ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Introduction to Machine Learning

Neural networks (continued)

Dr. Kfir Levy
Learning and Adaptive Systems (las.ethz.ch)

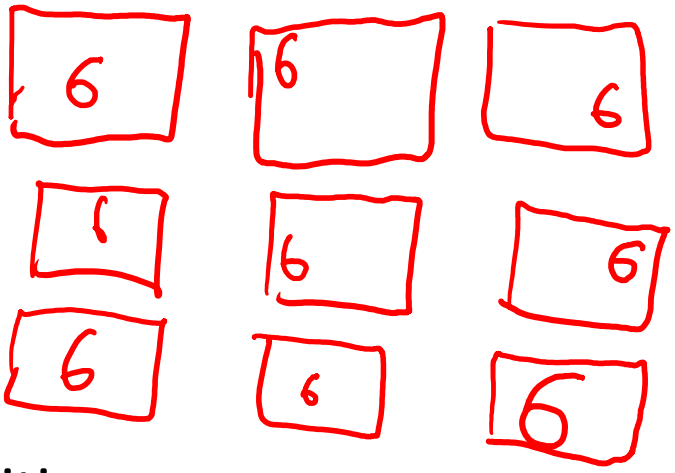
Invariances

- Predictions should be unchanged under some transformations of the data, e.g.
  - Classfication of handwritten-digits

shift invariance

rotation invariance

scale invariance



  - Speech recognition

pitch invariant features

speed of speech

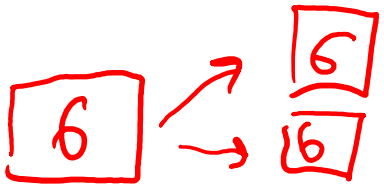
Invariances (contd.)

- How to encourage a model to learn specific invariances?

SIFT ~ Shift invariant feature transformation

Cepstrum (speech recognition features)

- Augmentation of the training set

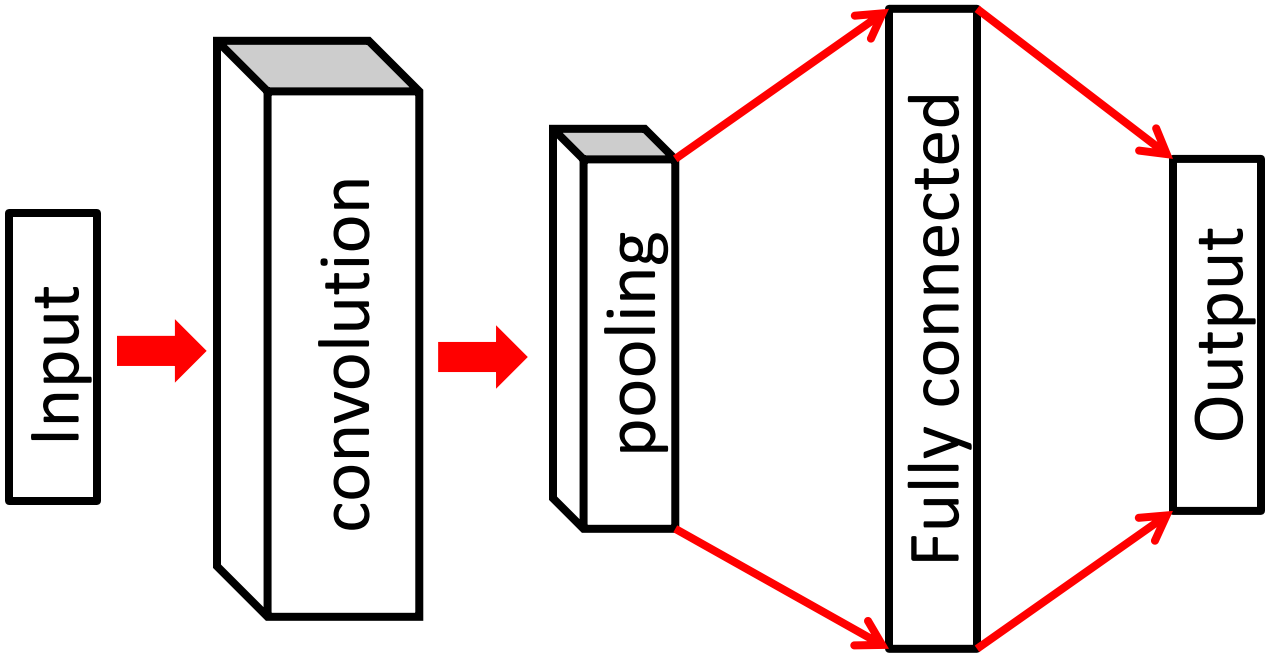


- Special regularization terms
- Invariance built into pre-processing
- Implement invariance into structure of ANN (→ e.g. using convolutional neural networks)

