Convolutional Neural Networks

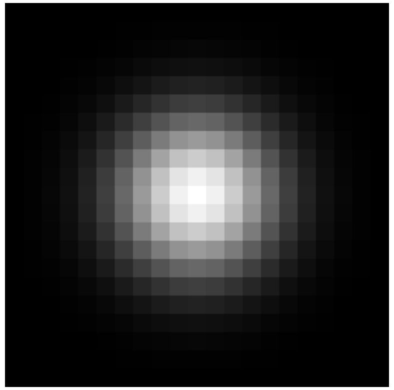
Computer Vision – Lecture 07

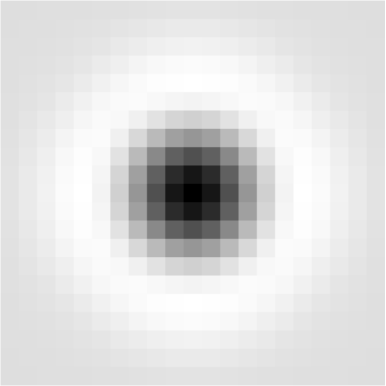
Further Reading

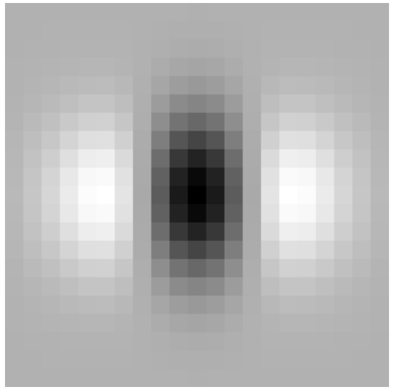
- Slides from F Li and slides from M Niessner
- Slides from E Gavves
- Deep Learning Book, by G,B,C, Chapter 9
- Foundations of Computer Vision, Torralba, Isola, Freeman

Convolutional Filters

- We have seen many useful convolutional filters:
  - Gaussian Blur
  - Edge Filters
  - Sharpening
  - Laplacian of Gaussian
  - Gradient Filters
  - …
- They have been hand-crafted from equations and intuitions.







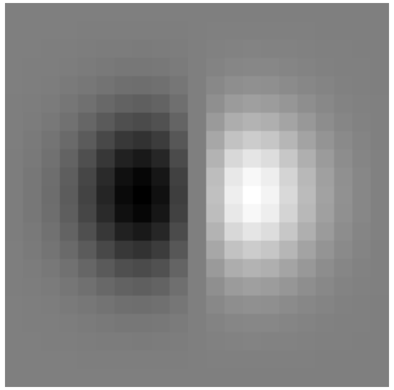


Image Classification in 2 Steps

1. Compute image embeddings.
2. Learn a classifier on the training set.

Many different approaches

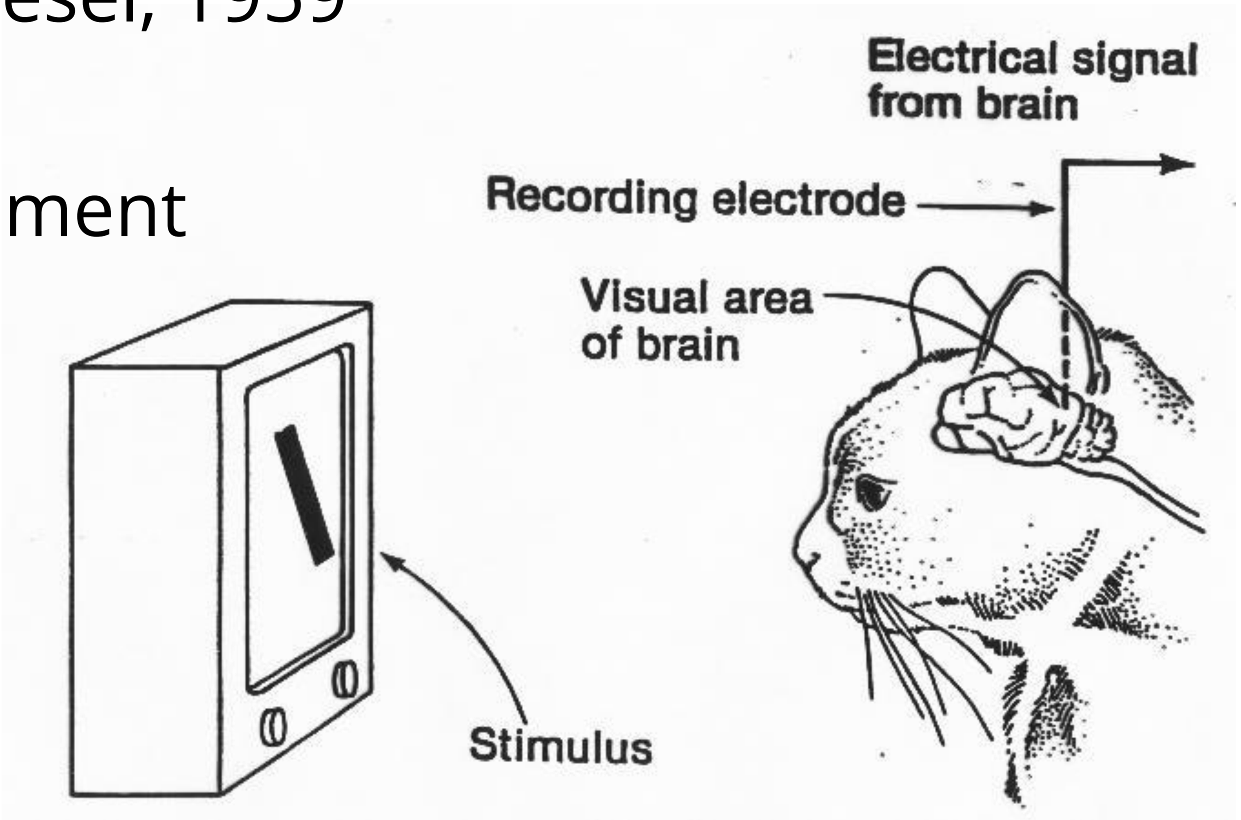
Image embeddings: FFT, BoW, HOG, Fisher Vectors, etc…

Classifiers: Linear Regression, SVMs, Kernel SVM, Random Forest, etc…

Biological Motivation

Receptive fields of single neurones in the cat's striate cortex, D. H. Hubel and T. N. Wiesel, 1959

- Measure single neuron excitement in the visual cortex.
- Neurons respond to oriented edges

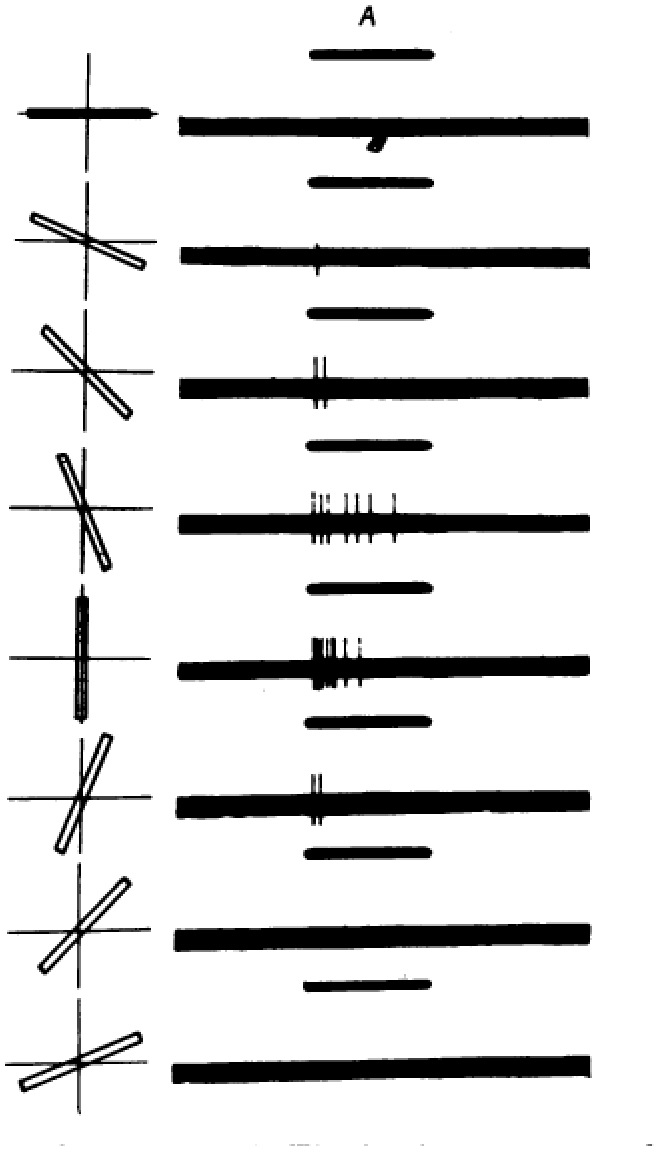


Biological Motivation

video source

Edge “Filters”

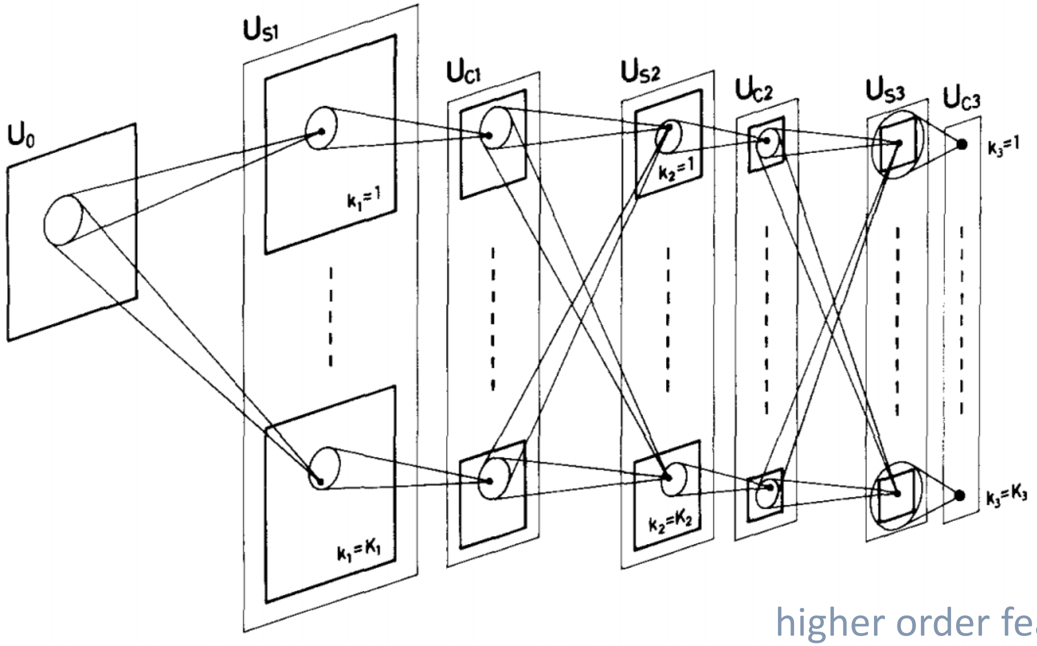
- A individual neuron responds to a quite precise angle of the edge.
- There are neurons for every angle.
- Remember: SIFT Descriptor.



