02456 – Week 4
Convolutional Neural Networks

Jes Frellsen

Technical University of Denmark

22 September 2025

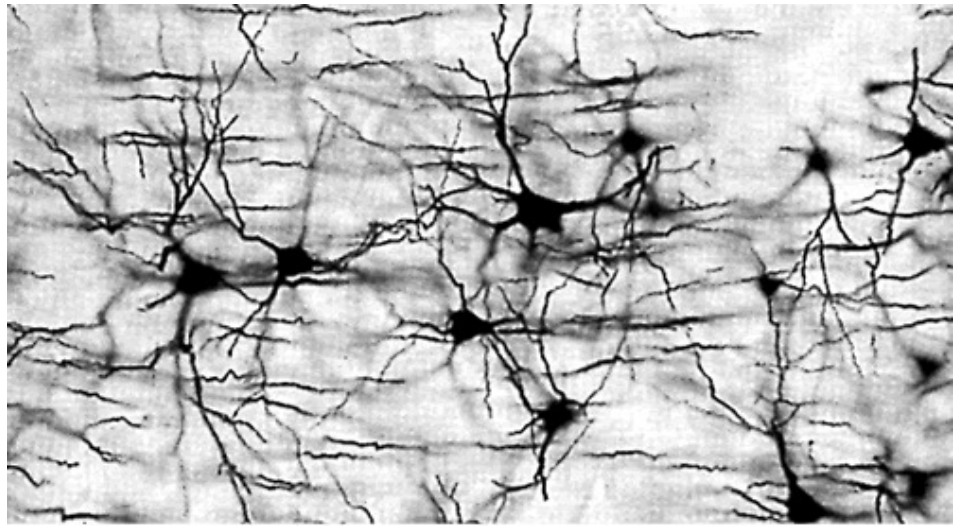
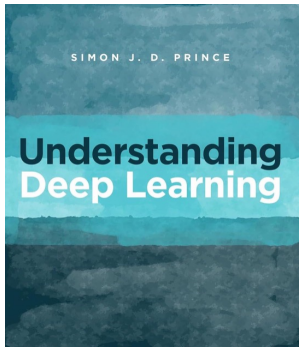


Image from: https://alanturing.net

Menu of the day



Basic **supervised** learning with NNs

- Week 1: Neural nets
- Week 2: Learning
- Week 3: Tricks of The Trade

Specialised architectures

- Week 4: CNNs
- Week 5: RNNs
- Week 6: Transformers

- Week 7: Unsupervised
- Week 8: Mini-project

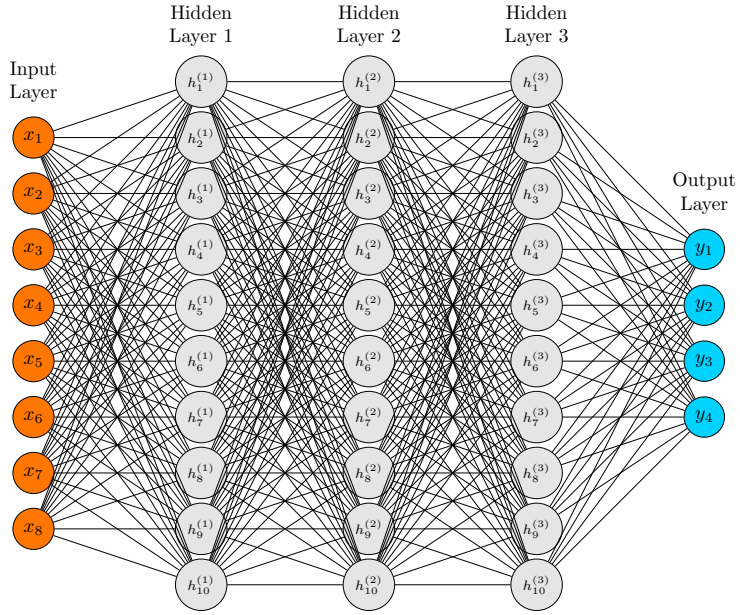
Lecture

- Motivation for CNNs
- Details of CNNs (ch 10)
- Applications of CNNs (ch 10)

Exercises

- **Notebooks 4.1-4.3**:
  - CNNs in **PyTorch**!
- **Problems:** 10.1, 10.6, 10.16

Recap: Neural networks and learning



We can write the **output of layer ℓ** as

$$f^{(\ell)}(\mathbf{h}) = \sigma\left(W^{(\ell)}\mathbf{h} + \mathbf{b}^{(\ell)}\right),$$

where σ is a non-linear **activation function**.

The **joint function** is then

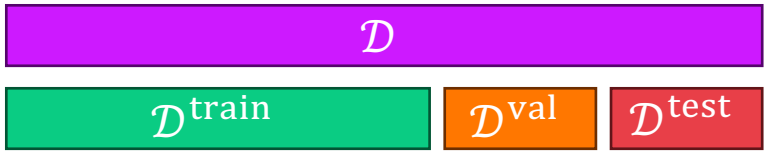
$$f_\phi(\mathbf{x}) = \mathbf{y} = f^{(4)}\left(f^{(3)}\left(f^{(2)}\left(f^{(1)}(\mathbf{x})\right)\right)\right)$$

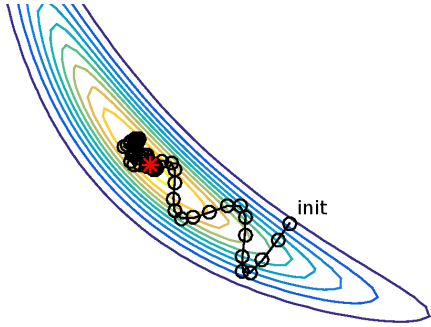
A **loss function** $L(\phi)$ captures the **mismatch** of $f_\phi(\mathbf{x}_i)$ and y_i: $\hat{\phi} = \operatorname{argmin}_\phi L(\phi)$

Use the **negative log-likelihood**

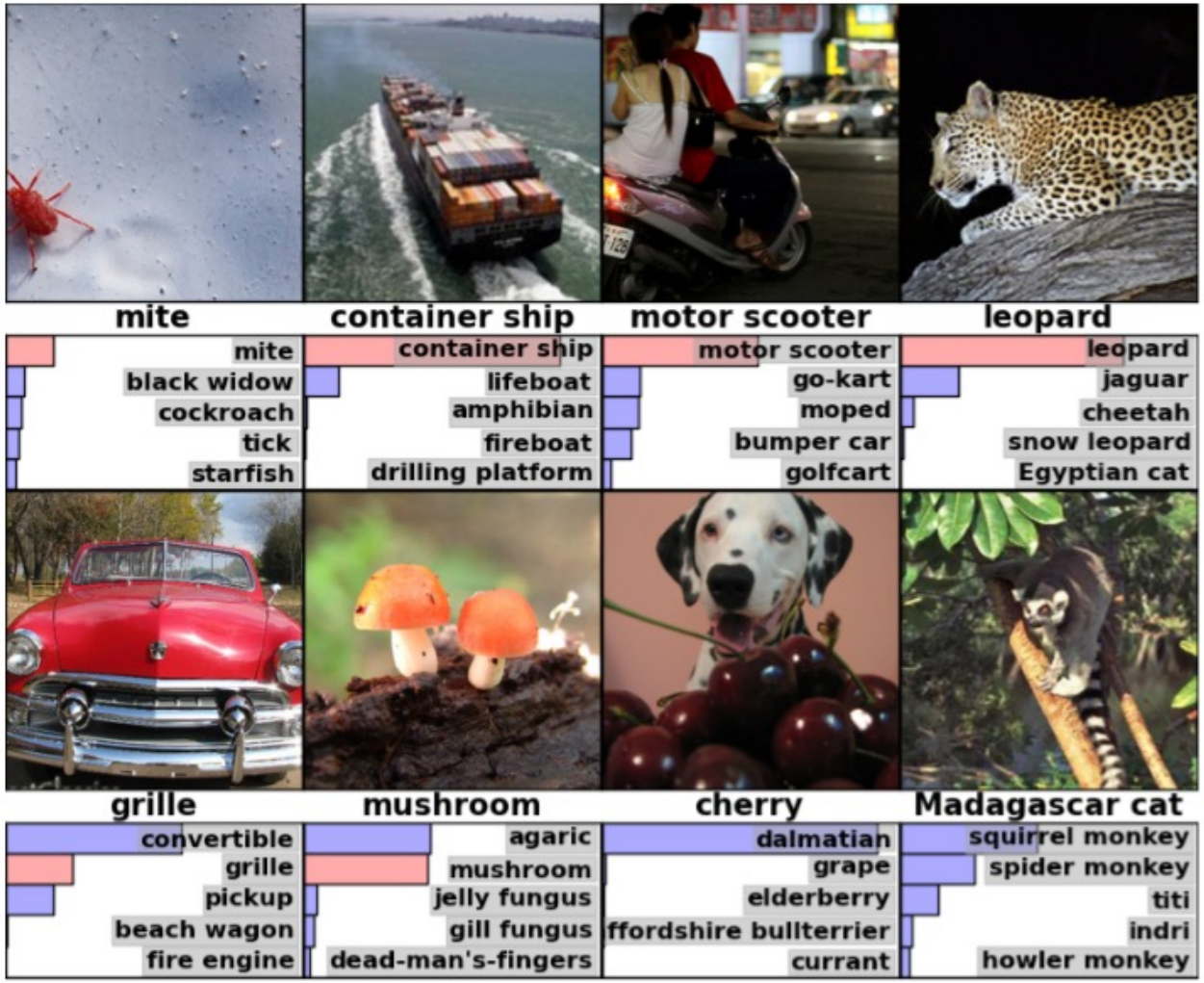
$$L(\phi) = -\sum_i \log p\left(y_i \middle| f_\phi(\mathbf{x}_i)\right)$$

- Use **gradient descent** to find $\hat{\phi}$
  - **Backpropagation** gets gradients $\nabla_\phi L(\phi)$
  - **Initialisation** is important
- Split **data set** in three
  - **Val**: Select hyper parameters
  - **Test**: estimate generalisation error





