Machine Learning

Rudolf Mayer

April 2nd, 2025

Outline

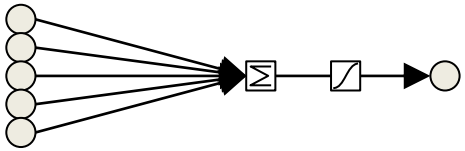
- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

Outline

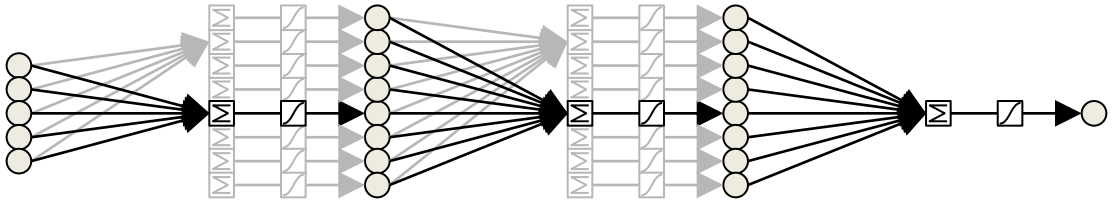
- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

Short Recap

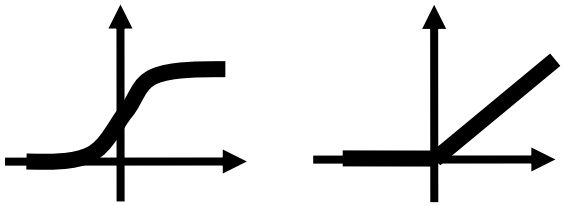
- MLP / Neural Networks
  - (One) building block: Perceptron



  - Multi Layer Perceptron
    - Fully connected, directed, feed-forward network
    - Hidden layers

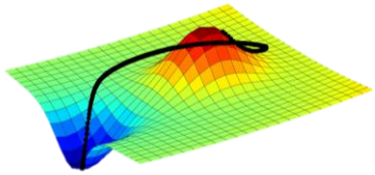


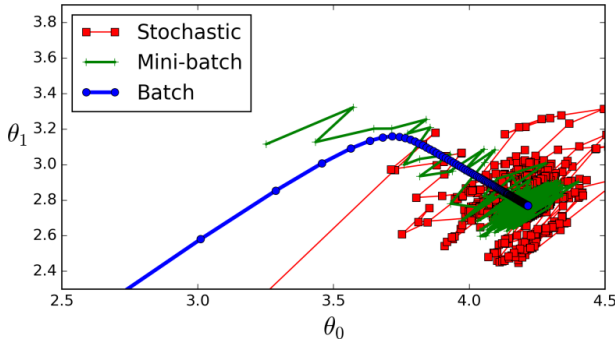
Short Recap



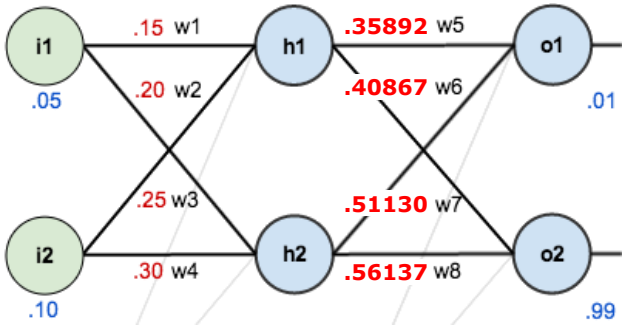
- MLP / Neural Networks
  - Activation functions (more today)
  - Training via back propagation
    - Adapt by gradients of error / loss

$$\frac{\partial L}{\partial W} \quad \frac{\partial L}{\partial \theta}$$



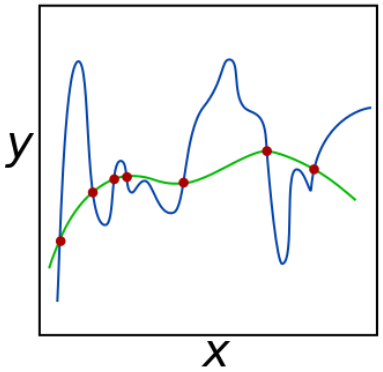


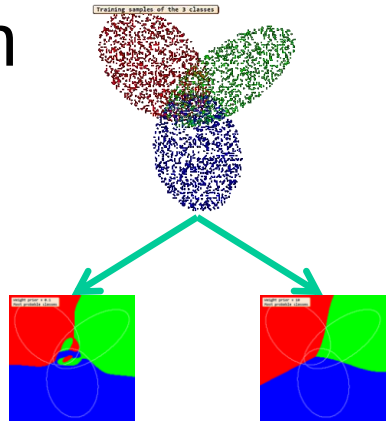
  - Online / batch / mini-batch
  - Example training

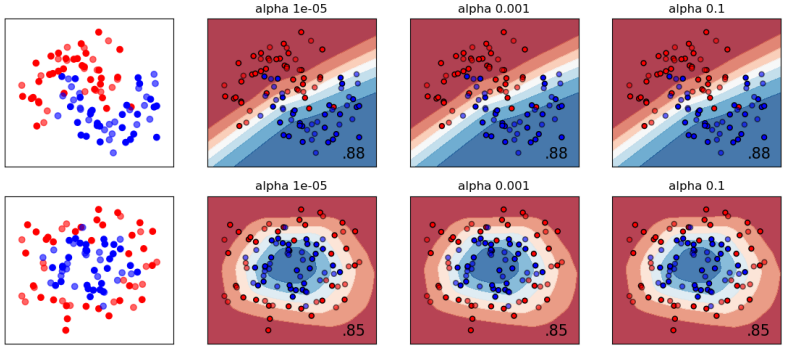


Short Recap

- Regularisation





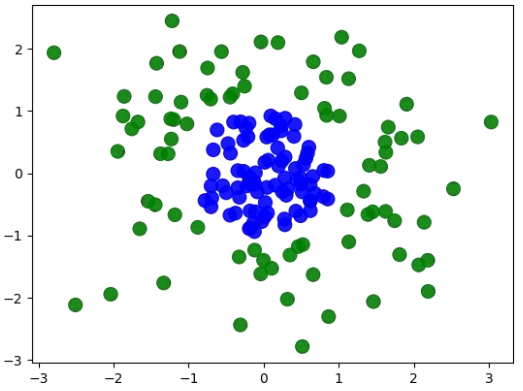


- Vanishing / exploding gradient

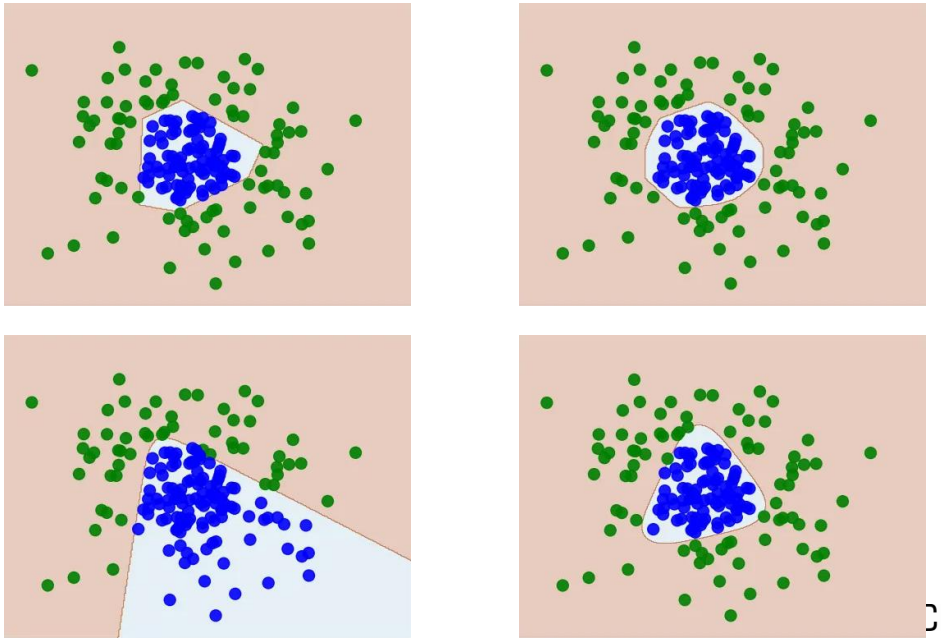
$$\theta \leftarrow \theta - \eta \cdot 0$$

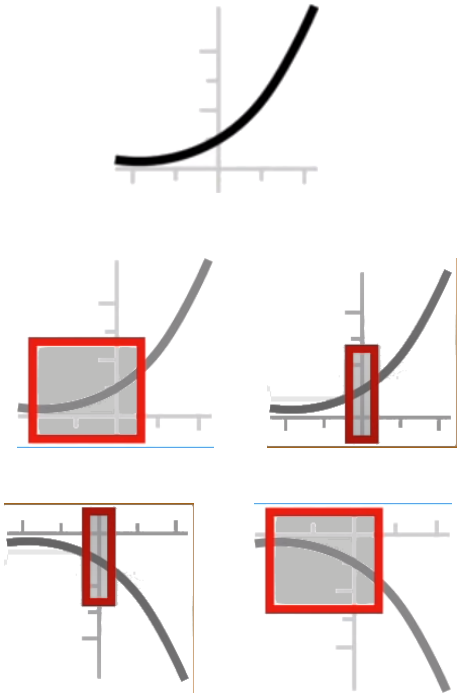
$$\theta \leftarrow \theta - \eta \cdot \infty$$

Recap: Function approximation



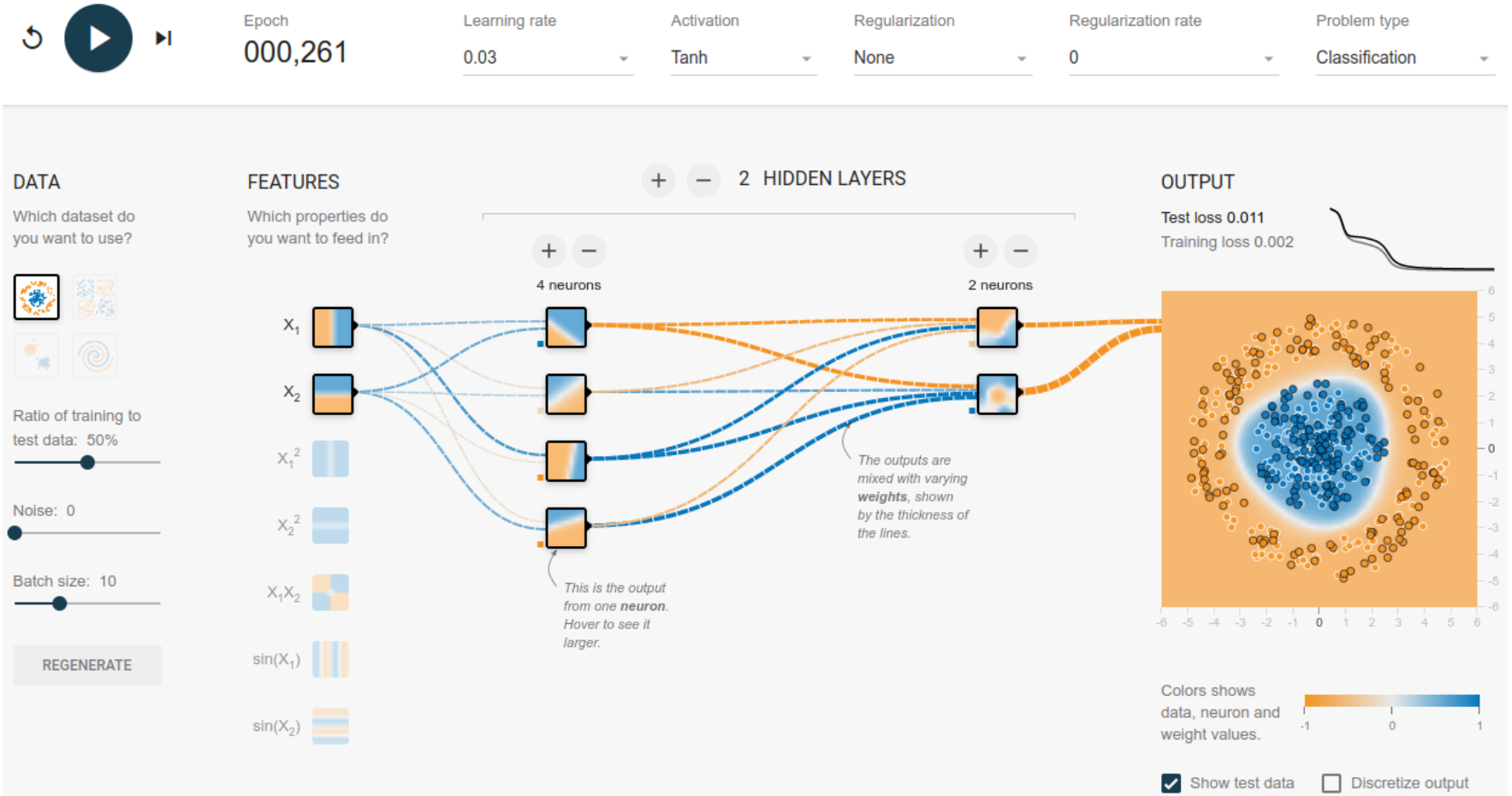
- Decision: piece-wise combination of decision boundaries (in each neuron)
  - Depends on
    - Activation function (shape)
    - Number of neurons (“complexity”)
- Dataset: 2 classes, not linear separable
  - RELU
  - Sigmoid





Short Recap

- Interactive demo: https://playground.tensorflow.org

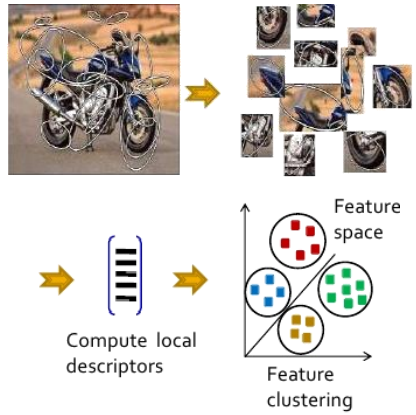


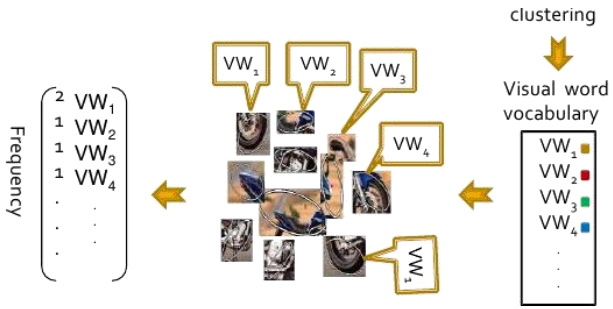
Short Recap

	F1	F2	F3	F4	Label
Sample_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	L_1
Sample_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	L_2
Sample_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	L_3
.	.				.
.		.			.
.			.		.
Sample_n	$v_{n,1}$	$v_{n,2}$	$v_{n,3}$	$v_{n,4}$	L_n

- Feature extraction
  - Convert complex media type to nur
  - Extract characteristic descriptors
    - E.g. the count of words, rather than the word on the 5 position
  - Text

  - Image, Audio, …

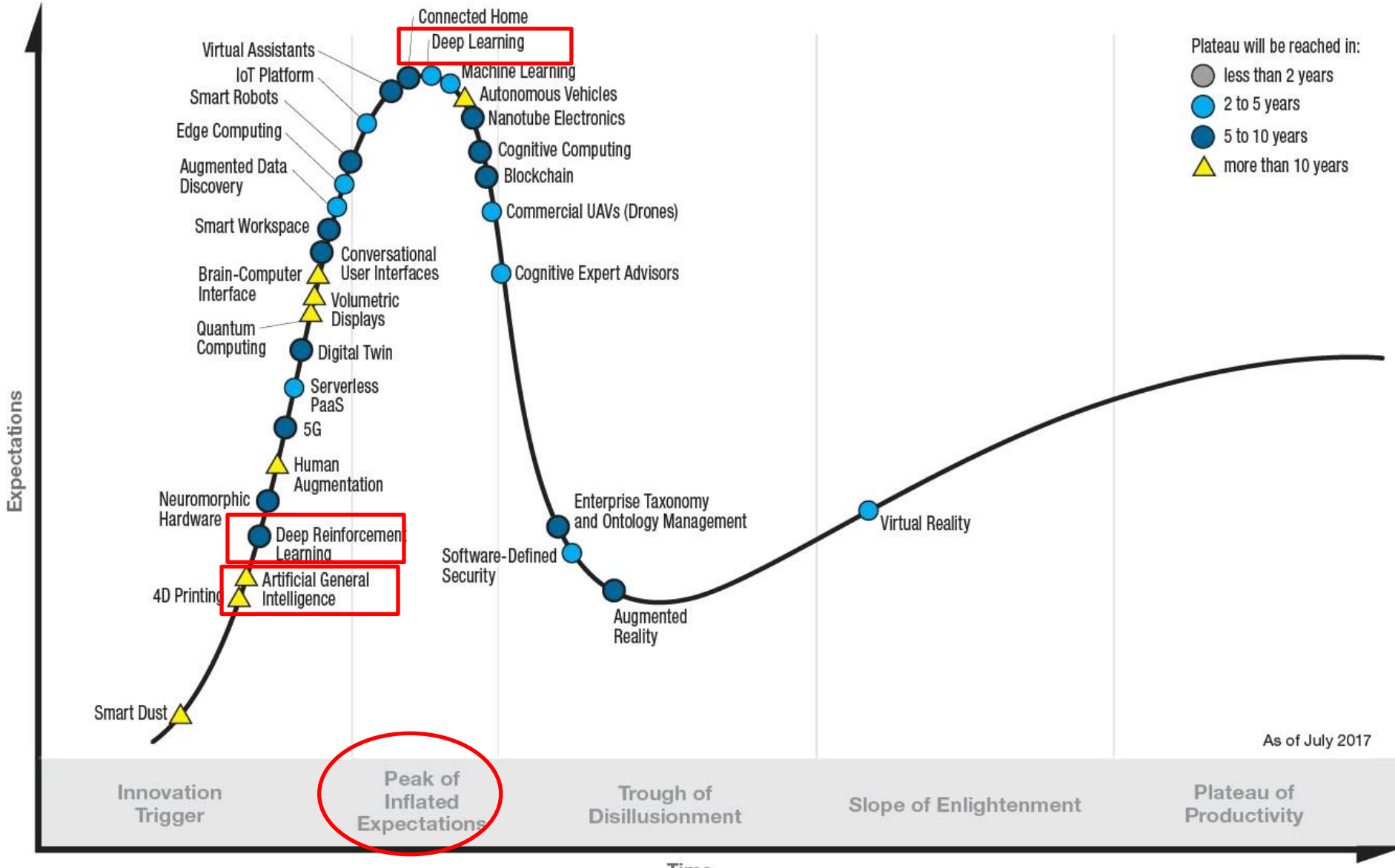




Outline

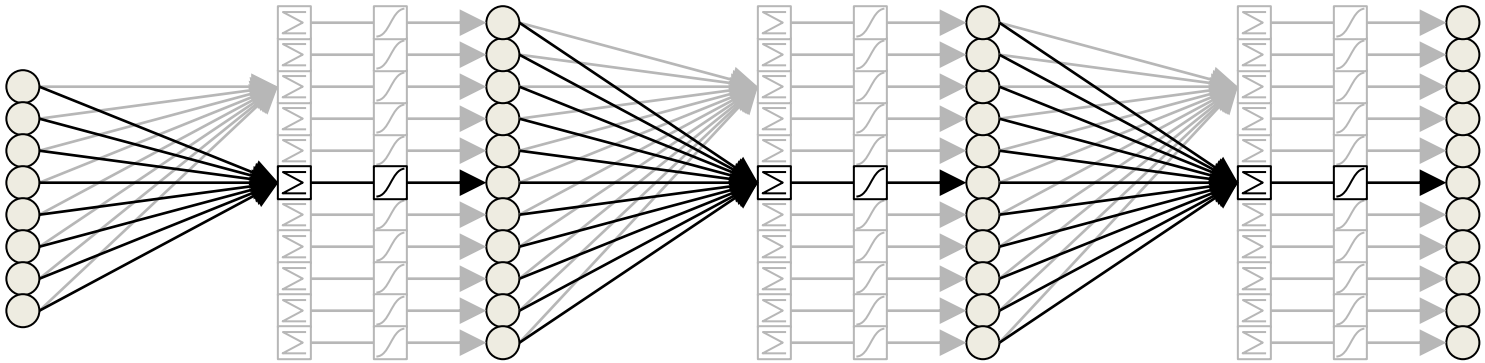
- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

Machine learning – Gartner Hype Cycle



Now what is Deep Learning?

- A term often misused, confused and hyped ☺
- Mostly associated with neural networks
- Generally: learning is considered deep when there are many (hidden) layers

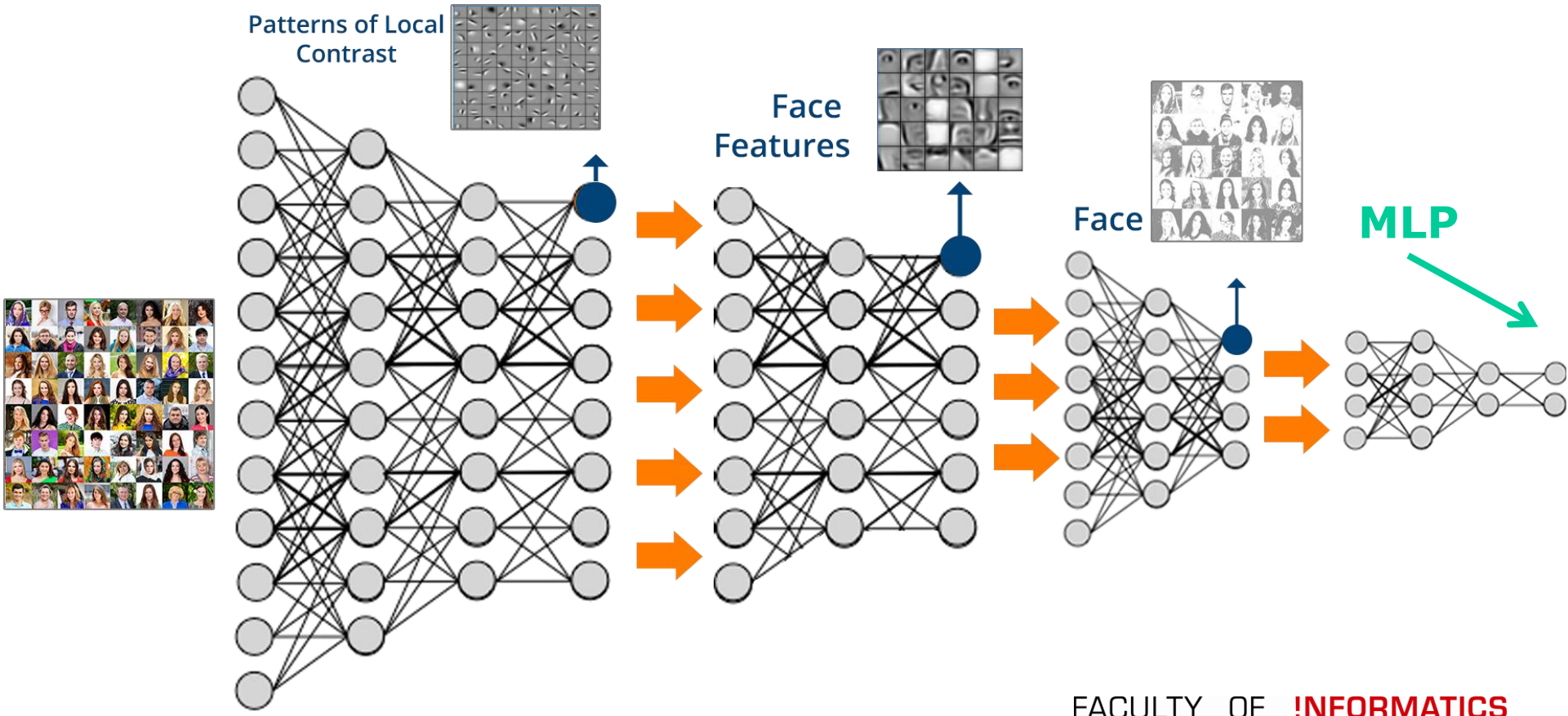


- *What is deep?*
  - Some definitions say >= 2 layers (e.g. of a neural network)

  ➔ i.e. even a relatively simple MLP is “deep learning”

Now what is Deep Learning?