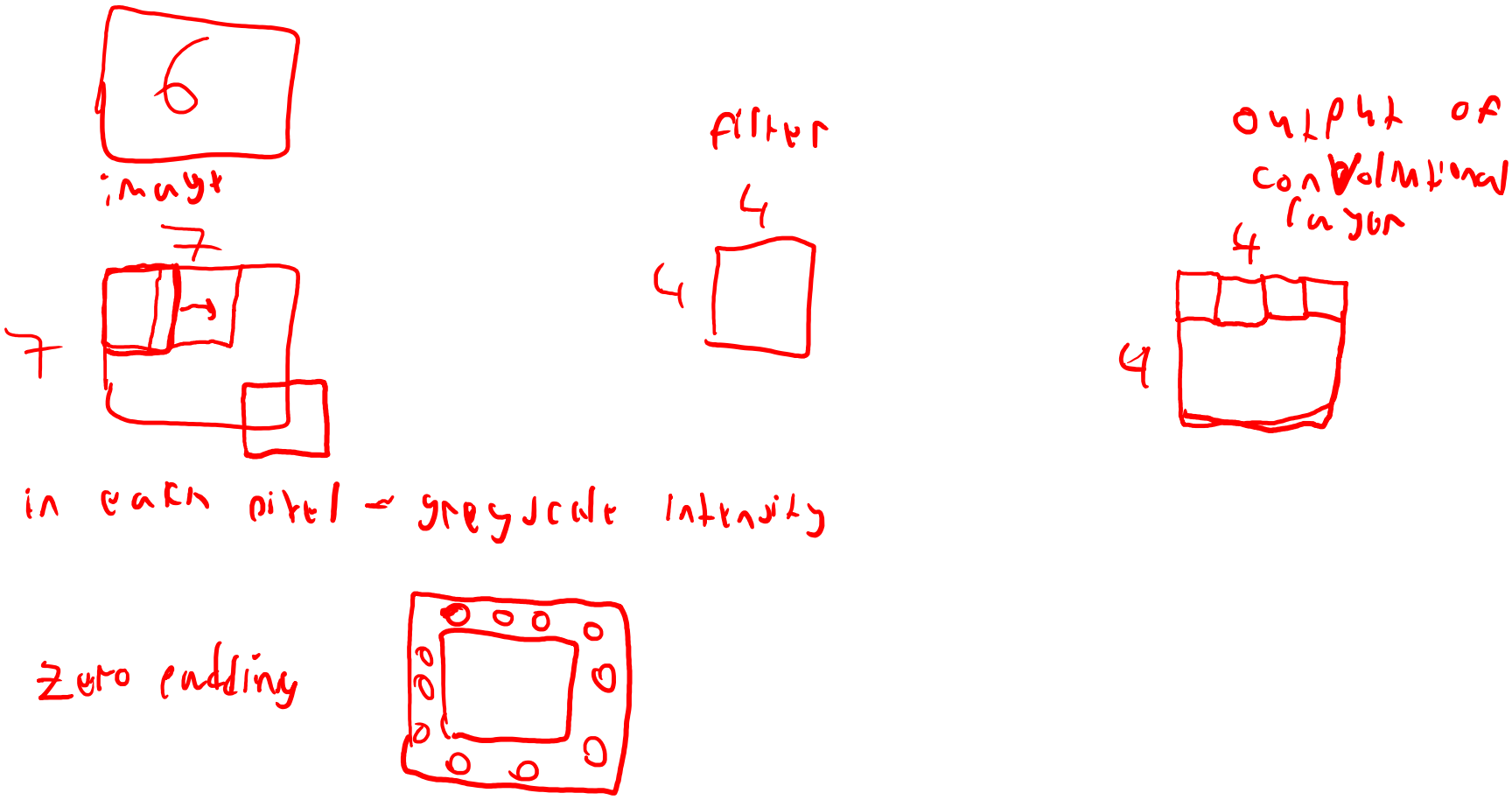
Convolutional neural networks

- Convolutional neural networks are ANNs for specialized applications (e.g., image recognition)
- The hidden layer(s) closest to the input layer shares parameters: Each hidden unit only depends on all „closeby“ inputs (e.g., pixels), and weights constrained to be identical across all units on the layer
- This reduces the number of parameters, and encourages robustness against (small amounts of) translation
- The weights can still be optimized via backpropagation