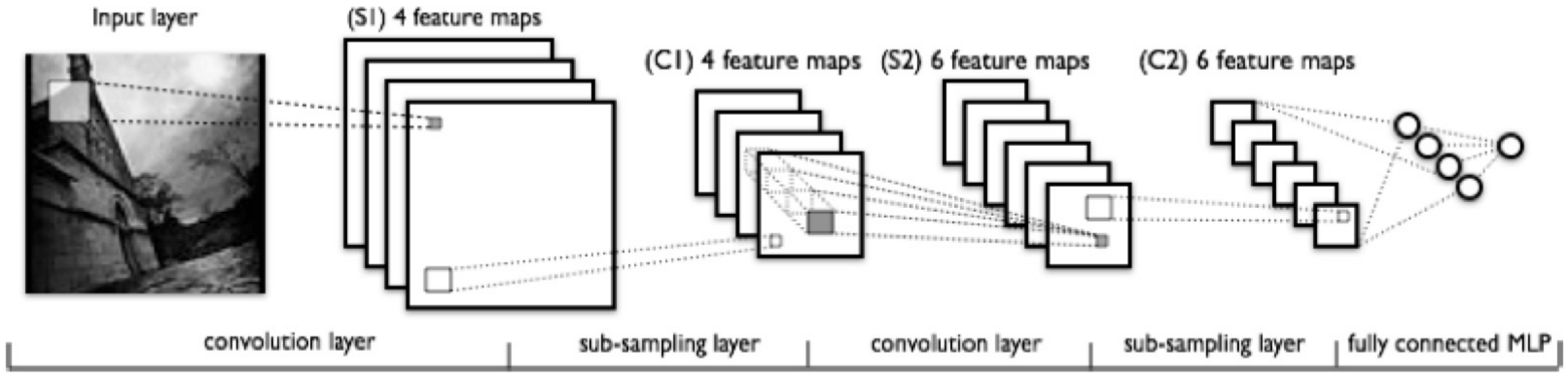
Motivation: Image classification



ImageNet

- **1,000 classes**
  - Including many types of dogs!
- **~1.2M training images**
- **Performance measure**: is the correct class within top, e.g., 5% (due to ambiguity)

Convolutional neural networks



$$\begin{bmatrix} 10 & 0 & -10 \\ 0 & 0 & 0 \\ -10 & 0 & 10 \end{bmatrix}$$

High-level idea:

- **Learn to extract features** from image
- **Same weights** applied across positions
  - Weight sharing
  - Translation equivariant
- Sub-sampling (pooling)
  - Reduce spatial dim.
  - Increase num. features
- **MLP at the end**

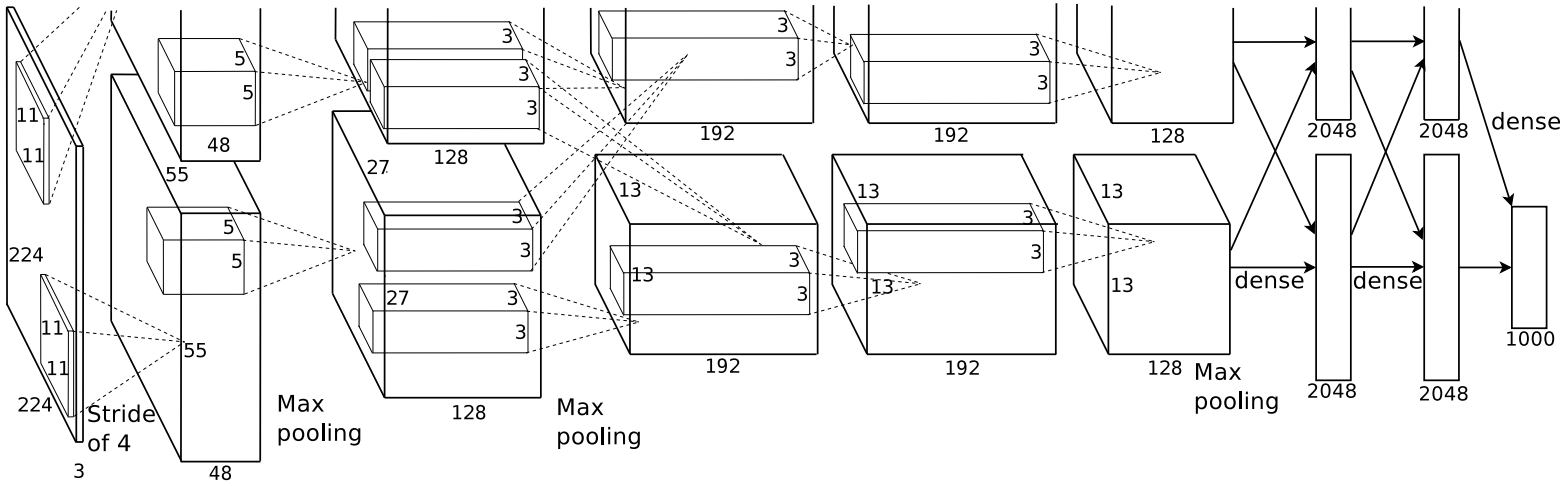
Feature engineering vs engineered models

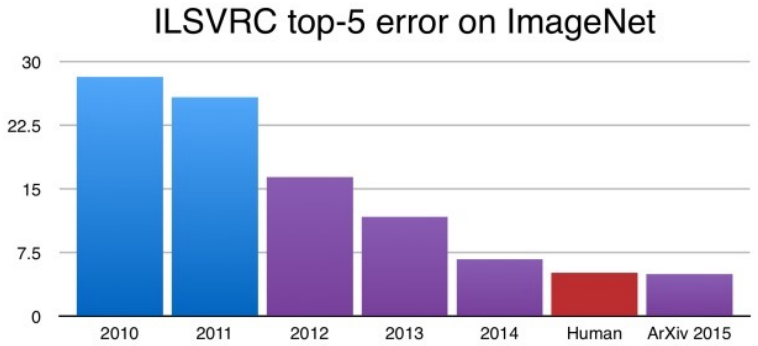
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca





AlexNex: Learned filters in first layer

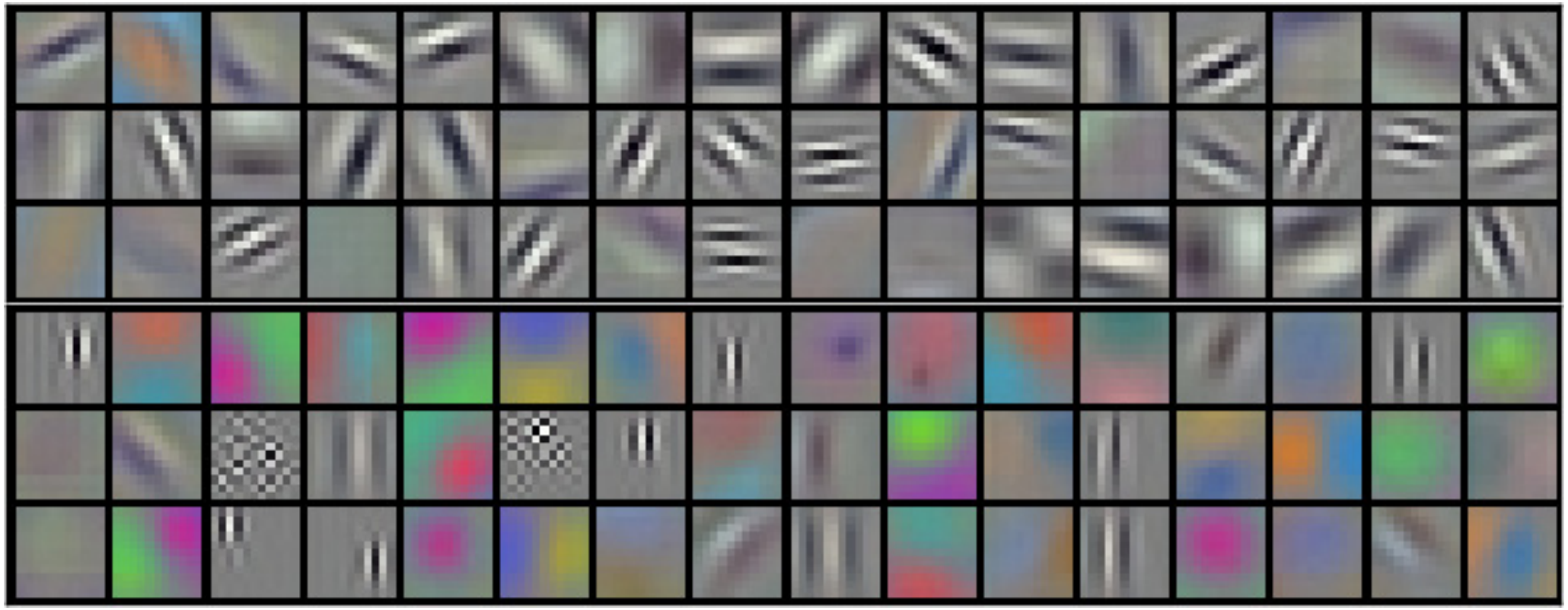
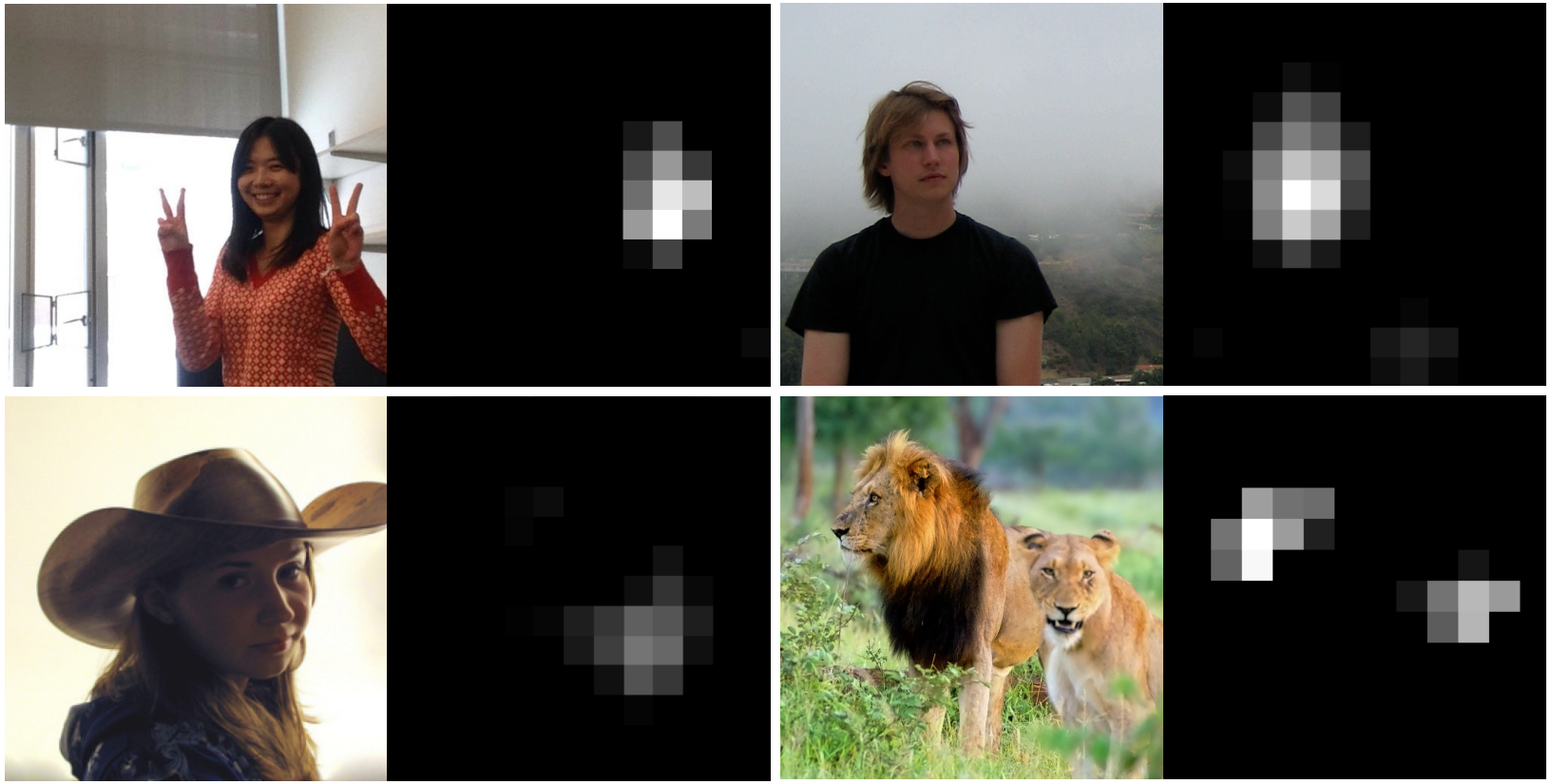