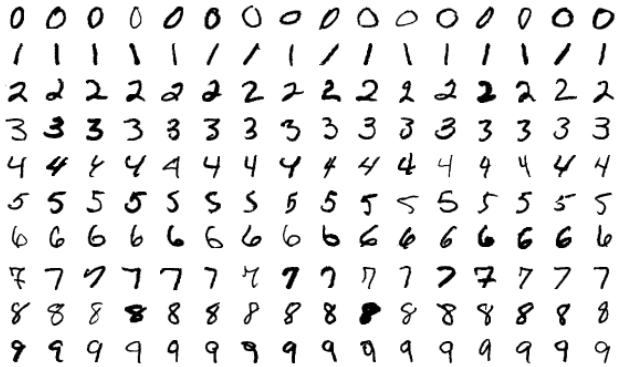
- When there are really many layer (MLPs/fully connected layers and other processing elements, such as feature extraction; some containing layers themselves)



DL Advances: Image Classification

- Pre-ImageNet: relatively small datasets
  - MNIST: 10 digits, CIFAR 10/100, …
  - Small # classes (10-100), low resolution (eg. 32 x 32 x 3)
  - Performance saturated: difficult to make significant advancements
  - MNIST benchmark results: http://yann.lecun.com/exdb/mnist/

K-NN, Tangent Distance	**1.10** LeCun et al. 1998
K-NN, shape context matching	**0.63** Belongie et al. IEEE PAMI 2002
Convolutional net LeNet-1	**1.70** LeCun et al. 1998
Convolutional net LeNet-4	**1.10** LeCun et al. 1998
Convolutional net LeNet-4 with K-NN instead of last layer	**1.10** LeCun et al. 1998
Convolutional net LeNet-5, [no distortions]	**0.95** LeCun et al. 1998
Convolutional net LeNet-5, [distortions]	**0.80** LeCun et al. 1998
Convolutional net Boosted LeNet-4, [distortions]	**0.70** LeCun et al. 1998
Trainable feature extractor + SVMs [affine distortions]	**0.54** Lauer et al., Pattern Recognition 40-6, 2007
unsupervised sparse features + SVM, [no distortions]	**0.59** Labusch et al., IEEE TNN 2008
Convolutional net, cross-entropy [elastic distortions]	**0.40** Simard et al., ICDAR 2003
large conv. net, random features [no distortions]	**0.89** Ranzato et al., CVPR 2007
large conv. net, unsup pretraining [elastic distortions]	**0.39** Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [no distortions]	**0.53** Jarrett et al., ICCV 2009
large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions]	**0.35** Ciresan et al. IJCAI 2011
committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	**0.27** Ciresan et al. ICDAR 2011
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	**0.23** Ciresan et al. CVPR 2012

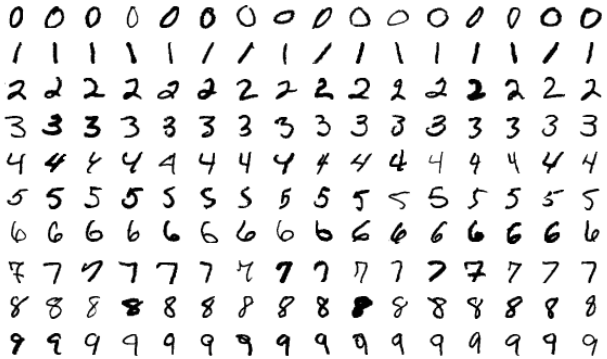


DL Advances: Image Classification

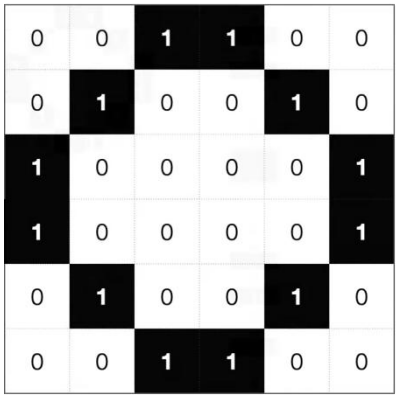
- MNIST Dataset
  - Inputs are very uniform
  - All pretty much same size
  - All pretty much centred
  - Inputs: pixel values at specific positions
    - MNIST: grey-scale (otherwise: RGB or other colour space)

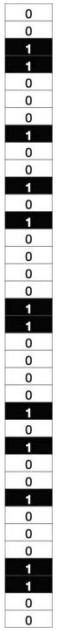
- How to train a standard Neural Network (MLP) on that dataset?

DL Advances: Image Classification



- MNIST Dataset
- How to train an MLP?
  - Convert 2D input matrix to 1D vector (of greyscale values)
  - Feed into the MLP

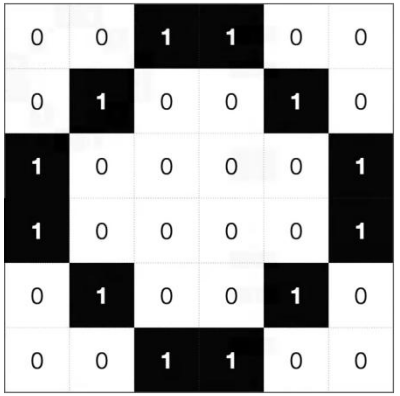


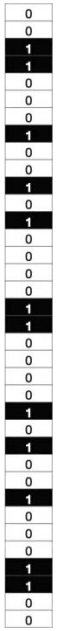


- Does not work well for “regular” datasets

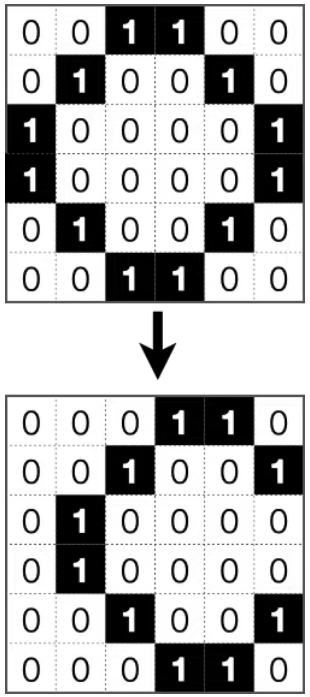
DL Advances: Image Classification

- MLP on MNIST



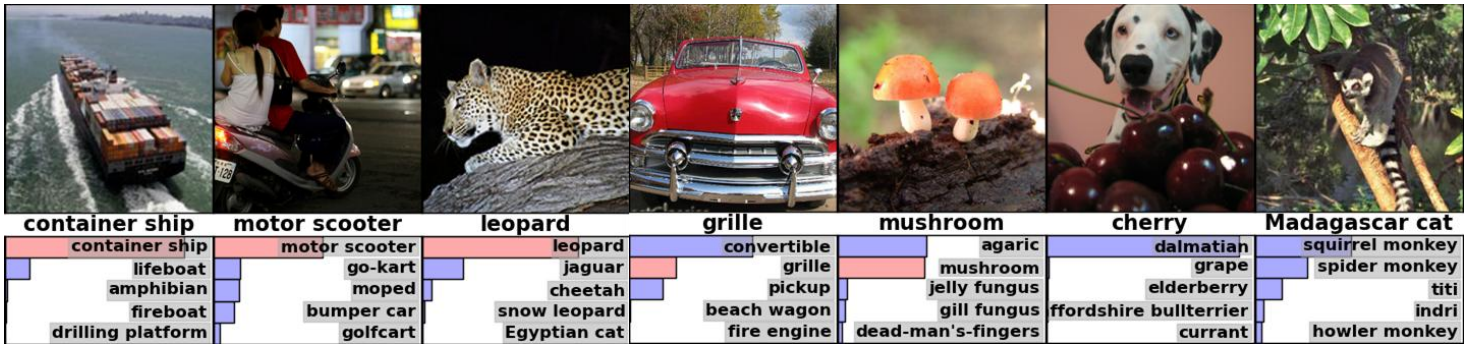


- Does not work well for “regular” datasets
  - Not centered (i.e. the object might be in the top-left, or bottom-riogh, etc...)
  - > 1 object
  - Different size
  - Rotated
  - ……



DL Advances: Image Classification

IMAGENET

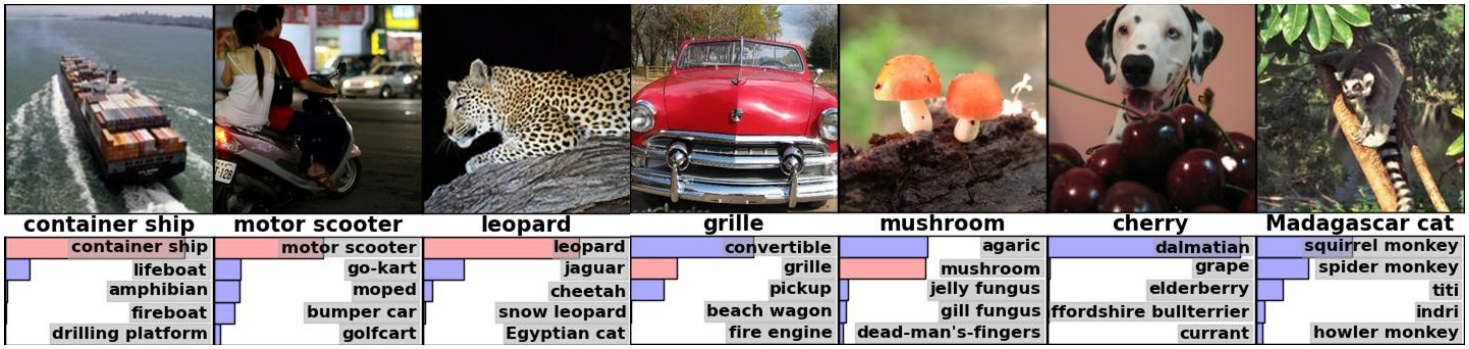


- Large Scale Visual Recognition Challenge (ILSVRC)
  - Started 2009, annotated by mechanical turk
  - Much larger scale than any previous
  - 1.2 million training images in 1000 classes
  - Task: assign labels (classes) to each image
  - Evaluation: set of 150.000 images
  - Metric: prediction among top-1 or top-5 labels

Deng et al. Imagenet: A large-scale hierarchical image database. *Computer Vision and Pattern Recognition, 2009*

DL Advances: Image Classification

IMAGENET



2012: 2nd-best entry: 26.2%

AlexNet achieves 16.4% error (first deep net)

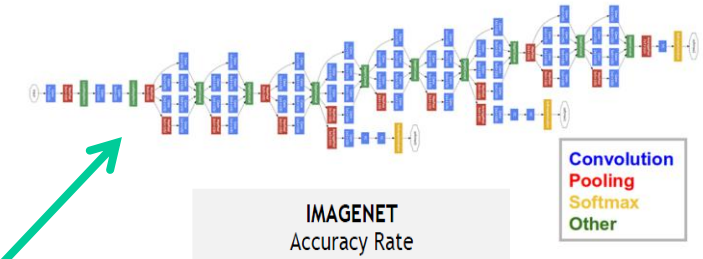
2013: Clarifai: 11.7% error (most entries are DL)

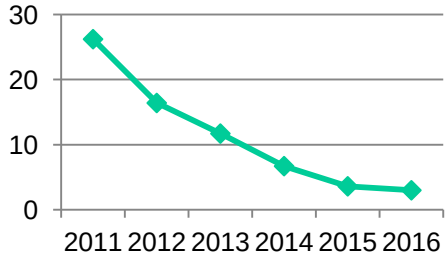
2014: GoogLeNet: 6.7% error (22 layers)

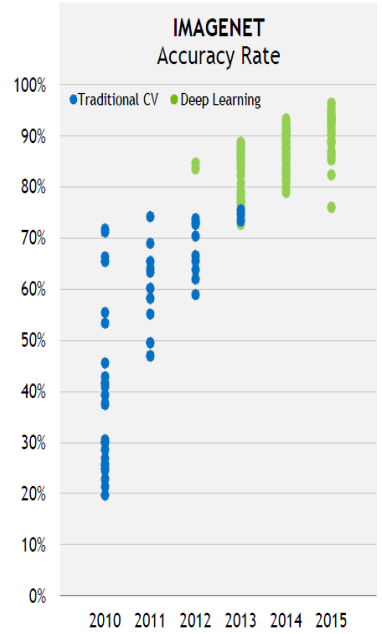
2015: ResNets: 3.6% error (138 layers !)

2016: 2.99% error (ensembles of GoogLeNet, Resnet, ..)

2017: 2.25% error



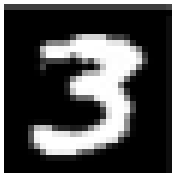




FACULTY OF !NFORMATICS

Now what is Deep Learning?

- **"Traditional" machine learning**
  - Express problem as a function, automatically adapt parameters to fit input data to desired output



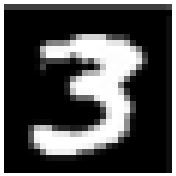
$$y = f(x; \theta)$$

adapt this

"3"

to predict this

- **Deep learning**
  - Learnable function is a **stack of many simpler functions** (often having the same form)



$$y = f(f(f(x;\theta_1);\theta_2);\theta_3)$$

adapt this

"3"

to predict this

Now what is Deep Learning?

$$y = f(f(f(x;\theta_1);\theta_2);\theta_3)$$

adapt this

to predict this

"3"

- Deep learning
  - Learnable function is a **stack of many simpler functions** that often have the same form
  - Often, it is an artificial neural network (ANN, e.g. a DNN)
  - Often, one tries to minimize hard-coded steps
    - Such as feature extraction
  - Most commonly used simple functions:
    - Matrix product: $f(x; \theta) = W^T x$
    - 2D Convolution: $f(x; \theta) = x * W$
    - Subsampling / pooling
    - Element-wise nonlinearities (sigmoid, tanh, rectifier)

Now what is Deep Learning?

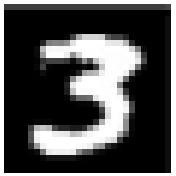
$$y = f(f(f(x;\theta_1);\theta_2);\theta_3)$$

adapt this

to predict this

“3”

- It is convenient to express repeated function application as a sequence



f(x) f(x) f(x)

“3”

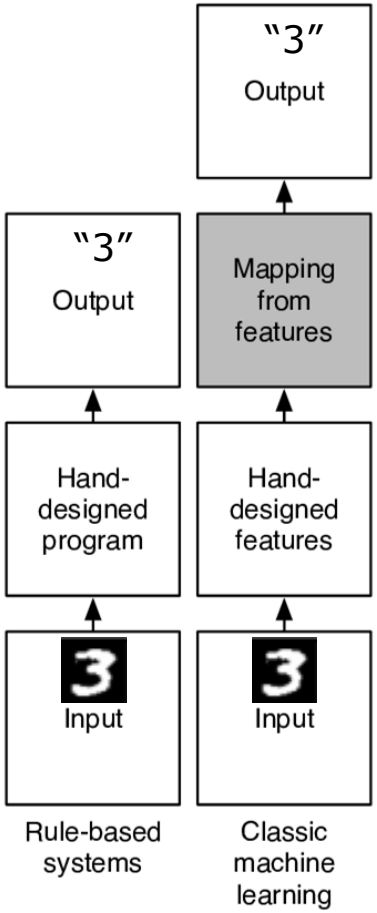
- Computation in sequential steps of processing, often termed “layers”

Learning Paradigms

Rule-based systems:
- Write algorithm by hand
  (if ... then ...)

graphic: Y. Bengio, Deep Learning, MLSS 2015

Learning Paradigms



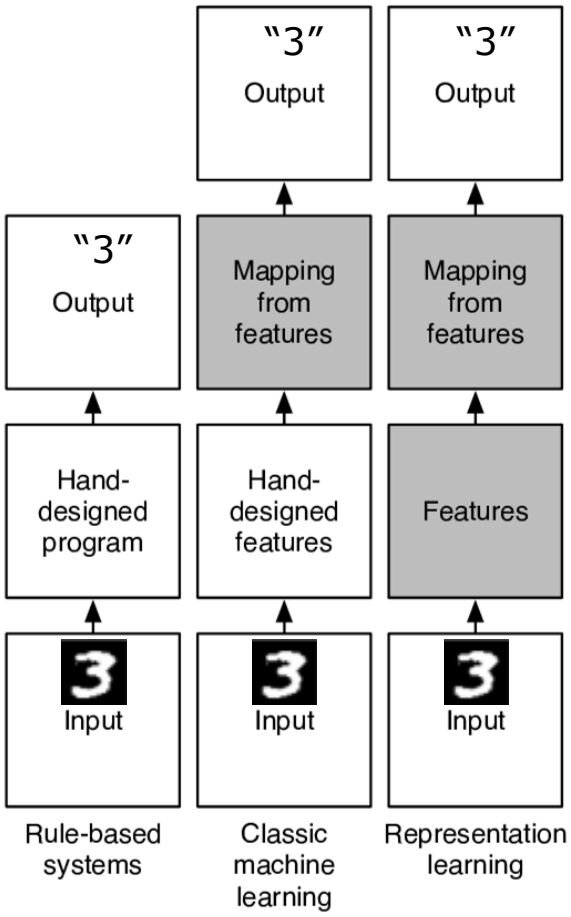
Rule-based systems:
- Write algorithm by hand

(if ... then ...)

Classic machine learning:
- ***Write feature extractor*** by hand
- Train classifier on top

graphic: Y. Bengio, Deep Learning, MLSS 2015

Learning Paradigms



Rule-based systems:
- Write algorithm by hand

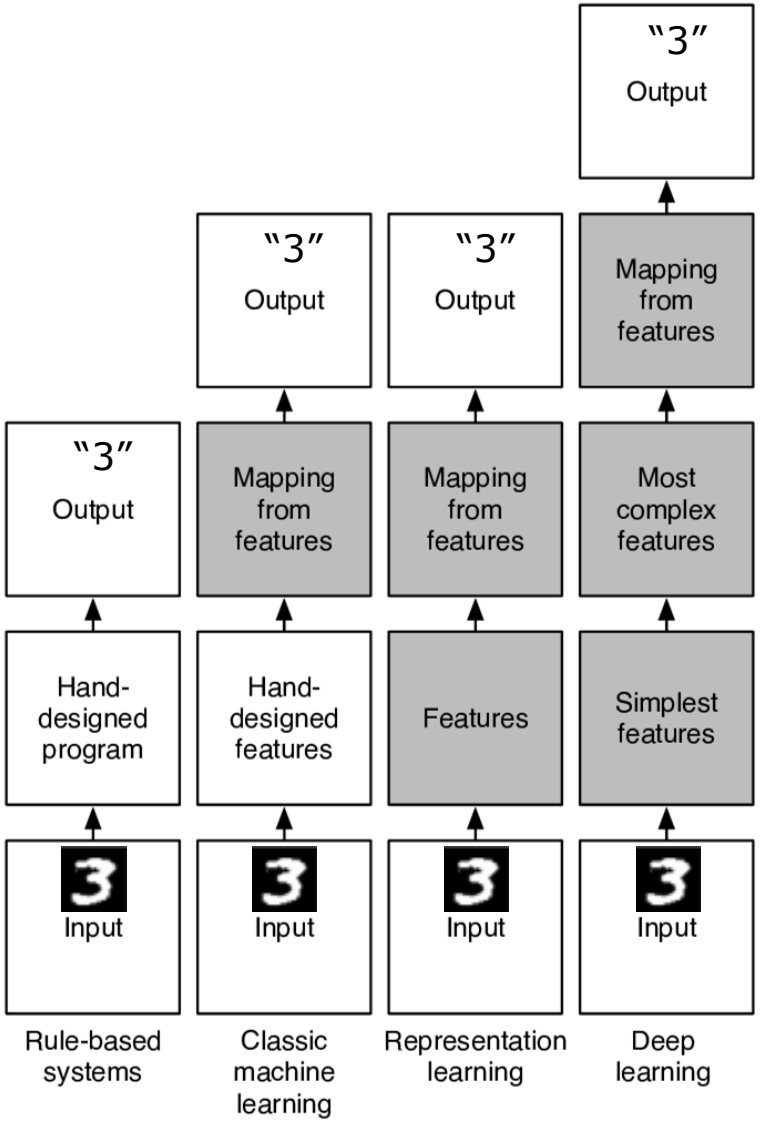
(if ... then ...)

Classic machine learning:
- ***Write feature extractor*** by hand
- Train classifier on top

Representation learning:
- ***Learn feature extractor*** (often unsupervised)
- Train classifier on top

graphic: Y. Bengio, Deep Learning, MLSS 2015

Learning Paradigms



Deep learning:

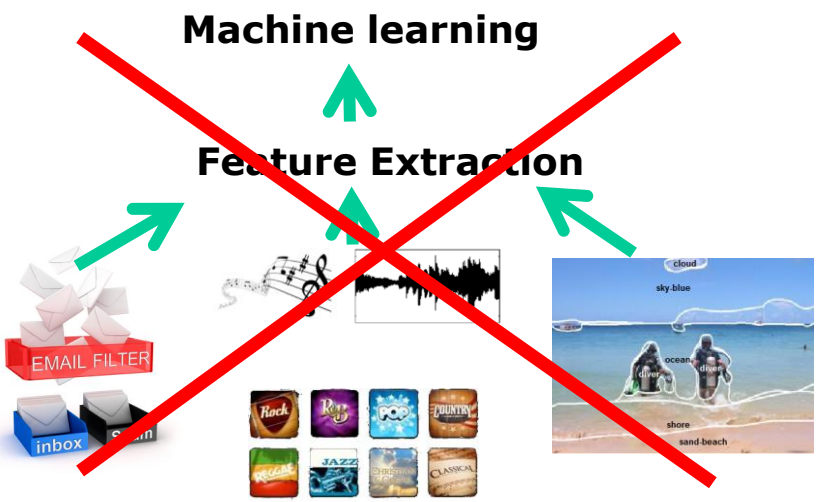
• Learn a stack of many simpler functions to map input to output.

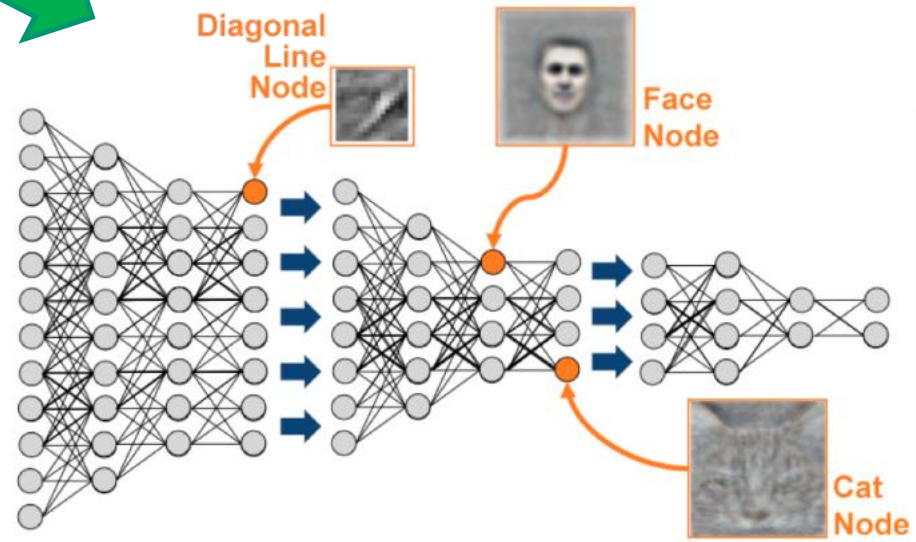
- Often, that stack is a *neural network*
- Often, it is trained on raw input: optimize features & classification together, *minimize hand-crafting*

"**end-to-end learning**"

graphic: Y. Bengio, Deep Learning, MLSS 2015

Deep Learning



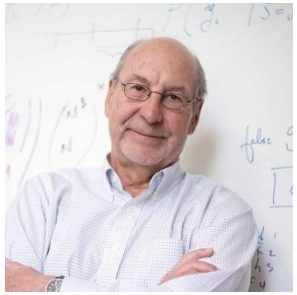


Mathematical Reasons for Going “Deep”

- A neural network with a **single hidden layer** of enough units can approximate **any continuous function** arbitrarily well

- In other words, it can solve whatever problem you’re interested in!

  (Cybenko 1998, Hornik 1989; 1991: activation function type not relevant)



Mathematical Reasons for Going "Deep"

But:

- "**Enough units**" can be a very large number. There are functions representable with a deep (and small!) network, but would require exponentially many units with a single layer.

  (e.g., Hastad et al. 1986, Bengio & Delalleau 2011)

- The proof only says that a shallow network ***exists***, it does not say how to find it.

- Evidence indicates that it is easier to train a deep network to perform well than a shallow one.

Why no Deep Learning in the 1980s?

Methodically, not a lot has changed since the 1980s. But:

- **Computers were slow** ➔ neural networks tiny, could not achieve (expected) high performance on real problems

- **Datasets were small** ➔ no large datasets that had enough information to learn the numerous parameters of (hypothetical) large neural networks

- **Nobody knew how to train deep nets.** Today, object recognition networks have > 10 layers (**or many more**). In the past, everyone was very sure that such deep nets cannot be trained. Therefore, networks were shallow and did not achieve good results.