CNN architecture

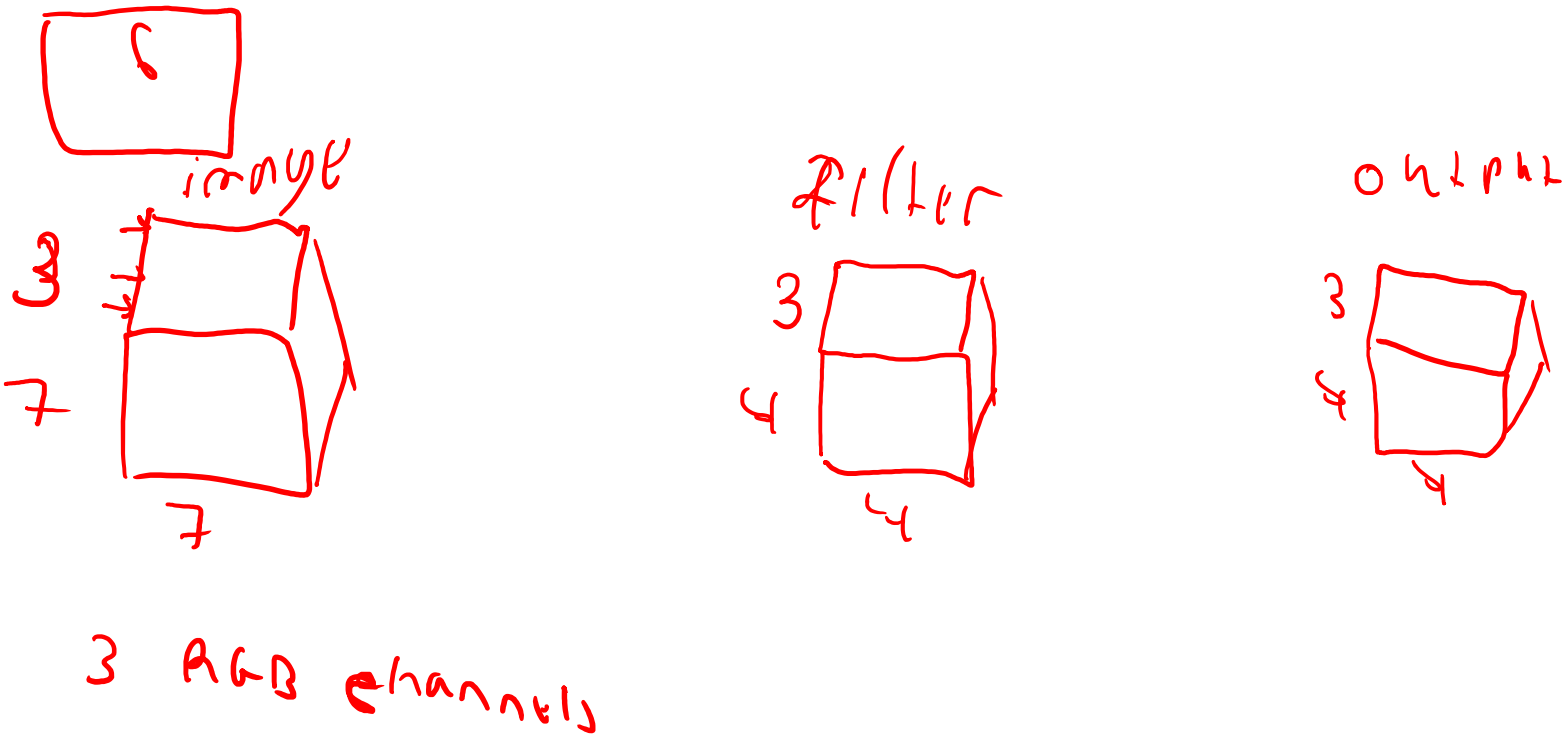


CNN examples

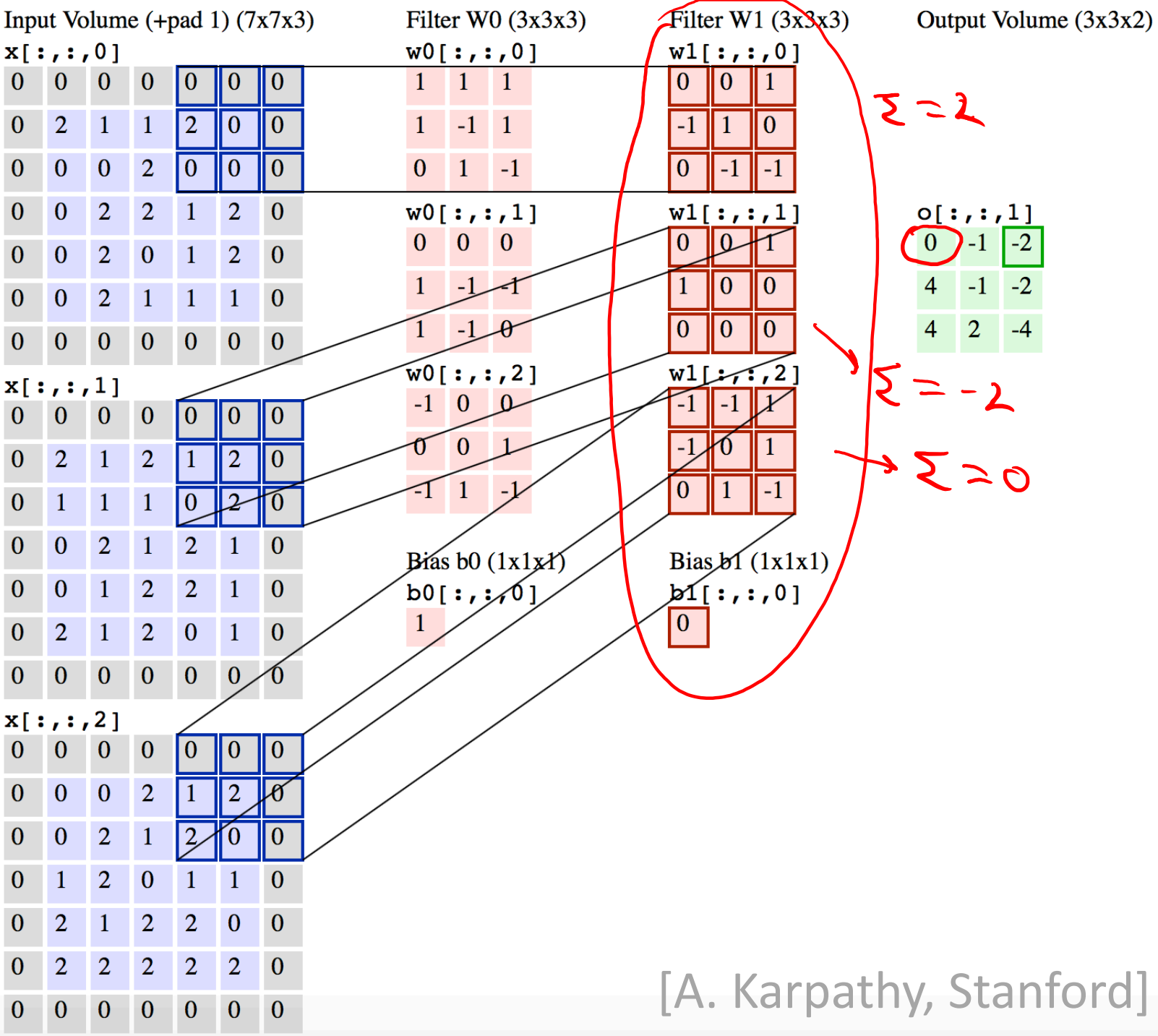


- **Terminology:** filters/kernels, stride, padding

CNN examples



- **Terminology:** filters/kernels, stride, padding



[A. Karpathy, Stanford]

Computing output dimensions

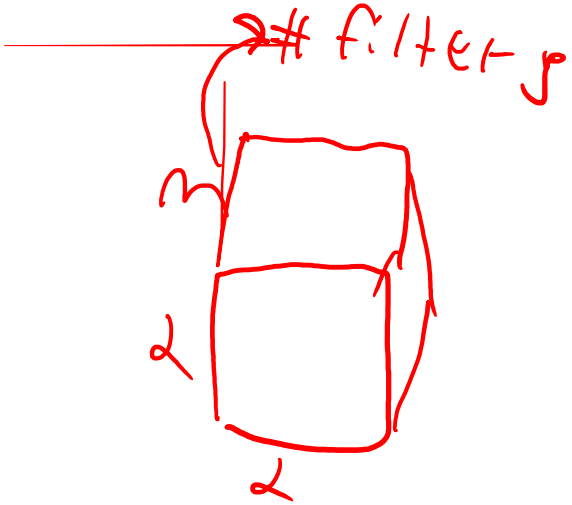
- Applying *m* different *f* x *f* filters
- To a *n* x *n* image
- Padding *p*
- Stride *s*