Image from: Krizhevsky, Sutskever & Hinton. Imagenet classification with deep convolutional neural networks. NeurIPS, 2012.

Emergent higher-level abstractions



Activations of the 151st channel of layer 5 in deep CNN:

- Sensitive to faces

Remember learned!

- We designs an architecture that make the network learn increasingly abstract filters

Yosinski, Clune, Nguyen, Fuchs & Lipson. Understanding neural networks through deep visualization. Deep Learning Workshop at ICML, 2015.

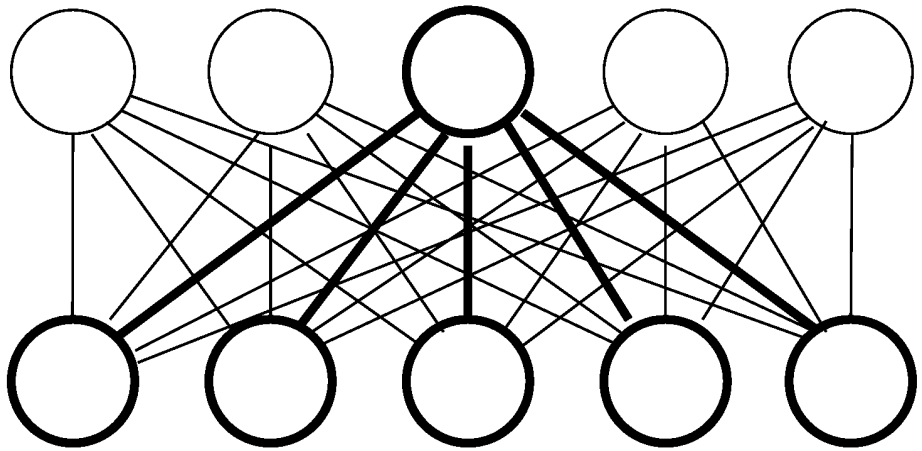
Convolutional Networks

Issues with Fully Connected Layers for Images

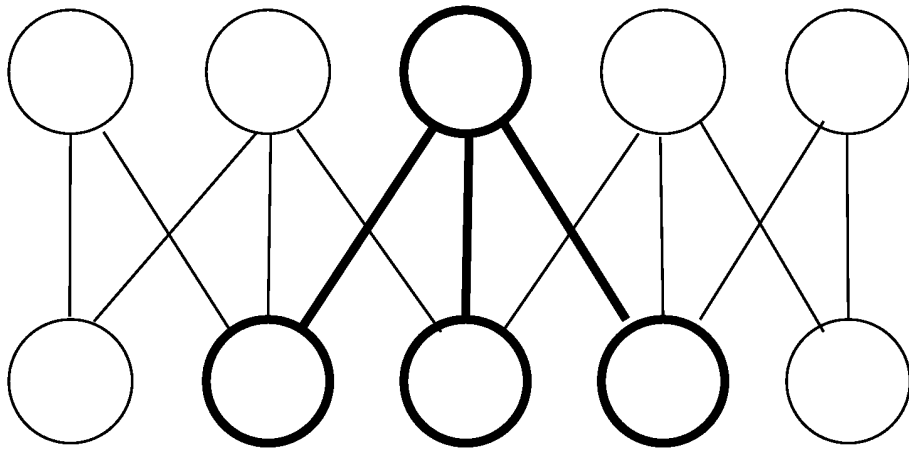
- **First weight matrix** has size $|\text{input}| \times |\mathbf{h}^{(1)}|$
  - For a 1000×1000 image: $10^6 \times |\mathbf{h}^{(1)}|$ parameters
- **No notion of locality** ("nearby" pixels are not treated specially)
  - Permuting pixels + permuting weights leaves output unchanged
- Interpretation of an image **not stable under translations**
  - We want layers that are **equivariant to translations**
- **Convolutional layers address all these issues**
  - Popularized by **Yann LeCun** et al. (**1989**)
  - Two ideas: **Local connectivity** and **parameter sharing**

LeCun, Boser, Denker, Henderson, Howard, Hubbard & Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation, 1989.

Local connectivity

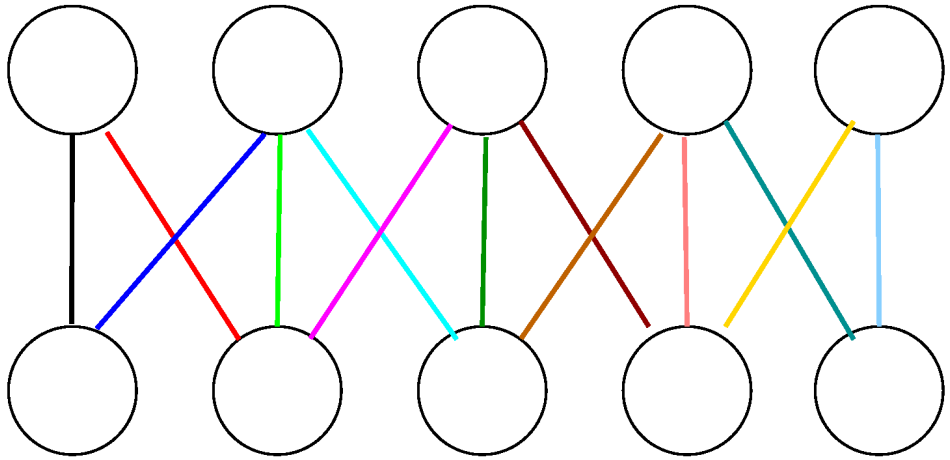


25 parameters

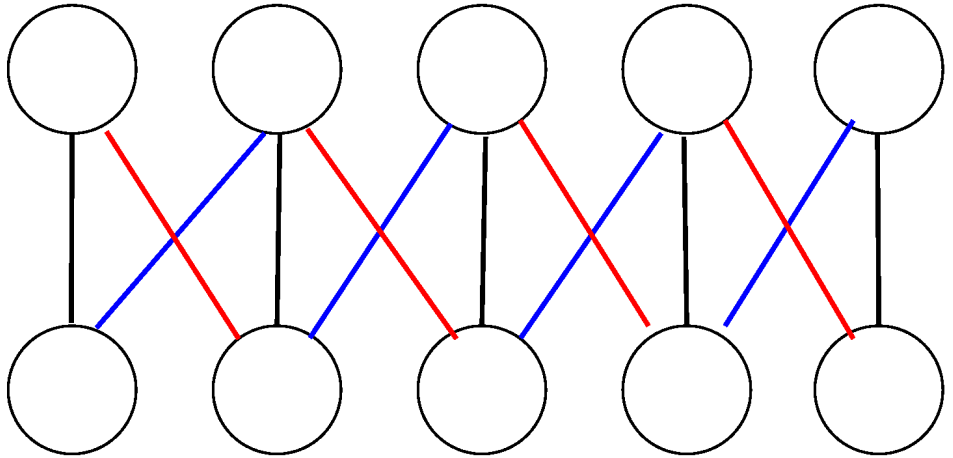


13 parameters

Parameter sharing



13 parameters

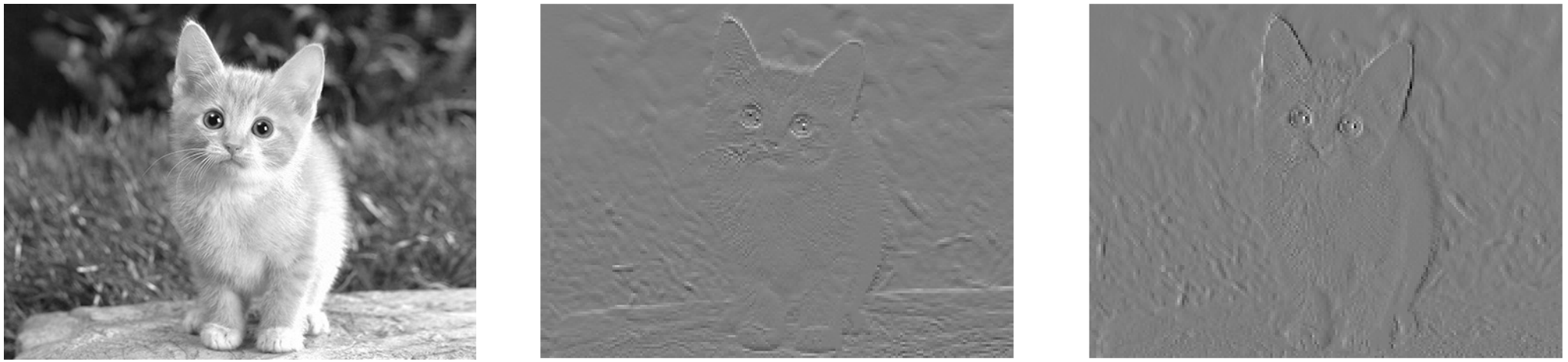


3 parameters

If we learn to extract a feature on place in the image, we can extract it in all places!