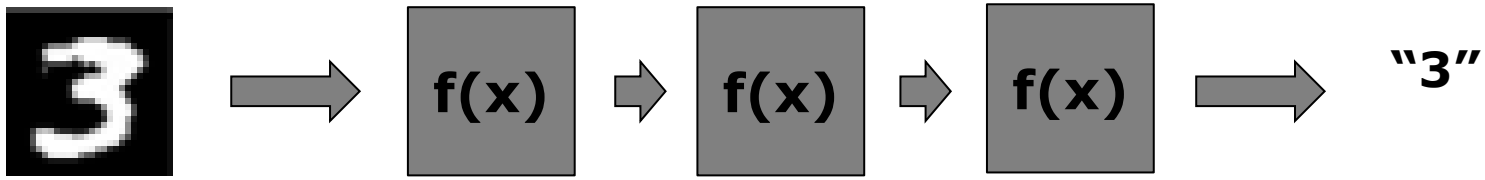
Ilya Sutskever, http://yyue.blogspot.co.at/2015/01/a-brief-overview-of-deep-learning.html

Availability of datasets

- *“Why Big Tech pays poor Kenyans to teach self-driving cars”*

- *https://www.bbc.com/news/technology-46055595*

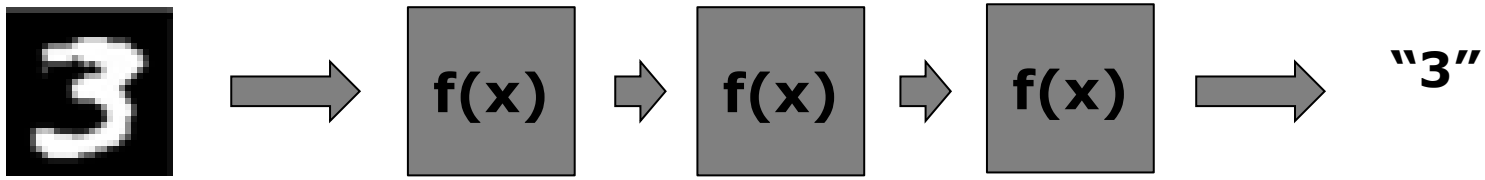
Training Deep Neural Networks



- **Training**: *heuristically find optimal weights* (cf. MLP)
  - **Randomly initialize** tuneable parameters (weights)
  - Define objective **function** telling how well the network does (on a set of training examples)
  - **Iteratively** adapt learnable parameters values to **minimize value of cost/loss function**
    (e.g. following simple gradient-based rule)

- For each step: solutions exist from the 1980s (cf. MLP)
- Recent advances since the 2000s added improvements along these steps

Popular Network Architectures

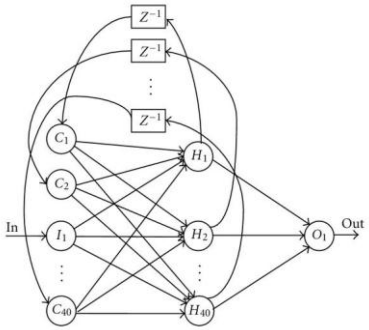


- Largest design space in employing deep learning
  - Which functions (or “layers”) to use, and in
  - Which combination / order / sequence?
    - Also: loss function, regularisation, learning rate, … (cf. MLP!)

- Several popular architectures proposed
  - Particular type of layers suited for particular problems
    - *Re-use architecture* for your specific problem, or even
    - *Re-use learned parameters* via *Transfer Learning*

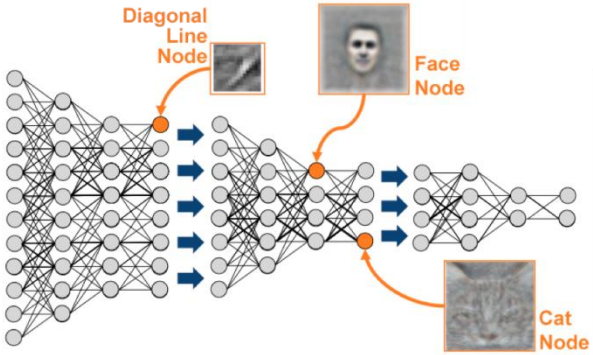
Deep Learning Architectures

- Deep (Artificial) Neural Networks (DNNs)
  - Anything with many (at least 2) hidden layers
    - E.g. a MLP
  - For any kind of data/problem



- Recurrent neural networks
  - E.g. Long short term memory network (LSTM) (1997!)
  - Mostly for time series / sequences

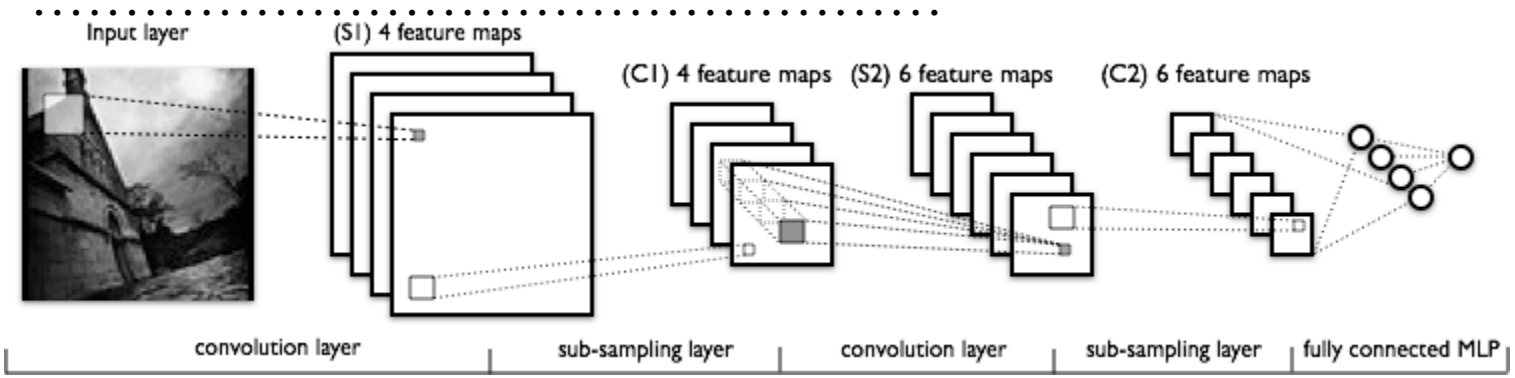
- Convolutional neural network
  - Mostly for image analysis (+audio, …)



Outline

- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

Convolutional Neural Network (CNN)



- Combines three types of layers:
  - Convolutional layer: performs 2D convolution of 2D input with multiple learned 2D kernels
  - Subsampling layer: replaces 2D patches by their maximum (“max-pooling”) or average
  - Fully-connected layer: computes weighted sums of its input with multiple sets of learned coefficients (~MLP)
- Applies nonlinear function after each linear operation

Face Recognition with CNNs

- Convolutional neural networks learn layers of “features”, from coarse to more detailed

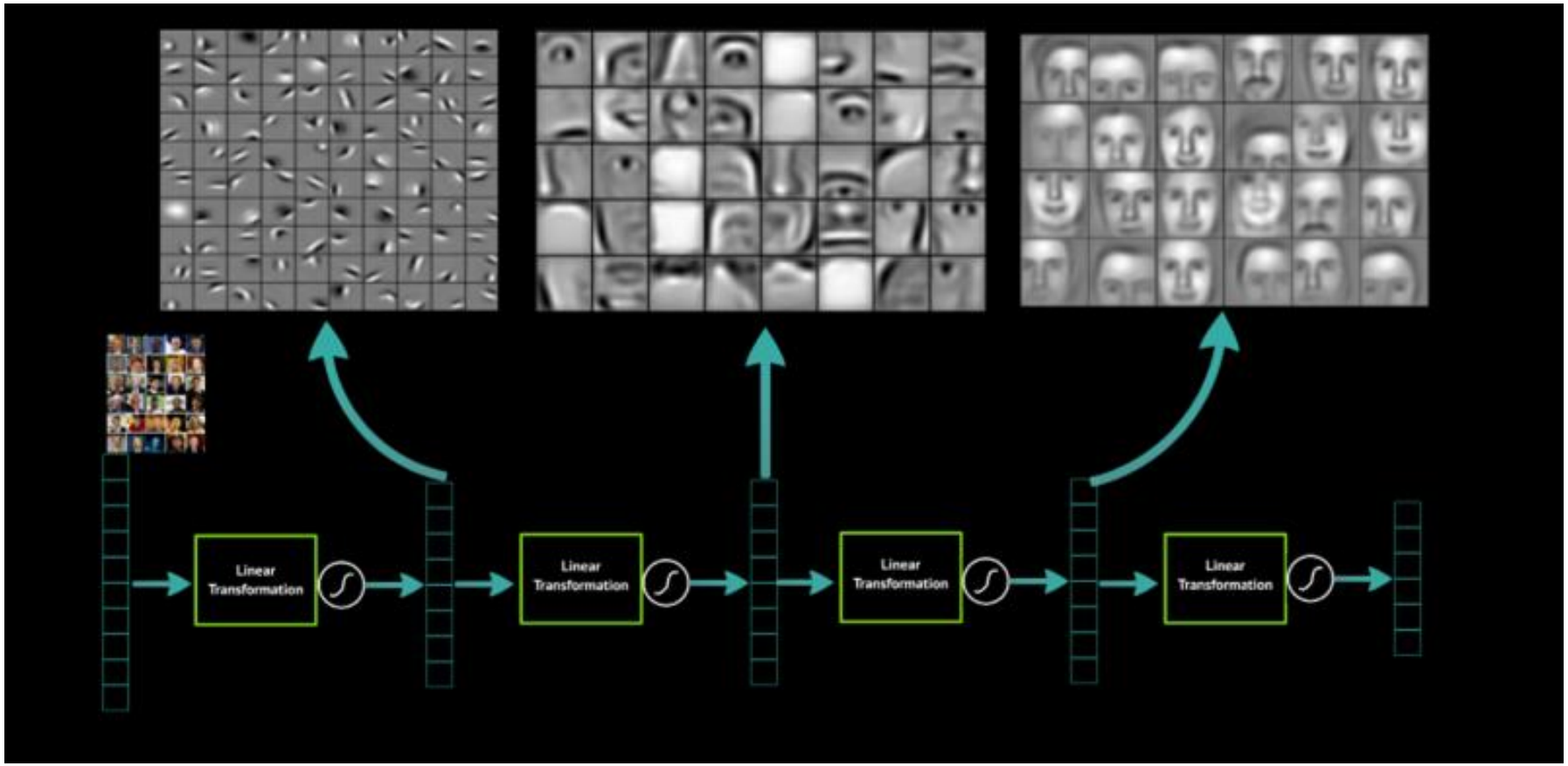
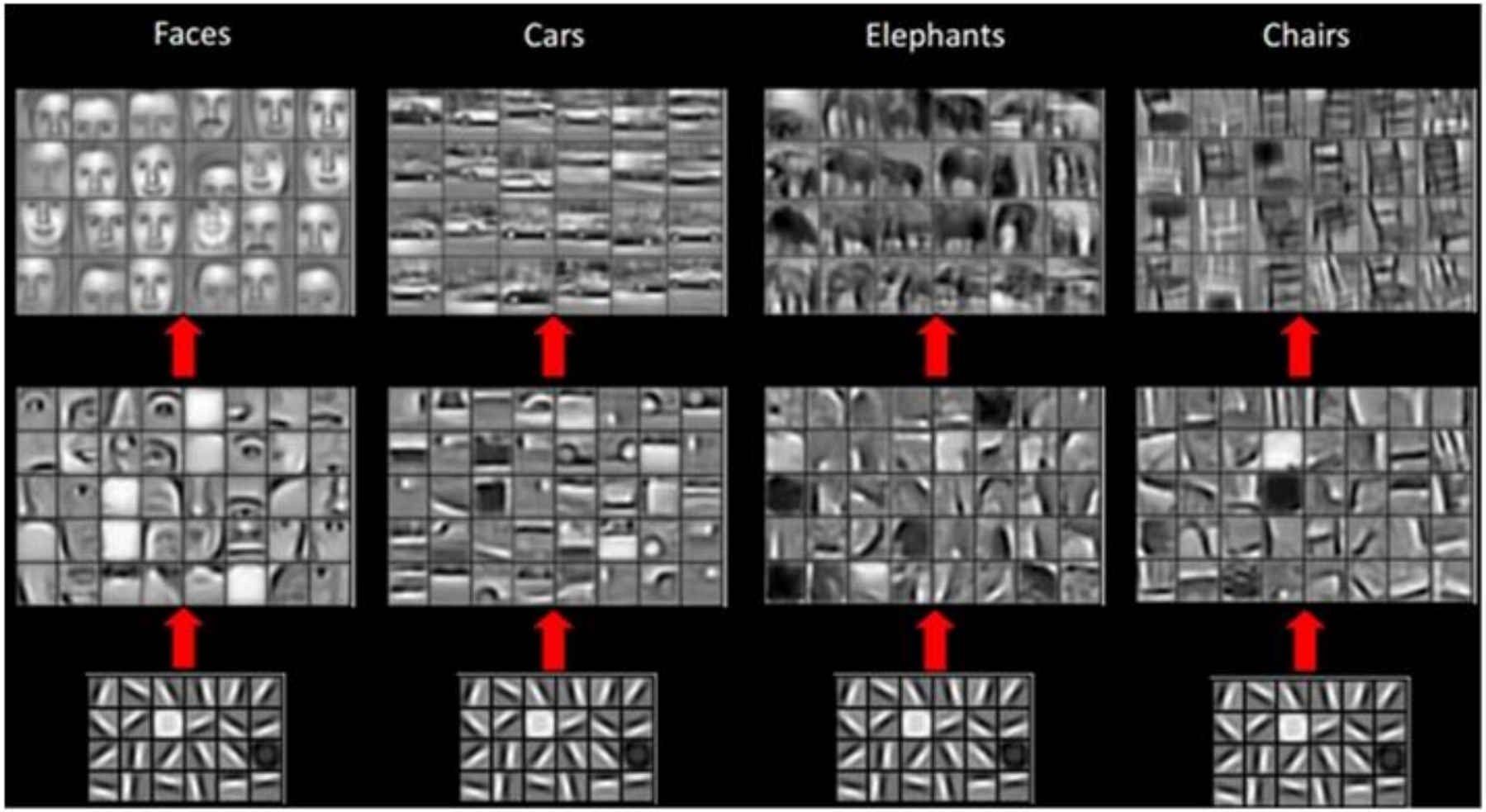


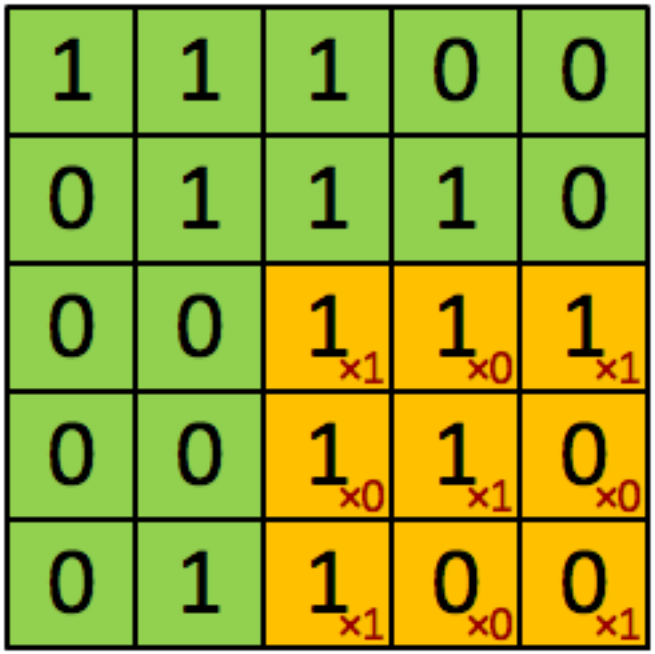
Image Object Recognition with CNNs



Src: http://stats.stackexchange.com/questions/146413/why-convolutional-neural-networks-belong-to-deep-learning

Convolutional Layer

- Convolution – step-wise multiply window of input matrix (tensor) with a kernel matrix
  ➔ Produce a new matrix (tensor) output
  - Convolution only sees a part of the input (“receptive field”)



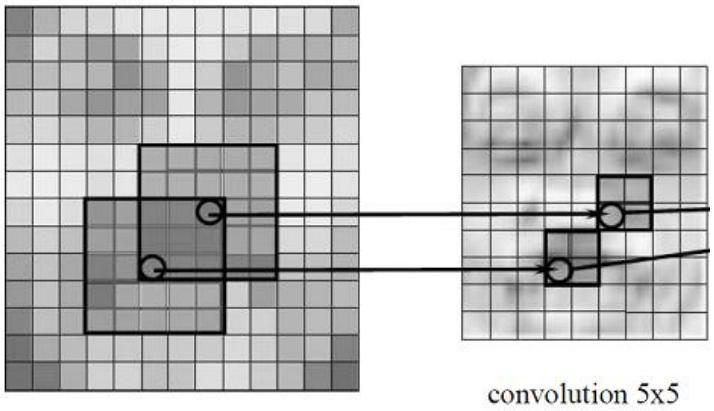
Image

4	3	4
2	4	3
2	3	4

Convolved Feature

Convolutional Layer

- What does a convolution do?
  - Applies local filter kernels (matrix / tensor)
  - *What does it remind you of?*
    - Similar concept in e.g. image processing
  - Kind-of feature extraction
    - E.g. Edge detection, etc...



convolution 5x5

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Outline

- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

What is a Convolution?

6 x 6 image

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$\circledast$

3 x 3 filter

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

$=$

4 x 4 result

-5			

3x1 + 0x0 + 1x-1 +
1x1 + 5x0 + 8x-1 +
2x1 + 7x0 + 2x-1 =
-5

What is a Convolution?

6 x 6 image

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$\circledast$

3 x 3 filter

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

$=$

4 x 4 result

-5	-4		

$$0x1 + 1x0 + 2x\text{-}1 +$$
$$5x1 + 8x0 + 9x\text{-}1 +$$
$$7x1 + 2x0 + 5x\text{-}1 =$$
$$-4$$

What is a Convolution?

6 x 6 image

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$\circledast$

3 x 3 filter

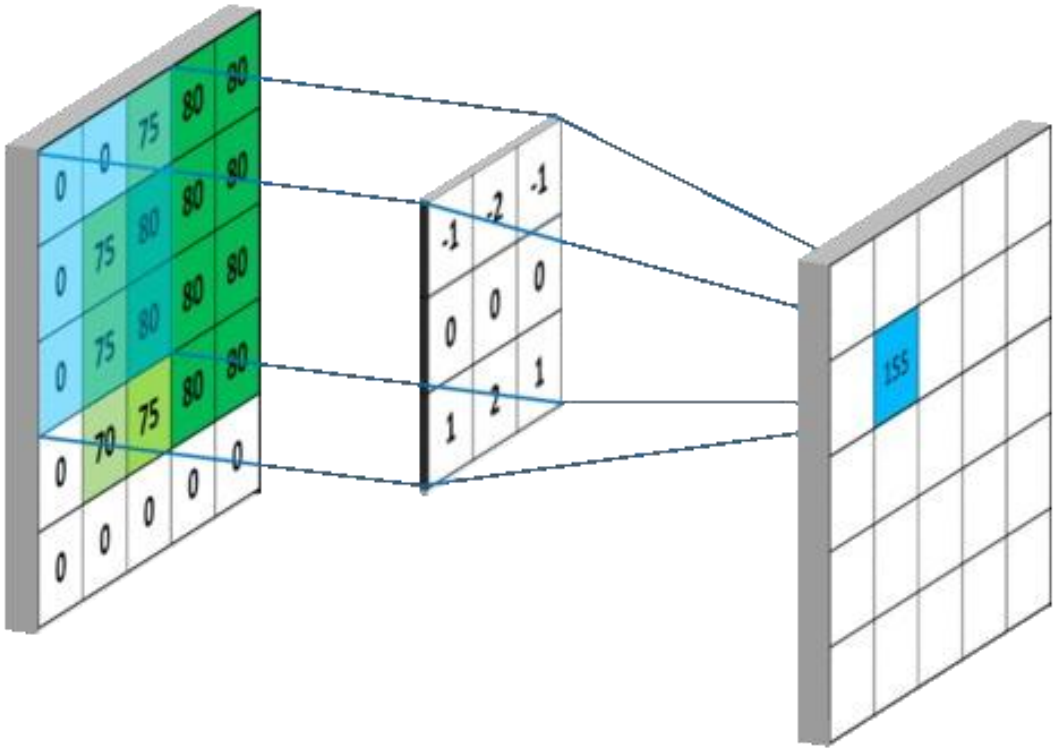
1	0	-1
1	0	-1
1	0	-1

$=$

4 x 4 result

-5	-4	0	8

What is a Convolution?



- https://mlnotebook.github.io/post/CNN1/

Convolutions: Padding

- Convolution result smaller than original image

$$n \times n \circledast f \times f \Rightarrow (n - f + 1) \times (n - f + 1)$$

$$6 \times 6 \circledast 3 \times 3 \Rightarrow (6 - 3 + 1) \times (6 - 3 + 1) = 4 \times 4$$

- Edge & corner pixels used much less than others

➔ Pad image with zeros at the edges

6 x 6 image with 1 pixel padding

0	0	0	0	0	0	0	0
0	3	0	1	2	7	4	0
0	1	5	8	9	3	1	0
0	2	7	2	5	1	3	0
0	0	1	3	1	7	8	0
0	4	2	1	6	2	8	0
0	2	4	5	2	3	9	0
0	0	0	0	0	0	0	0

$\circledast$

3 x 3 filter

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

=

6 x 6 result

-5					

Convolutions: Padding

- With *p* pixels padding: output size defined as:

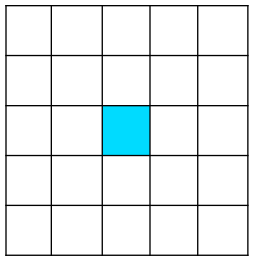
$$(n + 2p - f + 1) \times (n + 2p - f + 1)$$

- How to compute p to have output size *n x n*?

$$p = \frac{f - 1}{2}$$

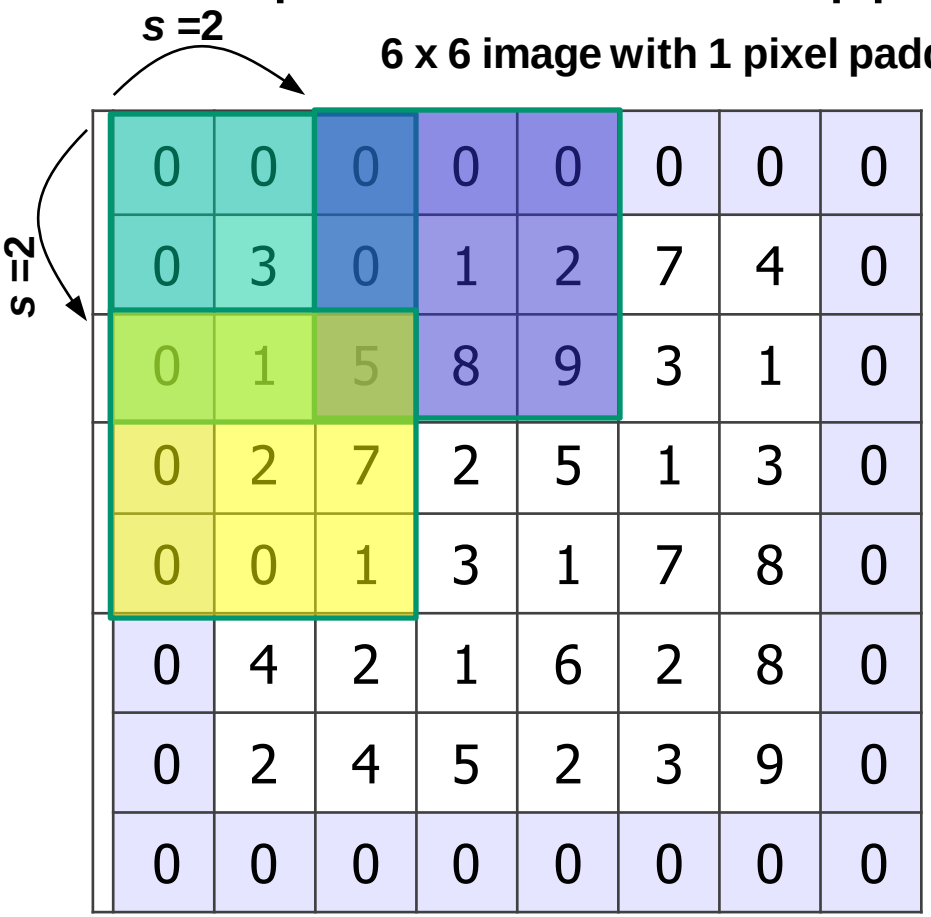
$$\begin{aligned} 3 \times 3 &\Rightarrow p = 1 \\ 5 \times 5 &\Rightarrow p = 2 \\ 7 \times 7 &\Rightarrow p = 3 \end{aligned}$$

- f is by convention odd
  - Such that p is well-defined
  - Such that the filter has a central pixel



Convolutions: Stride

- We can choose a stride parameter s to define the step size between applications of the filter



