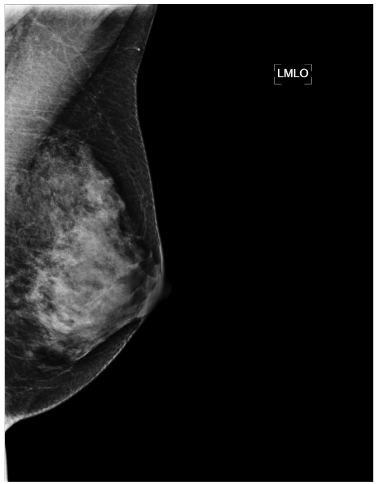
A Pai, Deep learning in medical image analysis workshop, 2017

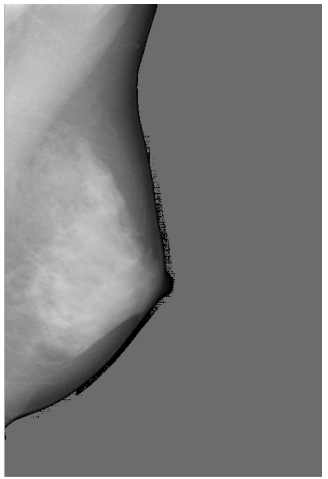
Normalization

You learn the normalization, but get sensitive to changes in imaging

Change of scanner (vendor, software, drift)

Change of population





A Pai, Deep learning in medical image analysis workshop, 2017