Image from by from slides by Tapani Raiko

Convolution *



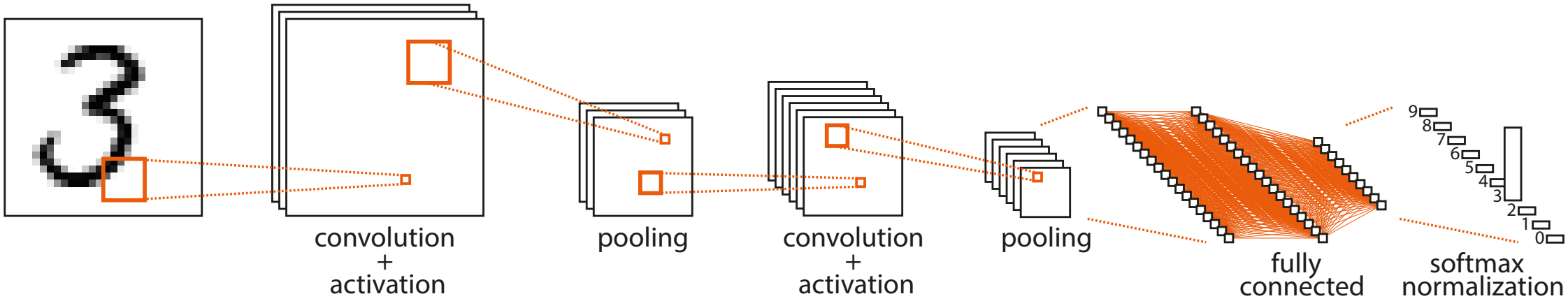
The convolution* on a 2D image $\mathbf{X}$ with a 2D kernel/filter $\mathbf{W}$

$$z_{ij} = (\mathbf{X} * \mathbf{W}) = \sum_m \sum_n X_{i+m,j+m} W_{m,n}$$

*) Strickly speaking a cross-correlation

Image from by from slides by Tapani Raiko

Convolutional Neural Networks



We use **convs** to construct a **CNN**

- Pooling to down-sample
- Fully connected layer at the end

Central: Parameter **sharing through filters**

- By sliding a filter give **feature map**
- **Multiple filters** give **multiple channels**
- Feature maps gets **increasingly abstract** away for the input

Image from: Boomsma & Frellsen. Convolutional models of molecular structure. MIA seminar, 2018.

What does a filter do?

filter

0	0	1
0	0	1
0	0	1

image

1	0	0	0	0	0
1	1	0	1	1	0
0	0	0	0	1	1
0	0	0	1	0	0
1	0	0	1	0	0
1	1	0	1	0	0

result

0	1	2	1
0	2	2	1
0	2	1	1
0	3	0	0

We can view the operation as an **inner product**

Image from: Boomsma & Frellsen. Convolutional models of molecular structure. MIA seminar, 2018.

Pooling?

image

1	0	0	0	0
3	2	0	1	1
0	0	0	0	2
0	0	0	1	0
1	0	0	3	0

avg pool

6/9	4/9
1/9	6/9

Image from: Boomsma & Frellsen. Convolutional models of molecular structure. MIA seminar, 2018.

Invariance and equivalence

A function f is **invariant** to a transformation t if

$$f\big(t(\mathbf{x})\big) = f(\mathbf{x})$$

A function f is **equivariant** to a transformation t if

$$f\big(t(\mathbf{x})\big) = t\big(f(\mathbf{x})\big)$$

In images we want **translational equivalence**

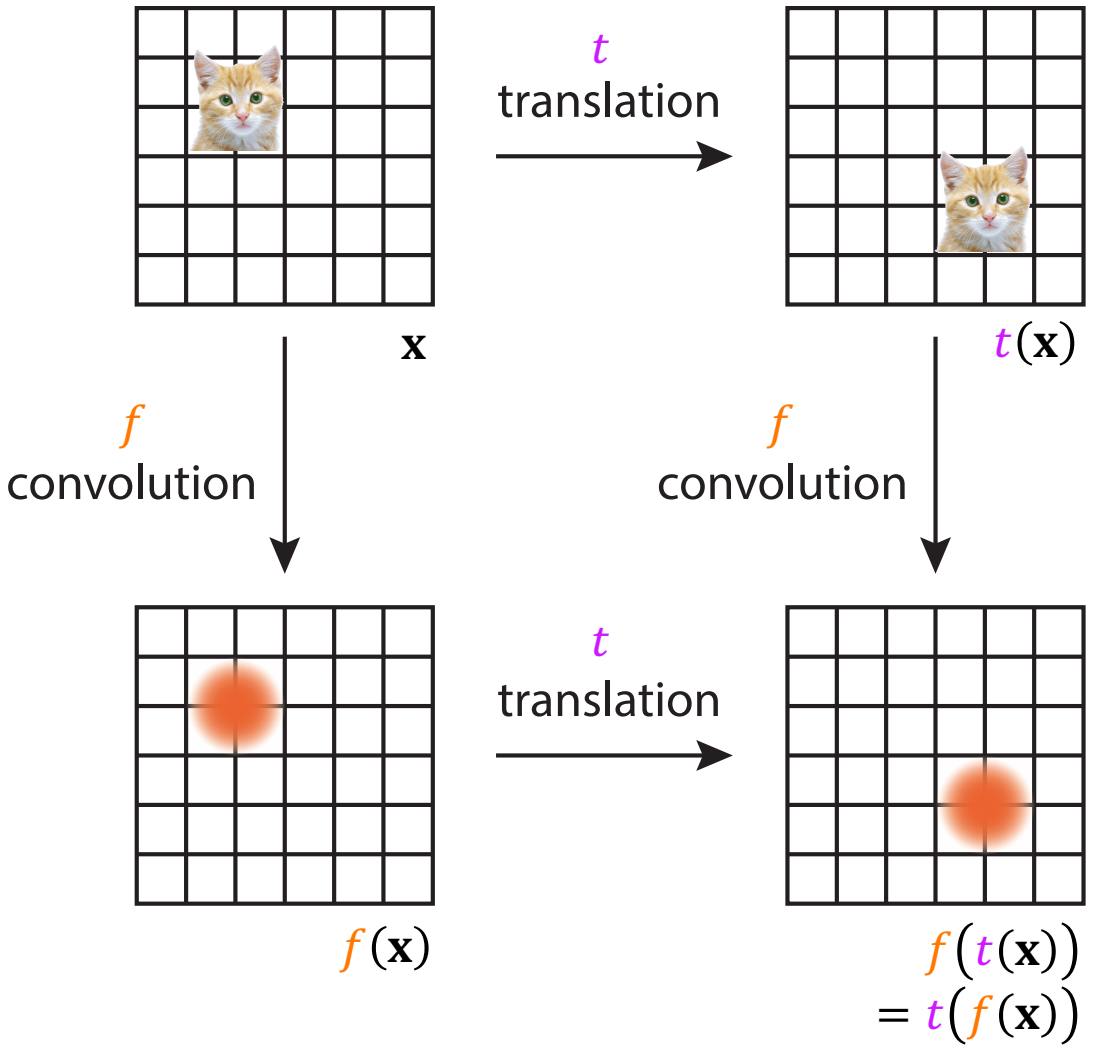


Image from: Boomsma & Frellsen. Convolutional models of molecular structure. MIA seminar, 2018.

Conv layers are translational equivariant

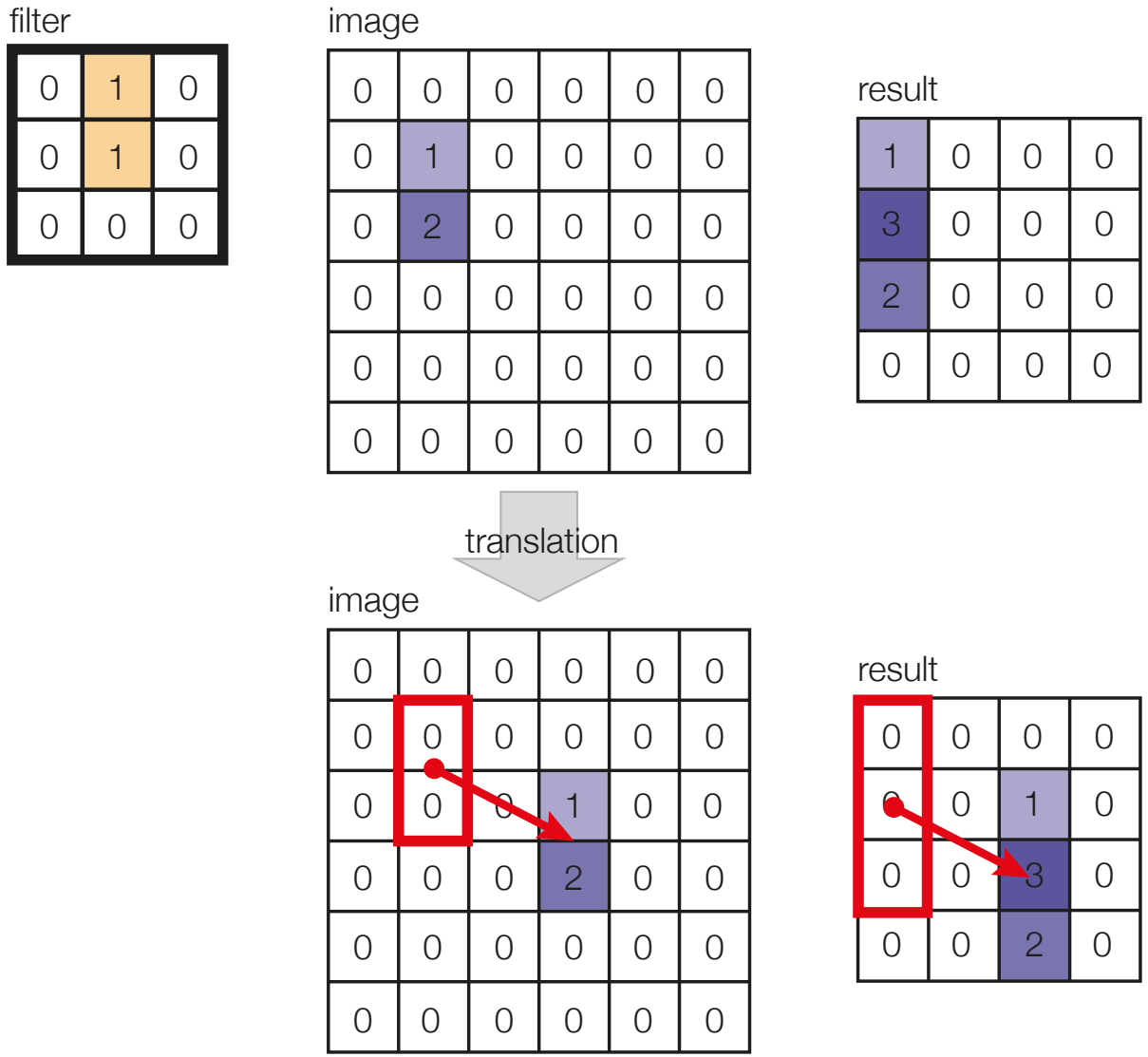
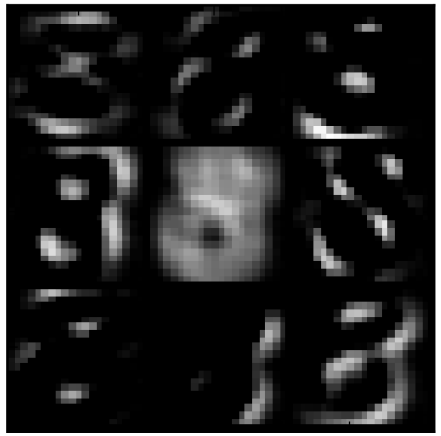
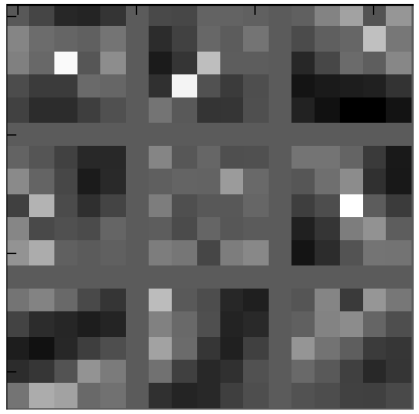


Image from: Boomsma & Frellsen. Convolutional models of molecular structure. MIA seminar, 2018.