3 x 3 filter

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

$\circledast$ =

3 x 3 result

1	-6	
13		

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

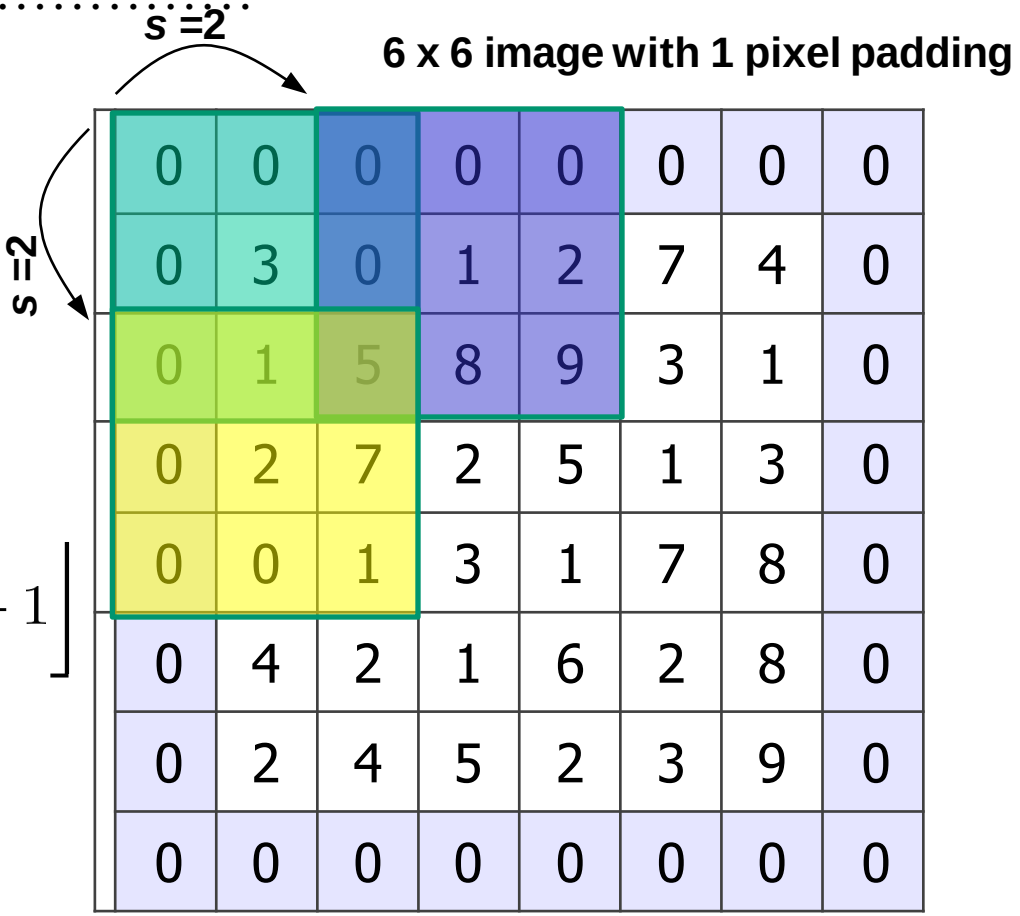
Convolutions: Stride

- *Output size?*
  - *Remember padding:*

$$(n + 2p - f + 1) \times (n + 2p - f + 1)$$

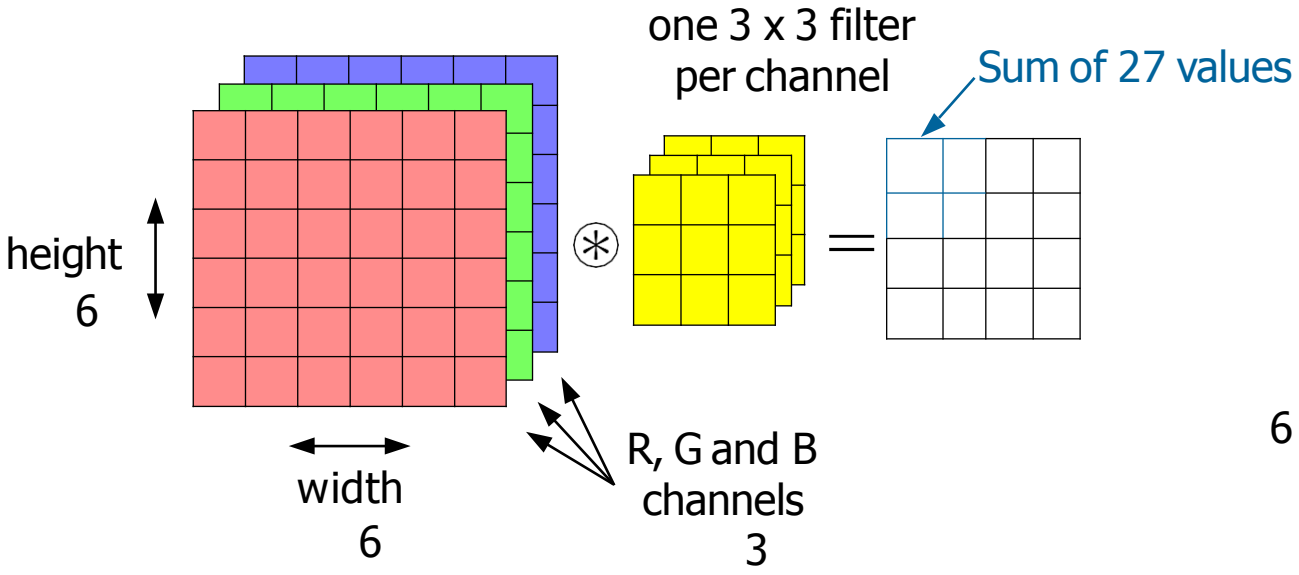
$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor$$

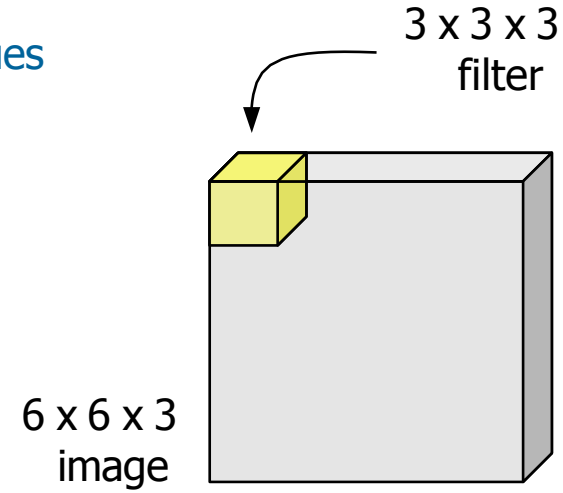
$$\frac{6 + 2 \times 1 - 3}{2} + 1 = 3$$



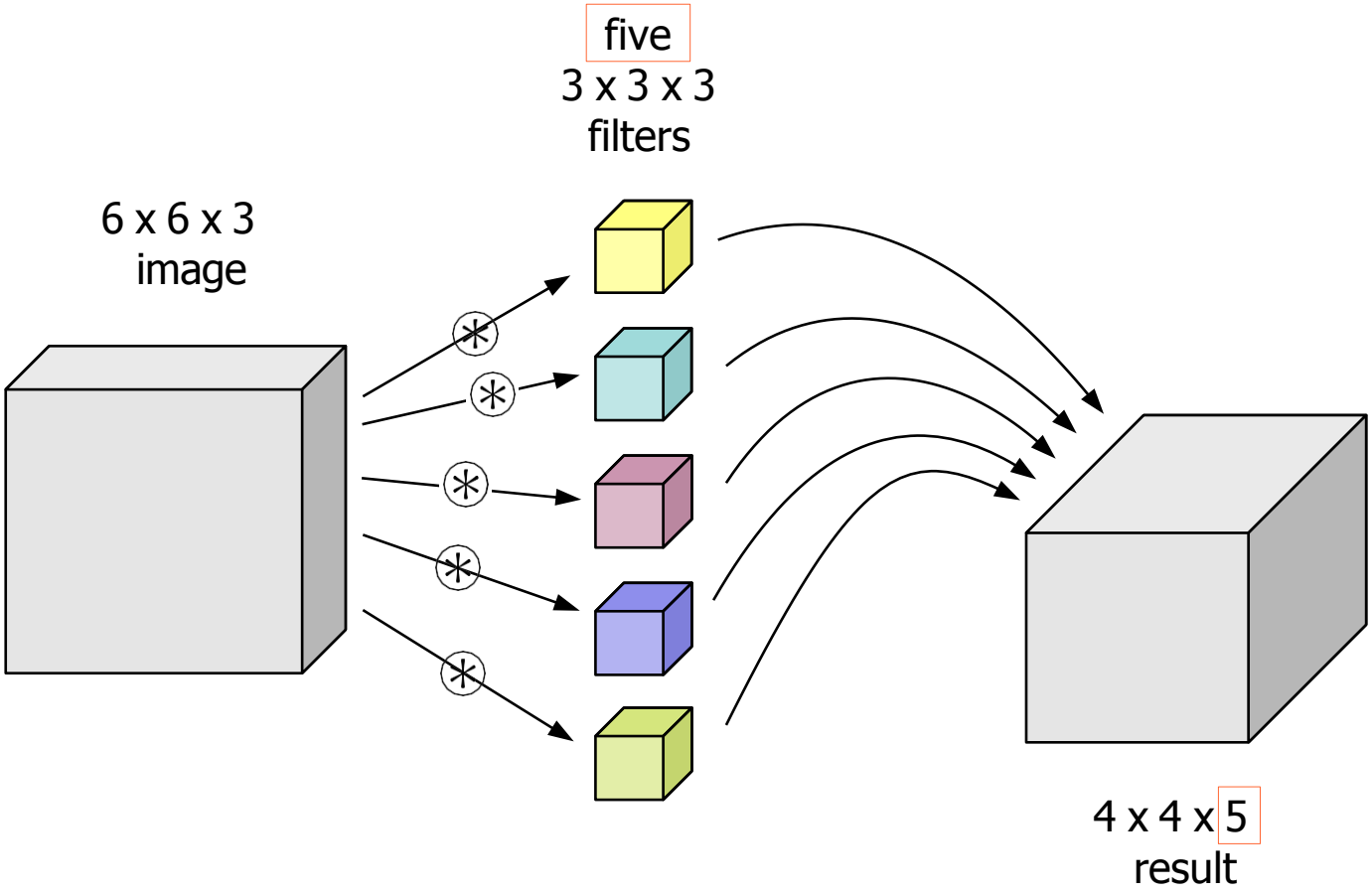
Convolutions over Volumes

- Input images are typically RGB (3 channels)
  ➔ volumes (tensors) of shape (width x height x 3)
- Filters should also have 3 channels
  ➔ volumes (tensors) of shape (f x f x 3)
  – Often drawn as 3D volumes





Convolutions over Volumes



No padding, stride 1

Outline

- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

Why is Visualisation Interesting?

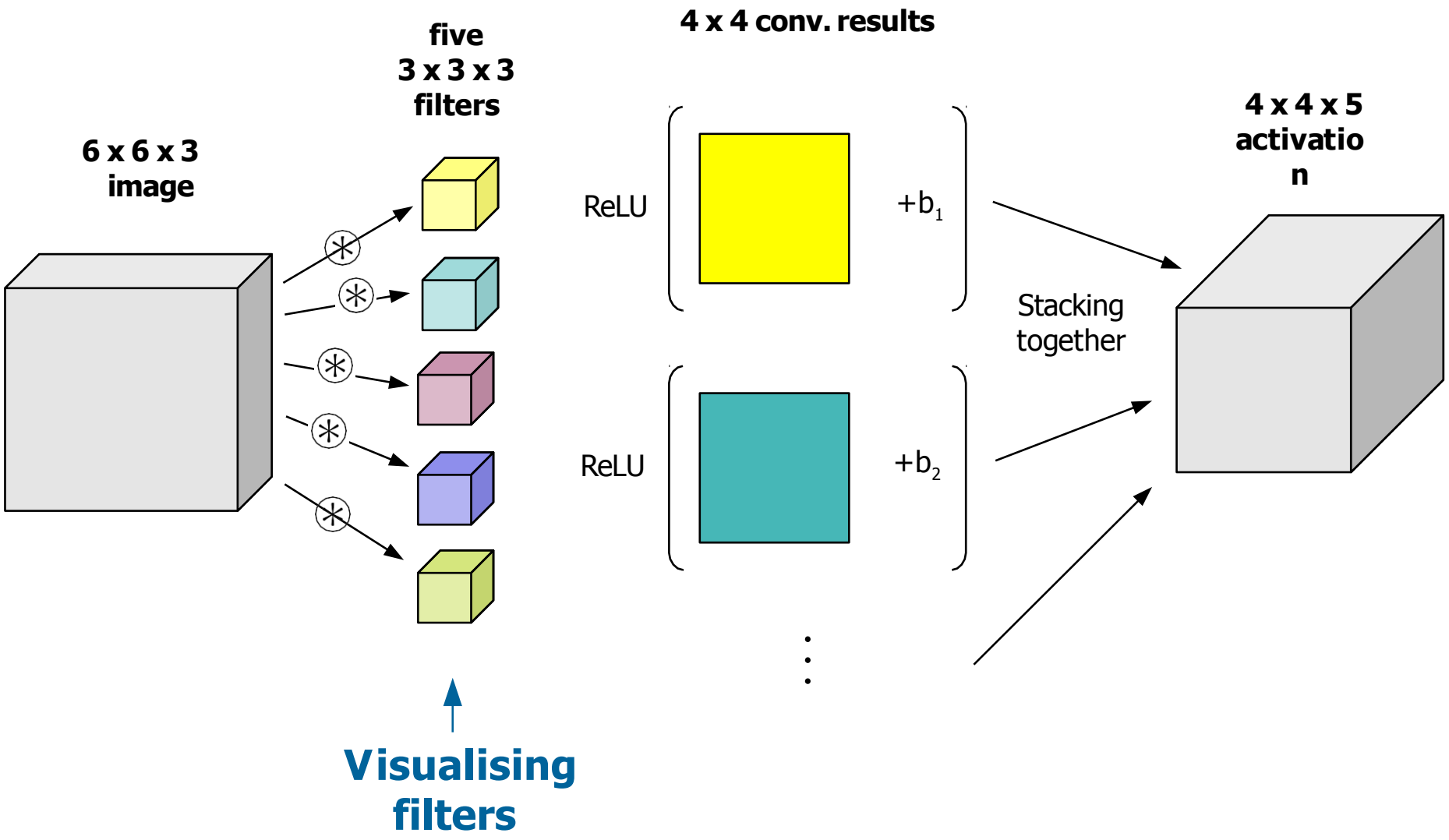
- By visualising “inner processes” of CNNs:
  Try to understand what is happening
  - Sort of “confirmation” that the network is capable of meaningful image interpretation

- Might trigger ideas to improve the network

  ➔ Not to directly improve **training** - try to understand, and improve network architecture

- Can use inner representations of images for other purposes (e.g. surrogate models, …)

Visualising Filters



Visualising Filters (First Layer)

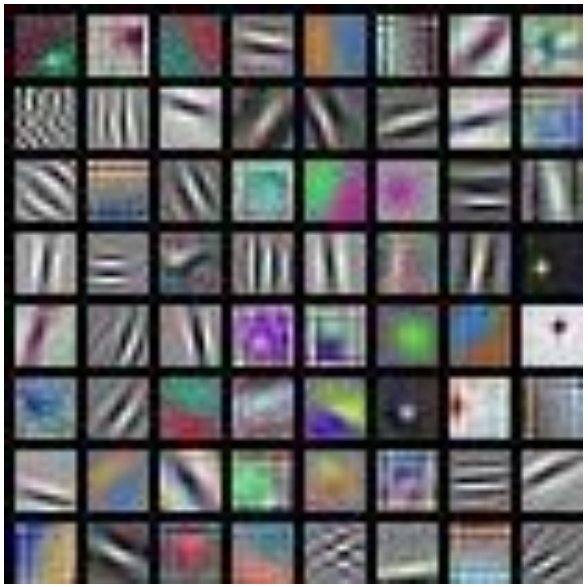
- Looking at filters after training *as if they were images themselves*

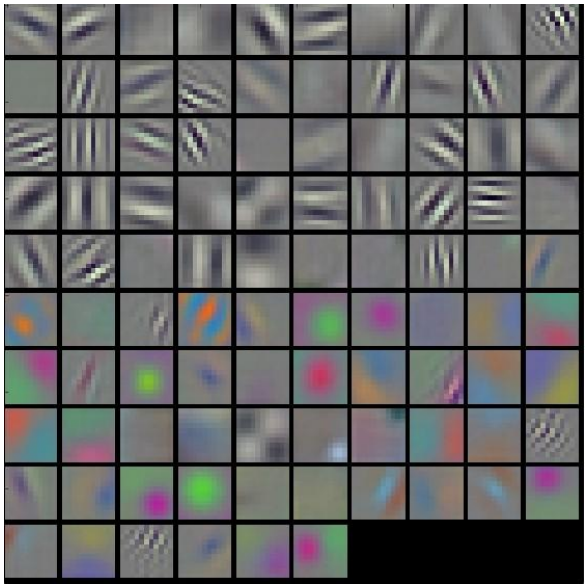
- Shows what CNN is “looking for” in first step
  - Edge detection filters at various angles and “sizes”
  - Color change detectors
  - Spot detectors
  - …

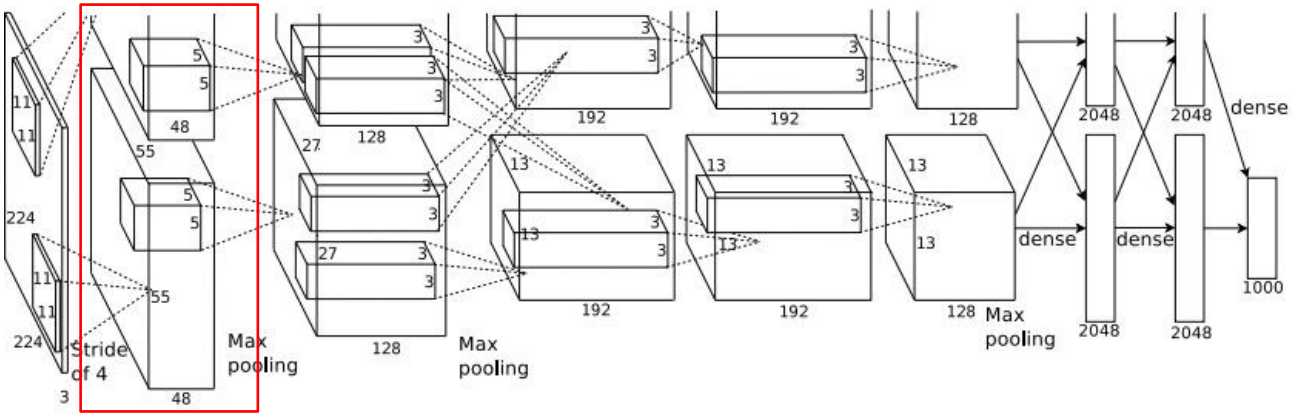
- Different CNN architectures end up having similar filters in first layer!

Visualising Filters (First Layer)

AlexNet:

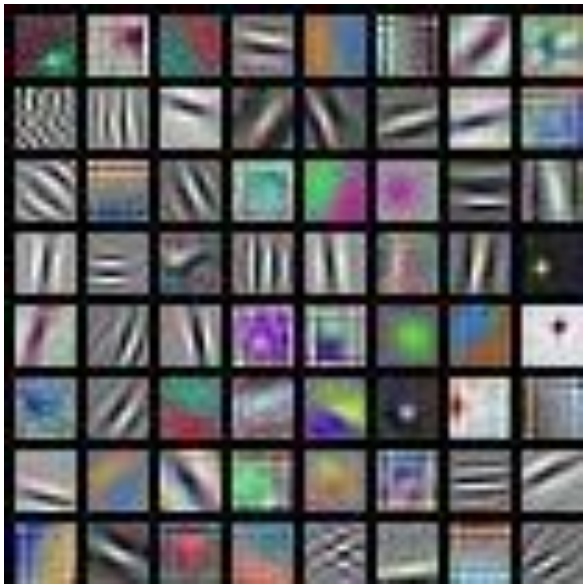




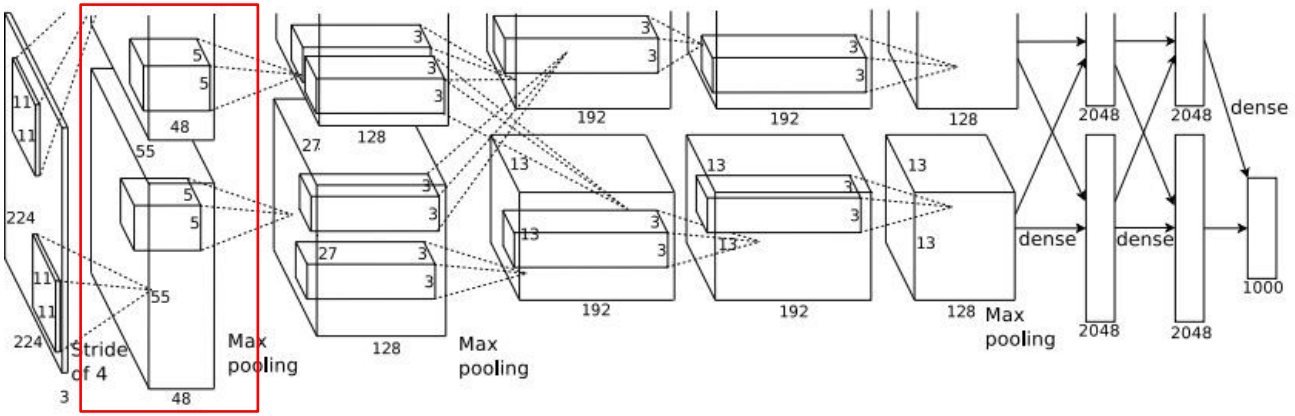


Visualising Filters (First Layer)

AlexNet:



ResNet-18,
ResNet-101,
DenseNet-121



Visualising Filters (Deeper Layers)

- At deeper convolutional layers
  - *More difficult to interpret the filters directly. Why?*
  - Inputs to these already are an intermediate result, rather than an image
    - (e.g., after the first convolution, ReLU, pooling)
  - Many channels
    - Cannot visualise them as RGB images directly
- Still see some spatial structure

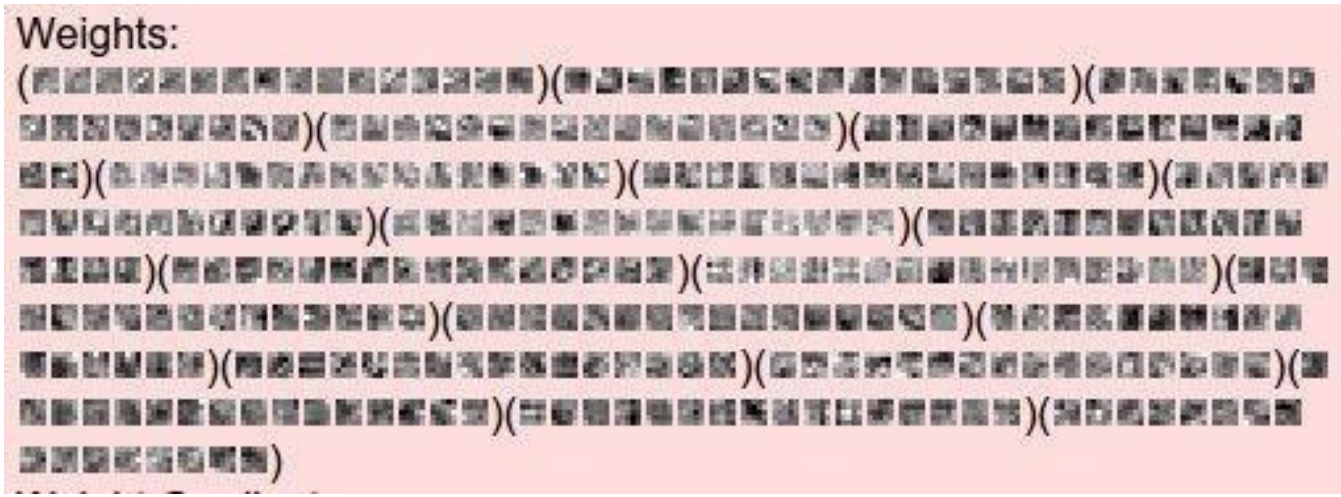
Visualising Filters (Deeper Layers)

Layer 1 weights
16 x 7 x 7 x 3

RGB

Layer 2 weights
20 x 7 x 7 x 16

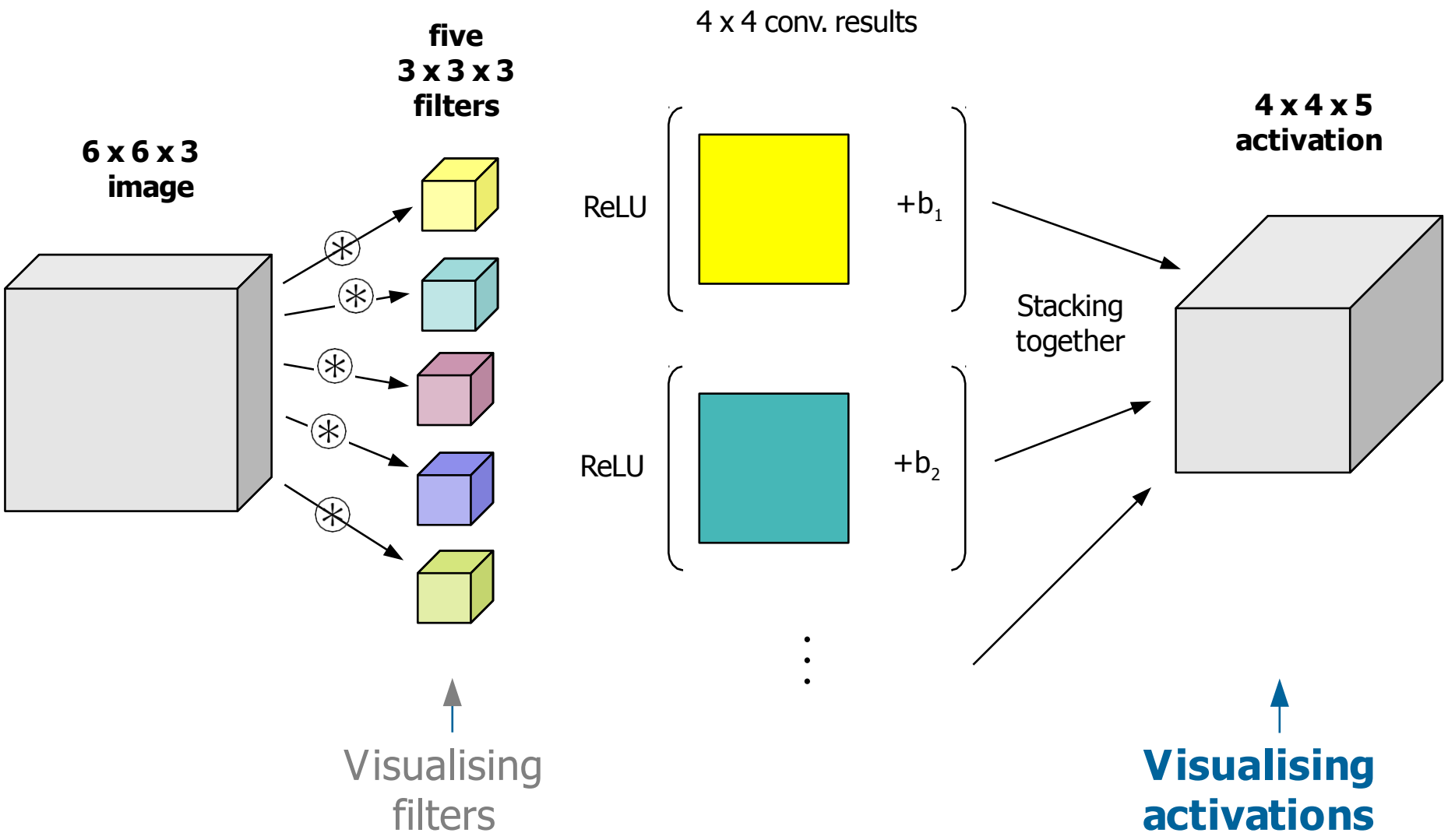
??