Neocognitron

A neural network, inspired by Hubel and Wiesel (1959):

- Cascaded structures: hierarchical and multi-layer



higher order features

Fukushima, K. "Neocognitron: A hierarchical neural network capable of visual pattern recognition." Neural networks 1.2 (1988): 119-130.

Neocognitron

A neural network, inspired by Hubel and Wiesel (1959):

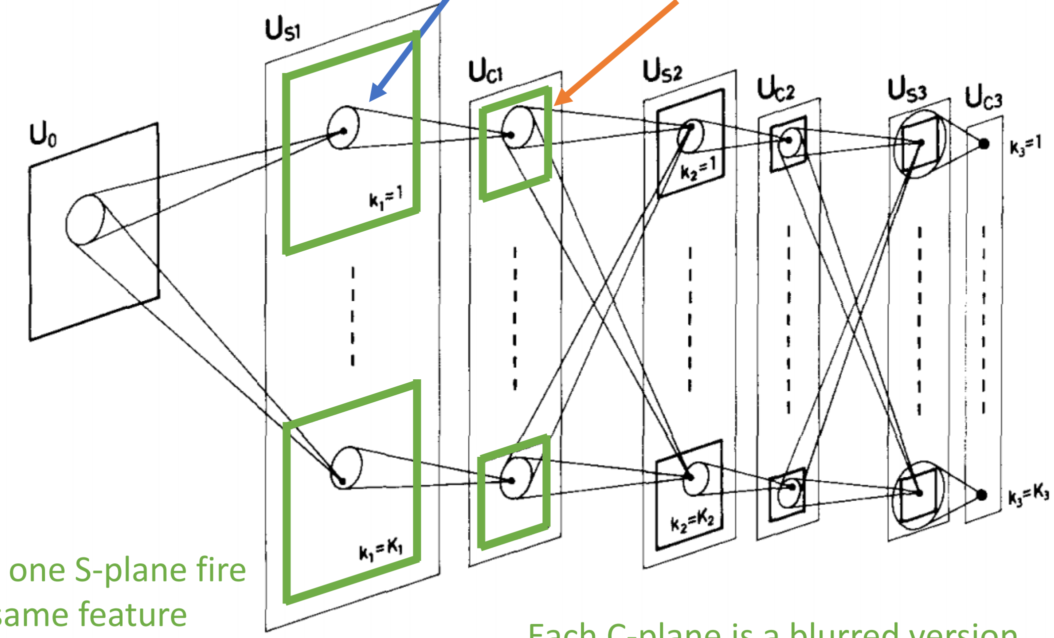
- Cascaded structures: hierarchical and multi-layer

- Different types of alternating “cells”
- Simple S-cells:
  extract local features
  (modifiable parameters)
- Complex C-cells:
  shift-invariance

- Arrangement in cell-planes,
  each responding to a feature,
  i.e. local connections with
  shared set of weights

each cell with a local connection area

ensure tolerance to small shifts of feature in the connection area



Cells on one S-plane fire on the same feature

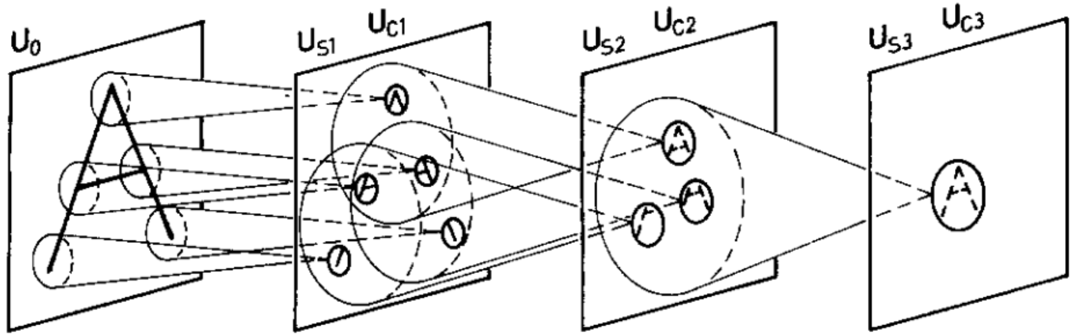
Each C-plane is a blurred version of its corresponding S-plane

Fukushima, K. "Neocognitron: A hierarchical neural network capable of visual pattern recognition." Neural networks 1.2 (1988): 119-130.

Neocognitron

Local features are gradually integrated and classified in higher layers:

- Basic features (edges, corners, etc.) in lower layers
- Global patterns in higher layers



Self-organizing maps:

- Initially no supervision (1980)
- Trained layer wise (1988)

Fukushima, K. "Neocognitron: A hierarchical neural network capable of visual pattern recognition." Neural networks 1.2 (1988): 119-130.

Towards CNNs

Convolutional Neural Networks (CNNs, LeCun 1989) are a category of multi-layer NNs with learnable weights and biases, designed such that they tackle common problems of ANNs.

- **Observation**: Inputs are structured, e.g. images

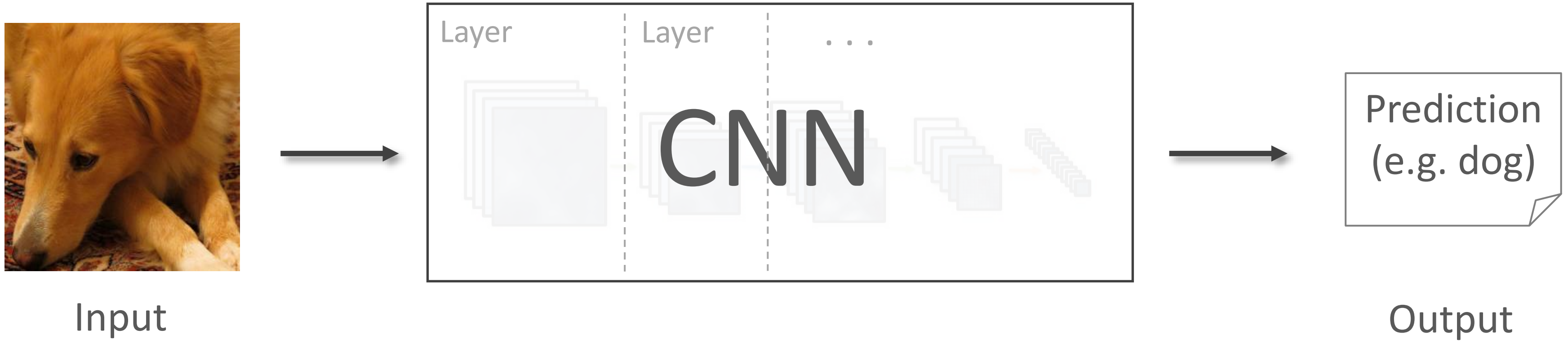
- **Key idea**: Invariance to shifts, scale and small distortions using
  - local weighted connections, i.e. local receptive field
  - shared weights across spatial locations
  - spatial sub-sampling

- **Main operations**:
  - Convolutions
  - Non-linearities
  - $\max(\cdot)$ functions

CNN Architecture

Network architecture generally composed by:

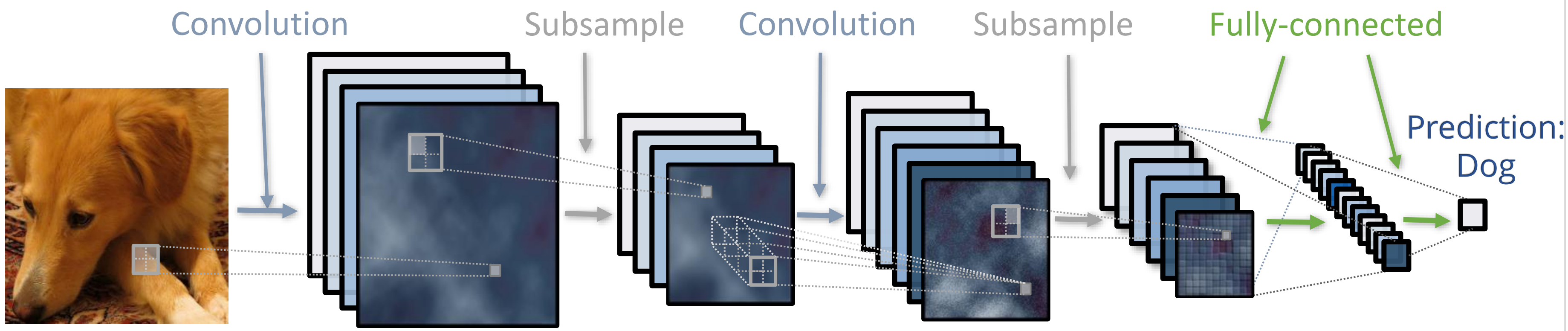
- “Filters” arranged in three dimensions: width, height and depth.
- Alternating convolutional (followed by a non-linear activation function) and sub-sampling layers to produce features at different levels of abstraction.
- Fully-connected layers that act as the final classifiers.