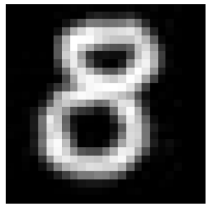
Example: CNN on MNIST

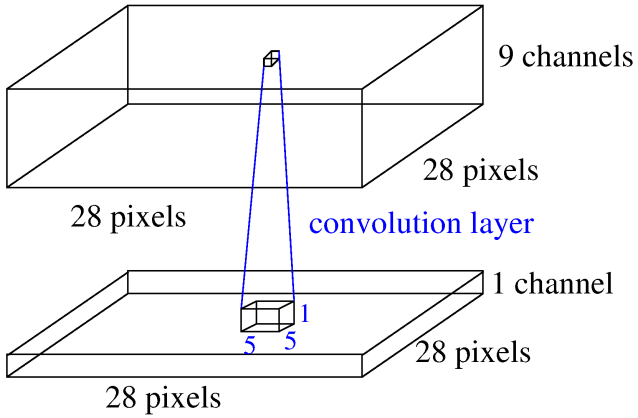


signals



weights





- MNIST data: 28×28 pixels and 1 channel
- First hidden layer: 28×28 pixels and 9 channels
- Use 5×5 filters

Image from by from slides by Tapani Raiko

Notation for convolutional layer

- Replace **vector** signals with **tensors** indexed by (x, y, c)
  - x, y are spatial coordinates and c is the channel
- Convolutions weights is also tensor $\mathbf{W} \in \mathbb{R}^{w \times h \times c_{\text{in}} \times c_{\text{out}}}$

$$h^{(j)}_{x,y,c_1} = \sigma\left(\sum_{c_0,m,n} h^{(j-1)}_{x+m,y+n,c_0} W_{m,n,c_0,c_1} + b_{c_1}\right)$$

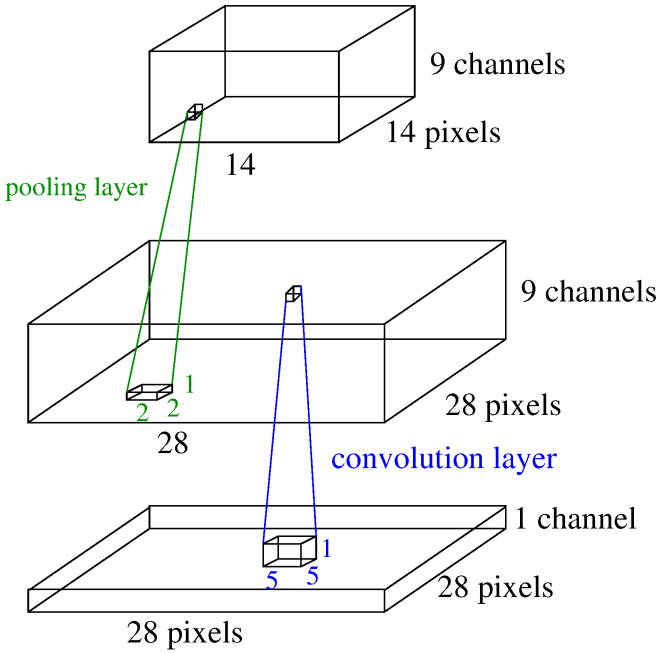
or

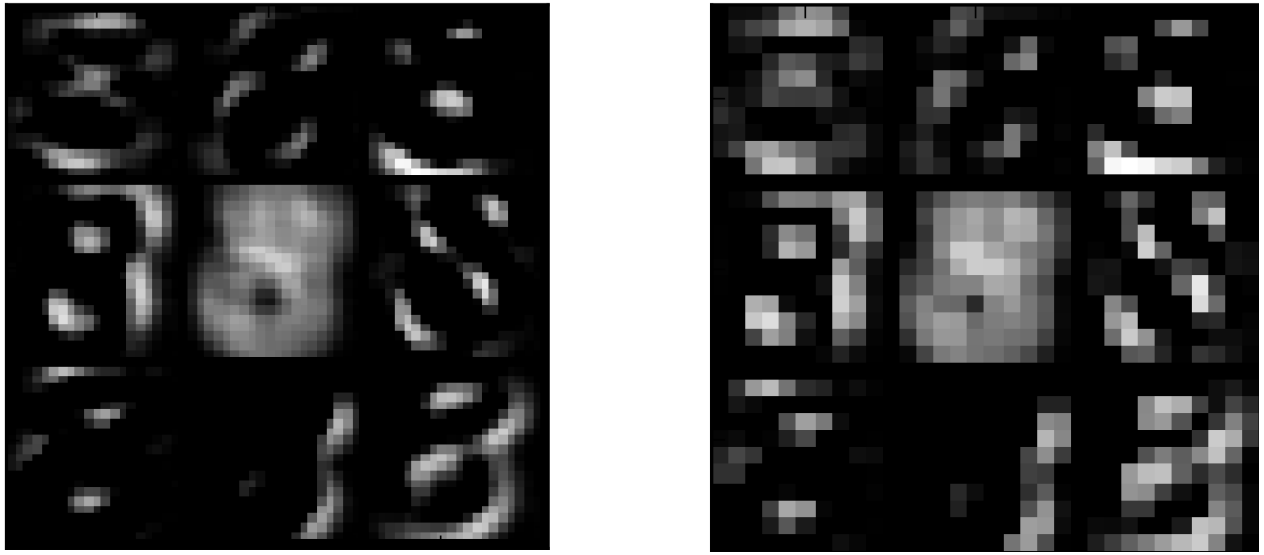
$$\mathbf{h}^{(j)}_{:,:c_1} = \sigma\left(\sum_{c_0} \mathbf{h}^{(j-1)}_{:,:,c_0} * W_{:,:,c_0,c_1} + b_{c_1}\right)$$

Padding: Input can be padded with zeroes to keep the same spatial size of the input and output.

Pooling

- Decrease resolution (pooling)
  - Increase channels going up (convolutions).
- Often down-sampling by hard-coded pooling layers.
- Here, use max(·) of 2×2 activations