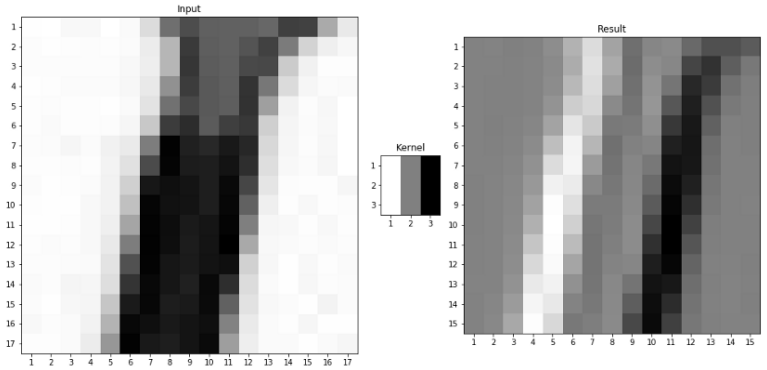
Layer 3 weights
20 x 7 x 7 x 20

??

Visualisation of Activations



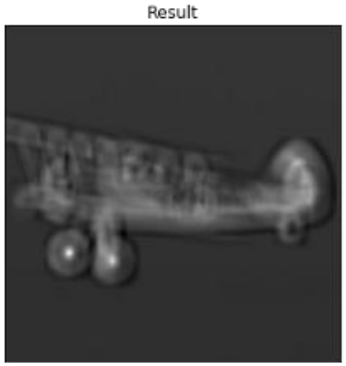
Visualisation of Activations



Kernel

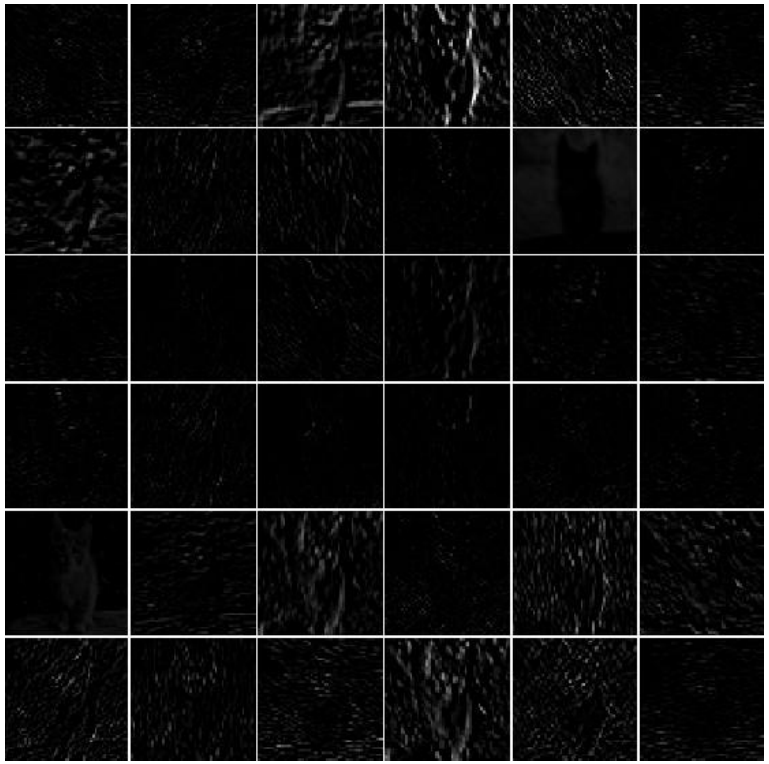
Darkest pixels: **smallest** values of kernel operation
Brightest pixels: **highest** values

Kernel

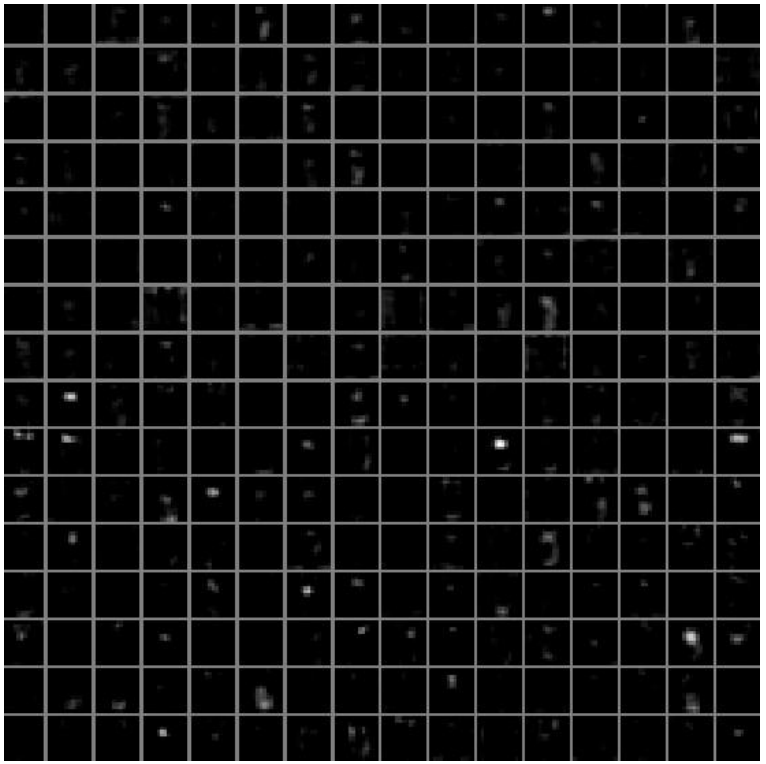


Kernel

Visualisation of Activations

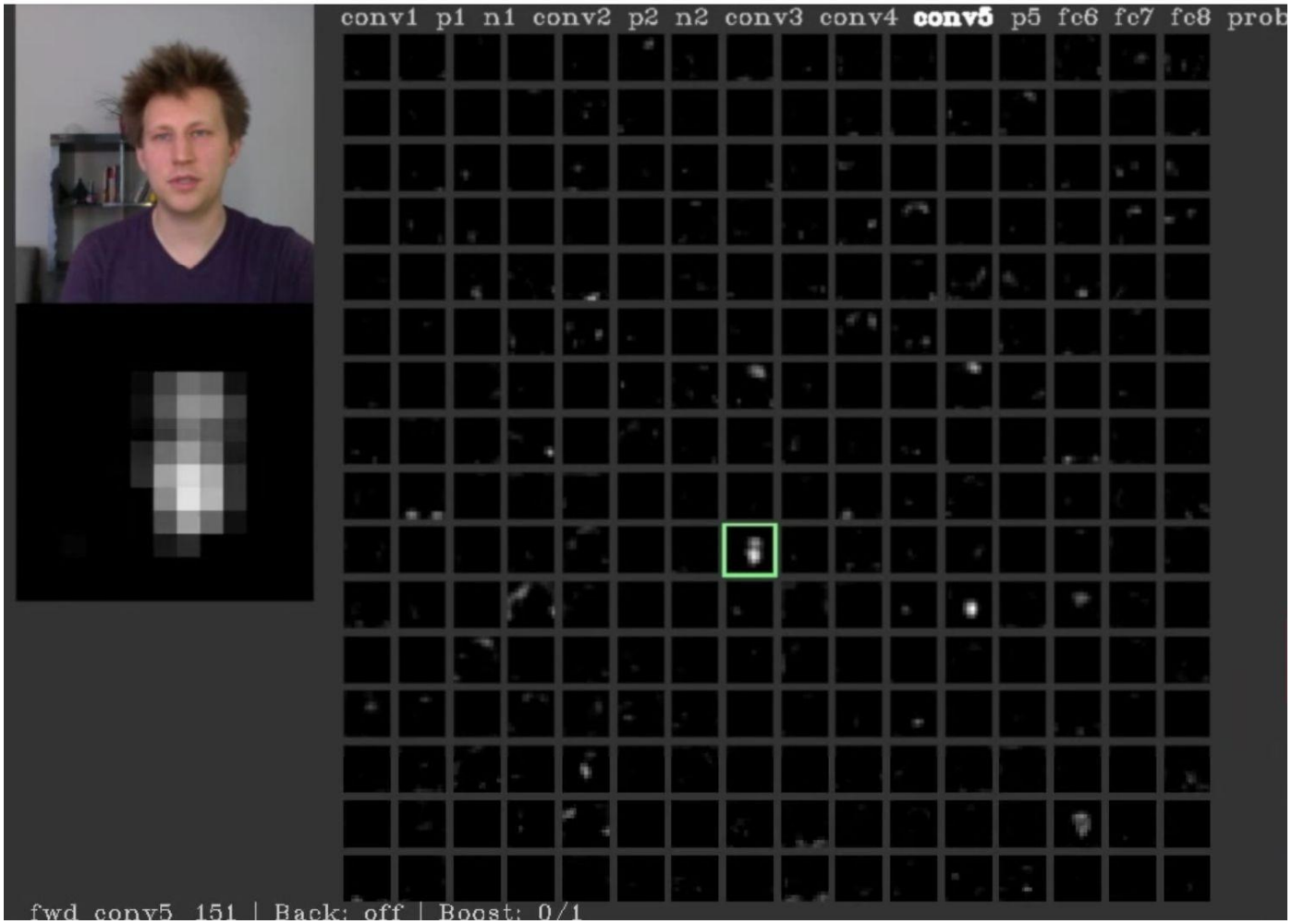


1st Layer



5th Layer

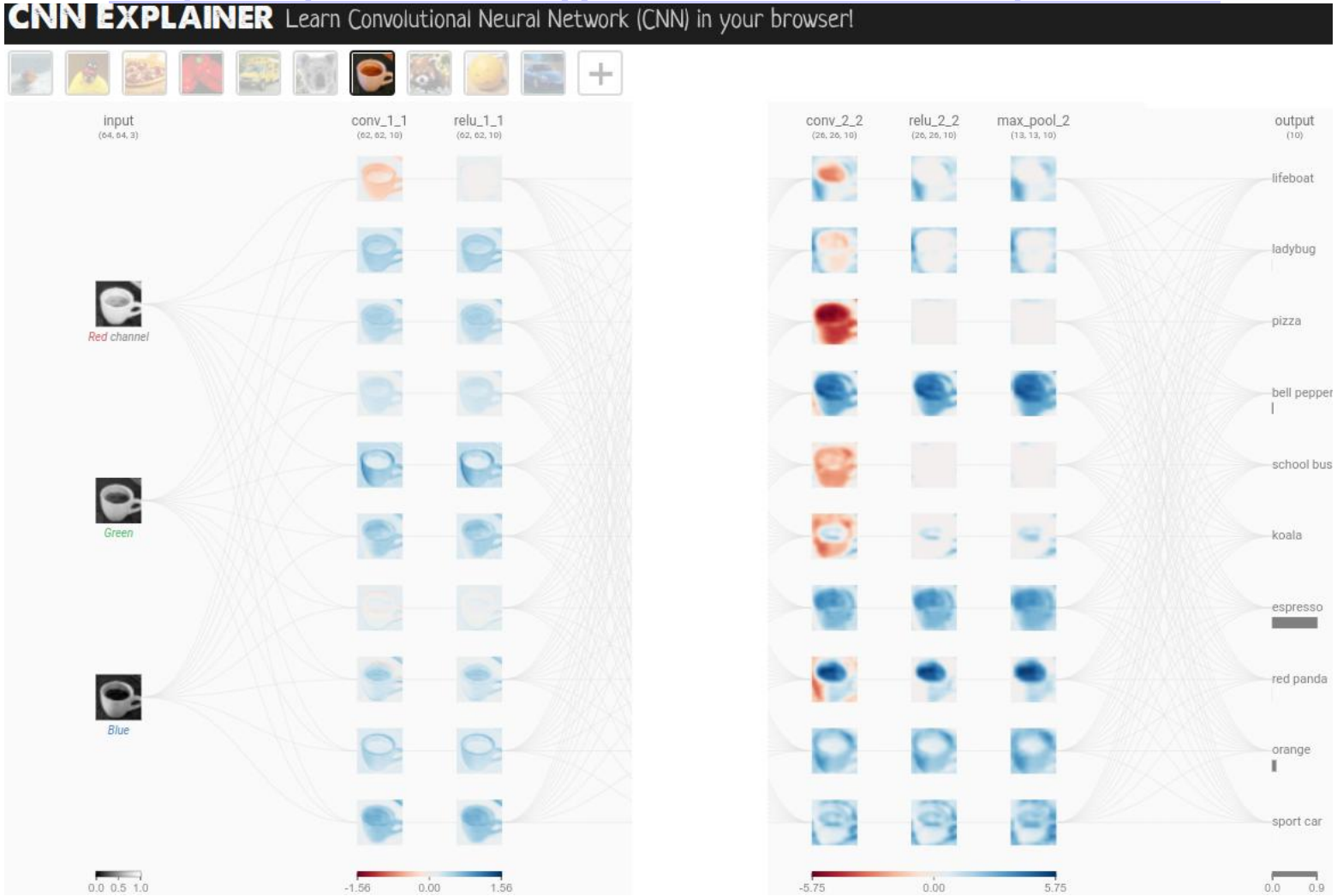
Demo



- http:/ yosinski.com/deepvis

Demo

- https://poloclub.github.io/cnn-explainer/



Visualising CNNs

- Used e.g. actively for Explainable AI (XAI)

- Approaches:
  - Anchors
  - Prototype images
  - Counterfactuals/adversarial examples, ...

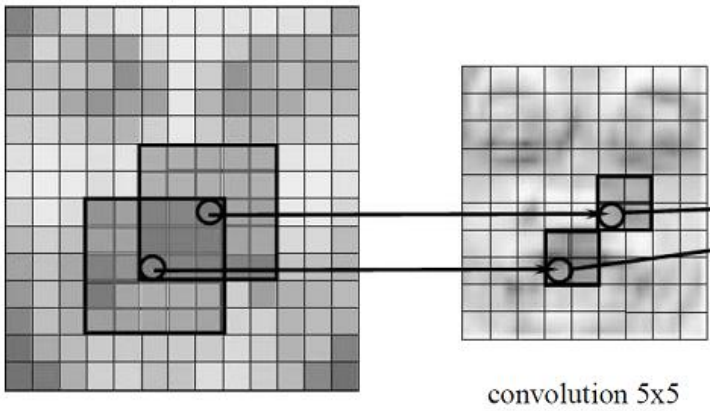
- *(A bit) more on that (a bit) later...*

Outline

- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

Convolutions vs Image processing ?

- Both apply local filter kernels (matrix)
- CNN: kernels are **learned**
  - And **not predefined** as they would be with feature extraction (or in **image processing** software)



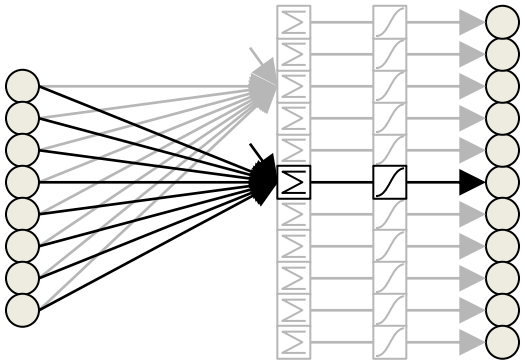
convolution 5x5

Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Fully Connected vs. Convolutional Layer

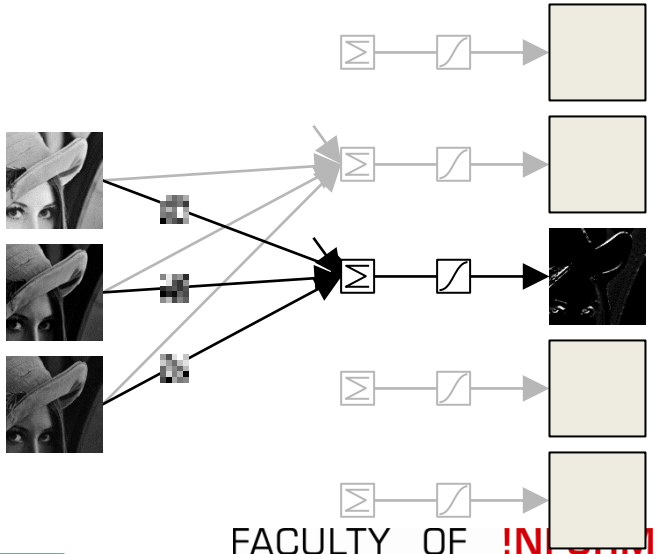
Fully-connected layer (~MLP)

- Each **input** is a **scalar** value
- Each **weight** is a **scalar** value
- Each output is the sum of inputs **multiplied** by weights.



Convolutional layer

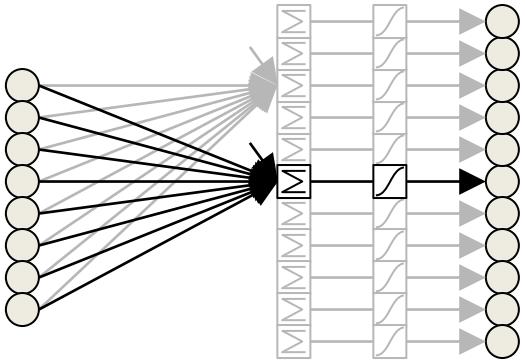
- Each **input** is a **tensor**
- Each **weight** is a **tensor**
- Each output is the sum of inputs **convolved** by weights.



Fully Connected Layer

Fully-connected layer:

- *Each **input** is a **scalar** value*
- *Each **weight** is a **scalar** value*
- *Each output is the sum of inputs **multiplied** by weights*



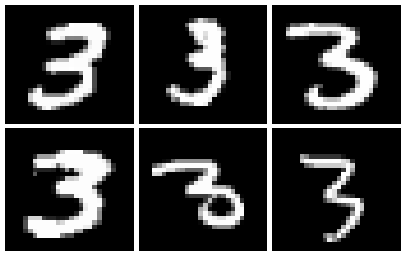
→ Swapping two inputs does not change the task – we can just swap the weights as well!

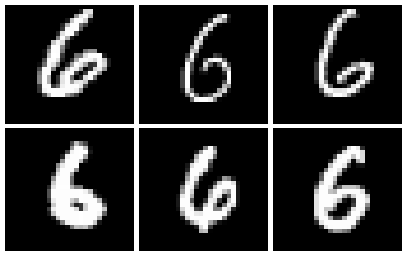
Example: distinguish *iris setosa*, *iris versicolour* and *iris virginica*

Input: (sepal length, sepal width, petal length, petal width)

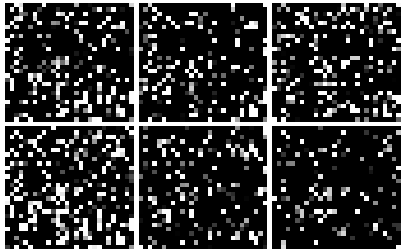
Equivalent input: (sepal width, petal length, sepal length, petal width)

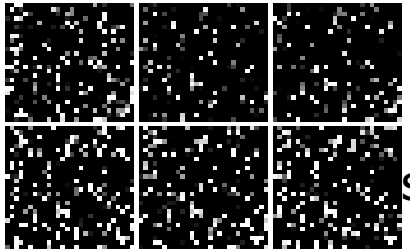
Fully Connected Layer



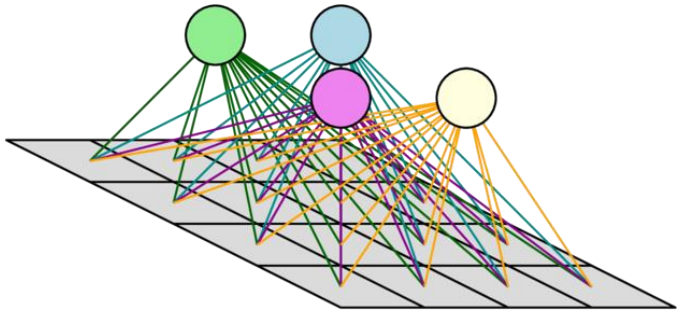


Original input





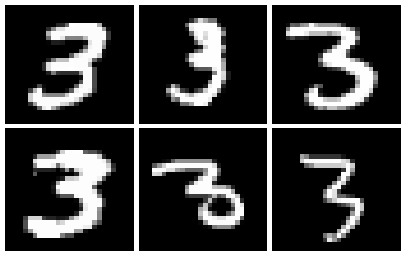
Permuted input
(not random ➔ same permutation for each input!)

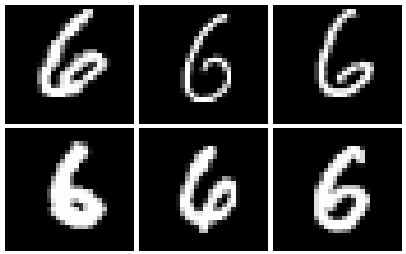


In a **fully**-connected layer, every input is connected with the following layer.

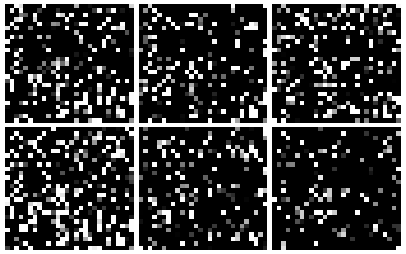
Consequently, the **order** (spatial arrangement) **does not matter!**

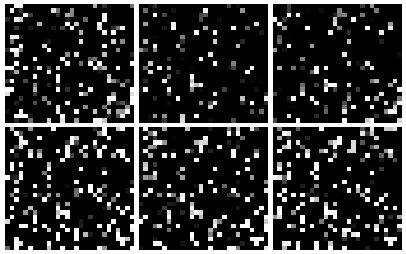
Fully Connected vs. Convolutional Layer





Original input



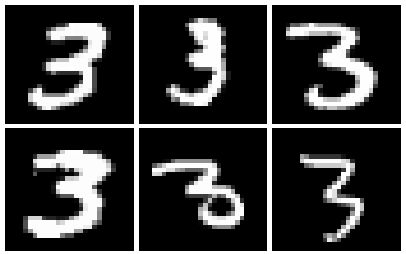


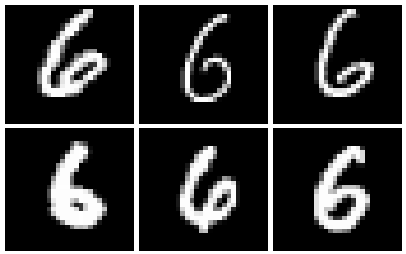
Permuted input

- For a fully-connected network: task is the same
- **For humans**, in permuted version: **impossible** to recognize the numbers!

