CS4442/9542b
Artificial Intelligence II
prof. Olga Veksler

Lecture 12

Computer Vision

Object Recognition with CNN

Some slides are from S. Seitz, S. Narasimhan, K. Grauman

Outline

- Object Recognition with Deep Neural Nets
- Convolutional Neural Network

Traditional Object Classification

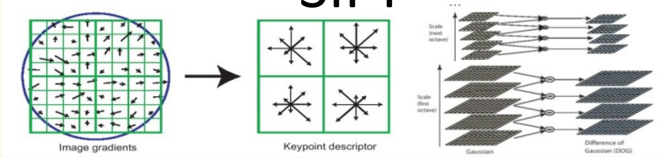
- Tradition Object Classification system

training data → extract hand-crafted features → train classifier

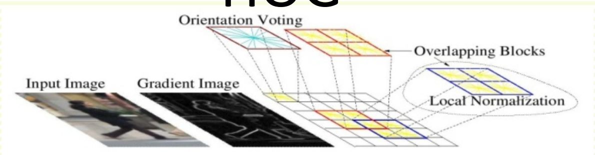
new data → extract hand-crafted features → apply classifier

- A lot of work to design good features by hand

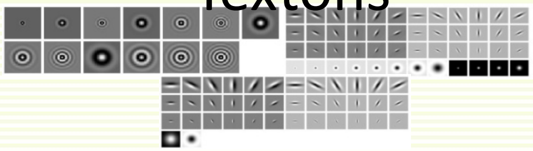
SIFT



HOG



Textons



Patches

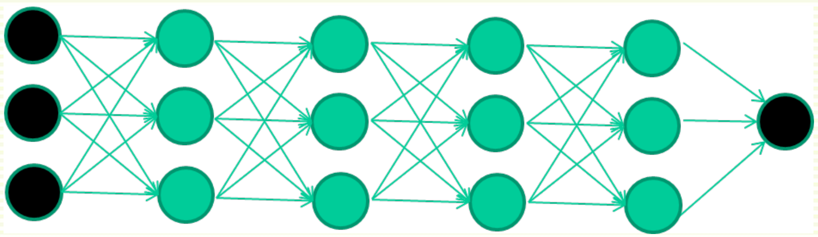
NN as Nonlinear Feature Mapping

- With NN, change in paradigm: instead of hand-crafting , learn features automatically from data

training data

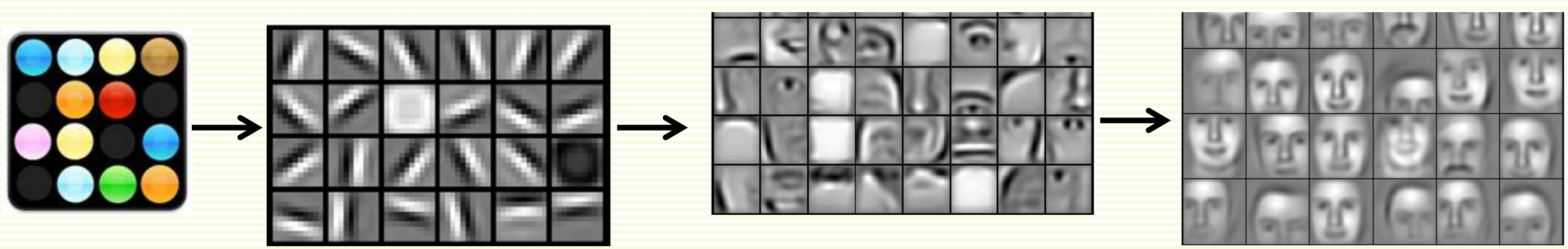
train DNN

raw image



Why Deep Networks: Hierarchical Feature Extraction

- Deep architecture works well for hierarchical feature extraction
  - hierarchies features are especially natural in vision
- Each stage is a trainable feature transform
- Level of abstraction increases up the hierarchy



Input layer
pixels

First layer
edges

Second layer
object parts

Third layer
objects

Early Work on Deep Networks

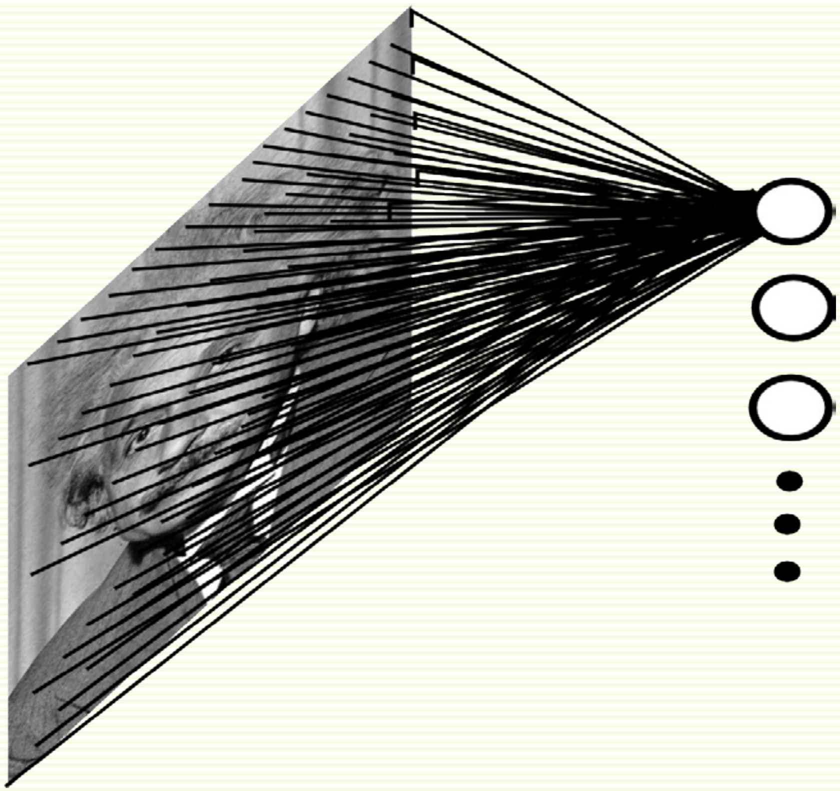
- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Networks (convnets)
  - Similarities to Neo-Cognitron
- Other attempts at deeply layered Networks trained with backpropagation
  - not much success
    - very slow
    - diffusion of gradient
  - recent work has shown significant training improvements with various tricks (drop-out, unsupervised learning of early layers, etc.)