CNN Architecture

- CNN architectures are generally composed of:

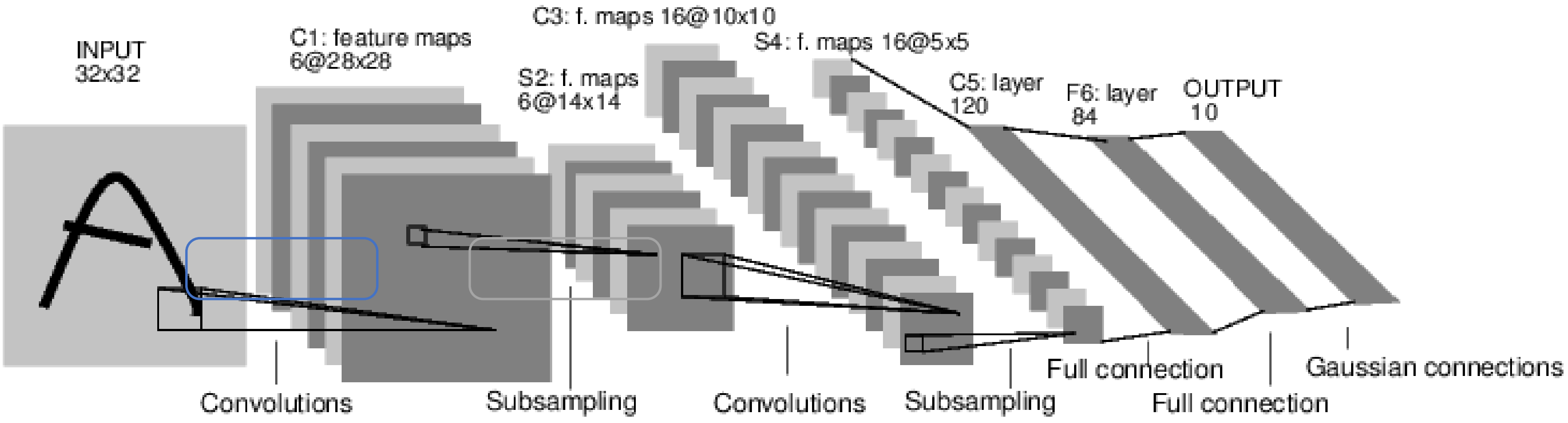
- "Filters" arranged in three dimensions: width, height and depth.
- Alternating convolutional (followed by a non-linear activation function) and sub-sampling layers to produce features at different levels of abstraction.
- Fully-connected layers that act as the final classifiers.



Sample CNN structure

LeNet-5

- First successful modern CNN architecture
- Introduced in 1998 for handwritten digit recognition
- Trained with back-propagation and gradient descent



LeCun et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

TITLE	CITED BY	YEAR
Distinctive image features from scale-invariant keypoints DG Lowe International journal of computer vision 60 (2), 91-110	77634	2004
Object recognition from local scale-invariant features DG Lowe International Conference on Computer Vision, 1999, 1150-1157	26433	1999
Gradient-based learning applied to document recognition Y LeCun, L Bottou, Y Bengio, P Haffner Proceedings of the IEEE 86 (11), 2278-2324	72812	1998

Convolutional Layer

Main operation of CNNs

- Local Connectivity

  Each neuron of a layer connects only to a local region of the previous layer (receptive field) and the dot product is performed between this region and the learnable weights.

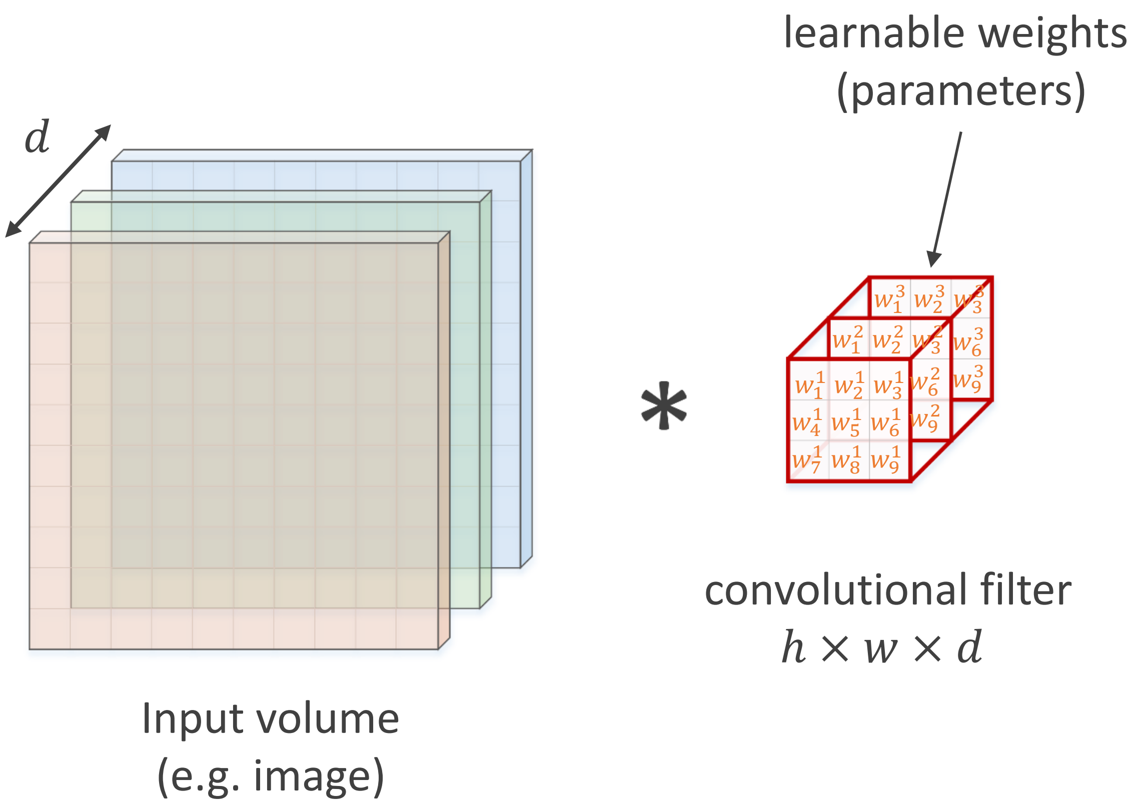
- Weight Sharing

  The same weights are used for every spatial location in the input *volume*.

$$f(x,y) * g(x,y) = \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} f(n,m) \cdot g(x-n, y-m)$$

- Before: apply the same filter to every channel.
- Now: filters will have separate weights for every channel.

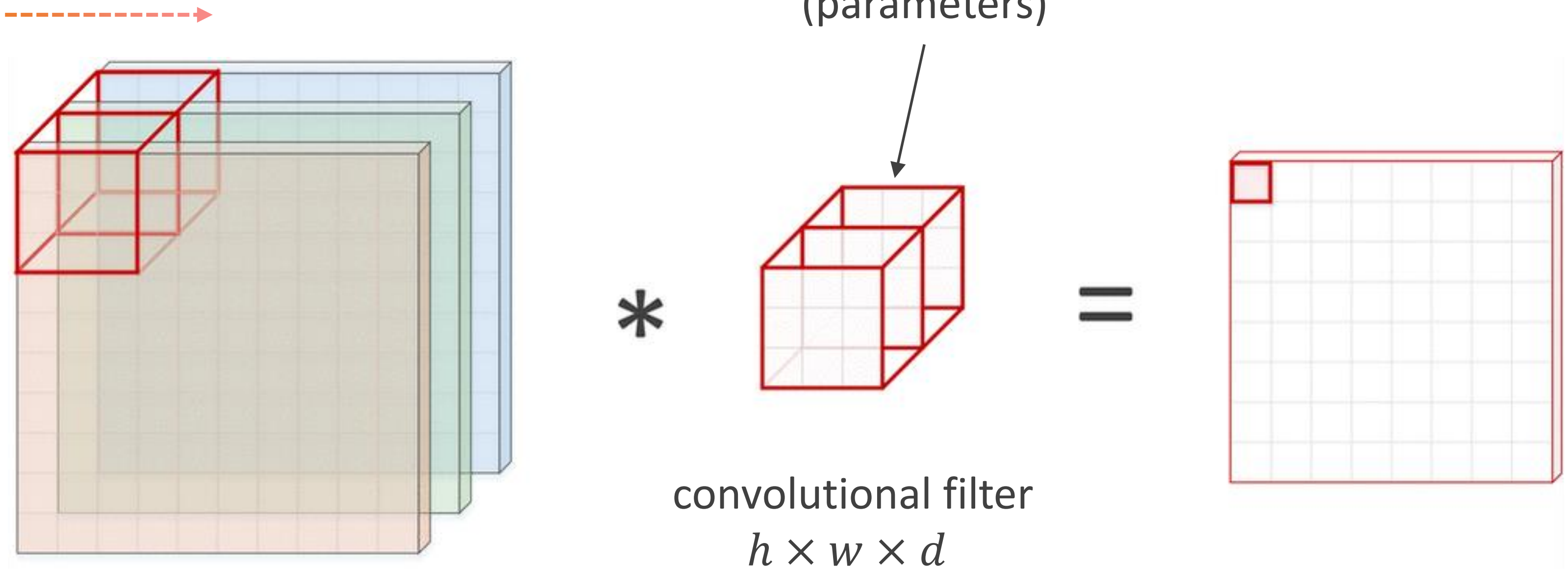
Convolutional Layer



Convolutional Layer

slide filter
over input

learnable weights
(parameters)