$f = 3$

$n = 7$

$p = 1$

$s = 2$

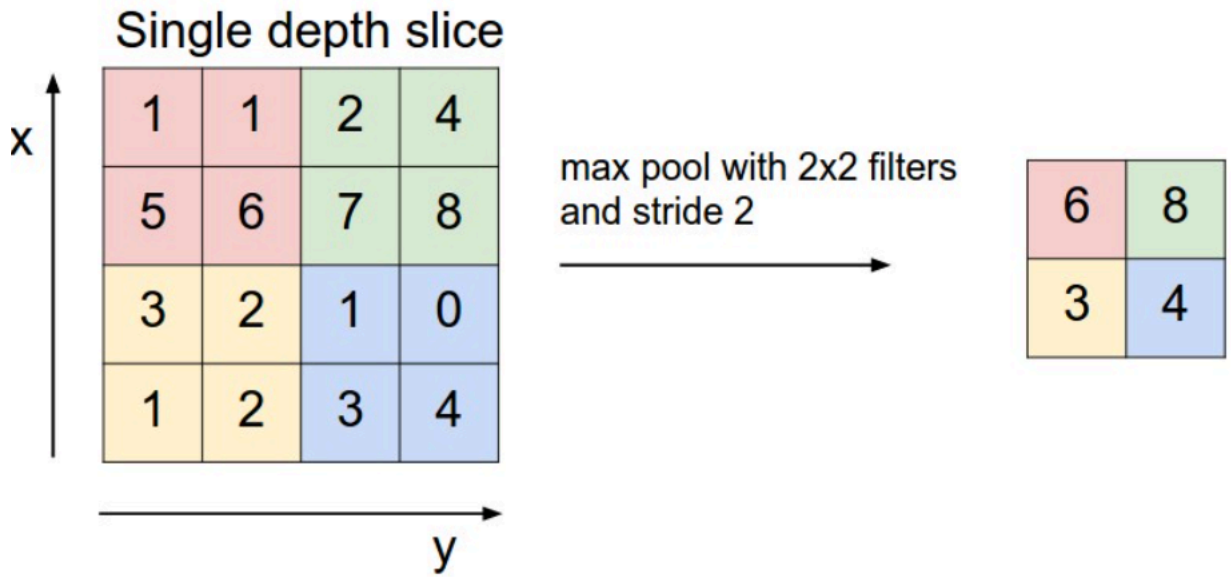


$$2 = \frac{n + 2p - f}{s} + 1$$

Pooling layers (subsampling)

- In some applications (e.g., image classification), it can make sense to aggregate several units to decrease the width of the network (and hence # of parameters)
- Usually, one considers either the average or the maximum value

Illustration of pooling layers

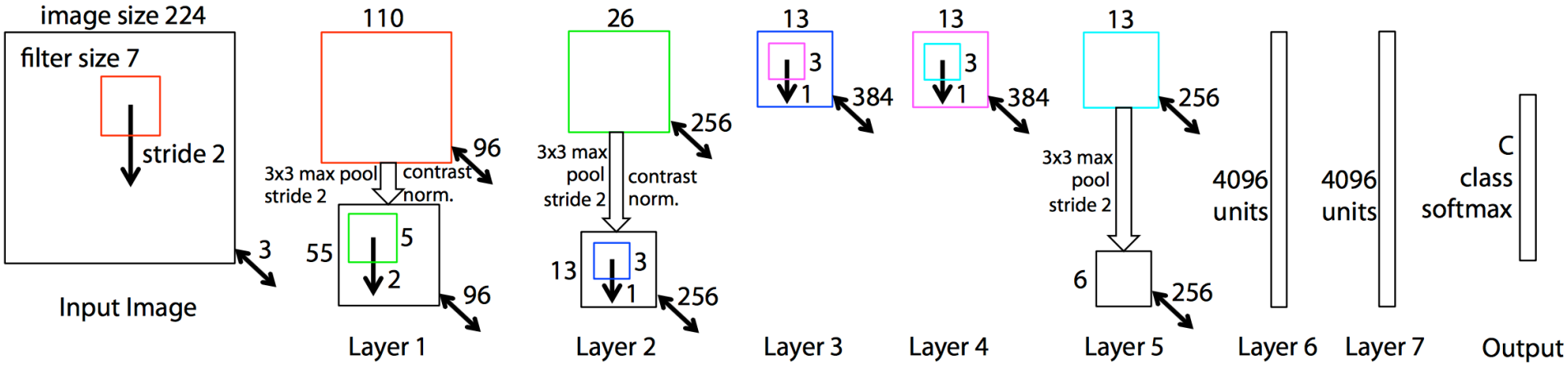


[A. Karpathy, Stanford]

CNN Demo

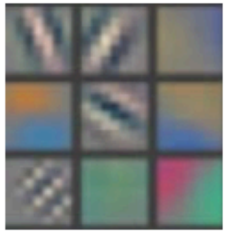
- http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Example CNN architecture



[Zeiler & Fergus '13]