ConvNets: Prior Knowledge for Network Architecture

- Convnets use prior knowledge about recognition task into network architecture design
  - connectivity structure
  - weight constraints
  - neuron activation functions
- This is less intrusive than hand-designing the features
  - but it still prejudices the network towards the particular way of solving the problem that we had in mind

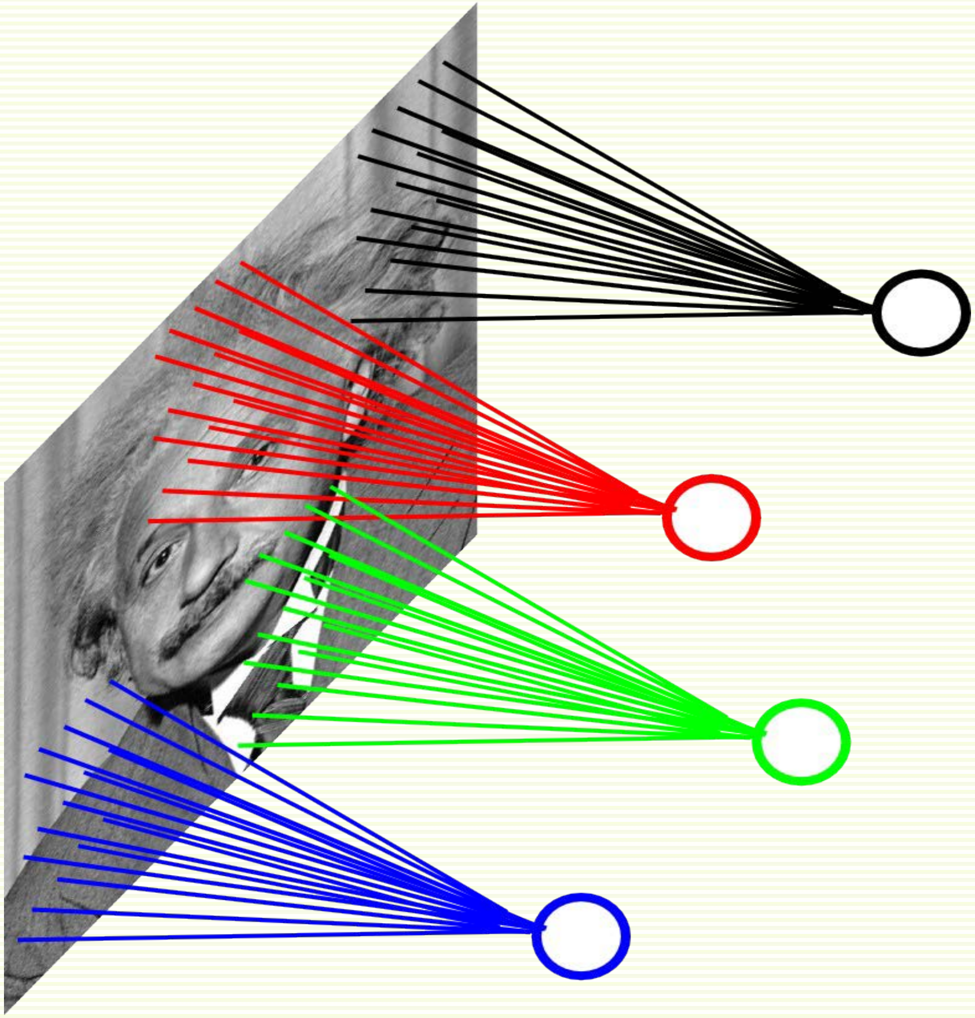
Convolutional Network: Motivation

- Consider a fully connected network
- Example: 200 by 200 image, $4\text{x}10^4$ connections to one hidden unit
- For 10^5 hidden units → $4\text{x}10^9$ connections
- But spatial correlations are mostly local
- Should not waste resources by connecting unrelated pixels



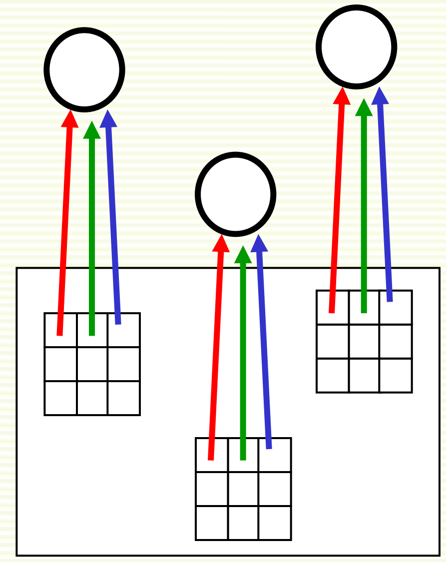
Convolutional Network: Motivation

- Connect only pixels in a local patch, say 10x10
- For 200 by 200 image, 10^2 connections to one hidden unit
- For 10^5 hidden units → 10^7 connections
- factor of 400 decrease



Convolutional Network: Motivation

- If a feature is useful in one image location, it should be useful in all other locations
  - *Stationarity*: statistics is similar at different locations
- All neurons detect the same feature at different positions in the input image
  - i.e. share parameters (network weights) across different locations
  - bias is usually not shared
  - also greatly reduces the number of tunable parameters