convolutional filter
$h \times w \times d$

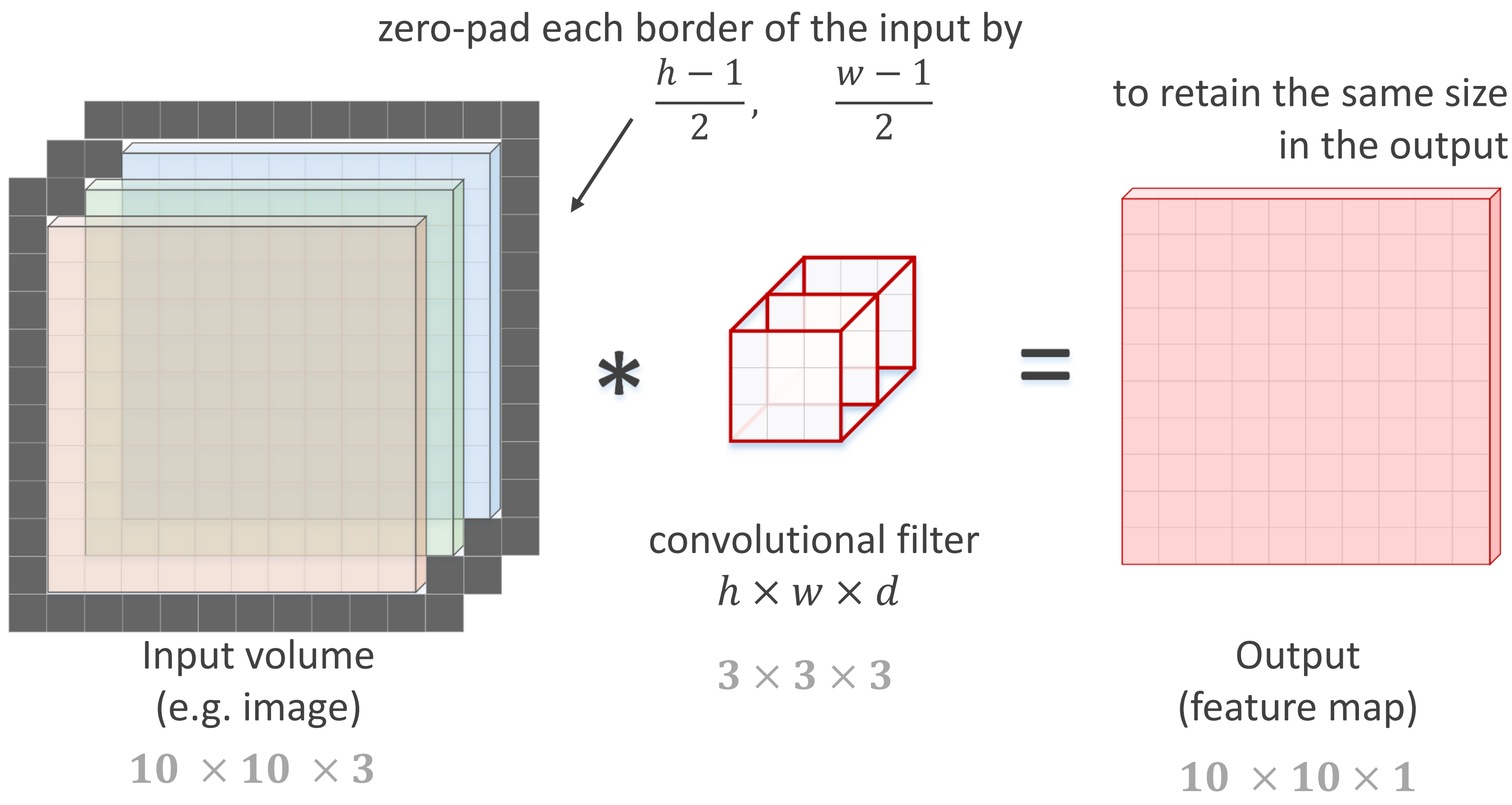
$\mathbf{3 \times 3 \times 3}$

Input volume
(e.g. image)
$\mathbf{10 \times 10 \times 3}$

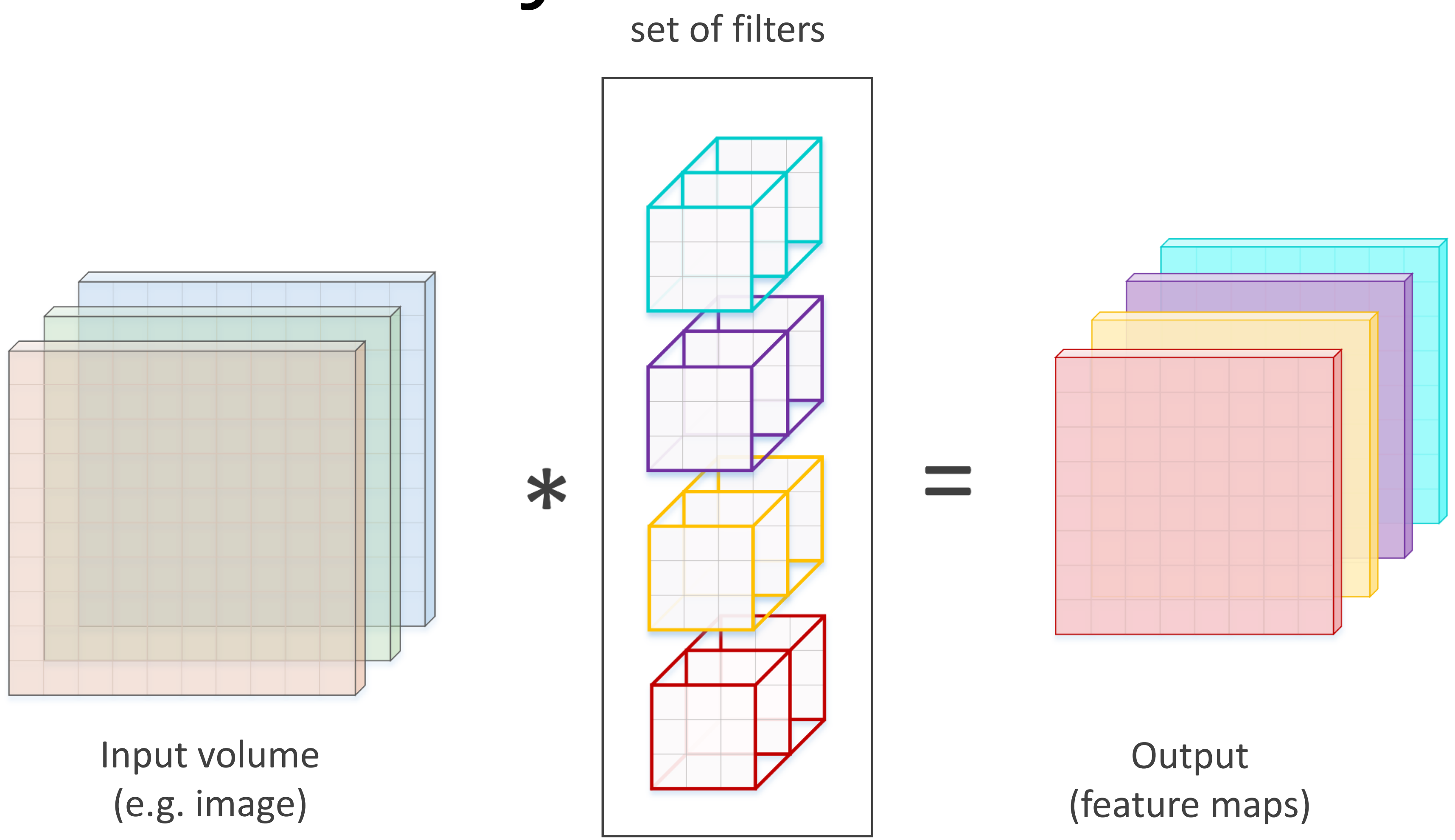
Output
(feature map)
$\mathbf{8 \times 8 \times 1}$

The filter slides spatially but operates (dot product) on all dimensions

Convolutional Layer



Convolutional Layer



learning multiple (different) filters → produce several feature maps → multitude of features

Convolutional Layer

Learnable parameters
i.e. the weights of the filters
biases added afterwards

Each filter learns to activate on some sort of *feature*

Other hyperparameters

- spatial extend: width w, height h
  number of channels: depth d
- number of filters f
- Stride s (step size)
  stride > 1 results in spatial sub-sampling of the feature maps
- Padding p on the input
  (on every side)

Size of resulting feature maps:

$$h_{out} = \frac{h_{in} - h + 2p}{s} + 1$$

$$w_{out} = \frac{w_{in} - w + 2p}{s} + 1$$

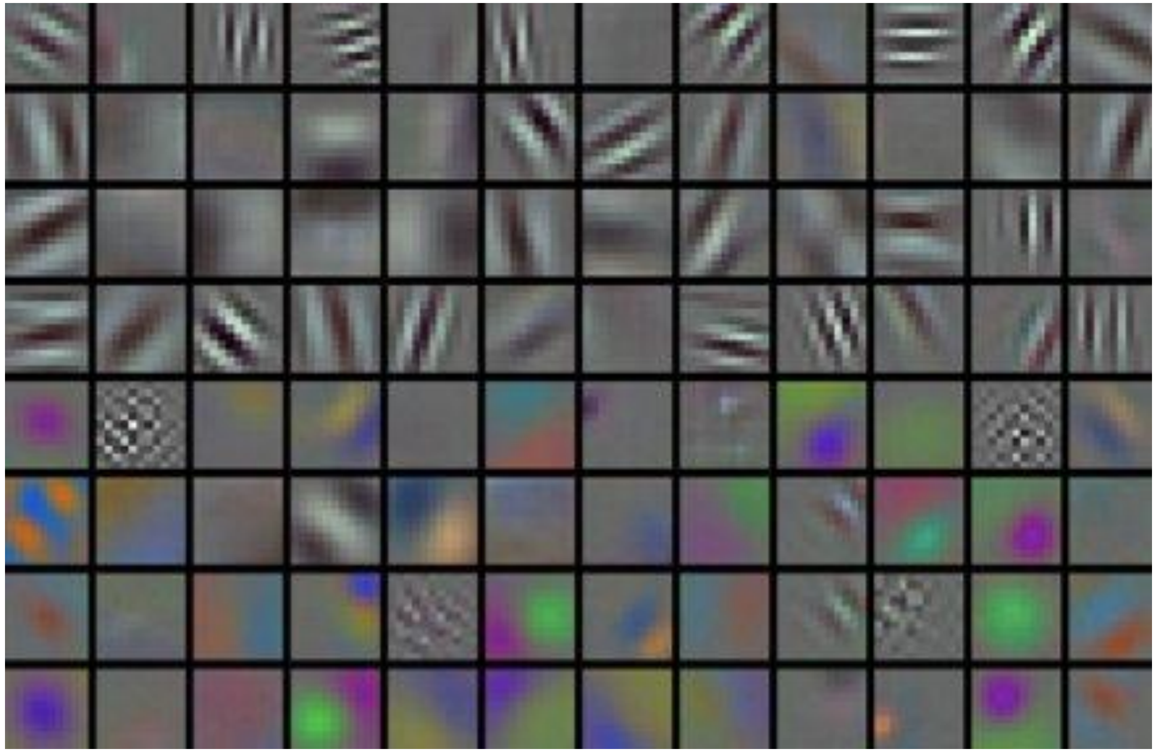
where $h_{in}, w_{in}, h_{out}, w_{out}$ are the height and width of input and expected output respectively

Practical Considerations

- Common filter sizes are 3×3, 5×5, etc.
  1×1 is also possible because it operates in depth too.
- The third dimension d is almost always the same as the number of channels in the input (but not necessarily).
- Padding does not need to be symmetric (but usually is).
- Stacking convolutions extracts features with a progressively higher level of abstraction.

First Layer Filters

First-layer filters from **AlexNet** (visualization of 96 $[11 \times 11 \times 3]$ filters):



First-layer learned features include basic elements, such as edges, blobs, colors, etc.

Parameter sharing thus appears to be reasonable: detecting e.g. an edge is important at any position of the input image.

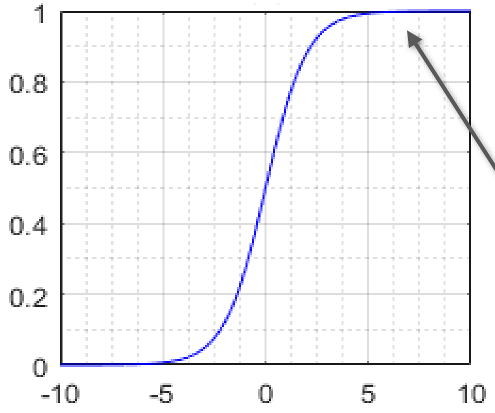
Krizhevsky, Sutskever, Hinton, "Imagenet classification with deep convolutional neural networks." NeurIPS 2012.

Activation Function

- Convolutions are linear operations
- Stacking them will still only give us a linear operation.
- Add a non-linear activation function in between.