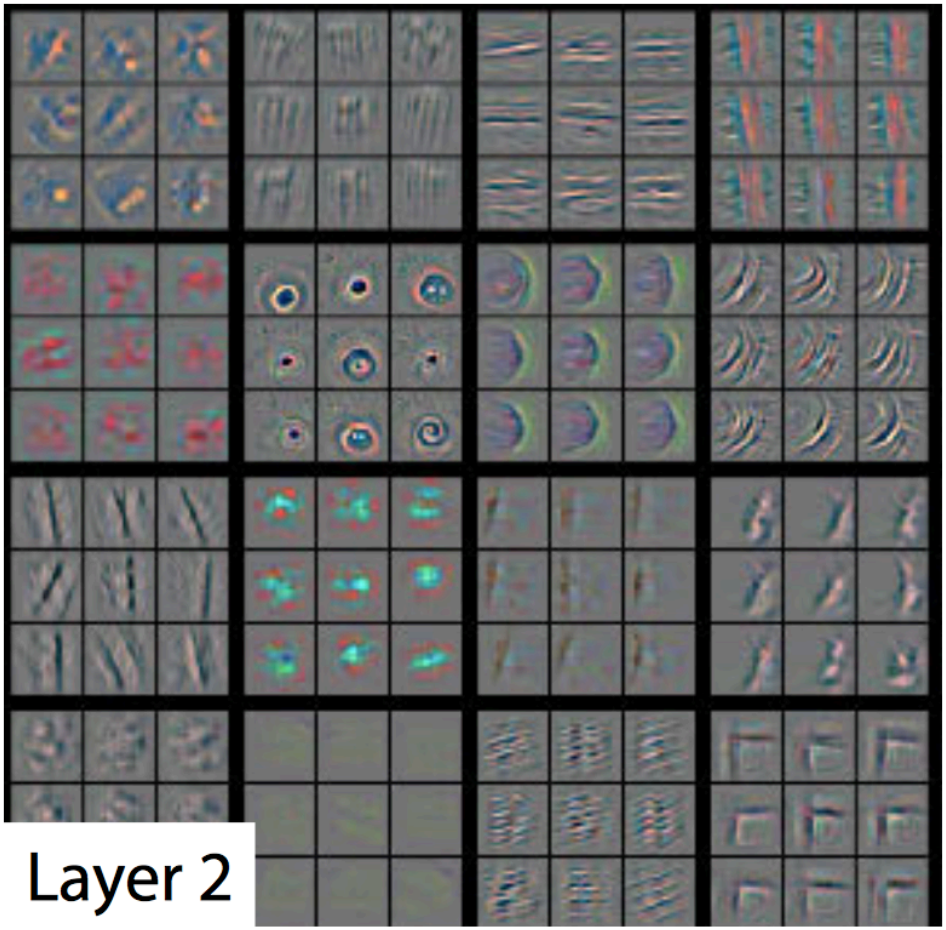
Visualizing the filters

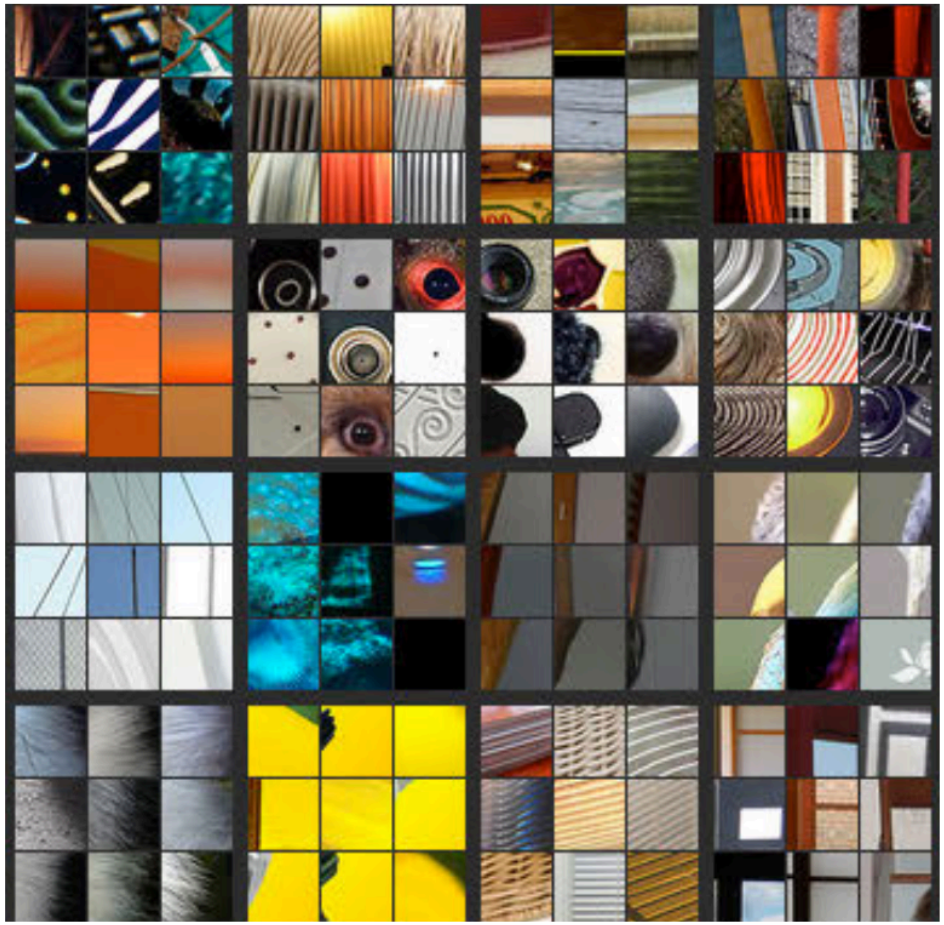


Layer 1

[Zeiler & Fergus ’13]