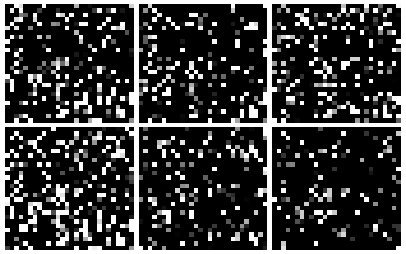
➔ But we want the machine to learn the same concepts that we know

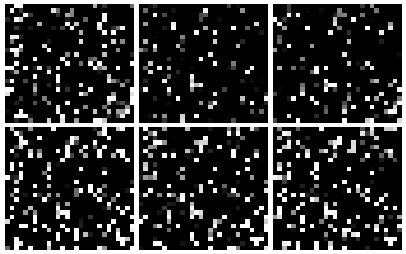
Fully Connected vs. Convolutional Layer





Original input



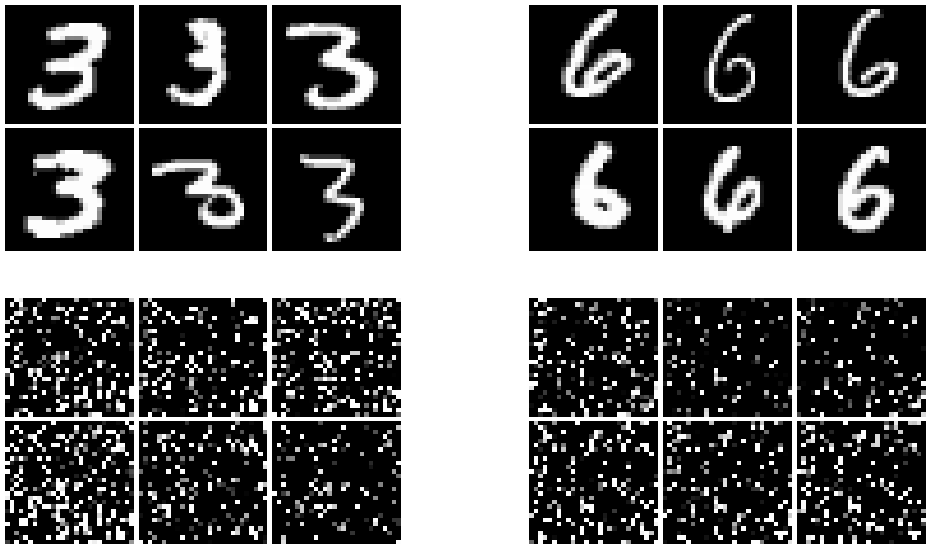


Permuted input

- Humans make use of **spatial layout** of visual data
  – We (our eyes) do not see a full image at once, but only a **local neighbourhood** around a focal point

➔ Connect each neuron to **small part of the input** image only

Fully Connected vs. Convolutional Layer



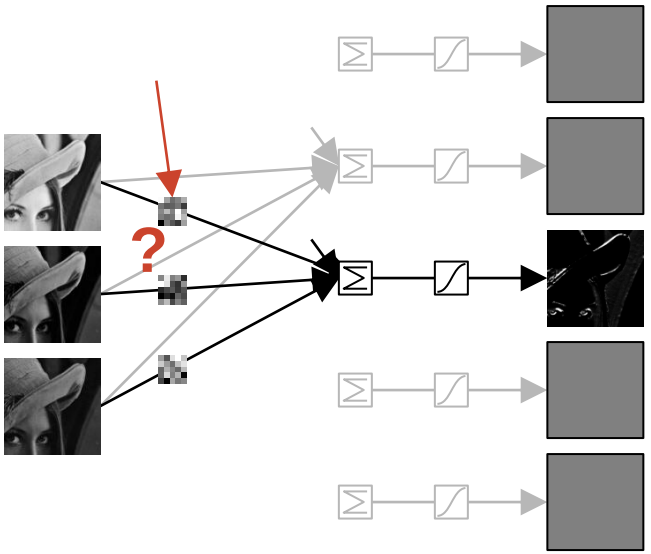
- Humans reuse “feature detectors”:
  – We do not have separate eyes to look at the upper and lower part of an image
  –Apply same **local feature detectors** to different positions

➔ **Share weights** between neurons applied to different parts of the input image

Convolutions: Kernel sizes

Which size for kernels?

(“receptive fields“)



3x3 kernel: Can only see small structure

7x7 kernel: Sees more, but more weights

224x224 kernel: Possibly sees full input

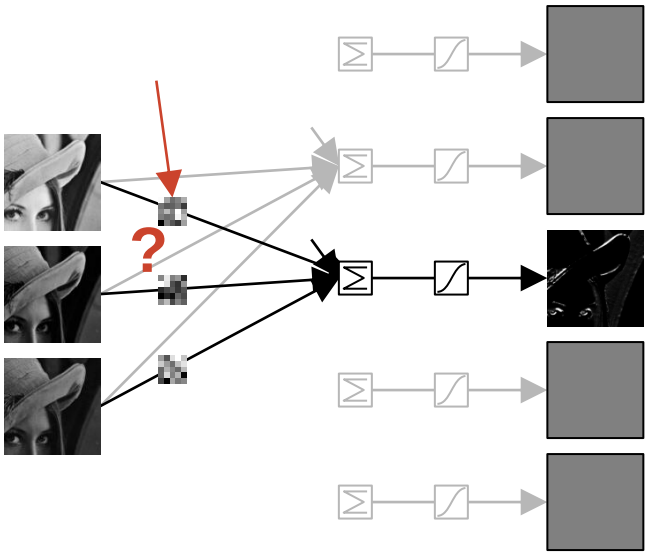
Equivalent to fully-connected then.

➔ Sweet spot somewhere in between?

Convolutions: Kernel sizes

Which size of kernels?

(“receptive fields“)



3x3 kernel: Can only see small structure

7x7 kernel: Sees more, but more weights

224x224 kernel: Possibly sees full input

Equivalent to fully-connected then.

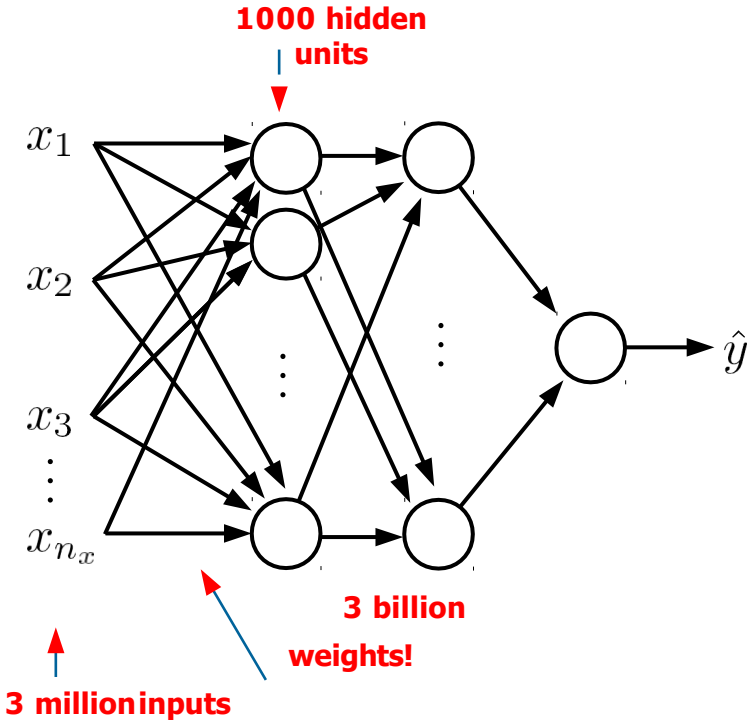
Stack of 3x3 convolutions: Three of those see as much as a 7x7 convolution, with 45% fewer weights (27/49) and two additional nonlinearities. Win-win situation!

→ **Motivation for going deeper!**

Motivation for Convolution (vs MLP)

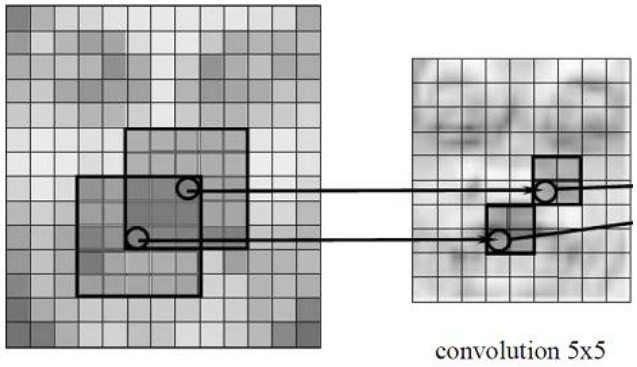
- Scalability of parameters

1000 x 1000
pixels
3 colorchannels



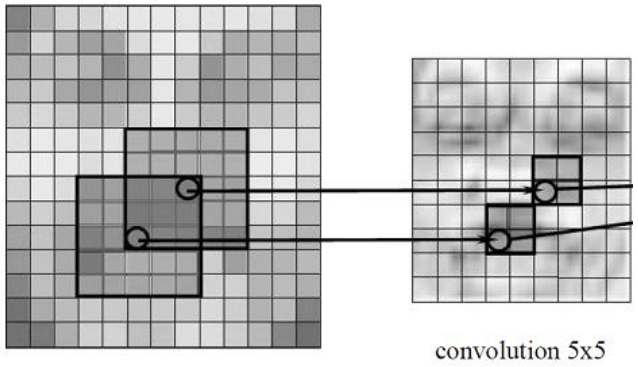
- Strong correlations in local pixel neighborhoods ➔ use that
- Tolerate image shift, rotation, etc.
  - Instead of a fixed “role” of every specific input pixel

Convolutions: Summary



- Input permutation **does** make a difference
- Also for the next layer: **Spatial layout is preserved**
- Can process large images with **few learnable weights**
- Weights are required to be applicable over the full image

Convolutions: Summary



Advantages:

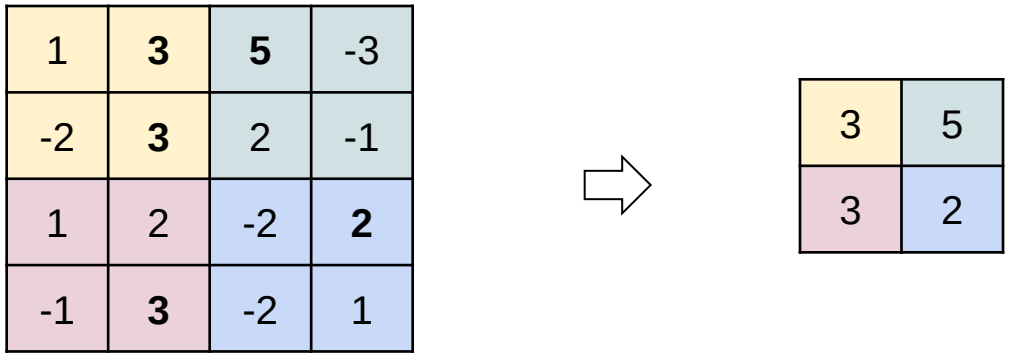
- Fewer parameters
- Faster learning
- Strong regularizer
- Learns spatial artefacts, i.e. repeating objects
- Less prone to overfitting!

Outline

- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

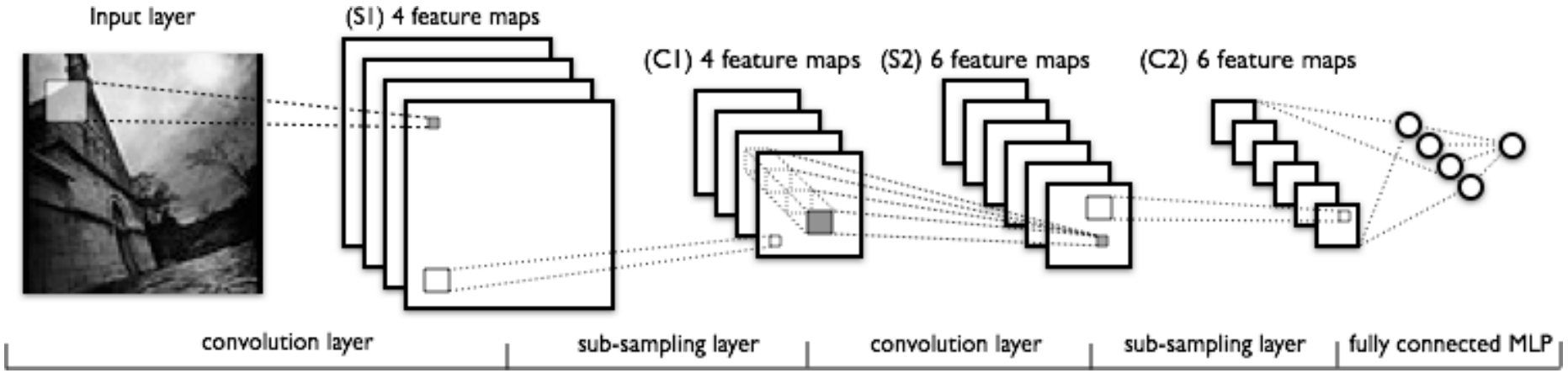
Pooling Step

- Second very important aspect of a CNN
  - (also called subsampling or downsampling)
  - A **pooling layer** reduces the size of feature maps (i.e. output of a CNN layer and thus the input to the next layer)



Max pooling: take the max. activation across small regions (e.g. 2x2, as in the example above)

CNN: Complete architecture example



a.k.a. Max-Pooling

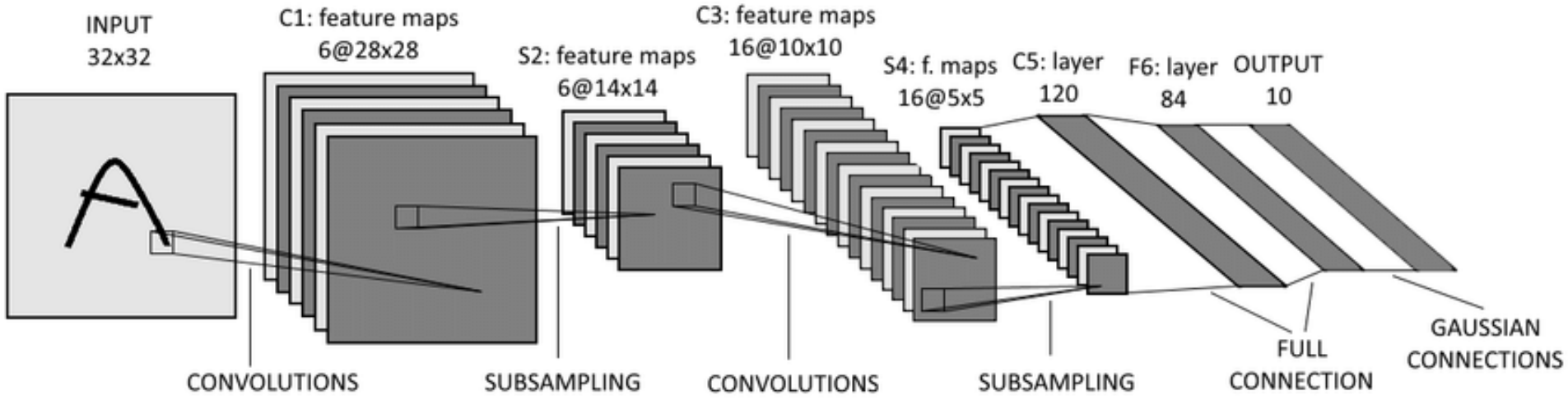
a.k.a. Max-Pooling

- Convolutional layers: local feature extraction
- Pooling layers: data reduction

- Fully-connected layers: integrate information over full input, produce output

CNN Architectures: LeNet

- Yann LeCunn, **1998 (!)**
  - 14,000 citations of the paper ☺
  - Arguably the first convolutional neural network



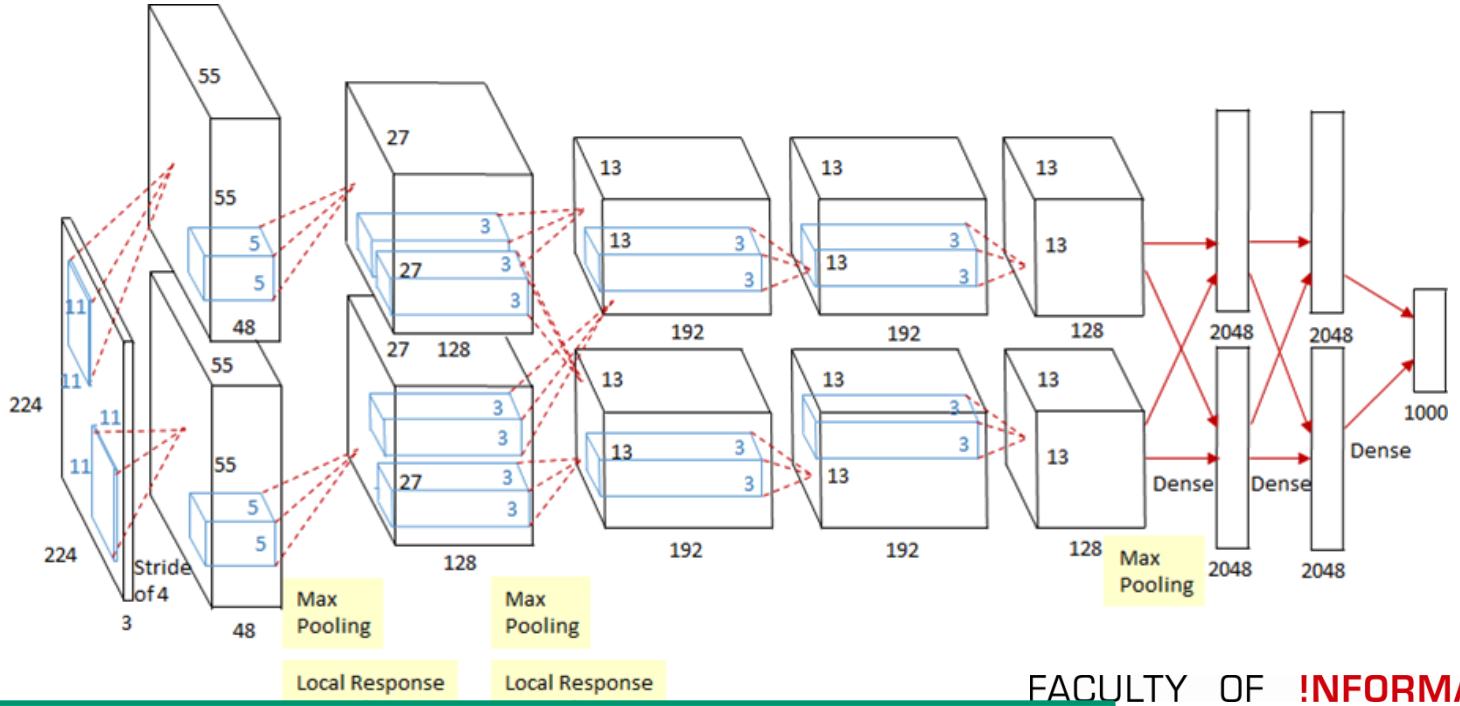
LeNet-5 Parameters:

C1: (5 5+1) 6 = 156 + S2: 6 2 = 12 + C3: (((5 5 (3|4|6)+1) 16) = 1516 + S4: 16 2 = 32 + C5: (5 5 16+1) 120 = 48120 + F6: 84 (120+1) = 10164

= 60,000

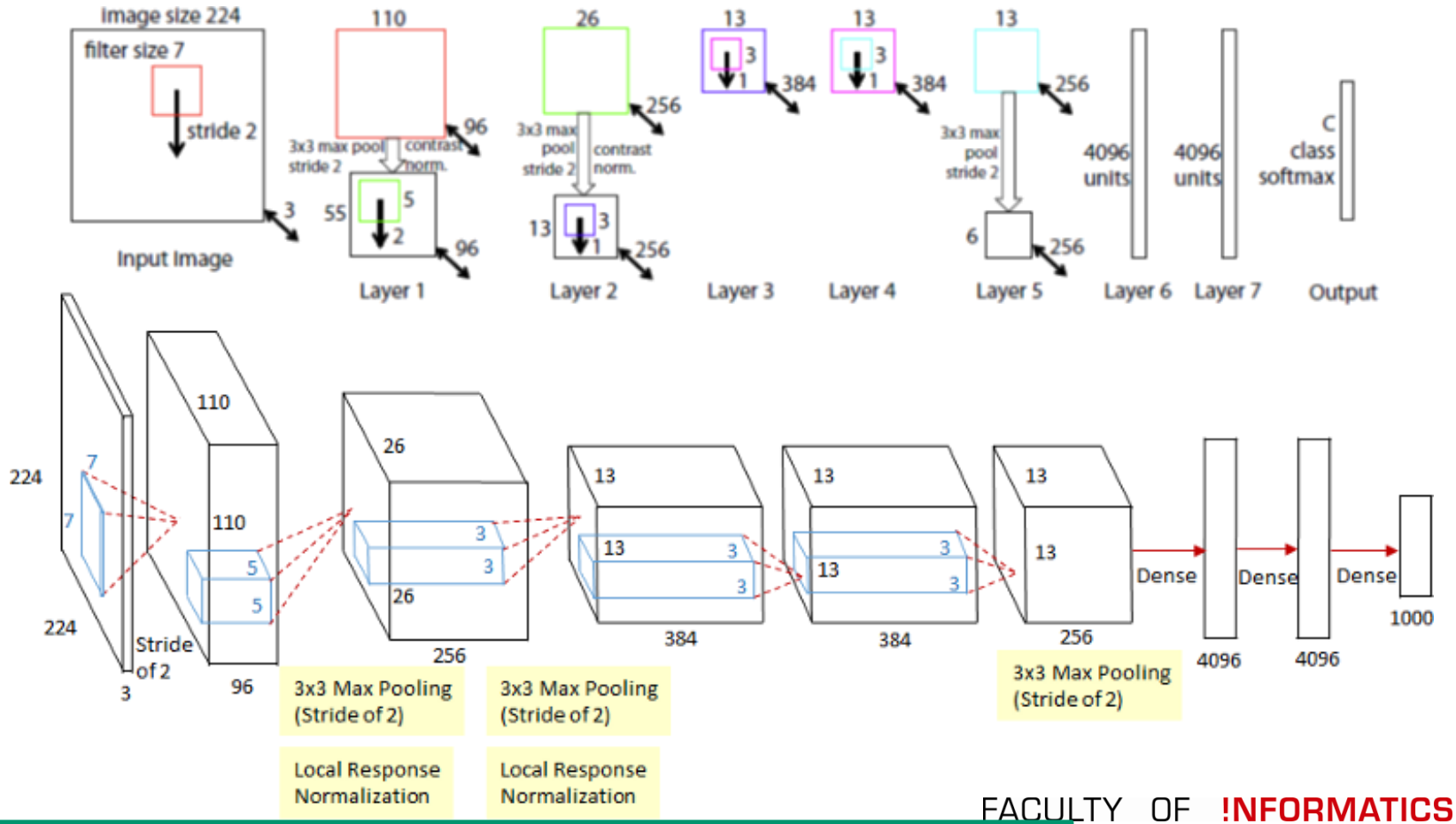
CNN Architectures: AlexNet

- Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, 2012
  - 30,000 citations of the paper ☺
  - 5 *convolutional* layers + 3 *Fully Connected* layers
  - Final layer: 1000-way softmax; *how many parameters?*
    - 60 million parameters, 650,000 neurons