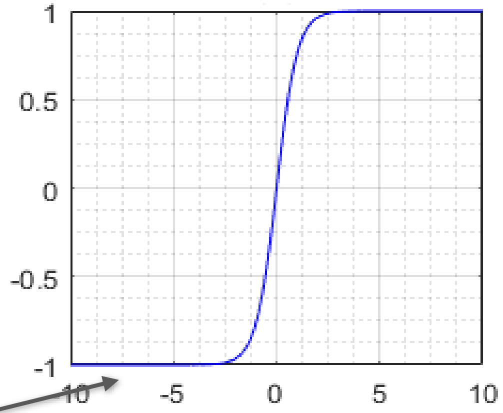
Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$

Output range: [0,1]



Tanh: $\tanh(x) = 2\sigma(2x) - 1$

Output range: [-1,1]



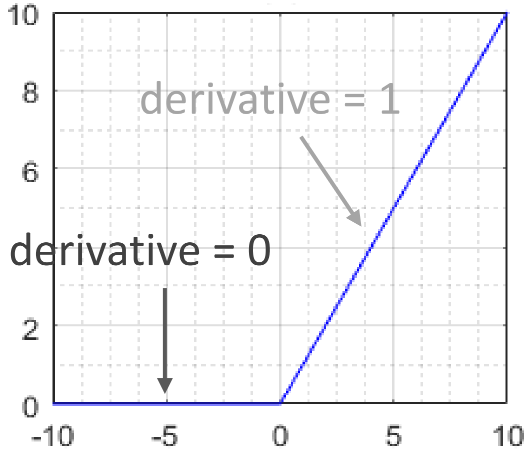
These functions saturate, making gradients very small → learning is very difficult.

Rectified Linear Unit (ReLU)

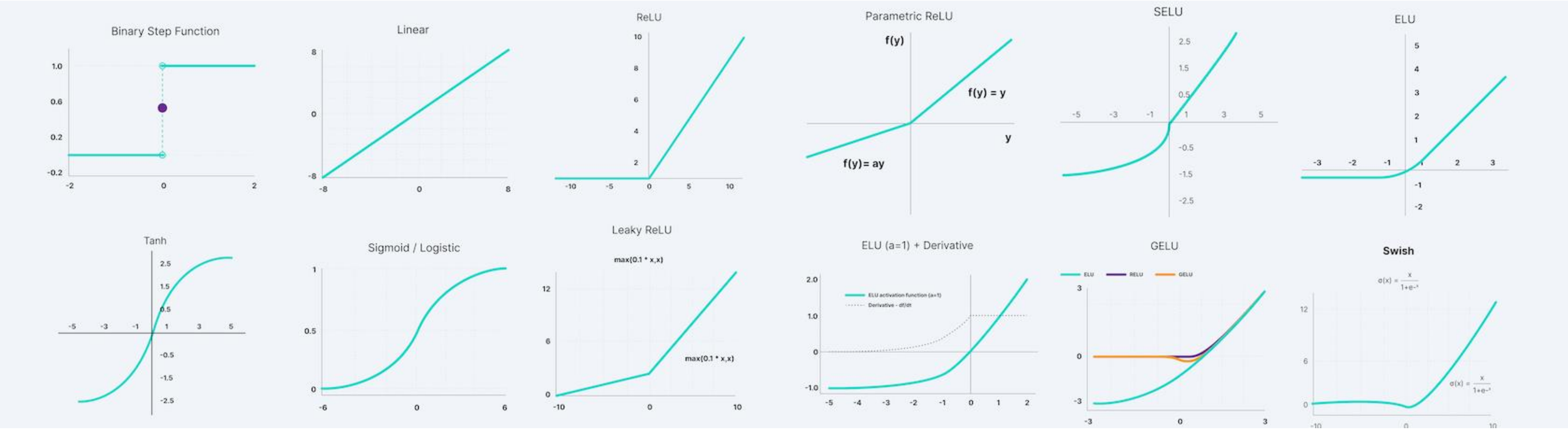
- Simply thresholds at zero
- Sparse activation
- Computationally efficient
- Non-saturating → speeds up convergence

Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$



Other Activation Functions



[image source]()

Pooling Layers

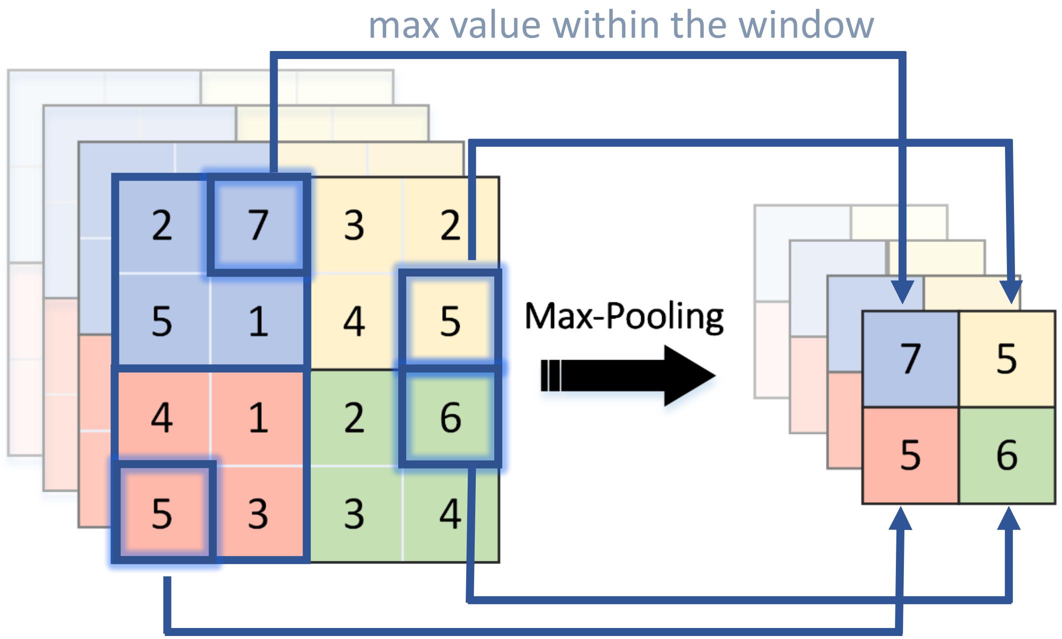
- Idea: reduce the resolution to understand content at different scales.

- Idea: reduce the resolution to save computations.

- Performs an element-wise operation on the feature maps in a local region, on each channel independently.

- Usually: $\mathbf{max}(\cdot)$ or $\mathbf{avg}(\cdot)$.

Max-Pooling Example



Example: Max-pooling

The window size is 2×2, applied with a stride of 2 (common case)

Pooling Layer

- Parameter-free layer
- Used for feature map spatial sub-sampling (with stride > 1).
- Controls the capacity of the network by reducing the resolution.
- Introduces some invariance to small transformations of the input, because precise spatial information is lost.

- Hyperparameters:
  - width w and height h of window
  - stride s
    overlapping of sliding window occurs if $s < w \ or \ s < h$

Size of resulting pooled maps:

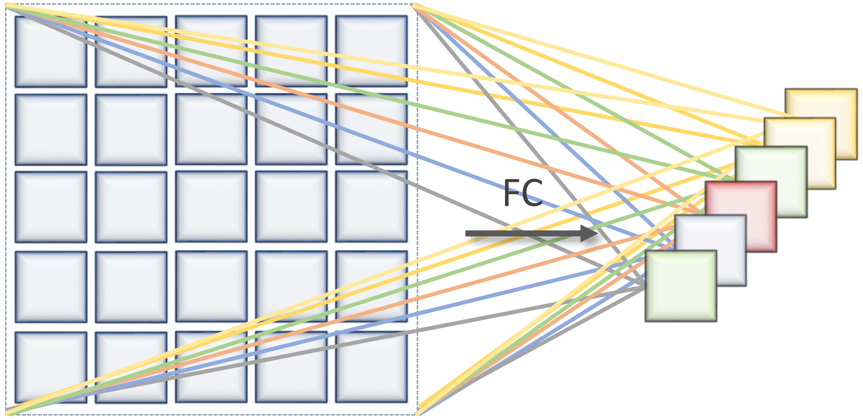
$$h_{out} = \frac{h_{in} - h}{s} + 1$$

$$w_{out} = \frac{w_{in} - w}{s} + 1$$

Fully Connected Layer

- Fully-connected layers follow the principle of the typical ANN weighted connections: each neuron in the output connects to all neurons of the input.
- Usually added as the last layers of the network / output layer.
- Implemented as a linear function, plus bias, followed by a non-linearity.
- Guarantee a full receptive field.

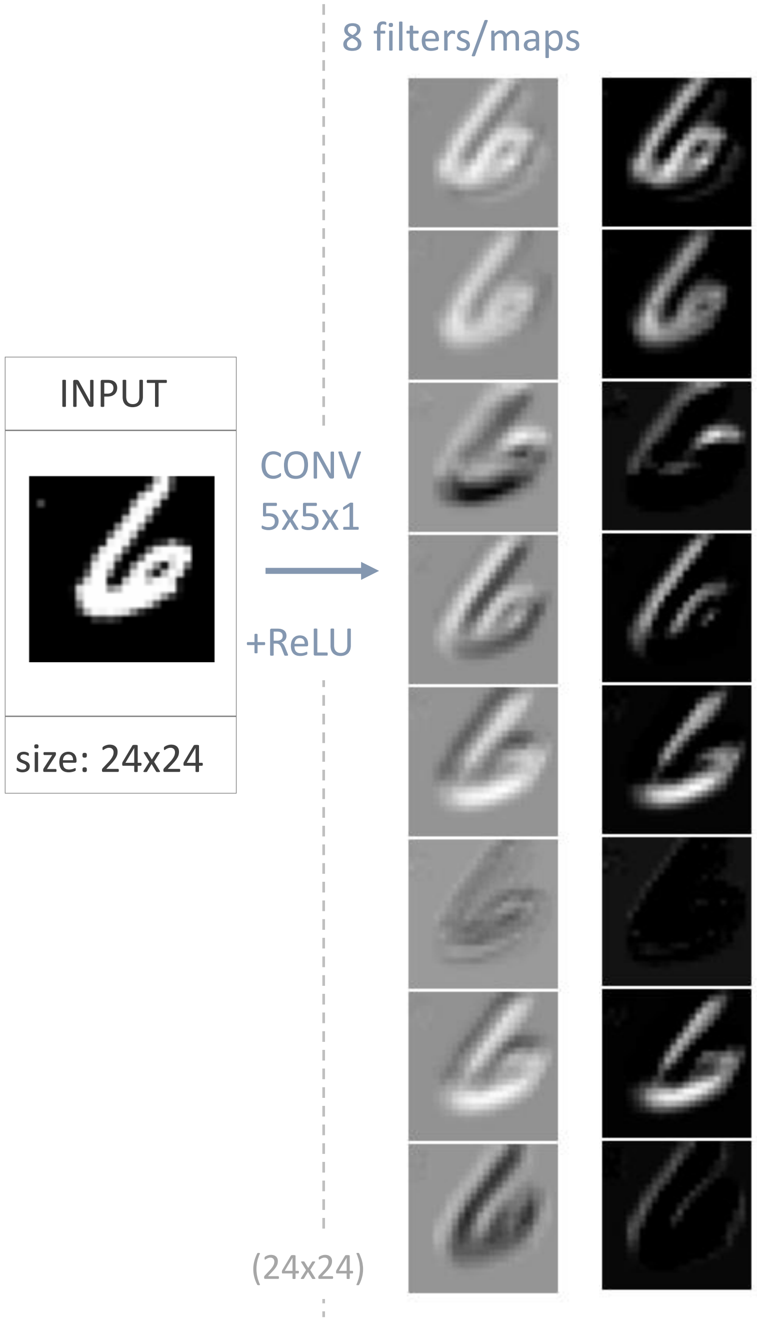
Fully Connected Layer



Input $h \times w \times d$
(here: $5 \times 5 \times 1$)