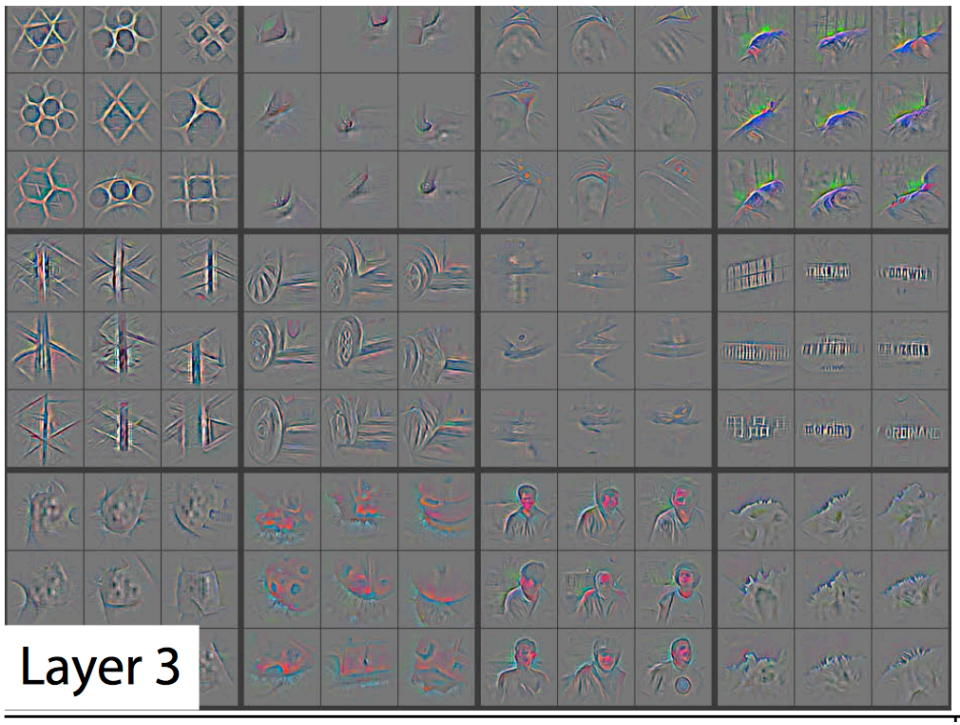
Visualizing the filters



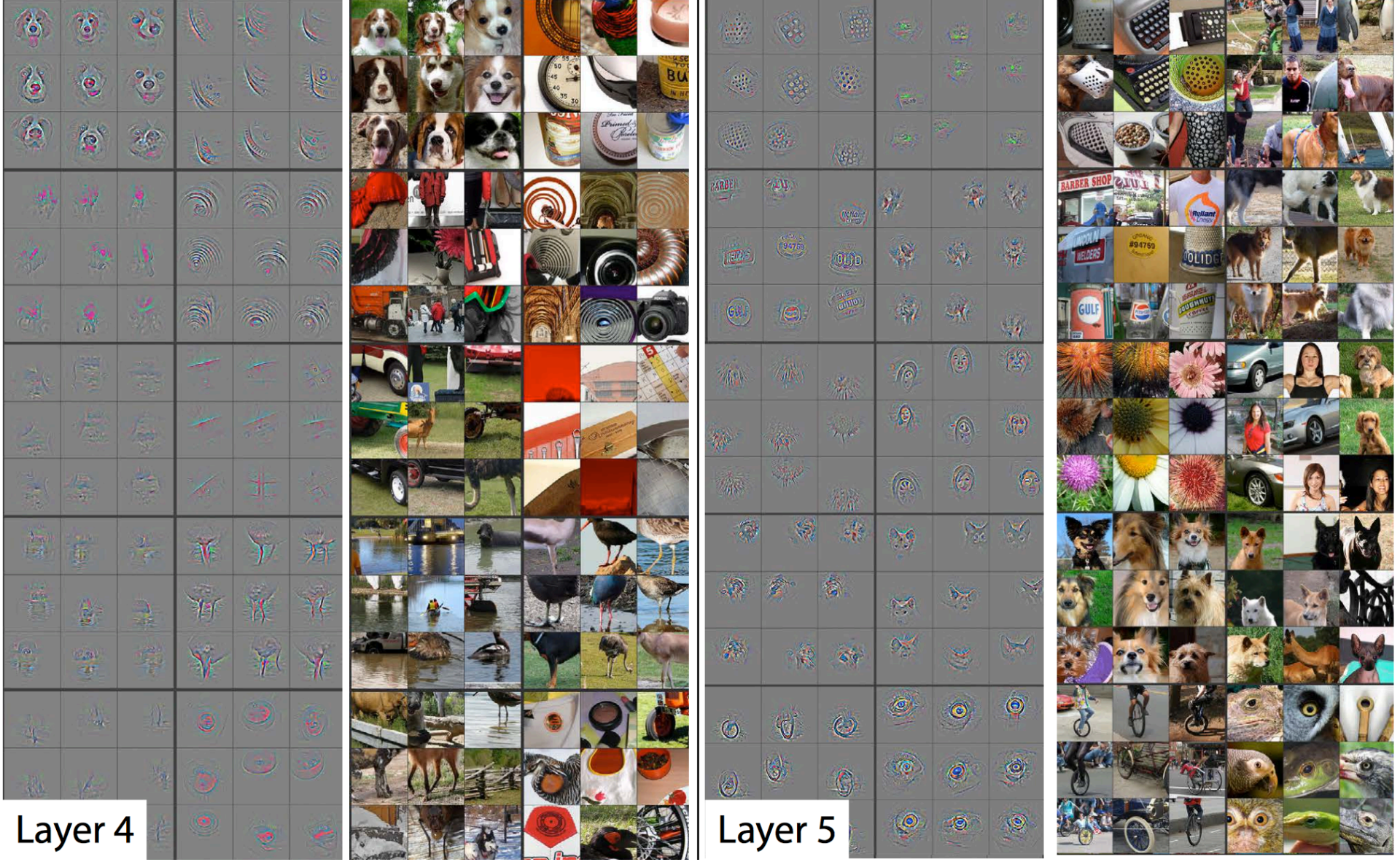


[Zeiler & Fergus ’13]

Visualizing the filters



[Zeiler & Fergus ’13]



[Zeiler & Fergus ’13]

Tutorial: Keras

Computational efficiency

- Key operation for backpropagation training:
  - Dense matrix vector multiplications
- Very suitable for implementation on General Purpose Graphical Processing Units (GPUs)
- Much work on special purpose hardware (e.g., TPUs)
- Responsible for large improvements in recent years

Choice of activation functions

- Several possible activation functions available
- Popular in the past: sigmoid / tanh activations
  - Differentiable!

- Currently used extensively: Rectified linear units (ReLUs)
  - Not differentiable ☹
  - Very fast to compute ☺
  - Gradients do not „vanish“ (important for deep networks) ☺

Parameter / architecture choice

- A number of parameters have to be chosen
  - Number and width of hidden layers
  - Use convolution / pooling layers? How many?
  - Skip connections
  - Type of activation functions
  - Weight initialization
  - Learning rate schedule
  - Regularization method
- Frameworks for autodifferentiation (e.g., Theano, Torch, TensorFlow)
- Can use cross-validation to compare models (but training is usually very expensive)

  ➔ Often use single validation („development“) set

  ➔ Much work on ML-based experimentation (“AutoML“)

Connection: ANNs vs. kernels