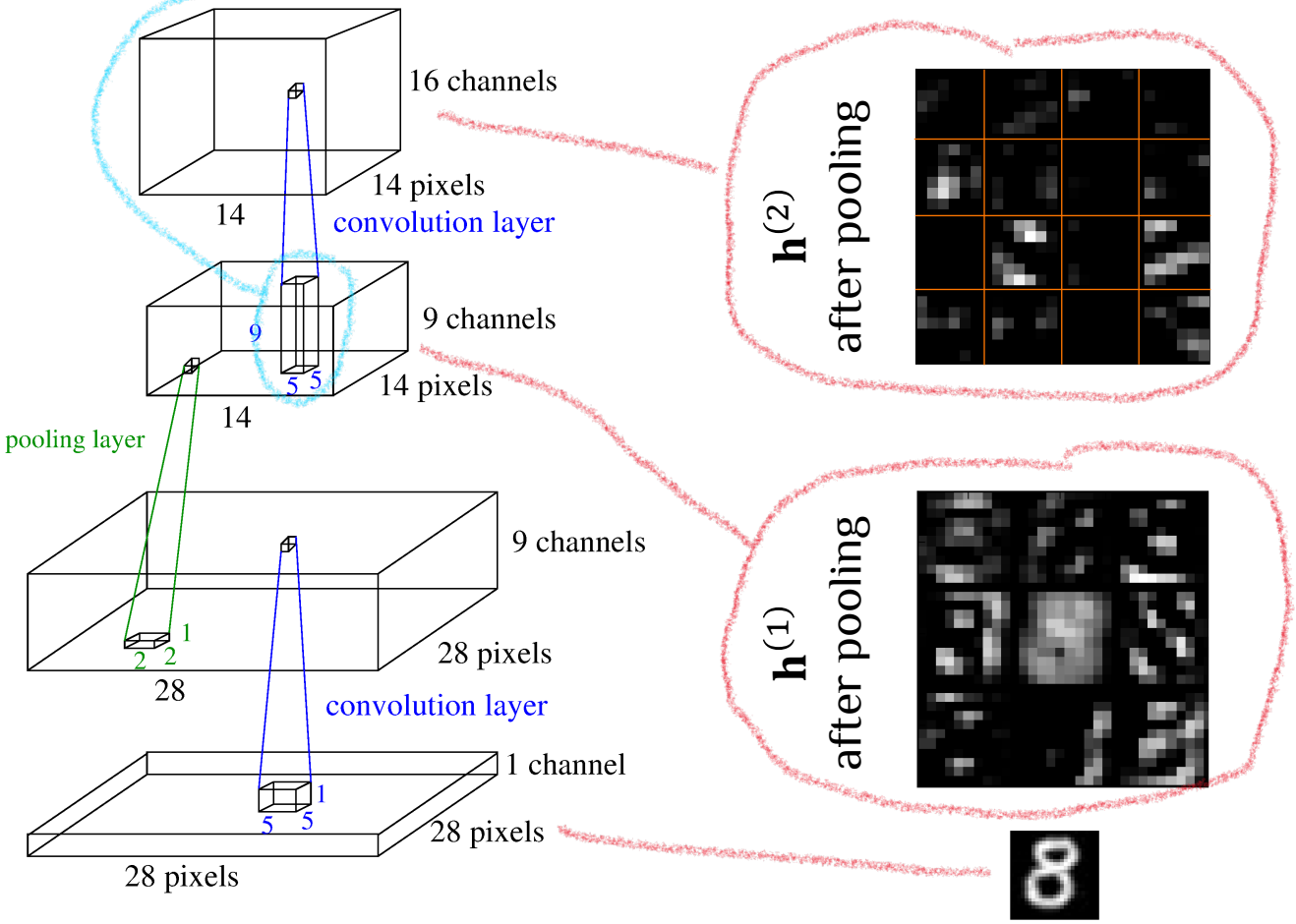


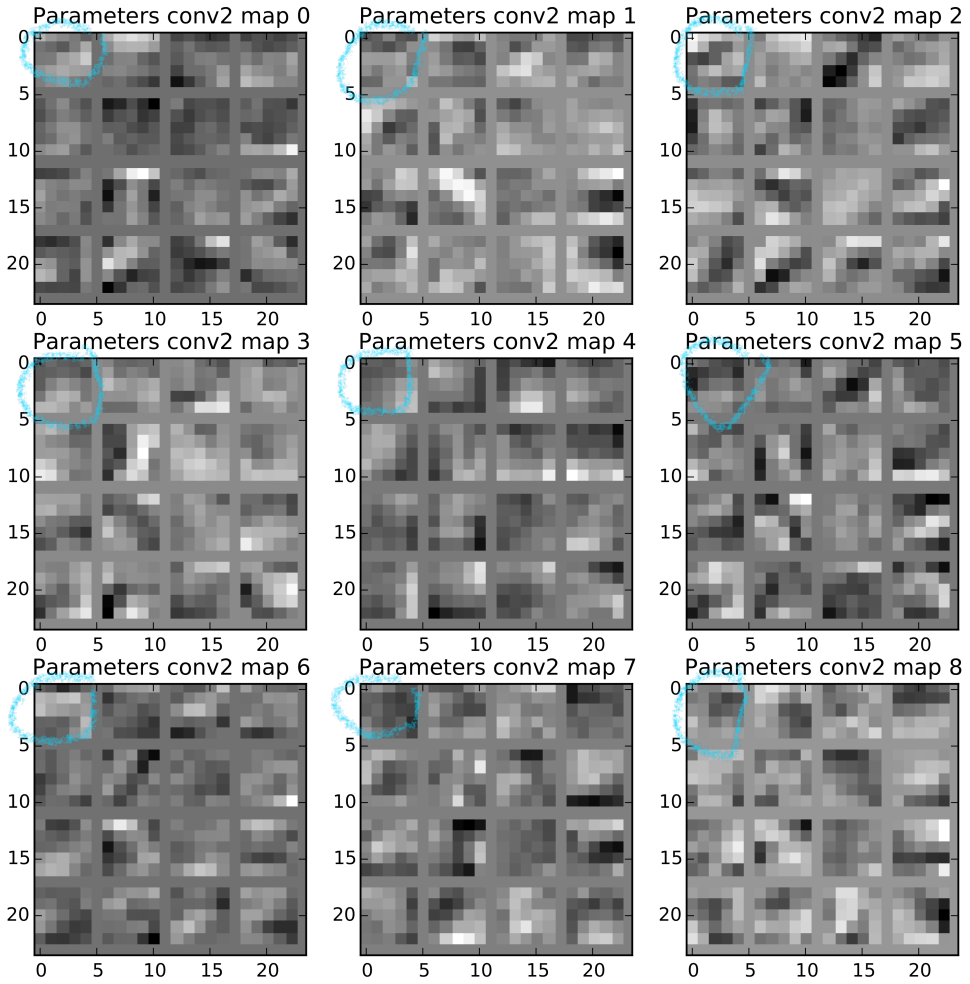


$\mathbf{h}^{(1)}$ before and after pooling

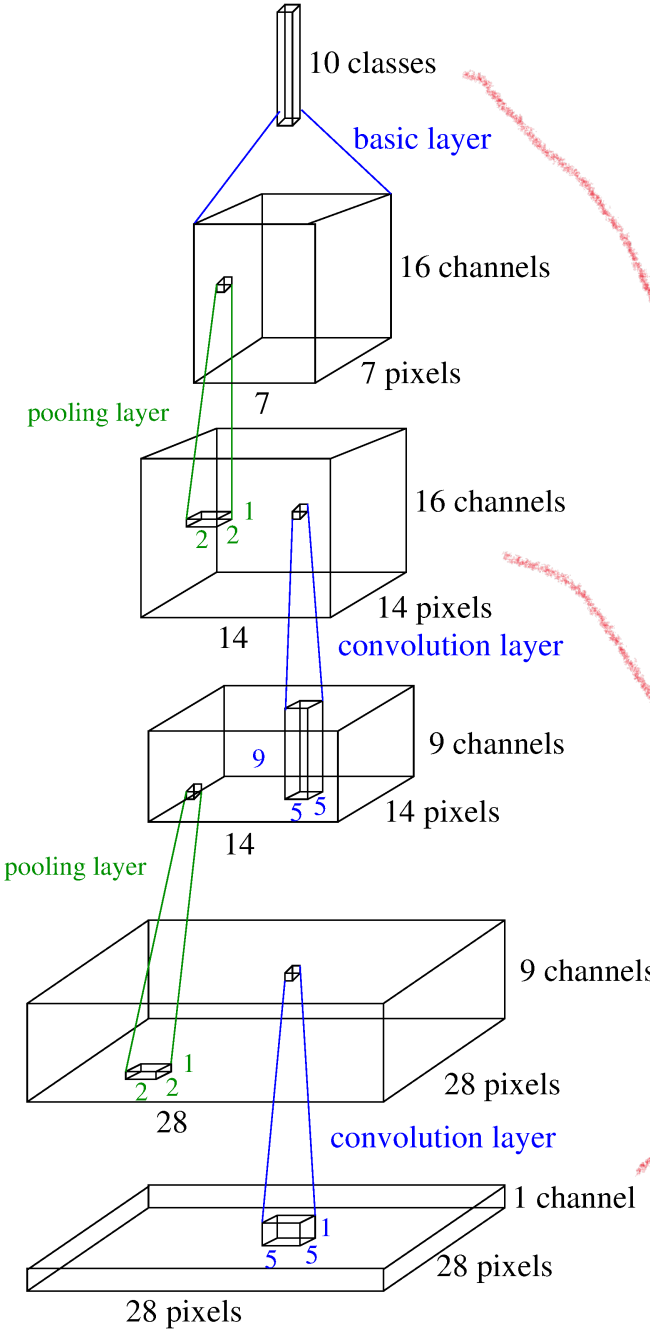
Stack more layers

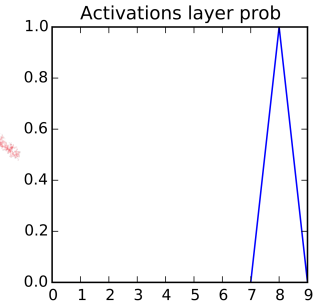
Note how each unit looks at all 9 channels of the previous layer