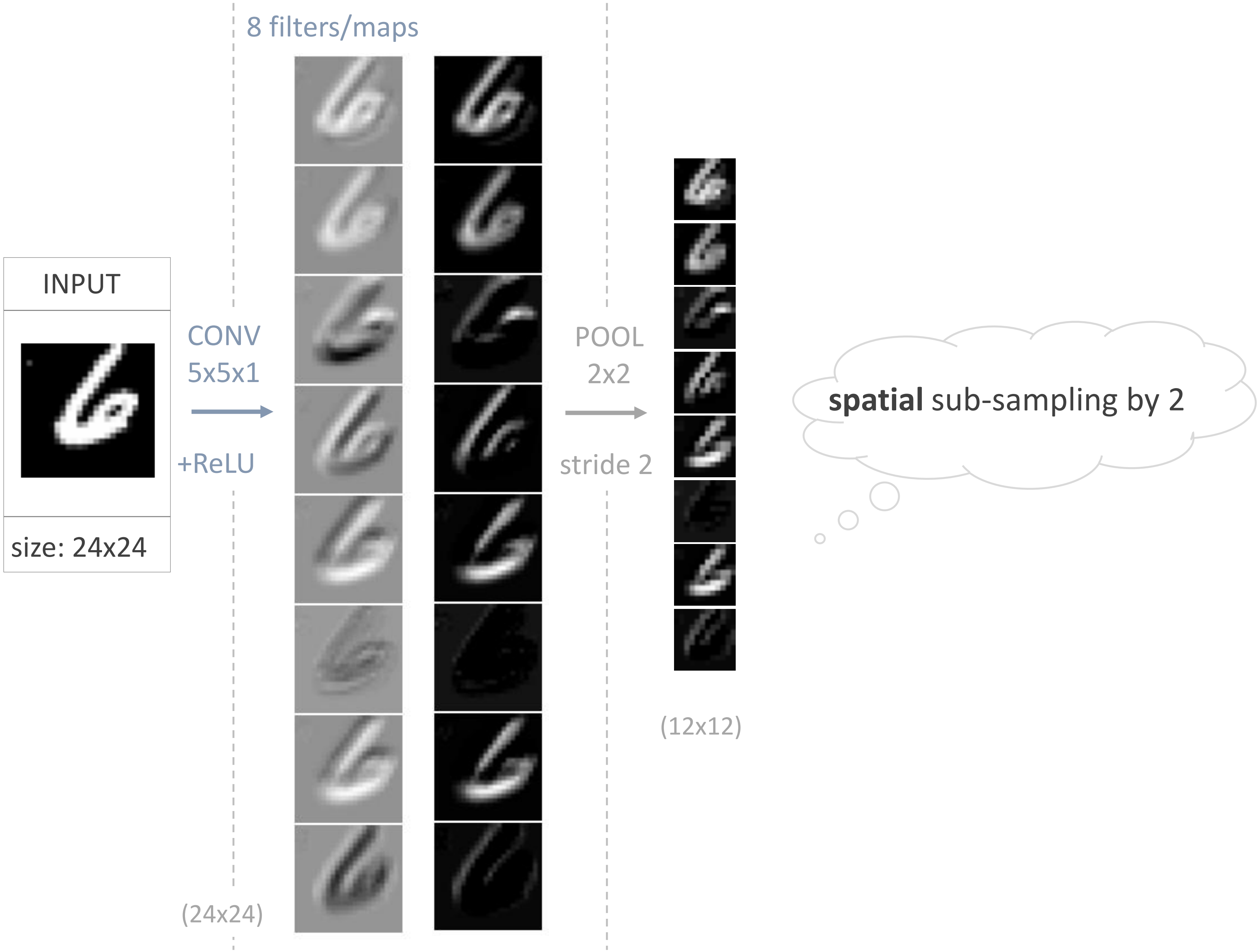
Output $1 \times 1 \times n$
n being the only hyperparameter
(here: $n = 6$)

Can be also seen as a convolutional layer with a set of filters of the **same size** as the input volume, i.e. n filters of size $h \times w \times d$

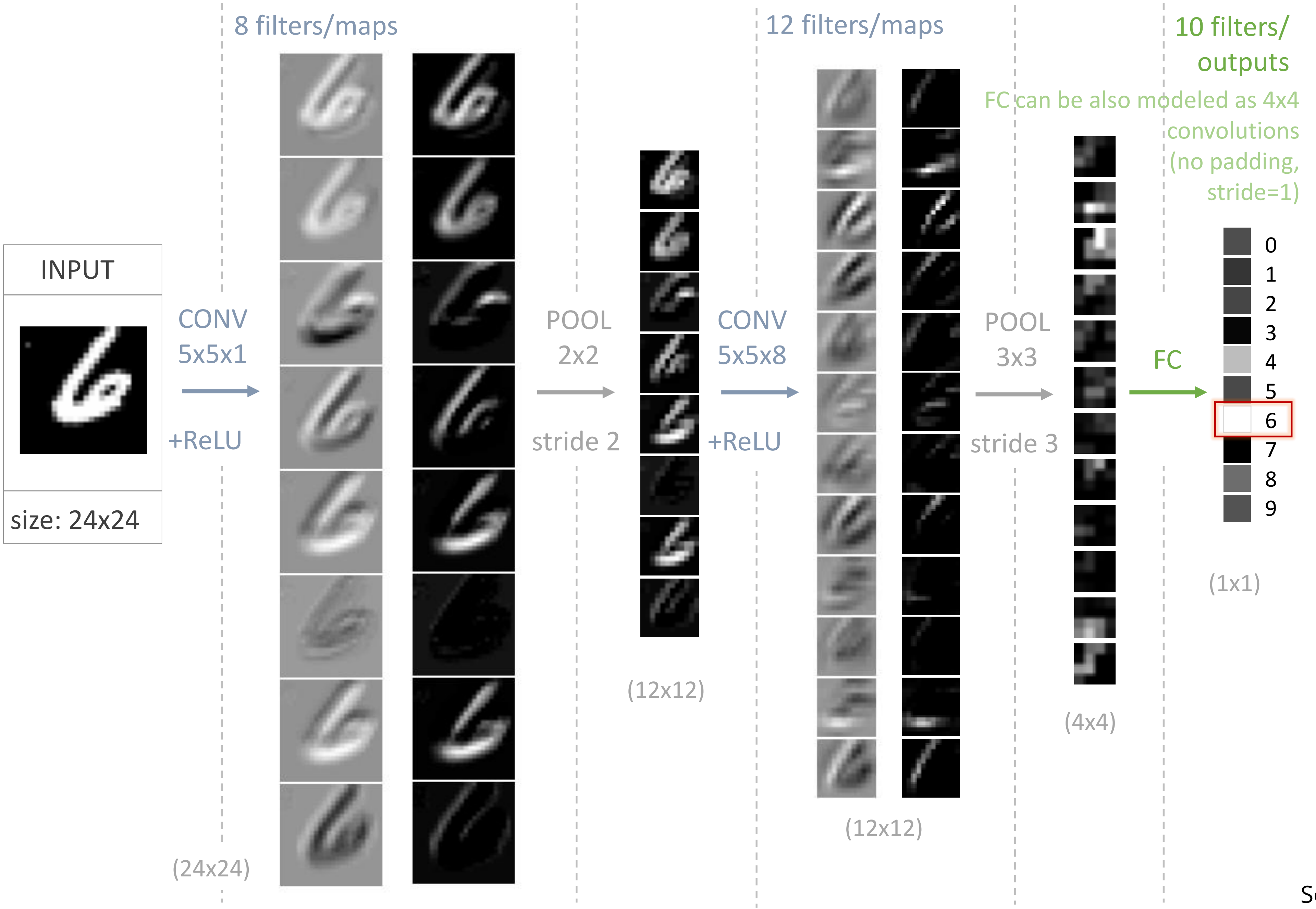


Notice how feature maps activate on different basic structures, depending on the corresponding filter

Source [A. Karpathy]



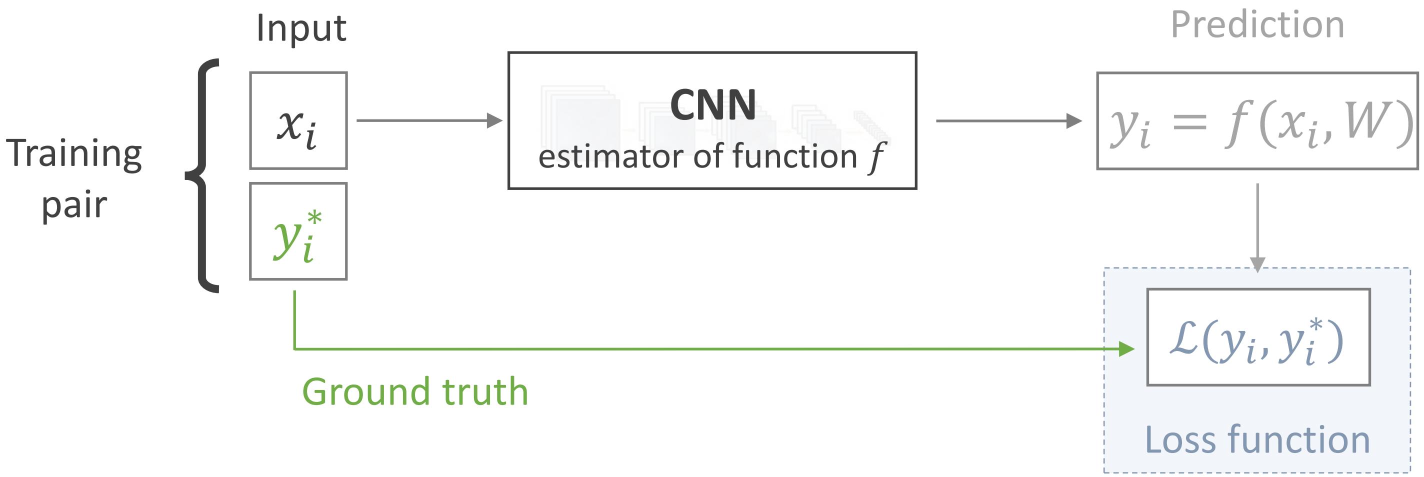
Source [A. Karpathy]



Source [A. Karpathy]

Loss function

- A CNN can be learned for different problems (classification or regression ones) by minimizing a specified objective, i.e. the *loss function*.
- It simply measures how well the CNN performs on the task.
- To do this, a "loss layer" receives the output of the CNN (prediction) and compares it to the ground truth of the given input.
- Stochastic Gradient Descent: the loss over the entire dataset must be written as the mean of the individual losses of the samples.