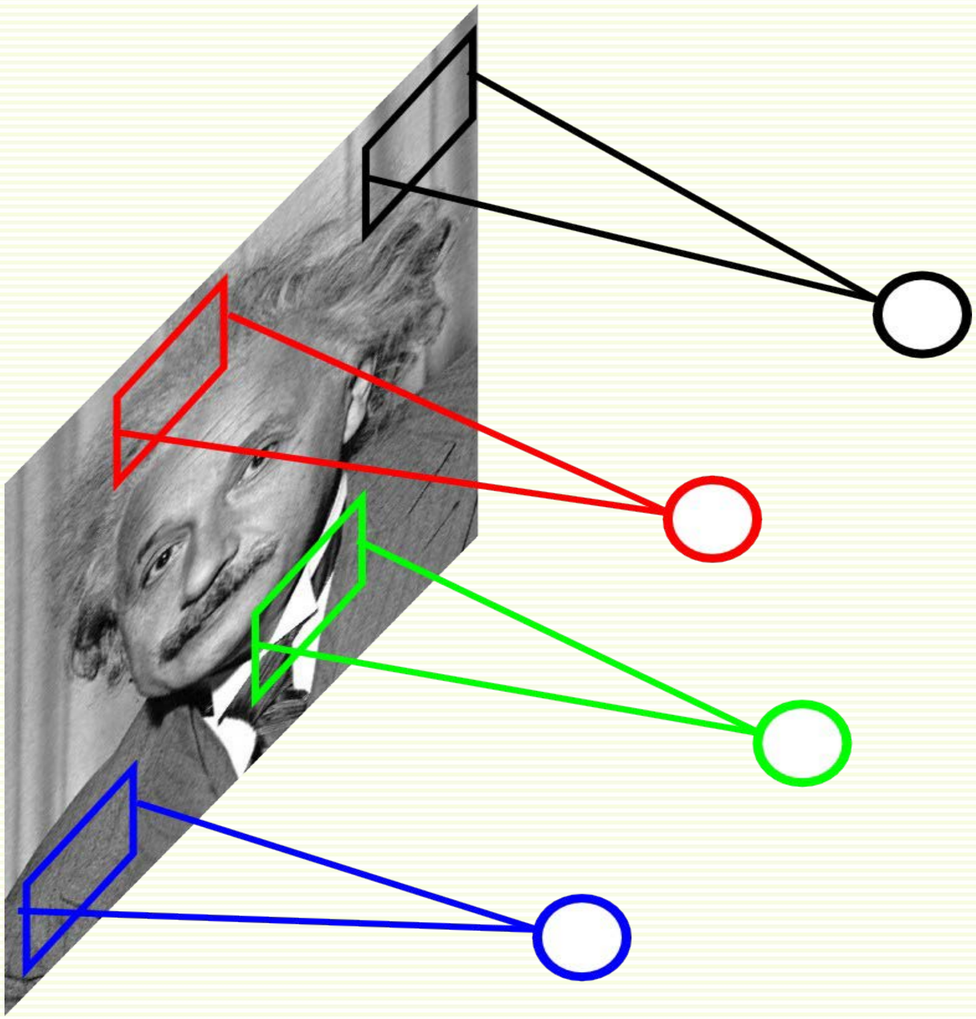
all red connections have the same weight

all green connections have the same weight

all blue connections have the same weight

ConvNets: Weight Sharing

- Much fewer parameters to learn
- For 10^5 hidden units and 10x10 patch
  - 10^7 parameters to learn without sharing
  - 10^2 parameters to learn with sharing