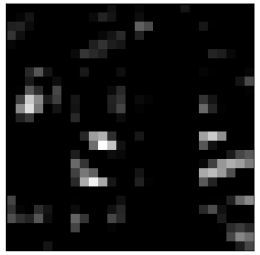


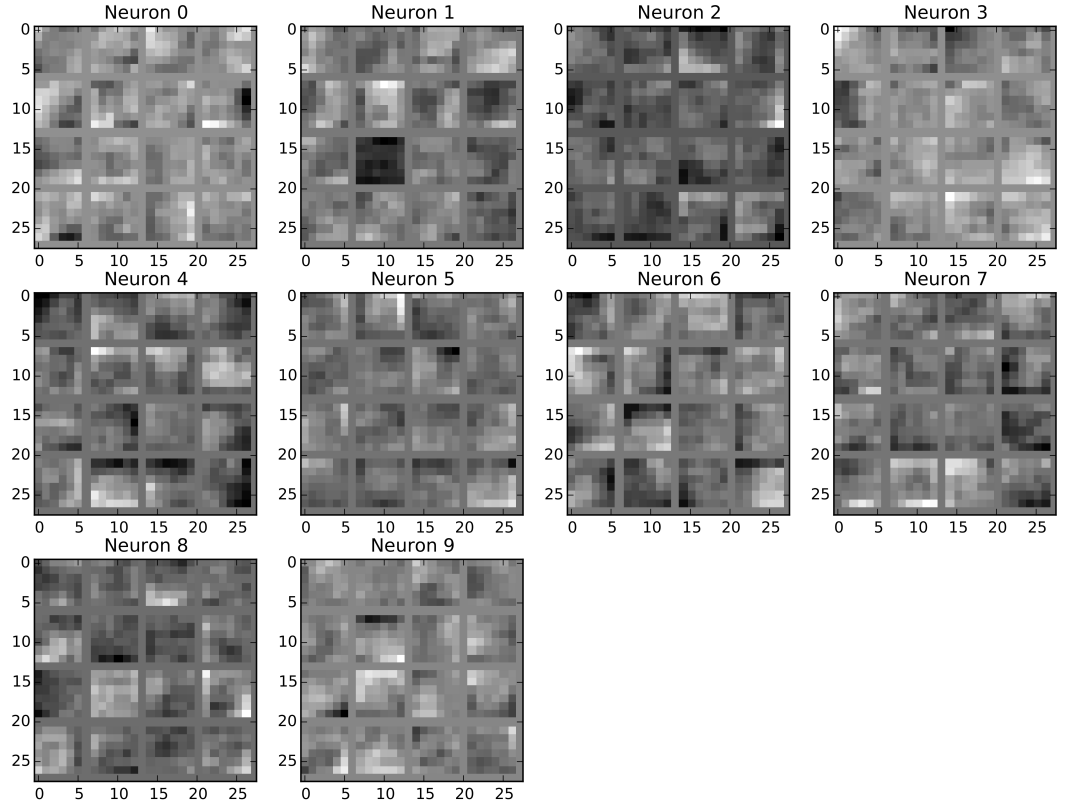


Whole network





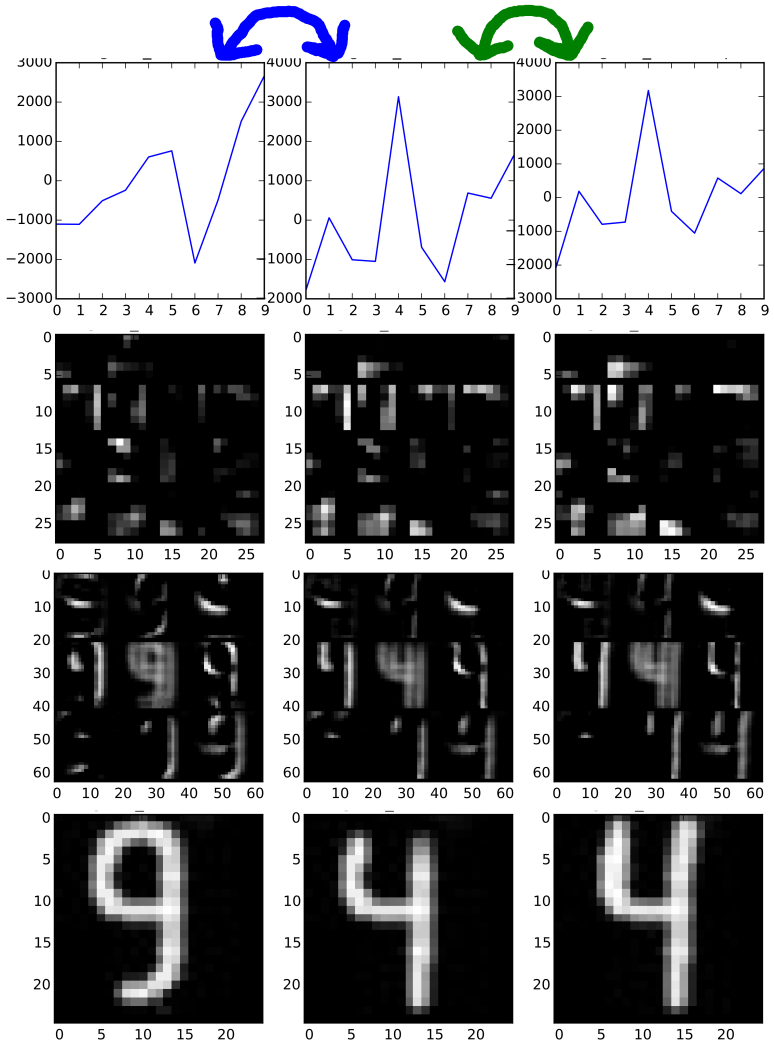


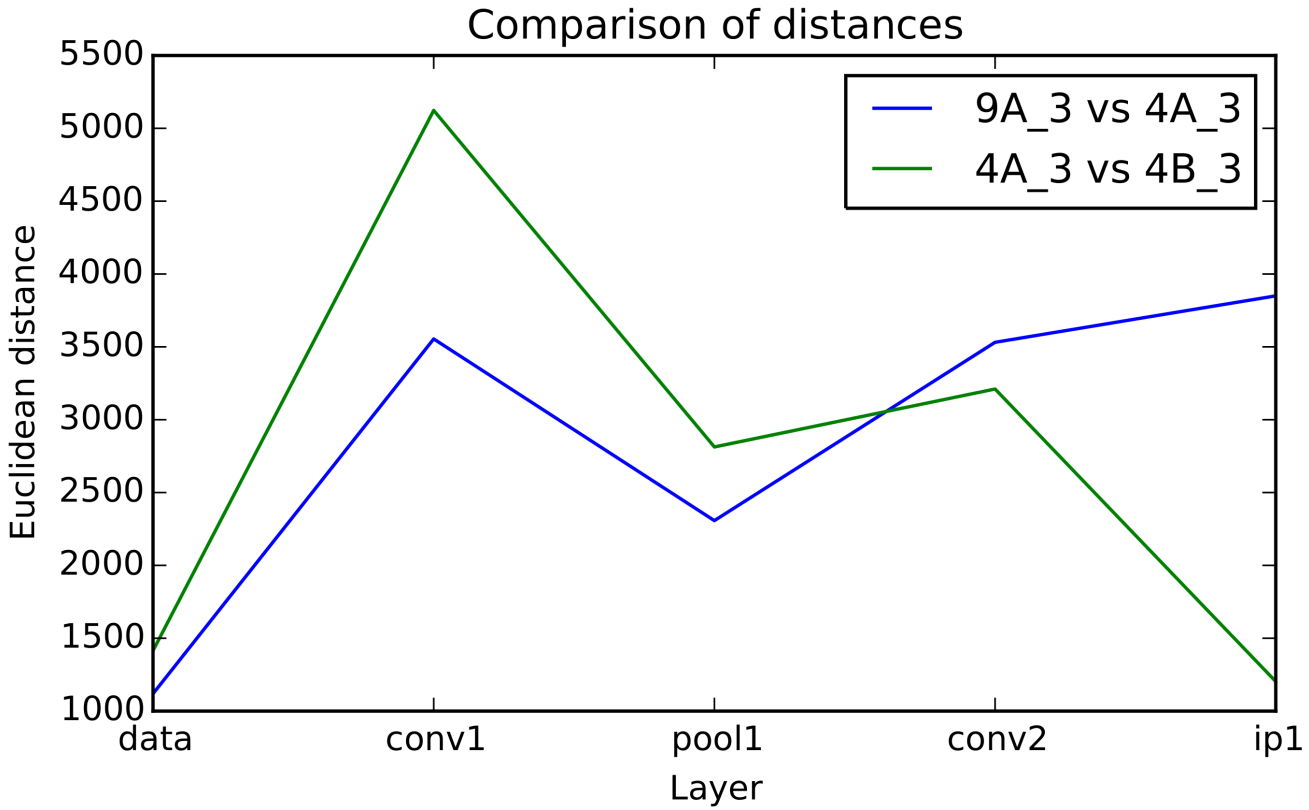


- At the top, the resolution drops to 1×1 pixel
- Train by back-propagation as usual

Image from by from slides by Tapani Raiko

Good learned representations?





- In pixel space, 9–4 is closer.
- In feature space, 4–4 is closer.

Image from by from slides by Tapani Raiko

Convolutional vs. non-convolutional

CNN example:

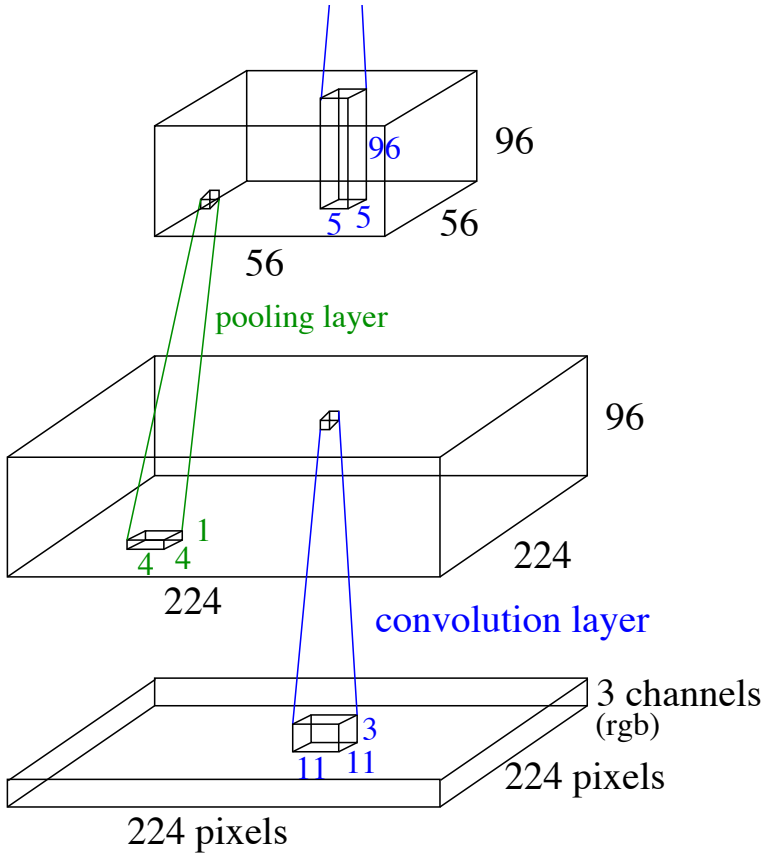
- Sizes of signals/features $\mathbf{h}$:
  $28 \cdot 28 = 784$ $28 \cdot 28 \cdot 9 = 7{,}056$ $14 \cdot 14 \cdot 16 = 3{,}136$
- Number of weights (ignoring biases):
  $5 \cdot 5 \cdot 9 + 5 \cdot 5 \cdot 9 \cdot 16 + 7 \cdot 7 \cdot 16 \cdot 10 = 225 + 3600 + 7840 = 11{,}665$

Compare to a MNIST FFNN example:

- Signals/features:
  $784, 225, 144$
- Weights:
  $28 \cdot 28 \cdot 225 + 225 \cdot 144 + 144 \cdot 10 = 176400 + 32400 + 1440 = 210{,}240$

Convolutional network has more signals/features but less parameter

Scaling up

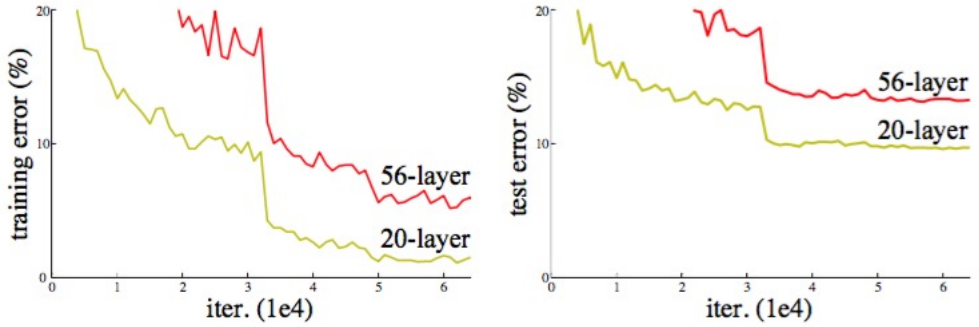


- **Trend in 2015 towards:**
  - Small filters (3×3)
  - Smaller pooling (2×2)
  - More layers (10 – 40)
- **Trends 2015+:**
  - Even smaller filters, sometimes 1×1,
  - Even more layers
  - Average pooling in transition from convolutional to fully connected layers

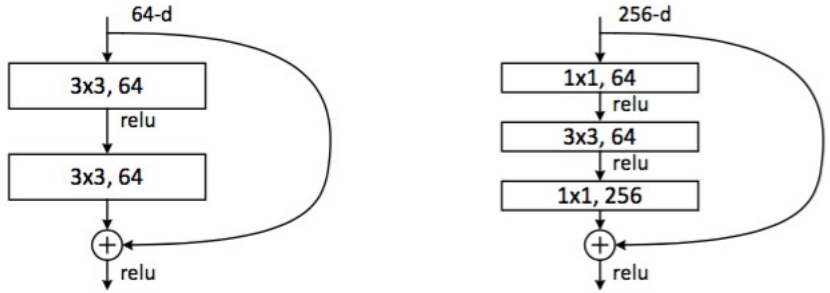
Image from by from slides by Tapani Raiko