Loss function

- A CNN can be learned for different problems (classification or regression ones) by minimizing a specified objective, i.e. loss function.
- It simply measures how well the CNN performs on the task.
- To do this, a “loss layer” receives the output of the CNN (prediction) and compares it to the ground truth of the given input.
- The loss over the entire dataset is (most often) the mean of the individual losses of the samples.

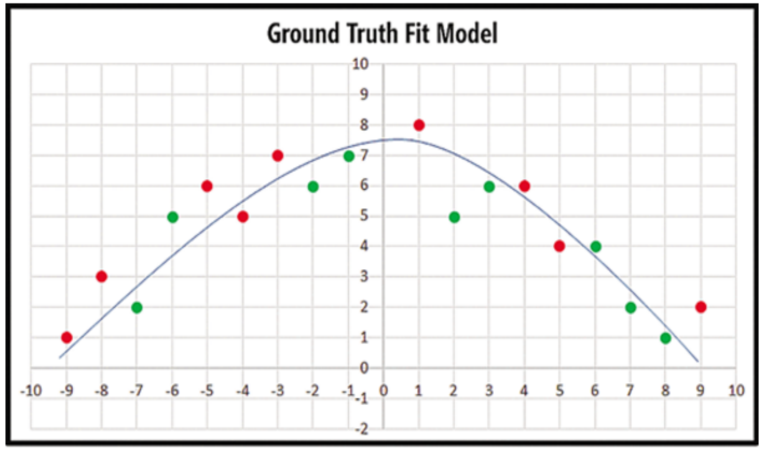
- Example:
  If our task is image classification,
  - the ground truth is the labeled category for the image
  - the prediction is a vector of scores, which represent the “probabilities” that the input belongs to each of the existing categories.

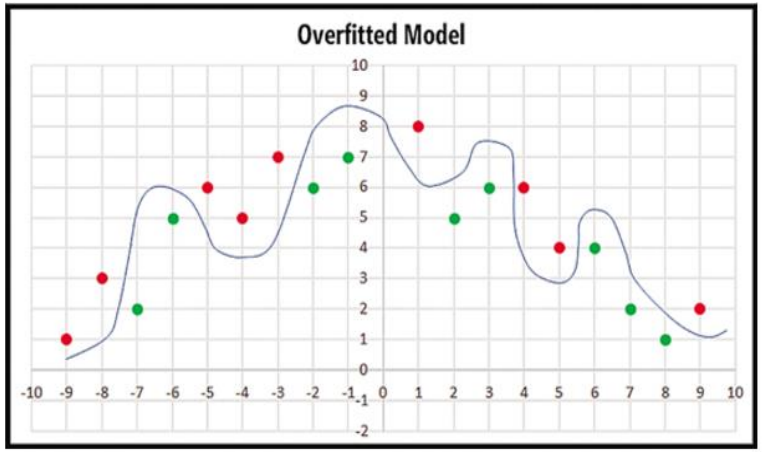
Loss function

- Classification: soft-max cross-entropy (Lecture 06)
- Regression:
  - for tasks where the output is continuous.
  - $\mathcal{L}_1(\boldsymbol{y}, \boldsymbol{y}^*) = \ |\boldsymbol{y} - \boldsymbol{y}^*|_1 = \frac{1}{n}\sum_{i=1}^{n}|y_i - y_i^*|$
  - $\mathcal{L}_2(\boldsymbol{y}, \boldsymbol{y}^*) = \|\boldsymbol{y} - \boldsymbol{y}^*\|_2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^*)^2$
  - Note that $\boldsymbol{y}, \boldsymbol{y}^*$ can have arbitrary dimensions depending on the task, e.g. vectors of regressed points or entire prediction maps.
    n would then be the number of points or pixels respectively.
- Task-specific loss functions that model some known properties of the problem.

Regularization

Helps generalization to unseen data, i.e. preventing over-fitting to the training samples.





In an over-fitted model, the predicted curve is not “regular”
Weights have very large or very small values

https://msdn.microsoft.com/en-us/magazine/dn904675.aspx

Regularization

- It includes methods for better generalization to unseen data, i.e. preventing over-fitting to the training samples.

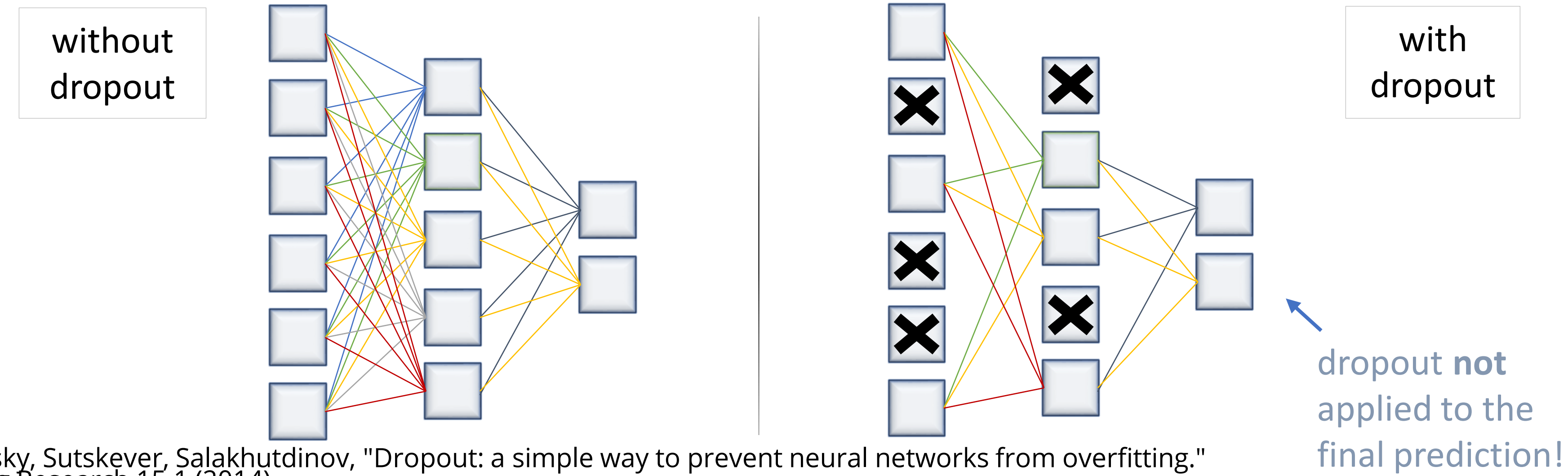
If a is too large, the networks tries to keep weights too small

- L2 regularization
  - Penalty term: $\boxed{R = \frac{1}{2} a w^2}$ (the squared magnitude of all parameters) a is the regularization strength (typically small, e.g. order of 10^{-4})
  - Favors weight "diffusion"
  - Weight update through gradient descent: $w_{t+1} = w_t - a w_{\mathrm{t}}$ (linear decay)

- L1 regularization
  - Penalty term: $R = a|w|$
  - Causes weight vector to become sparse and invariant to noisy inputs

Dropout

- Randomly "dropping out" neurons of a layer
  (with probability p, usually 0.5) at each iteration of training
- This effectively means making them inactive (setting to zero) so that they do not contribute in forward/backward passes
- Neurons do not learn to rely on the presence of other specific neurons
- Usually applied before the last fully-connected layer(s)



Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research 15.1 (2014)

Optimization methods

For a training iteration t and the current state of parameters denoted as w_t, an update is performed as:

$$w_{t+1} = w_t + \Delta w_{\mathrm{t}}$$

A variety of first-order solvers, popular for training CNNs:

- Stochastic Gradient Descent (SGD)
  Follow the negative gradient for a "mini-batch" of samples $\Delta w_t = -\lambda g_t$
  - Requires manual setting of learning
  - Manual annealing: decrease learning rate, if validation curve "plateaus" to prevent parameters from oscillating near local minima
- SGD with momentum
  Keep in memory previous weight updates $\Delta w_t = \rho \Delta w_{t-1} - \lambda g_t$
  - → Accelerates SGD progress when gradient points in the same direction as before and dampens oscillations

Rumelhar, Hinton, Williams, "Learning representations by back-propagating errors." Nature 323 (1986): 533-536.