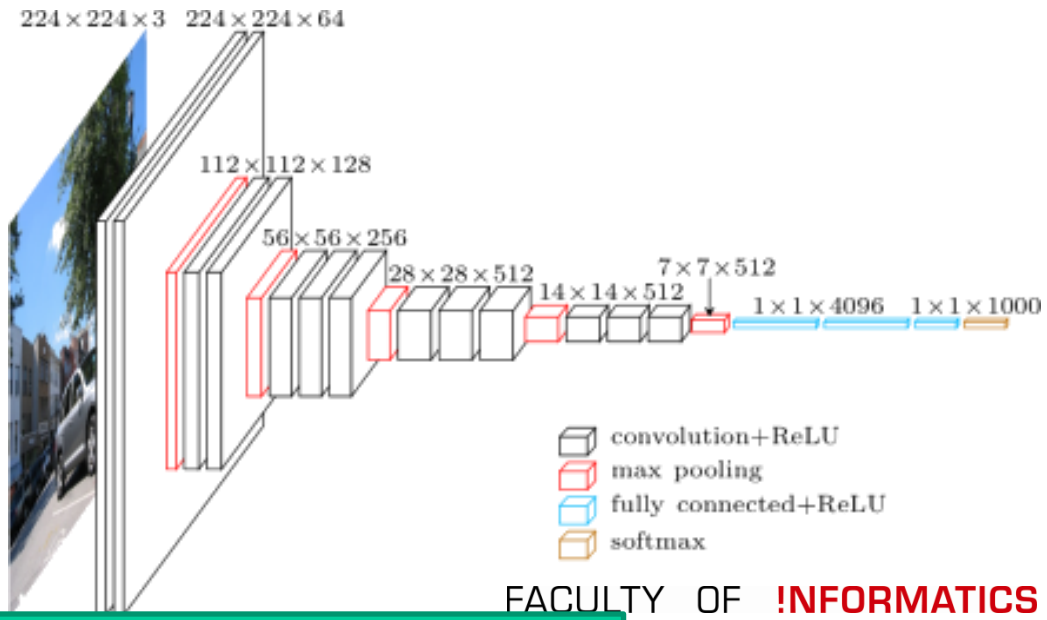
CNN Architectures: ZFNet

- Zeiler & Fergus; winner in 2013
  - Modification of AlexNet; also ~60 million parameters



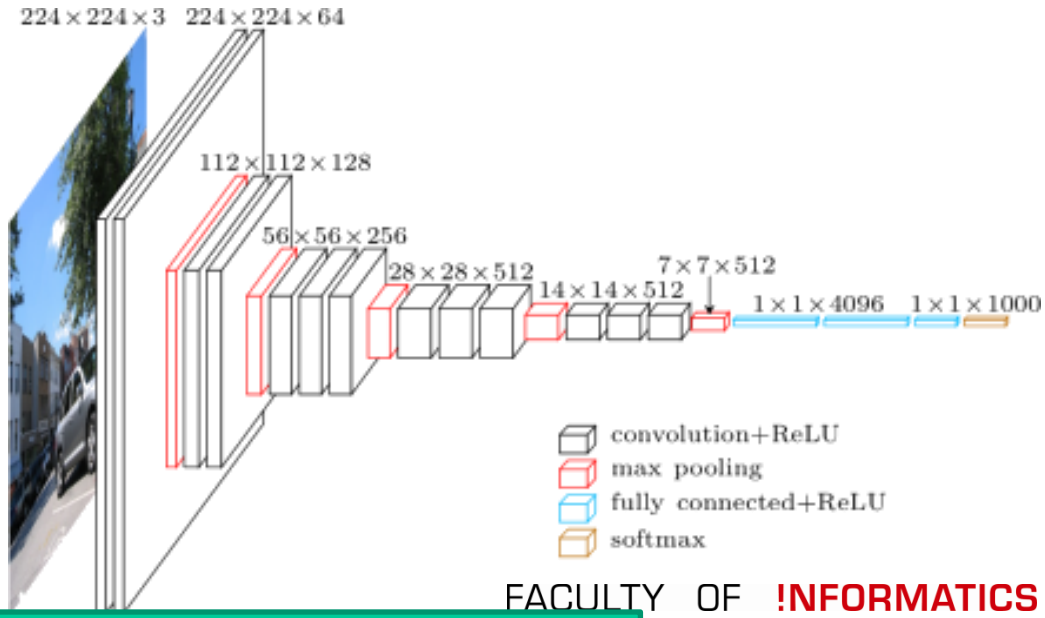
CNN Architectures: VGG Net

- VGG16 & VGG19
  - 2014, 2nd place
  - Based on ZF; deeper than AlexNet
  - Convolutional filters build on each other
    - Earlier layers are just line and edge detectors
    - Later layers combine the earlier layers into shape and face detectors



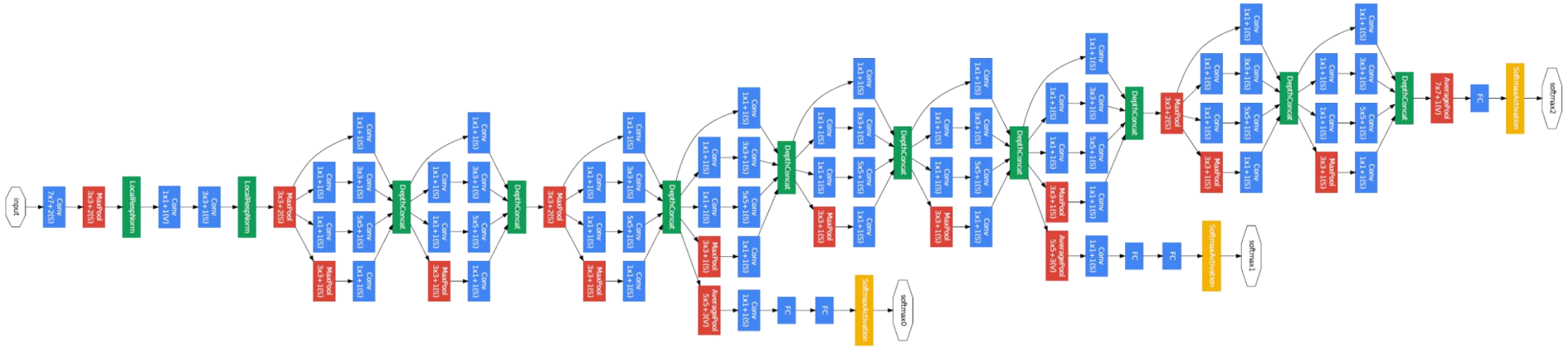
CNN Architectures: VGG Net

- *Number of parameters?*
  - 138 million!
    - 1st FC layer: 4096 (7 7 512) + 4096 = 102,764,544
    - 2nd FC layer: 4096 4096 + 4096 = 16,781,312
    - 3rd FC layer: 4096 1000 + 4096 = 4,100,096
    - Total FC layers = 123,645,952.



CNN Architectures: GoogLeNet (Inception)

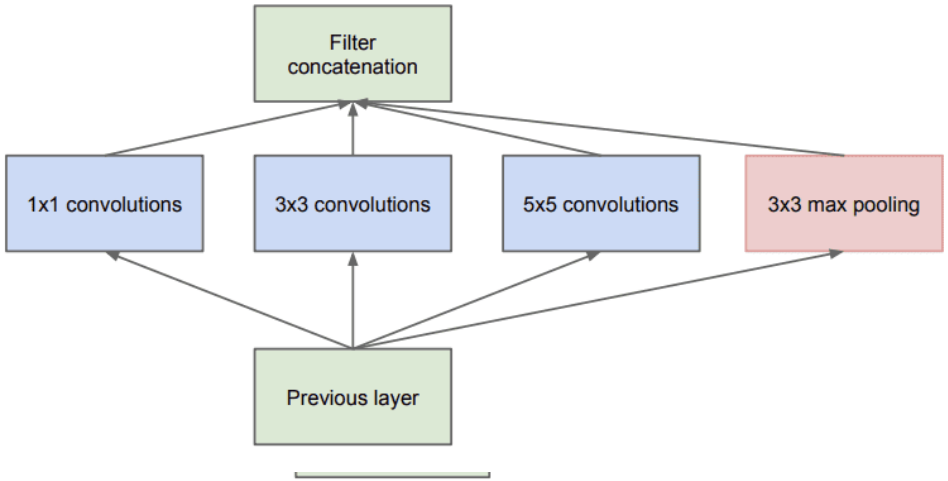
- Google, 2014
  - 22 layers, with auxiliary classifiers



  - Convolutions, Pooling, Fully Connected & softmax Multiple fully connected layers
  - Provide feedback to network at various depths
  - *Total parameters?*
    - ~5 million for v1 (13 million for v3)

CNN Architectures: GoogLeNet

- Building block: “Inception”

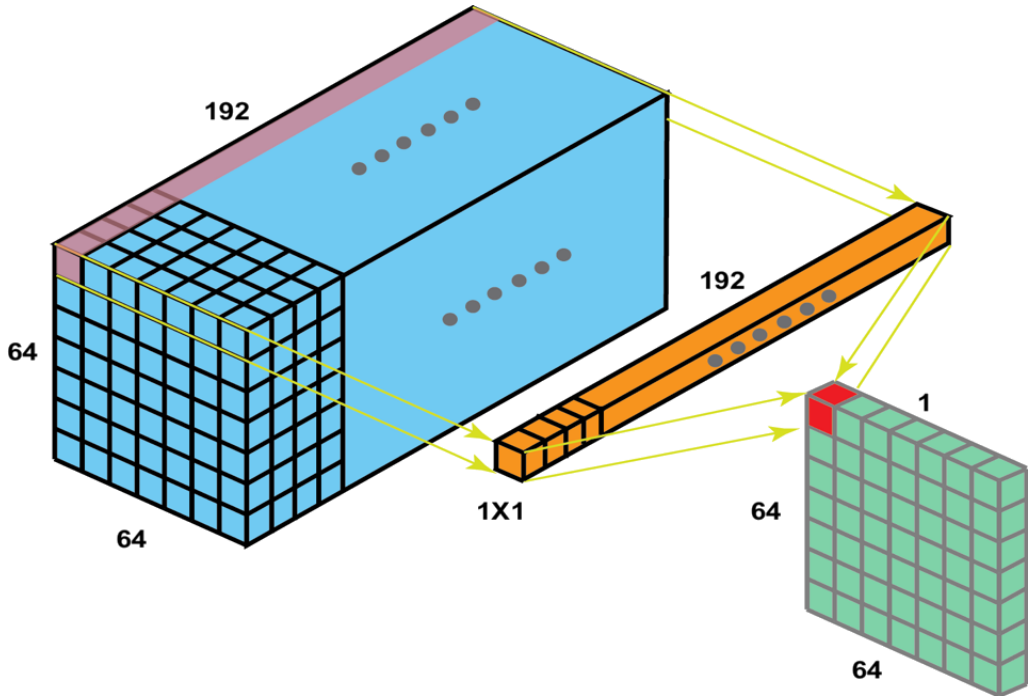


  - Instead of choosing a convolution kernel size..
    - ➔ Use multiple together
    - *Why not done before?*
    - Computationally expensive
    - ➔ 1x1 convolution reduces feature maps before that

1x1 Convolution

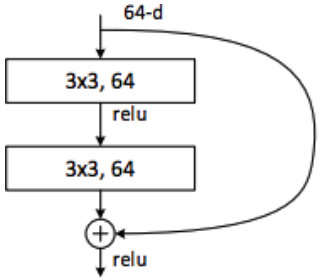
- Reduces input dimensionality
  - Not along the x/y axis of the output, but along the depth
    - Opposed to a pooling layer

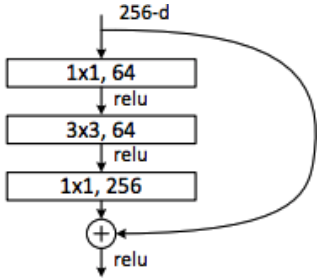
  ➔ Summarises multiple channels / feature maps to one
- More like a linear weighting / projection than a convolution

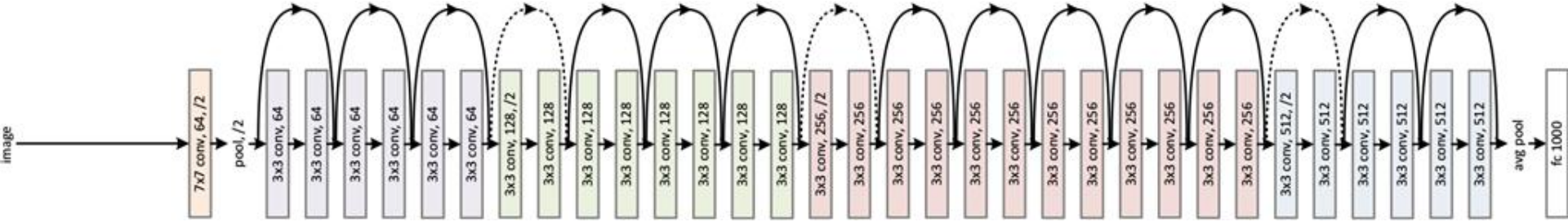


CNN Architectures: ResNet

- Microsoft, 2015
  - 152 layers, with shortcut connections (below: 38 layers)
  - Addresses vanishing gradients by residual blocks
    - Stacks residual blocks together



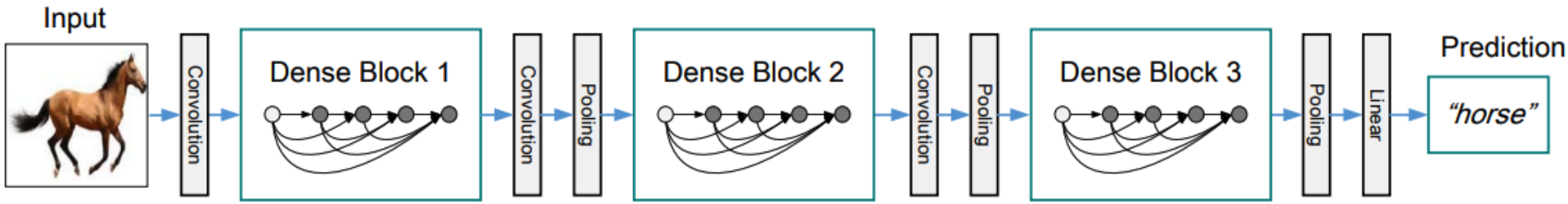




  - *Parameters?*
    - 25.6 million (ResNet 50)
    - 44.5 (Resnet 101
    - 60.2 (Resnet 152)

CNN Architectures: DenseNet

- Facebook, 2017
  - Builds on idea of residual blocks of ResNet
  - I.e. connections between layers
    - Here: each layer in a dense block has all previous ones as input

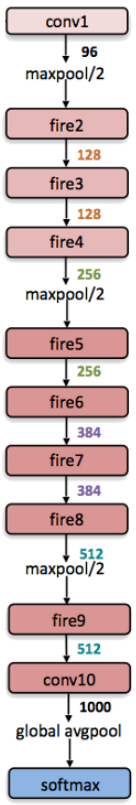


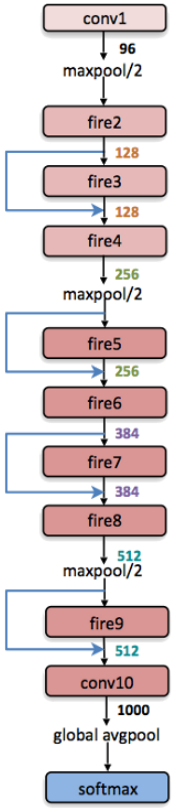
  - *Parameters?*
    - 0.8 million (DenseNet-100, k=12)
    - 15.3 million (DenseNet-250, k=24)
    - 40 million (DenseNet-190, k=40)

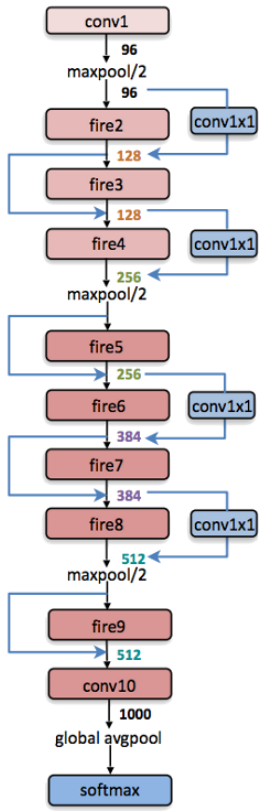
CNN Example: SqueezeNet

- 2016
  - Goal: reduce number of parameters ➔ why?
  - x50 times smaller than AlexNet
    - With DeepCompression even ~500x
  - Similar performance

- Other small networks:
  - SimpleNet, TinyNet, MobileNet, ..







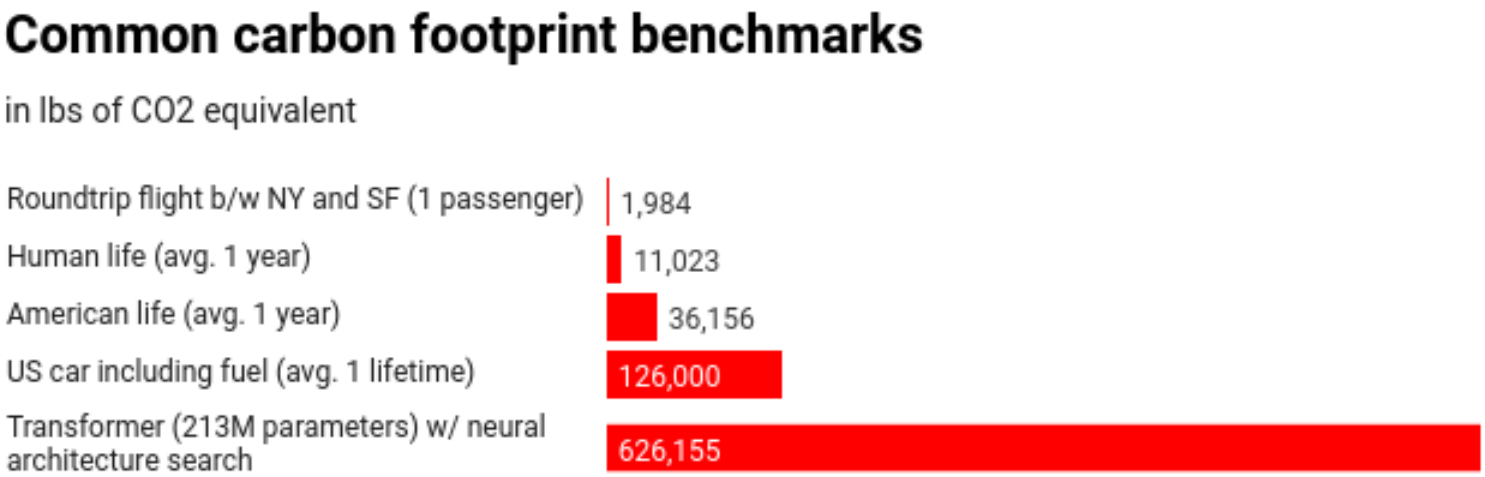
Neural Networks: Design Choices

- *Very **time** consuming for very large networks!*
  - "Efficient Neural Architecture Search via Parameter Sharing". Hieu Pham et al., 2018

Method	GPUs	Times (days)	Params (million)	Error (%)
DenseNet-BC (Huang et al., 2016)	–	–	25.6	3.46
DenseNet + Shake-Shake (Gastaldi, 2016)	–	–	26.2	2.86
DenseNet + CutOut (DeVries & Taylor, 2017)	–	–	26.2	**2.56**
Budgeted Super Nets (Veniat & Denoyer, 2017)	–	–	–	9.21
ConvFabrics (Saxena & Verbeek, 2016)	–	–	21.2	7.43
Macro NAS + Q-Learning (Baker et al., 2017a)	10	8-10	11.2	6.92
Net Transformation (Cai et al., 2018)	5	2	19.7	5.70
FractalNet (Larsson et al., 2017)	–	–	38.6	4.60
SMASH (Brock et al., 2018)	1	1.5	16.0	4.03
NAS (Zoph & Le, 2017)	800	21-28	7.1	4.47
NAS + more filters (Zoph & Le, 2017)	800	21-28	37.4	**3.65**
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	**3.87**
Hierarchical NAS (Liu et al., 2018)	200	1.5	61.3	3.63
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3	–	3.60
Progressive NAS (Liu et al., 2017)	100	1.5	3.2	3.63
NASNet-A (Zoph et al., 2018)	450	3-4	3.3	3.41
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4	3.3	**2.65**
ENAS + micro search space	1	0.45	4.6	3.54
ENAS + micro search space + CutOut	1	0.45	4.6	**2.89**

Neural Networks: Design Choices

- *Very **resource** consuming for very large networks!*



Outline

- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

Data Augmentation

Goal: Increase **invariance** to **irrelevant properties**

cat facing left

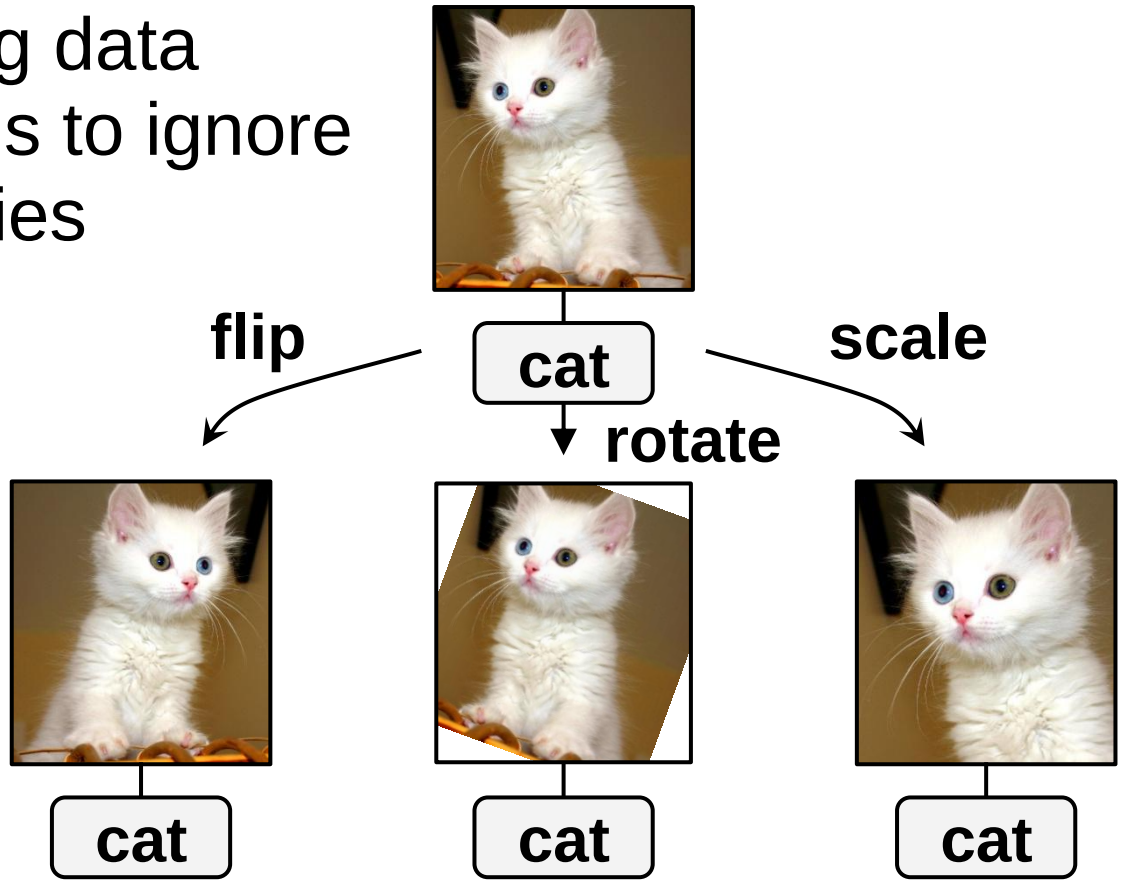
cat facing right

Possible solutions:

- Collect more (more diverse) training data
- Design invariant features or invariant model
- **Data augmentation**

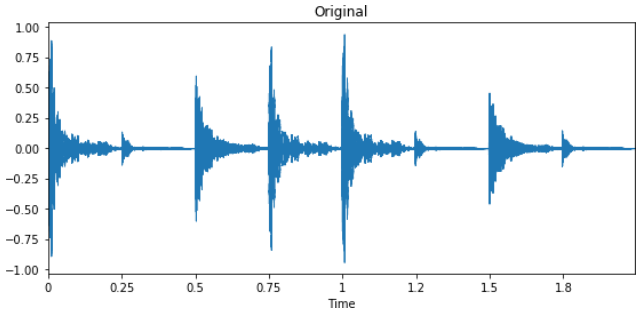
Data Augmentation

- Transform training data
  ➔ classifier learns to ignore irrelevant properties
- E.g. orientation,
  rotation,
  scale,

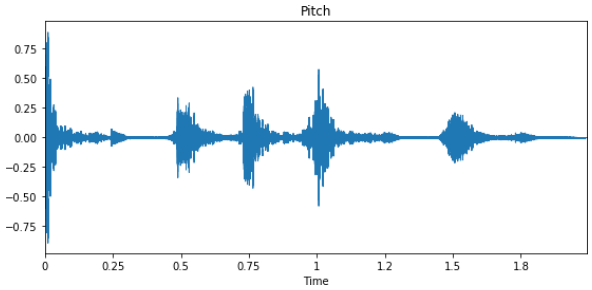


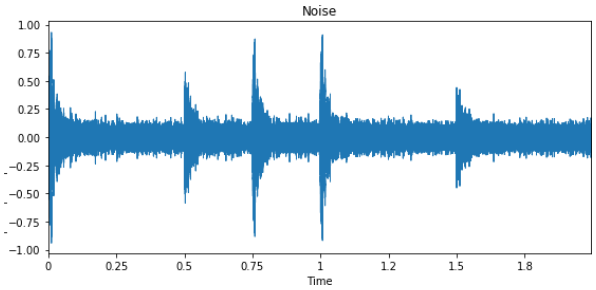
- Also: color, contrast, etc.