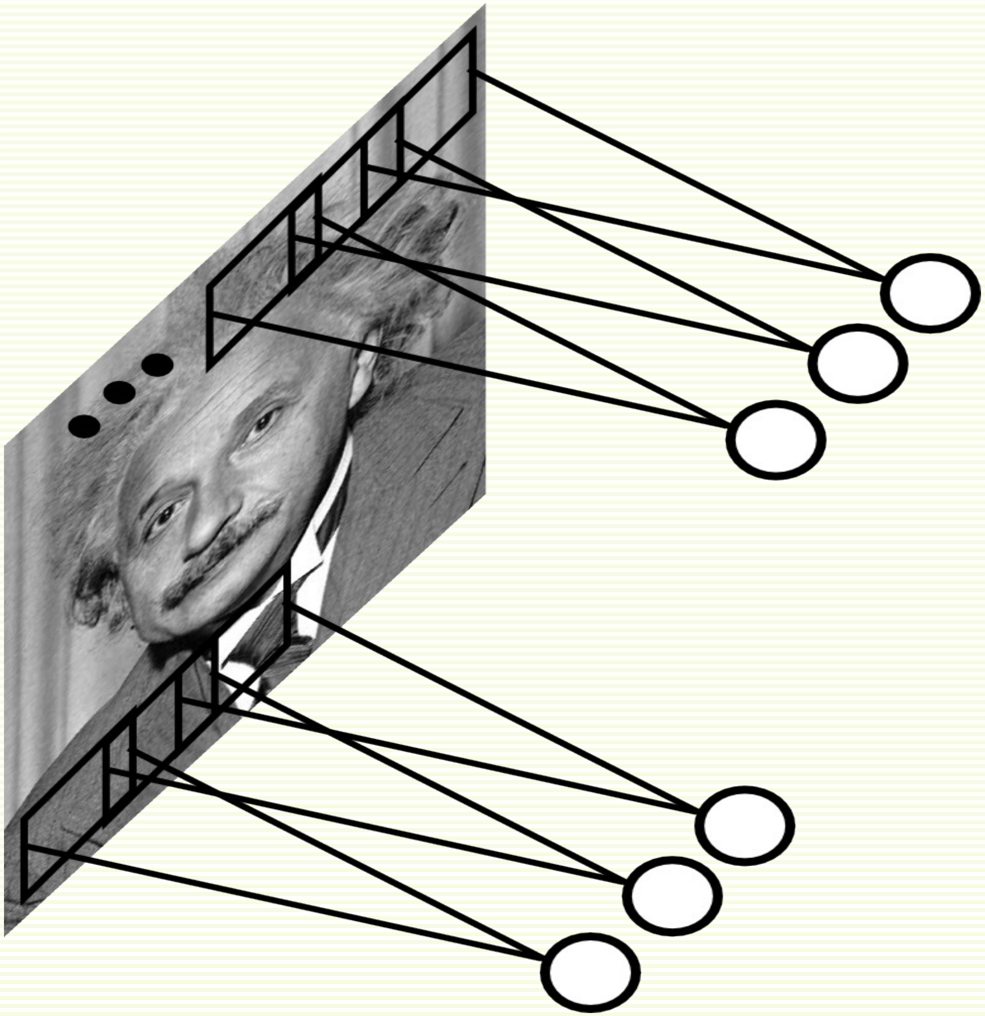


Weight Sharing Constraints

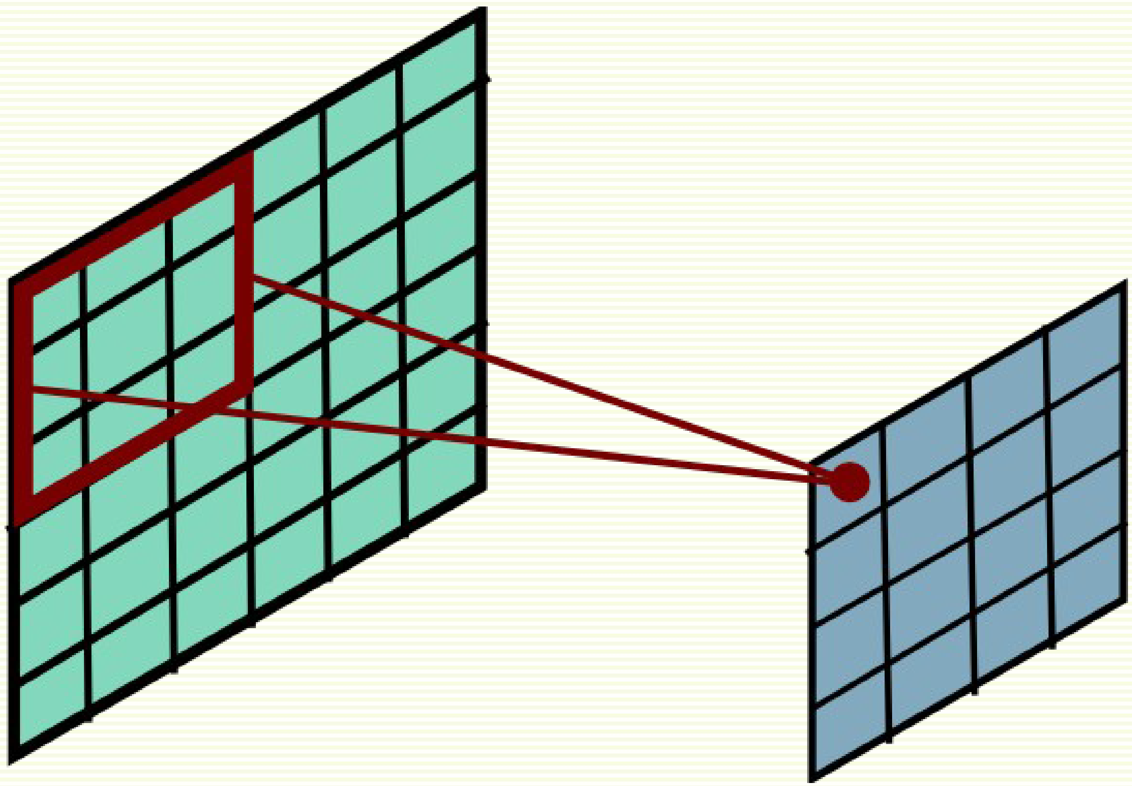
- Easy to modify backpropagation algorithm to incorporate weight sharing
- Compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.
  - if the weights started off satisfying the constraints, they will continue to satisfy them
- To constrain $\mathbf{w_1} = \mathbf{w_2}$, we need $\Delta\mathbf{w_1} = \Delta\mathbf{w_2}$
- Before we used $\dfrac{\partial \mathbf{L}}{\partial \mathbf{w}_1}$ to update $\mathbf{w_1}$ and $\dfrac{\partial \mathbf{L}}{\partial \mathbf{w}_2}$ to update $\mathbf{w_2}$
- Now use $\dfrac{\partial \mathbf{E}}{\partial \mathbf{w}_1} + \dfrac{\partial \mathbf{E}}{\partial \mathbf{w}_2}$ to update $\mathbf{w_1}$ and $\mathbf{w_2}$, use