Adam (Adaptive Moment Estimation)

Additionally keep an exponentially decaying average of previous gradients:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \rightarrow \widehat{m_t} = m_t / (1-\beta_1^t)$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \rightarrow \widehat{v_t} = v_t / (1-\beta_2^t)$$

1st moment (mean)
2nd moment (variance)

bias correction

$$\Delta w_t = -\frac{\lambda}{\sqrt{\widehat{v_t} + \varepsilon}} m_t$$

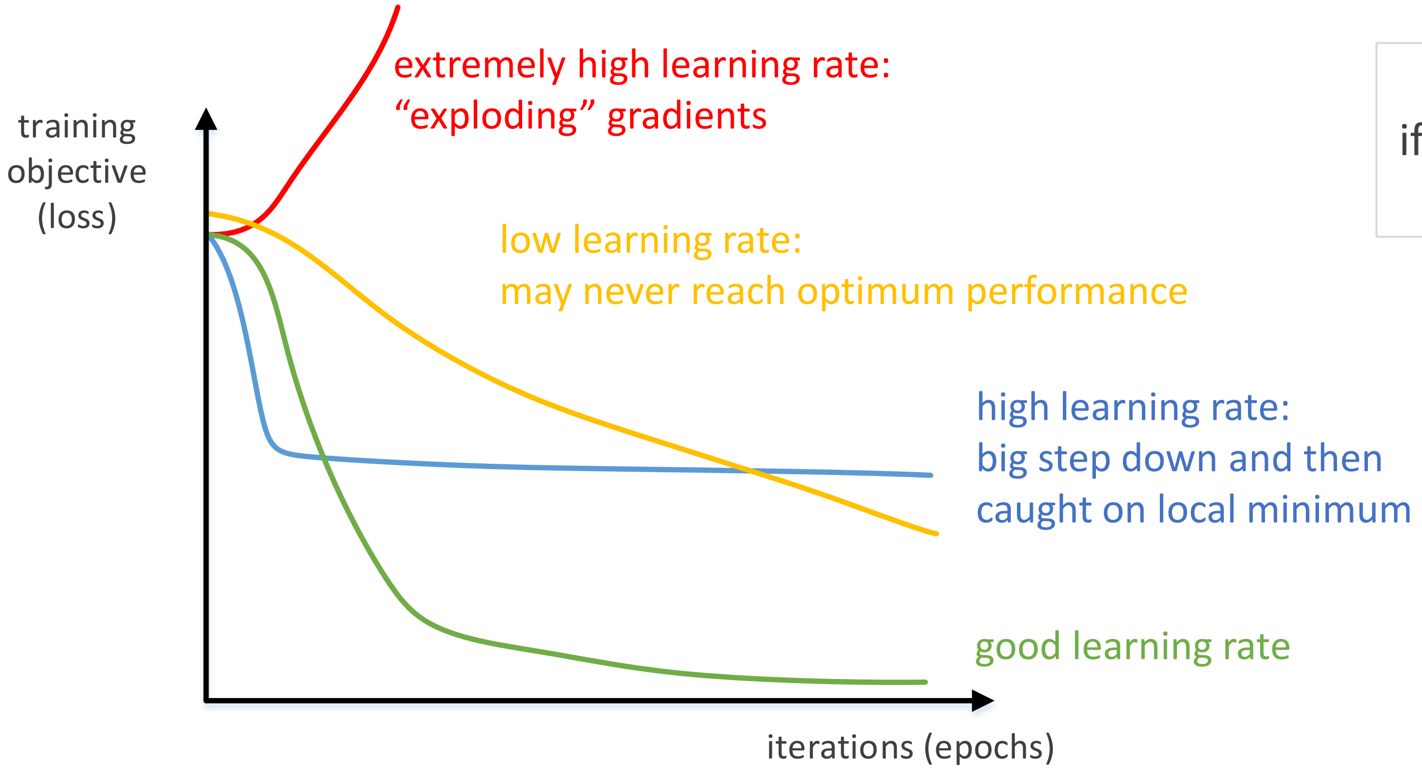
Suggested decay: $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Initial averages: zeros

Kingma and Ba. "Adam: A method for stochastic optimization." ICLR'15.

How to train your network

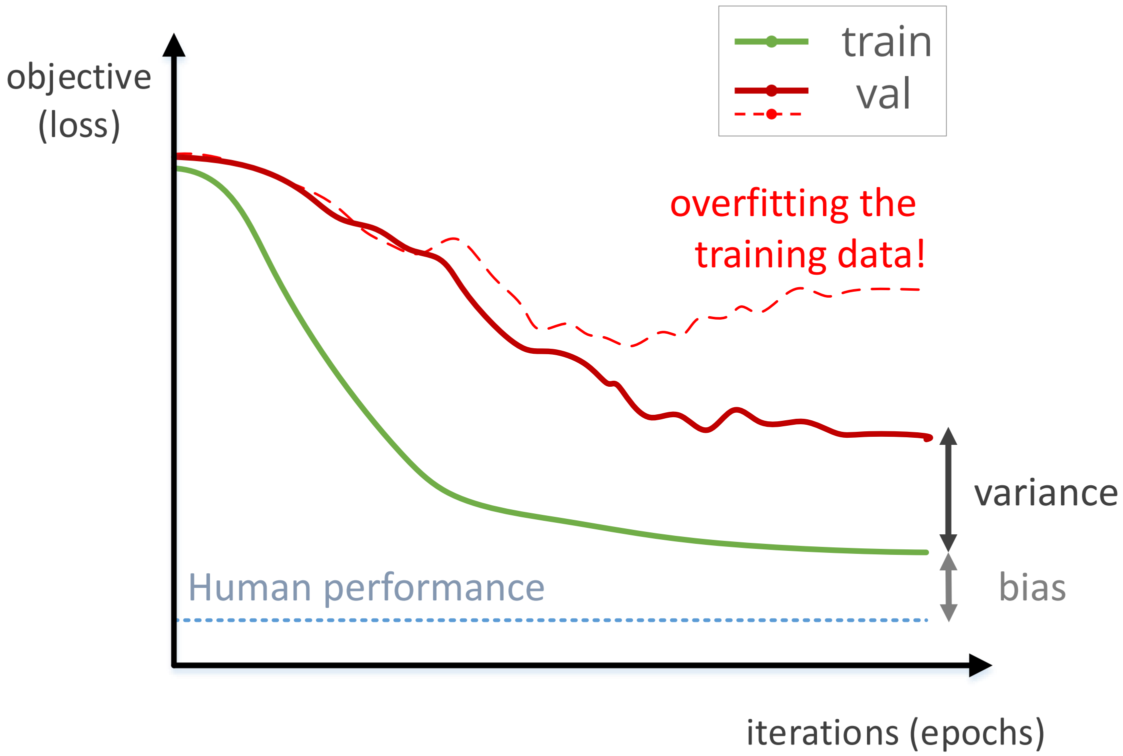
Looking for the right learning rate…



As for the validation curve, if it “**plateaus**” then decrease the learning rate

How to train your network

Validation vs Training (when data comes from the same distribution)



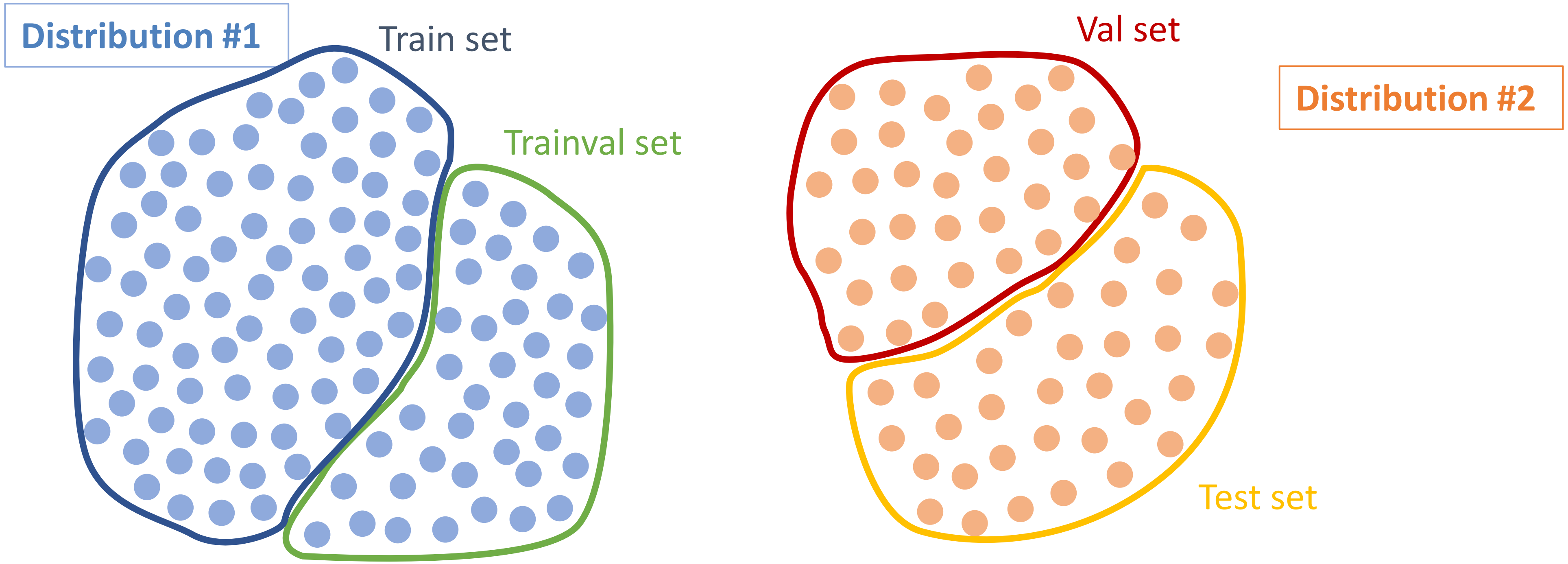
If **bias** is high:
Train a bigger model
or train longer

If **variance** is high:
Try more data,
augmentations,
regularization (e.g. dropout)

If **overfitting**:
Try more data or
early stopping

How to train your network

When data comes from different distributions…

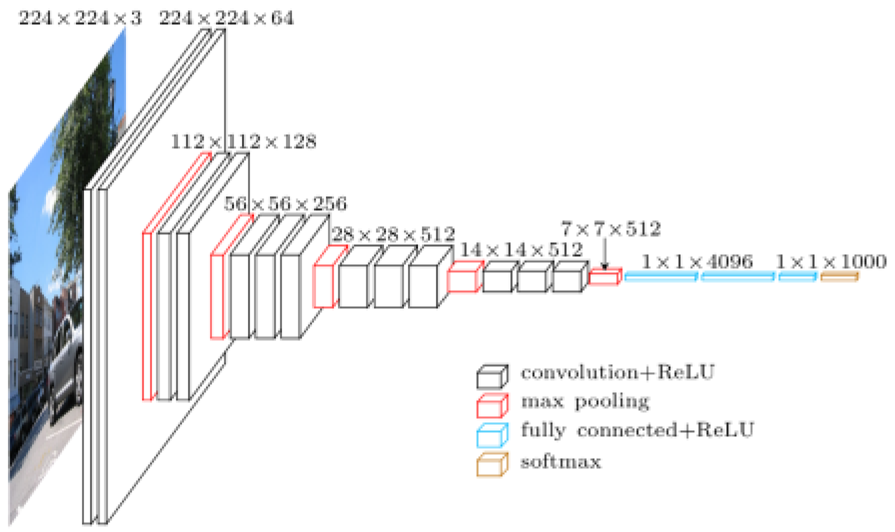


- If val error high: Get more data of distribution #2 (similar to the test scenario)
- If val error low, but test error high: Get more validation data

Architecture Example: VGG

Very Deep Convolutional Networks for Large-Scale Image Recognition, Simonyan and Zisserman, 2014

- Different configurations: 11-19 layers
- 3x3 convolutions
- 3 large FC layers in the end



Architecture Example: ResNet

Deep Residual Learning for Image Recognition, K He et al., 2015

- Introduces residual connections (next lecture)
- Again: different sizes
  popular R18, R50, R101, R152
- Building blocks:
  - Convolution (mainly 3x3)
  - ReLU
  - Pooling
  - 1 FC layer in the end