- When using the *tanh*-activation function, an ANN with a single hidden layer learns functions of the form

$$f(\mathbf{x}) = \sum_{i=1} w_i \tanh \theta_i^T \mathbf{x}$$

- This is exactly the same type of functions learned with kernels, when using the *tanh* kernel

$$f(\mathbf{x}) = \sum_{i=1} \alpha_i \tanh \mathbf{x}_i^T \mathbf{x}$$

$k(x_i, x) = \langle \phi(x_i), \phi(x) \rangle$

- **Difference**: kernels optimize α's only ➔ convex
  ANNs optimize both w's and θ_i ➔ non-convex

Comparison: Kernels vs. ANNs

Method	*Kernels*	*ANNs*
Advantages	Convex optimization, no local minima Robust against noise Models grow with size of data ☺	Flexible nonlinear models, with fixed parameterization Multiple layers discover representations at „multiple levels of abstraction“
Disadvantages	Models grow with size of data ☹ Don‘t allow „multiple layers“	Many free parameters / architectural choices that need to be tuned Often suffer from very noisy data

Supervised learning summary so far

Representation/ features	Linear hypotheses; nonlinear hypotheses with nonlinear feature transforms, kernels, learn nonlinear features via neural nets		
Model/ objective:	Loss-function	+	Regularization
	Squared loss, 0/1 loss, Perceptron loss, Hinge loss, cost sensitive losses, multi-class hinge loss		L^2 norm, L^1 norm, early stopping, dropout
Method:	Exact solution, Gradient Descent, (mini-batch) SGD, Reductions		
Evaluation metric:	Mean squared error, Accuracy, F1 score, AUC, Confusion matrices		
Model selection:	K-fold Cross-Validation, Monte Carlo CV		