Convolutional Layer

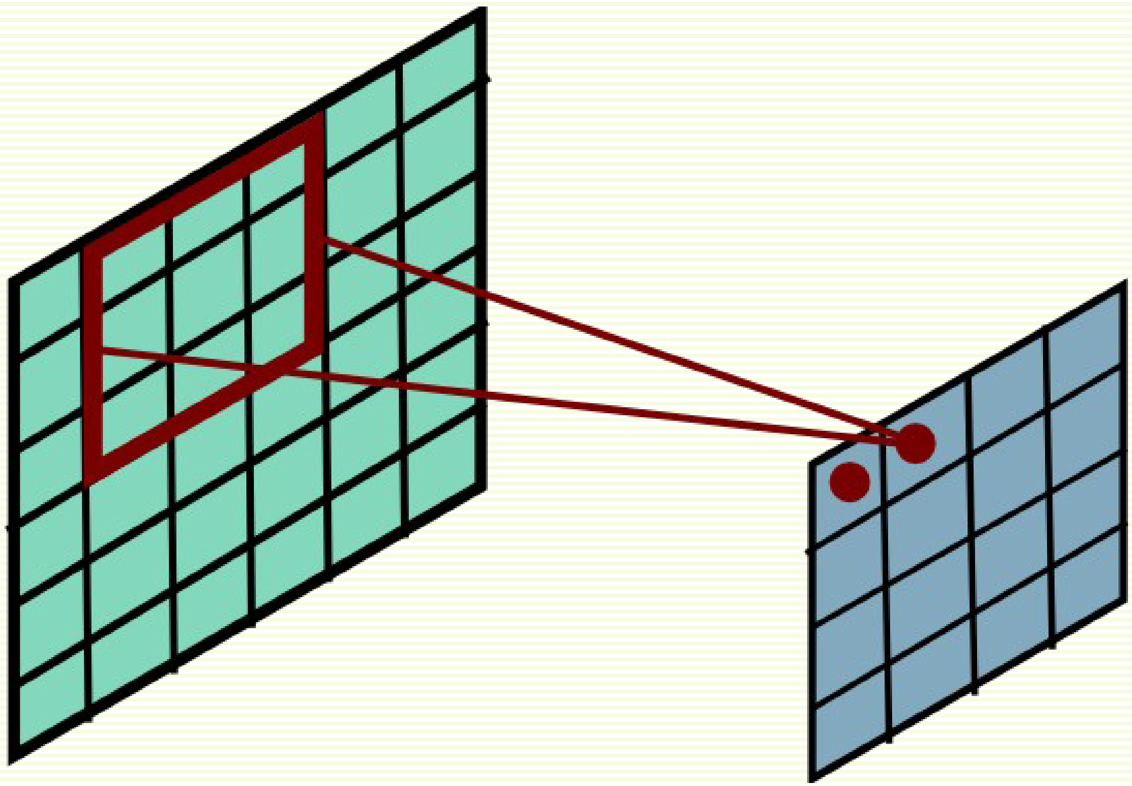
- Share parameters (network weights) across different locations
- Note similarity to convolution with some fixed filter
- But here the filter is learned