Deep residual nets

- **Observation**: it is difficult to fit very deep nets

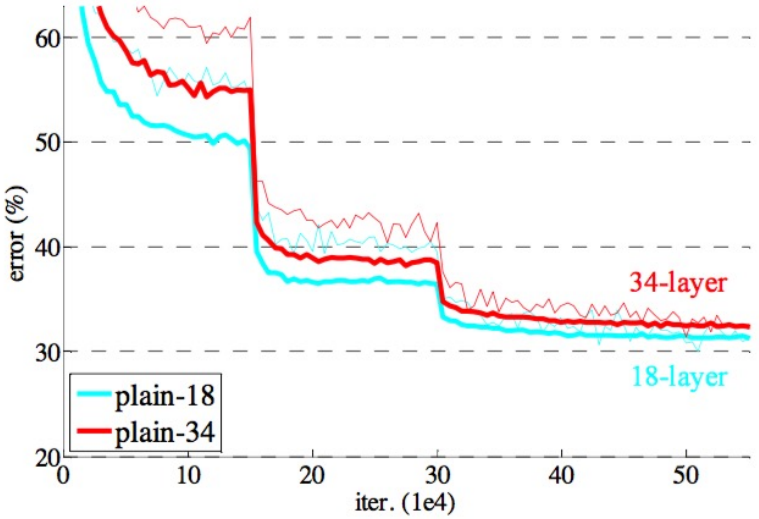


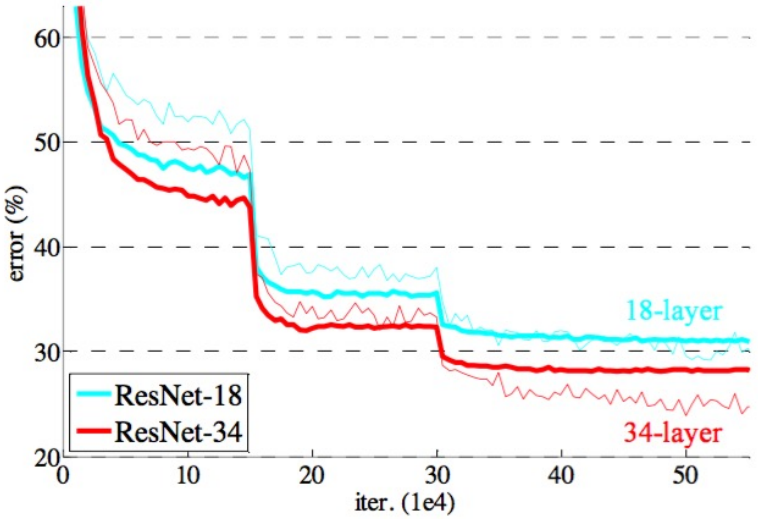
- **Solution**: skip layers $\mathbf{h}^{(\ell)} = \mathbf{h}^{(\ell-1)} + f_\phi\left(\mathbf{h}^{(\ell-1)}\right)$



- Initially $f_\phi\left(\mathbf{h}^{(\ell-1)}\right) \approx 0$ meaning net $\approx$ linear

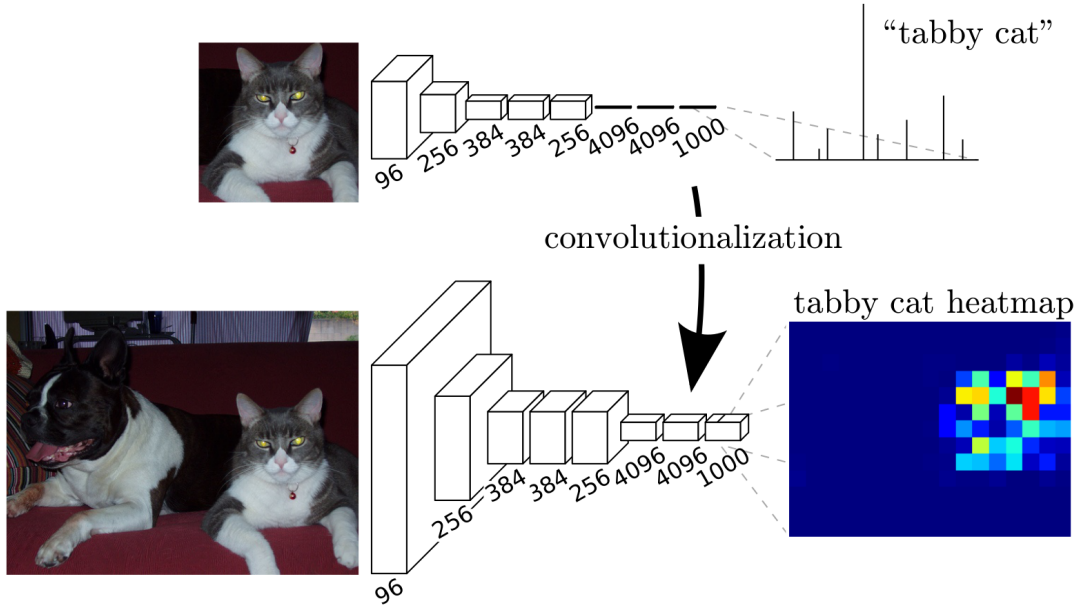
Deep residual nets: ImageNet results

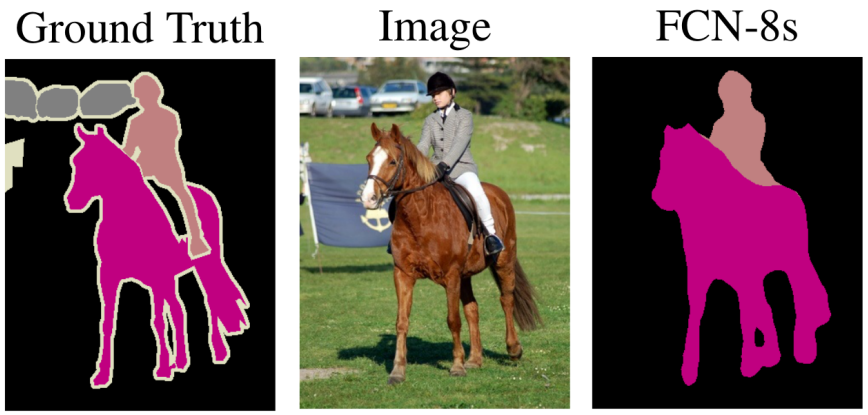




method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Application: Semantic segmentation

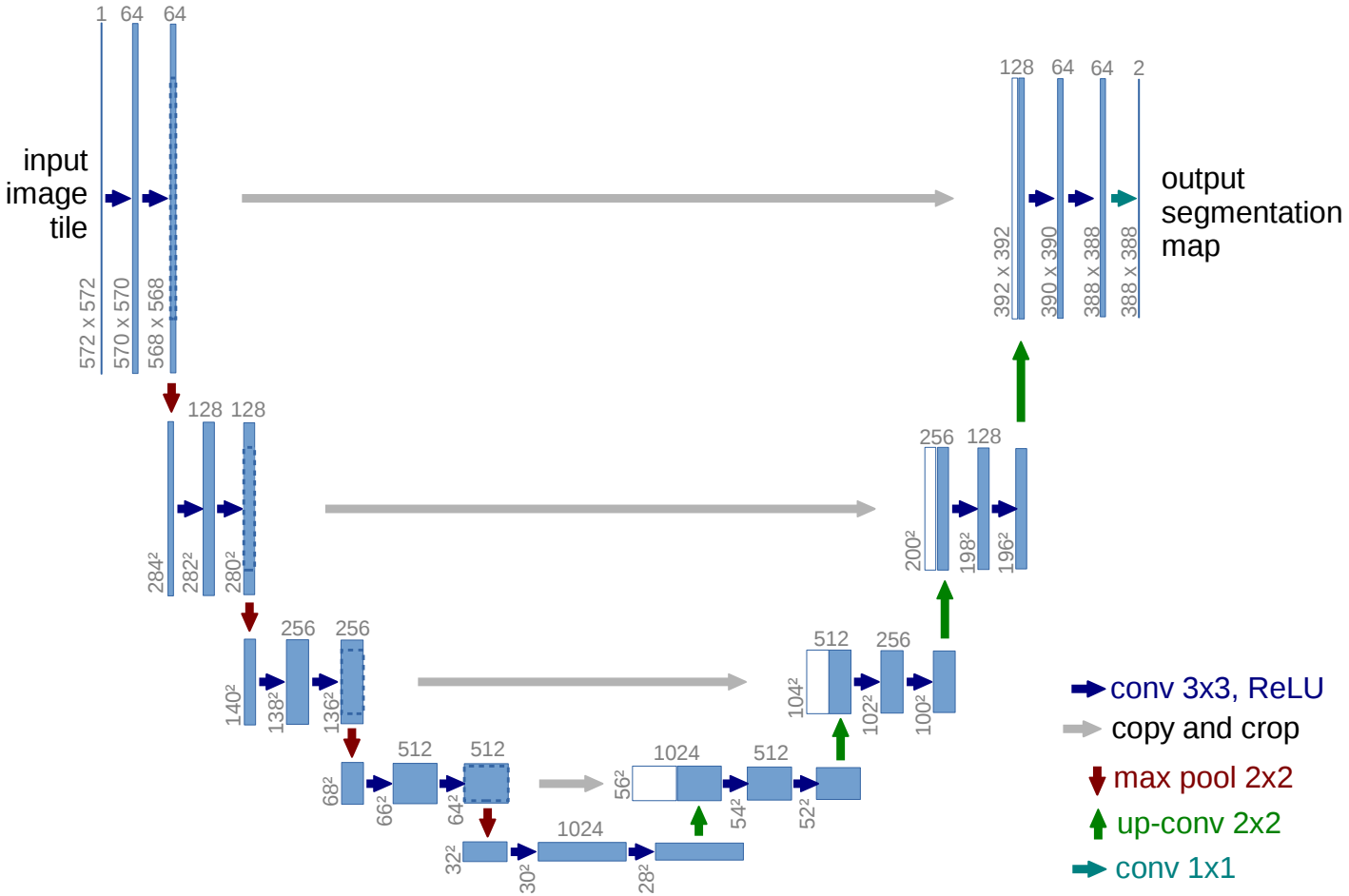




- Use a CNN to classify each pixel
- Lots of shared computation

Long, Shelhamer & Darrell. Fully convolutional networks for semantic segmentation. CVPR, 2015.

U-net: Utilising skip connections for segmentation



Ronneberger, Fischer & Brox. U-net: Convolutional networks for biomedical image segmentation. MICCAI, 2015.

Today's exercises!

- Three Python notebooks (`Week4 Convolutional`)
  - Implement CNNs in **PyTorch** MNIST and CIFAR-10
  - Transfer learning from CIFAR-10 to the Caltech101 datasets

- Three **problems** from to book on covered material

Thank you!