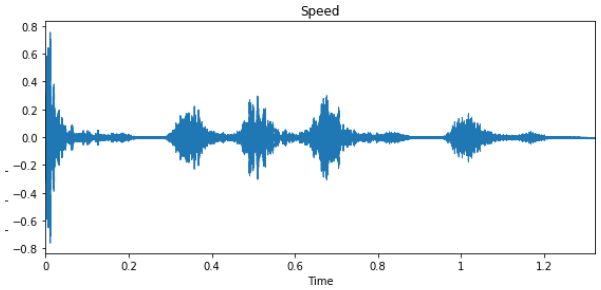
Data Augmentation



- Audio: time stretching, pitch shifting, noise, …







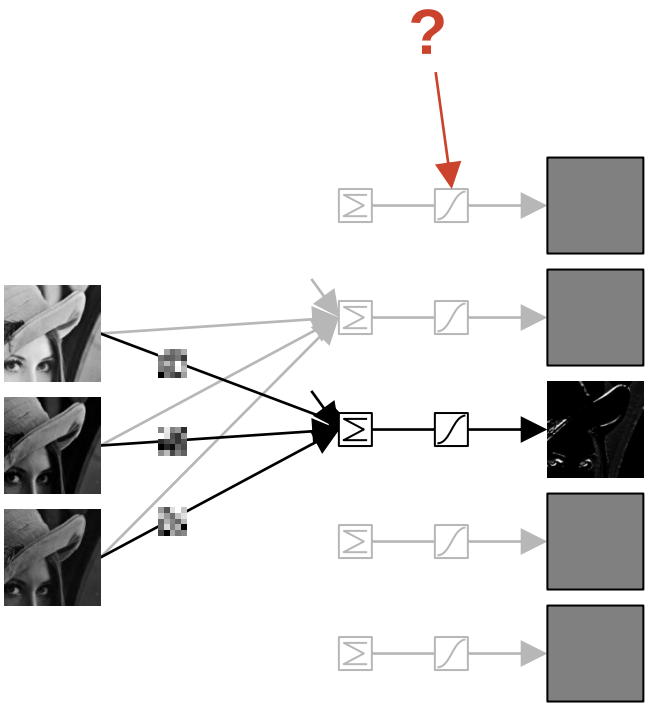
- Other modalities (e.g. video): similar operations
- Side effects:
  - Regularization: Less likely to learn data “by heart”
  - Increases apparent training set size
    - Can train larger model
    - Can use smaller dataset

Outline

- Short Recap
- Deep Learning
  - Intro / Definition
  - Convolutional Neural Networks
  - Convolutions in detail
  - Visualisation
  - Data Augmentation
  - Training Tweaks

Activation Functions

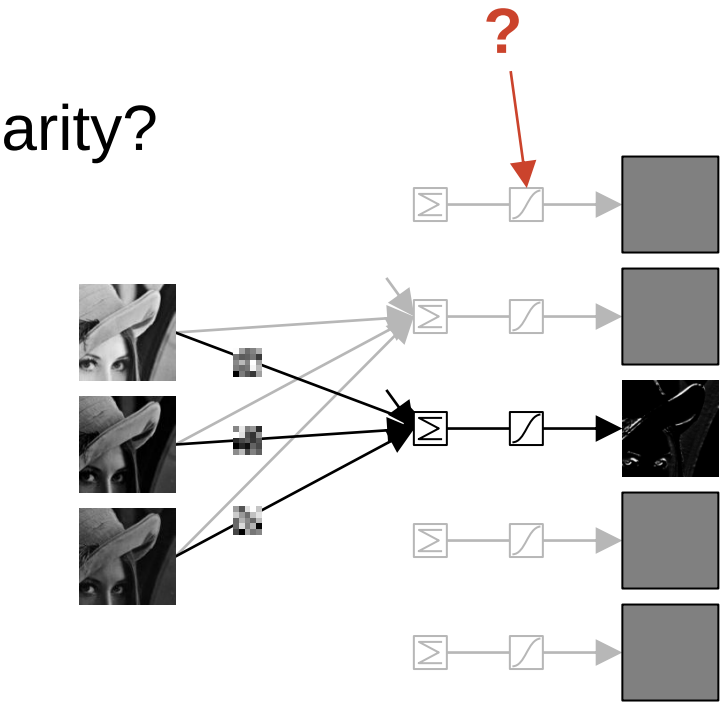
What shall the learnable function look like?



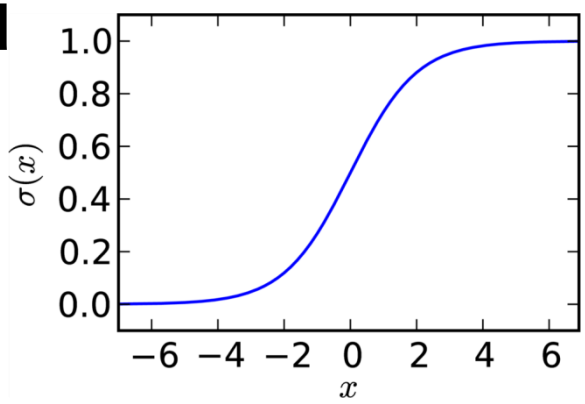
Activation Functions

What shall the learnable function look like?

- Specifically, what kind of nonlinearity?



Sigmoid

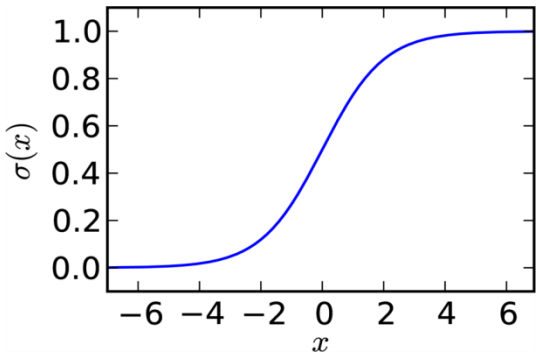


Problem:

saturates for large inputs (small slope ➔ weak gradient)

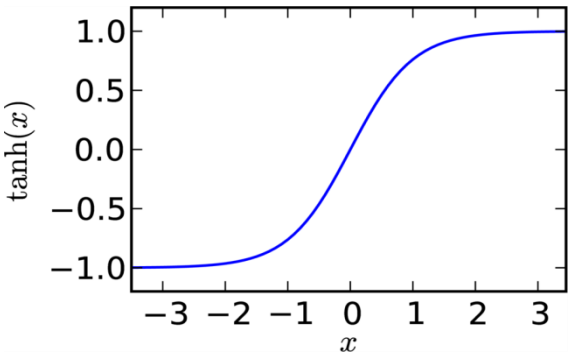
has nonzero mean (slows learning)

Linear Rectifier (ReLU)



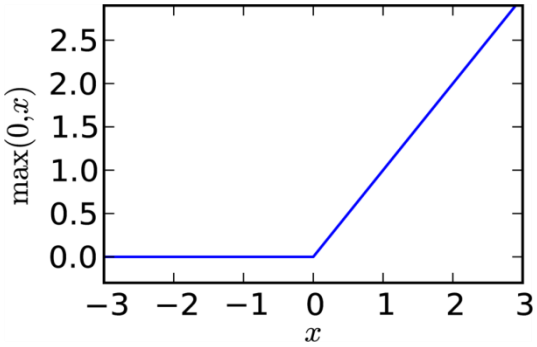
Sigmoid

saturates for large inputs (small slope, weak gradient)



TanH

saturates for large inputs (small slope, weak gradient!)

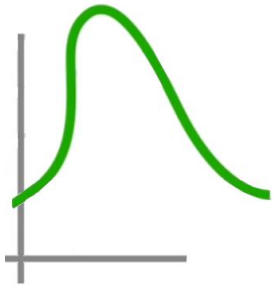


ReLU

Higher gradients also for larger inputs
➔ Avoids vanishing gradient

Benefits:
no saturation
low computational costs

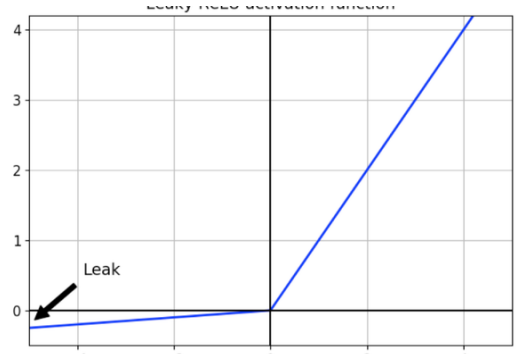
Shape of learned function?



Softplus

Variants of ReLU:

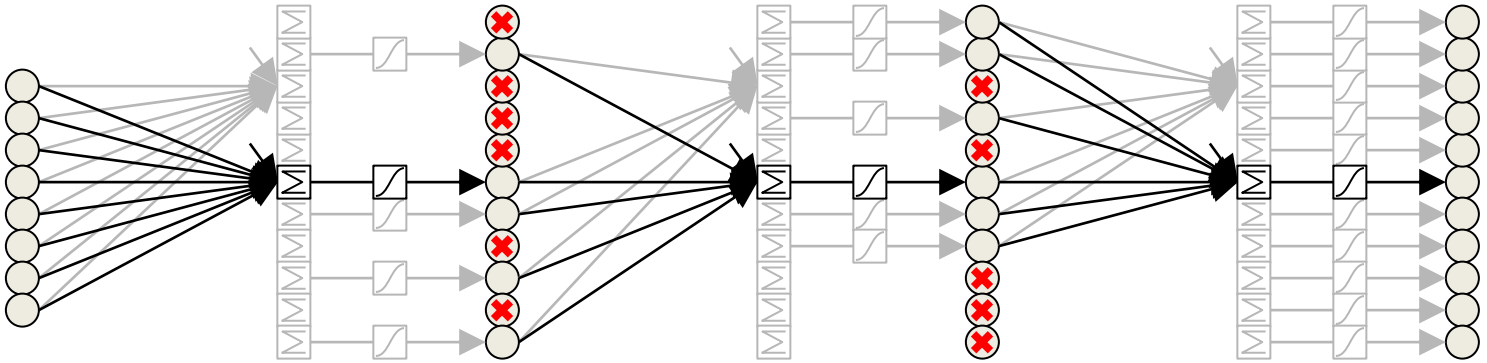
- Leaky Rectifier (LReLU)
  - small slope for <0
- Parametric Rectifier (PReLU), ...



ICML 2013: Rectifier Nonlinearities Improve Neural Network Acoustic Models

Dropout

Randomly “delete” a part of the units
for each training example (e.g. 10% or 50%)



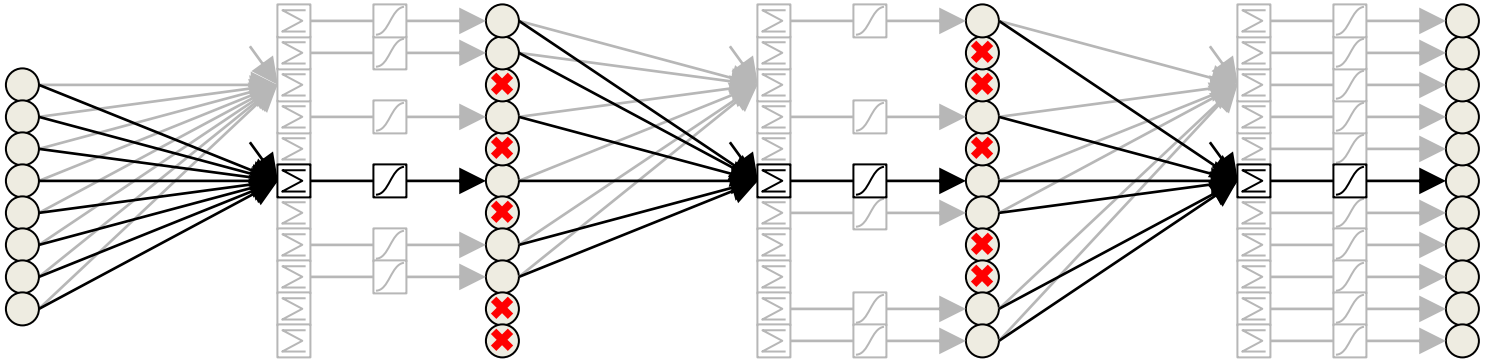
Makes the units more robust, prevents *co-adaptation of feature detectors (kernels)*

(they cannot rely on all inputs and neighbours being present)

Hinton et. Al, 2012: Improving neural networks by preventing co-adaption of feature vectors

Dropout

Randomly “delete” a part of the units
for each training example (e.g. 10% or 50%)



Makes the units more robust, prevents *co-adaptation*
(they cannot rely on all inputs and neighbours being present)

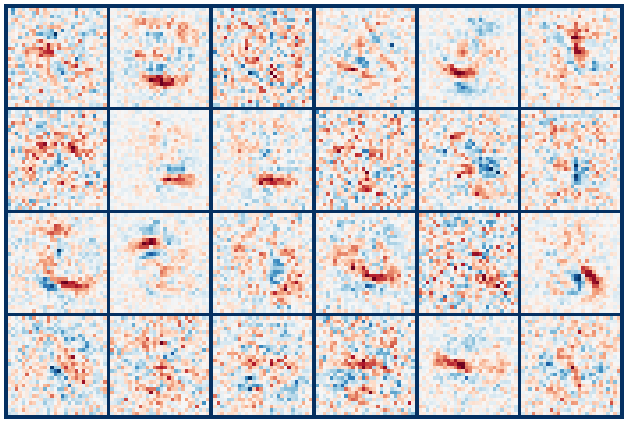
Hinton et. Al, 2012: Improving neural networks by preventing co-adaption of feature vectors

Dropout Example

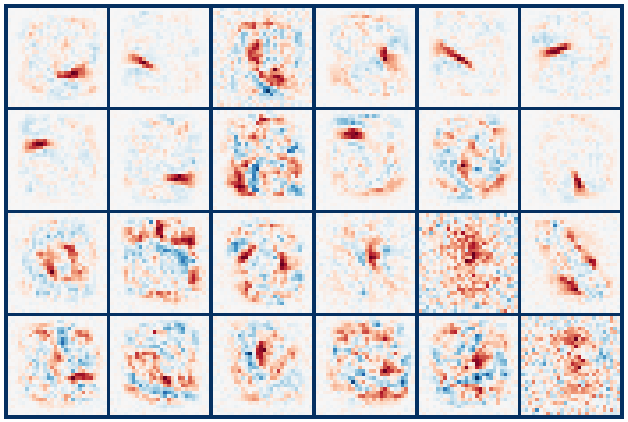
E.g. randomly delete half of the units for each training example.

MNIST digit recognition:

First-layer features after training:



No dropout:
noisy features, overfit to training set



20% input, 50% hidden dropout:
cleaner global features, more general

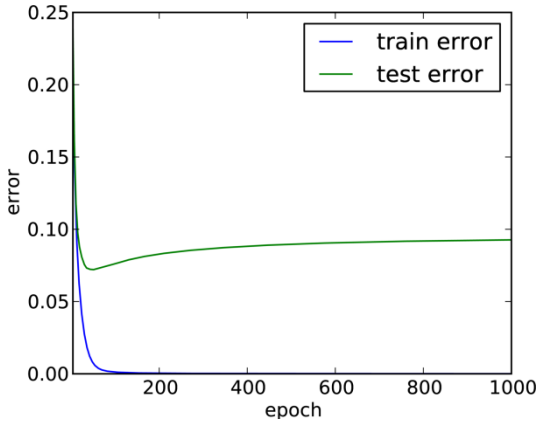
Hinton et. Al, 2012: Improving neural networks by preventing co-adaption of feature vectors

Dropout Example

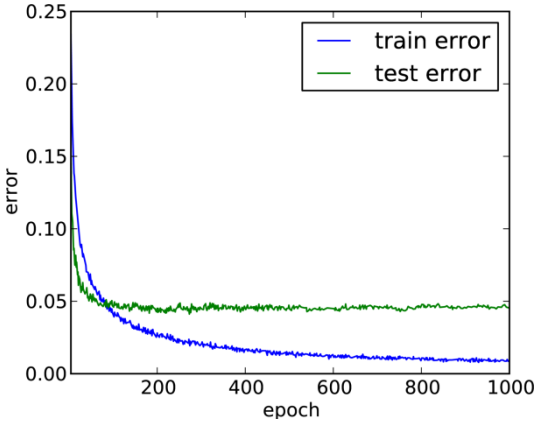
E.g. randomly delete half of the units for each training example.

MNIST digit recognition:

Train/Test error:



No dropout:
quick overfitting, 169 test errors

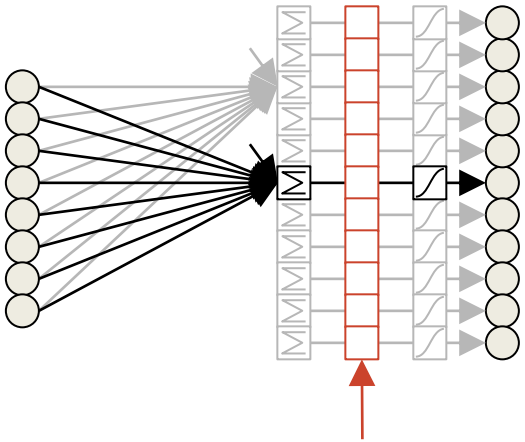


20% input, 50% hidden dropout:
test error plateaus, 99 test errors

Hinton et. Al, 2012: Improving neural networks by preventing co-adaption of feature vectors

Batch Normalisation

- Weight gradients depend on the unit activations
  - Very small or very large activations lead to small/large gradients
    - For ***input*** units, **standardizing** the **training data** avoids this
  - **Batch normalisation does the same for hidden units**
    - Even as mean/variance (distribution) changes (internal covariate shift) during training
    - Computationally cheap
  - Also works as regulariser!



batch normalisation

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Mini-batch Gradient Descent

- Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

  - Randomly select one sample to compute loss / error
  - Take small step in direction of negative gradient
  - Repeat until all samples have been processed (one epoch)

- Mini-batch: instead of update after single training sample: gradient averaged over mini-batch of samples (often 2^x instances)
  - Sweet spot between
    - Online learning (single example, very noisy gradient estimate, can only take small steps) and
    - Batch learning (all examples, too much computation per update step)

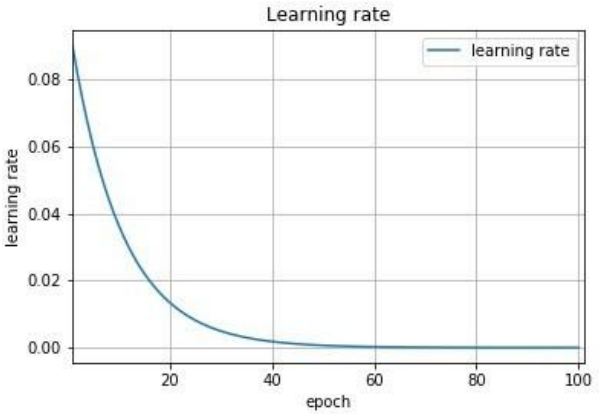
Gradient Descent

- How to set the learning rate α?
  - The learning rate is a hyperparameter of great influence
  - It's not easy to guess, so often many values are tried
  - To allow for better convergence, the learning rate can be adjusted over time using a decay function

Time decay: $\alpha_t = \frac{\alpha_0}{t}$

Exponential decay: $\alpha_t = \alpha_0 * \exp(-t * k)$

Exponential decayfunction



Gradient Descent

- Several alternatives to (stochastic) gradient descent proposed
- Aim: solve issue of (speed of) convergence
  - Gradient descent converges slowly to the optimum in low curvature regions
  - **Momentum** is a simple approach to speed up the optimizer
  - Might also help to avoid local optima

- Very active research field, new approaches frequently published
- Two selected approaches:
  - Momentum
  - Adaptive Moment Estimation (Adam)

Gradient Descent: Momentum

- Stochastic Gradient Descent (SGD) with Momentum:

$$v = \beta v - \alpha \frac{\partial L}{\partial \theta}$$

$$\Theta_{t+1} = \theta_t - v$$

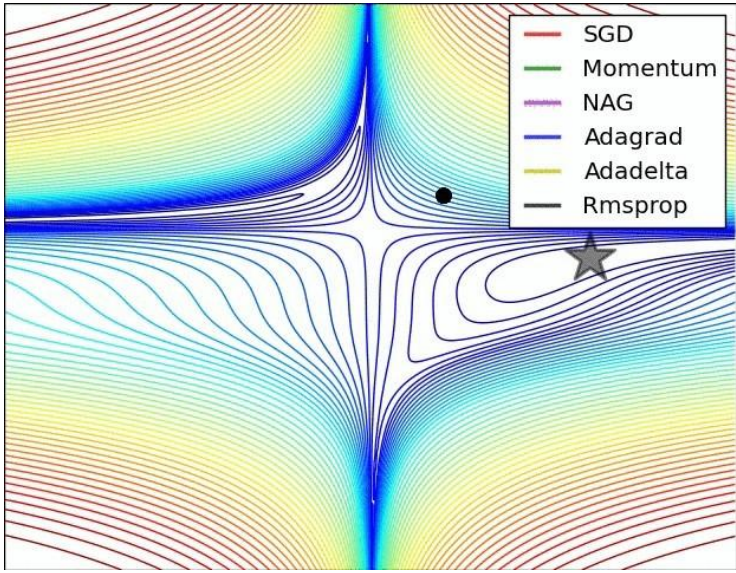
- Analogy: Ball rolling downhill (along the cost function)
  - Builds up momentum if subsequent gradients point into same direction ➔ doesn't slow down immediately when it's flat
  - Dampen velocity according to friction coefficient ß (e.g., 0.9)
  - Increase velocity in direction of negative gradient
  - Move according to velocity
  - Issues:
    - Often results in oscillations and instability in high-curvature regions
    - How to pick the damping coefficient?

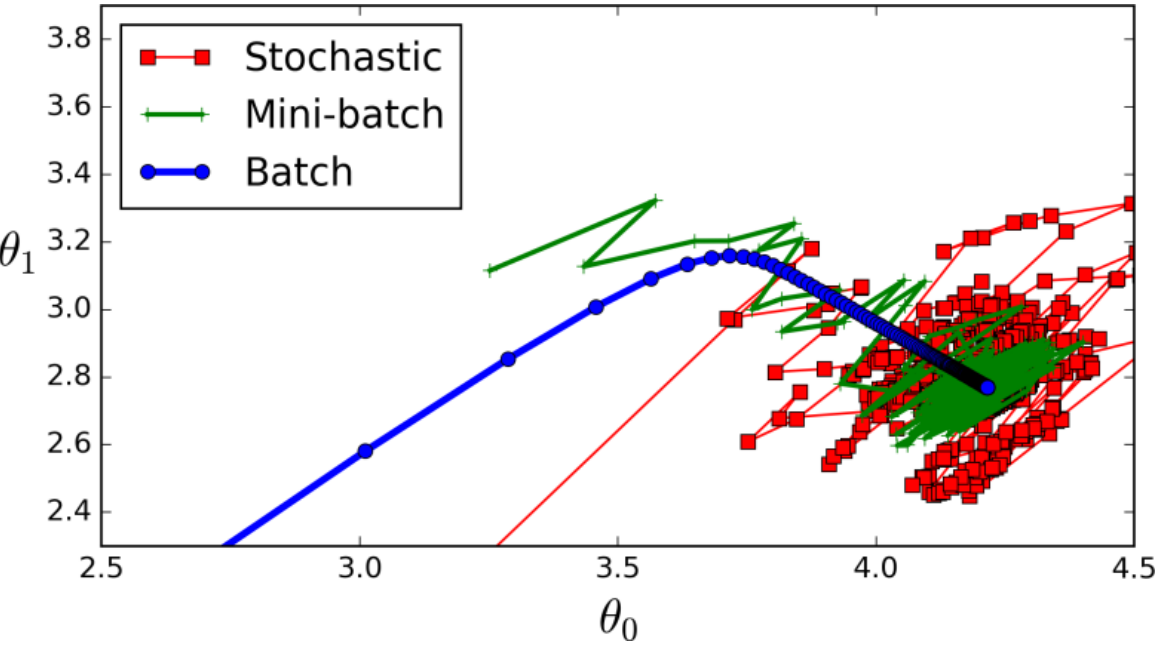
Adam (Adaptive Moment Estimation)

- Optimization methods that adapt the learning rate **for each parameter**
  - Compute velocity (first moment): exponential moving average over past gradients (as before)
  - Compute second moment estimate: exponential moving average over past gradient magnitudes
  - Move according to velocity, divided by second moment

- Intuition: counter notoriously **small** gradients by **upscaling**, and **large** gradients by **downscaling**
  - Separately for each weight

Optimization: Many Flavours





Questions ?