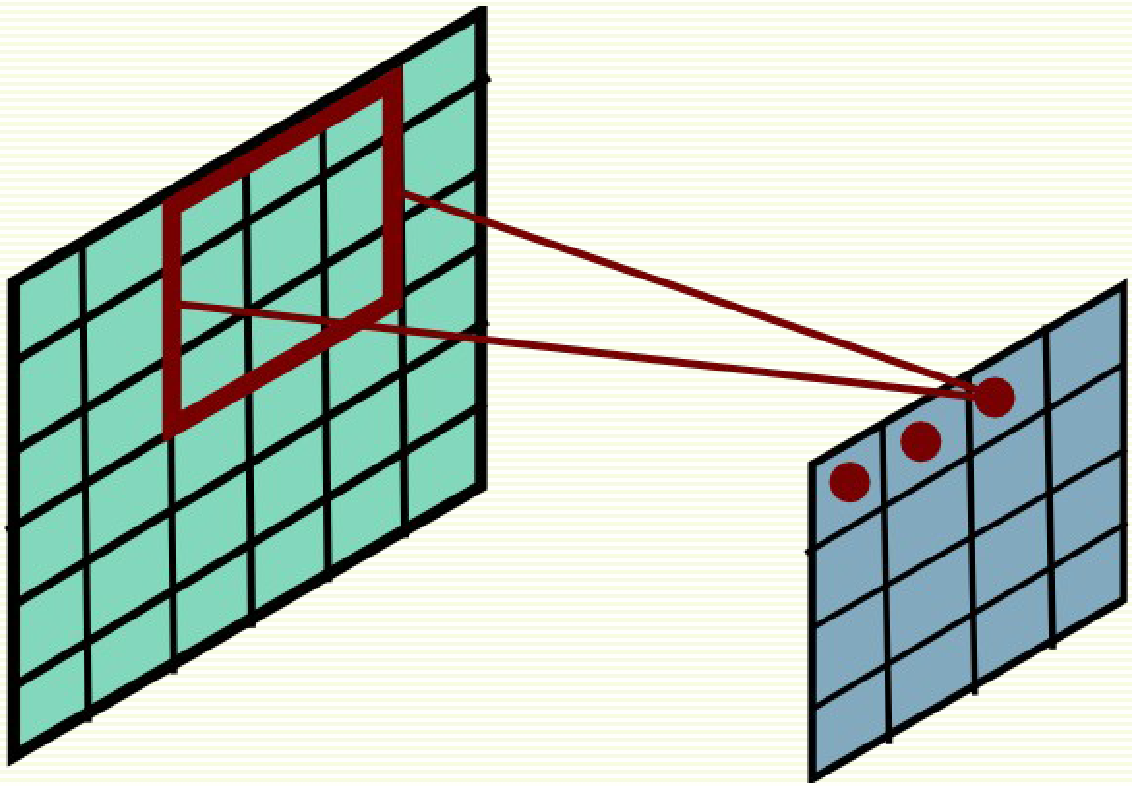
Convolutional Layer



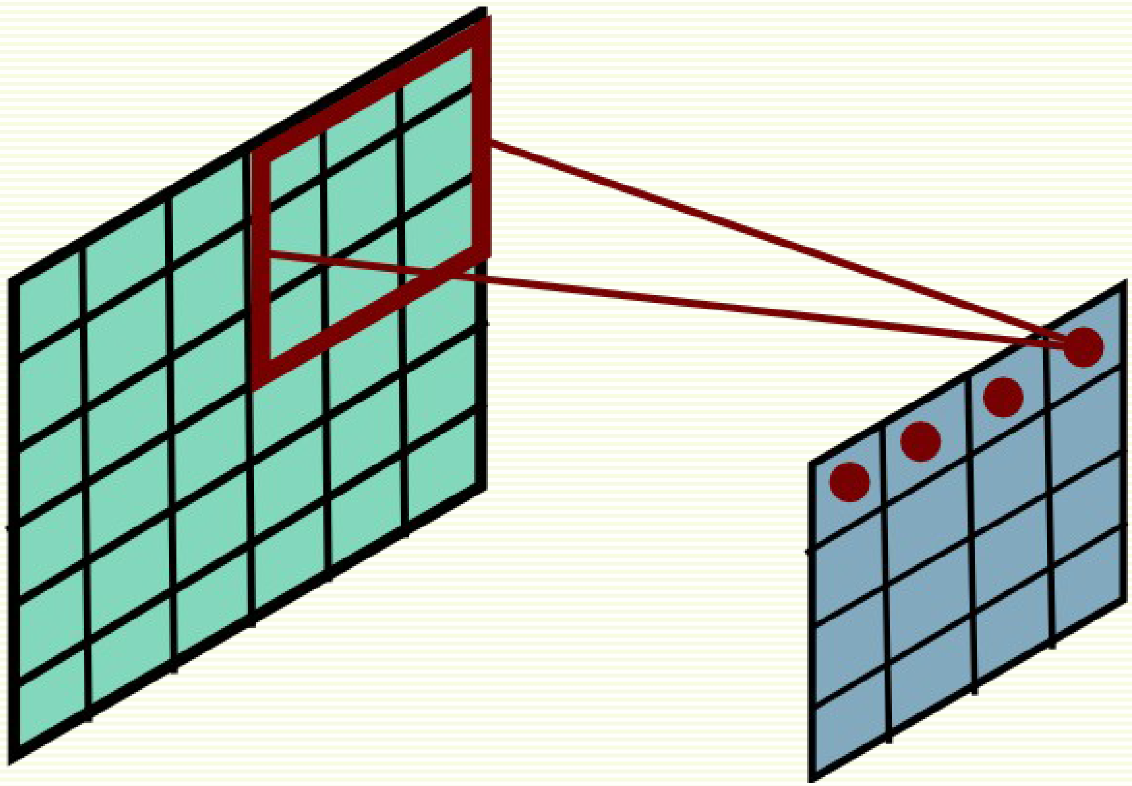
Convolutional Layer



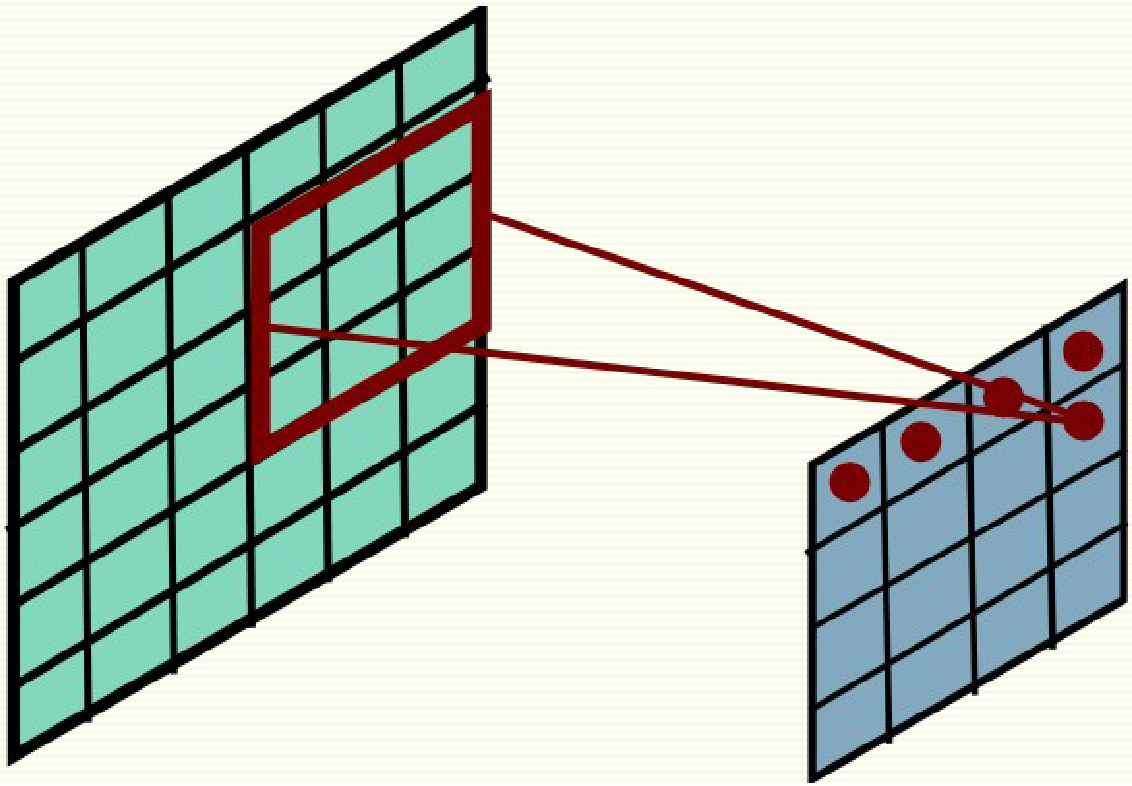
Convolutional Layer



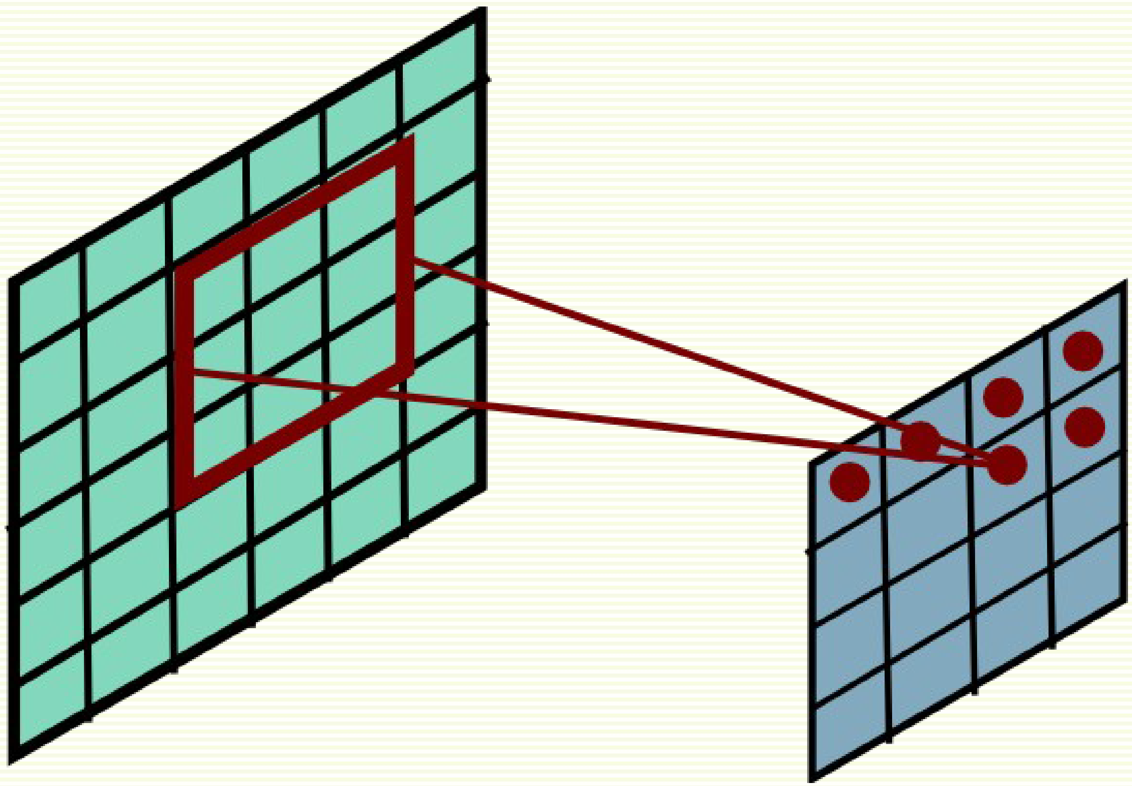
Convolutional Layer



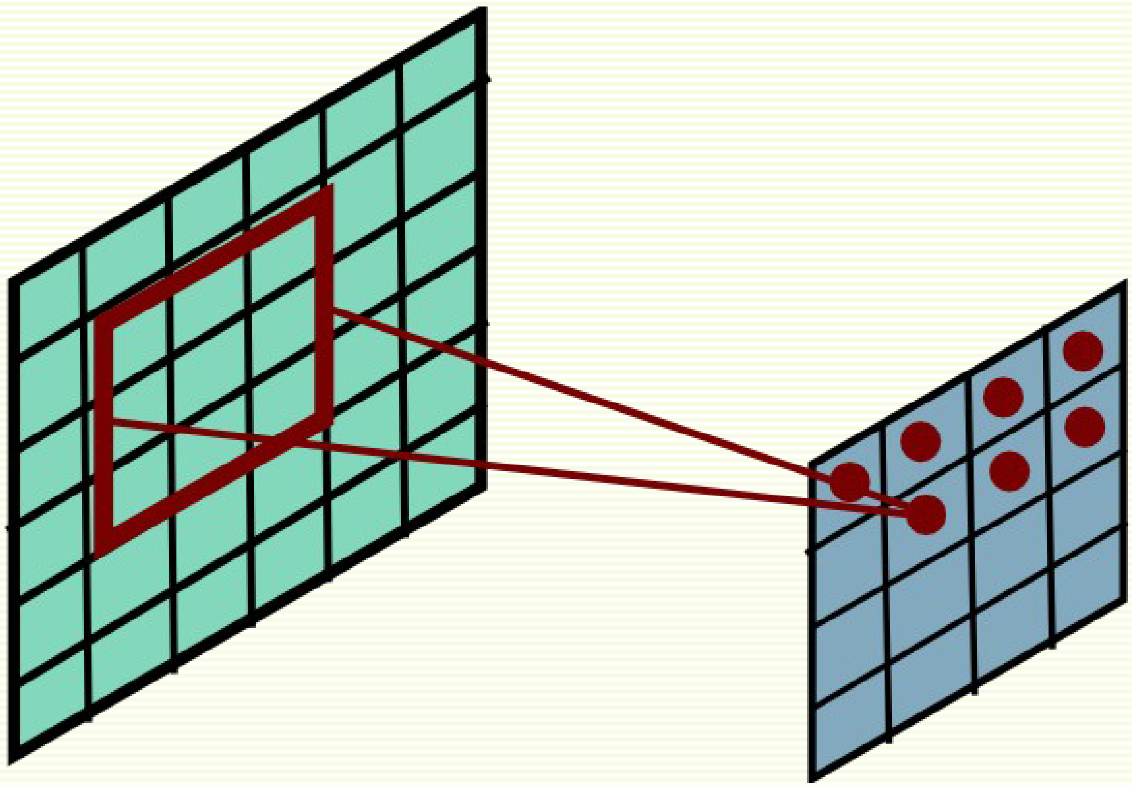
Convolutional Layer



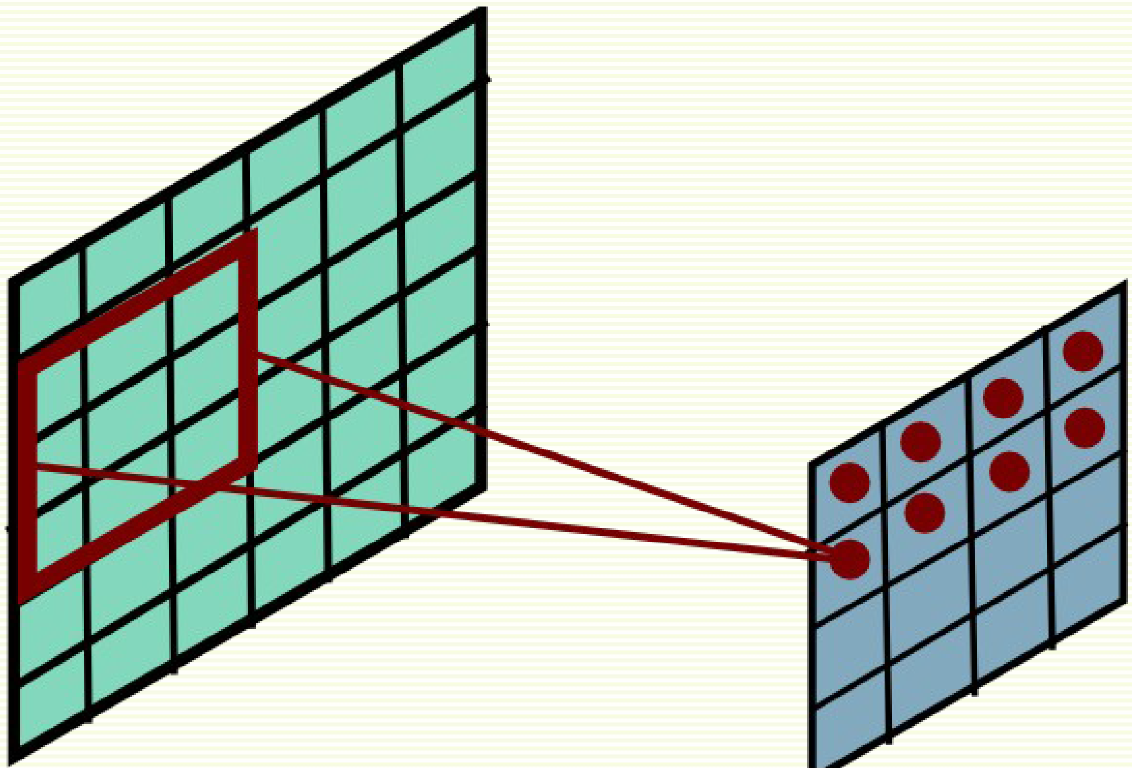
Convolutional Layer



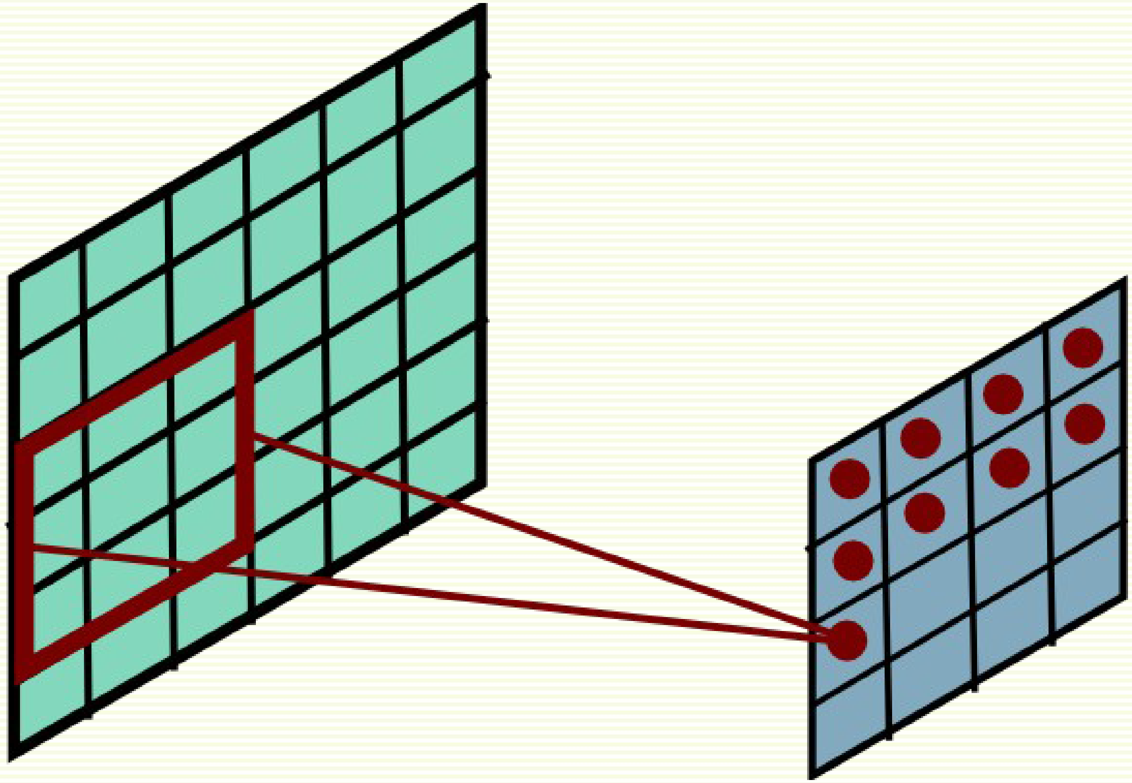
Convolutional Layer



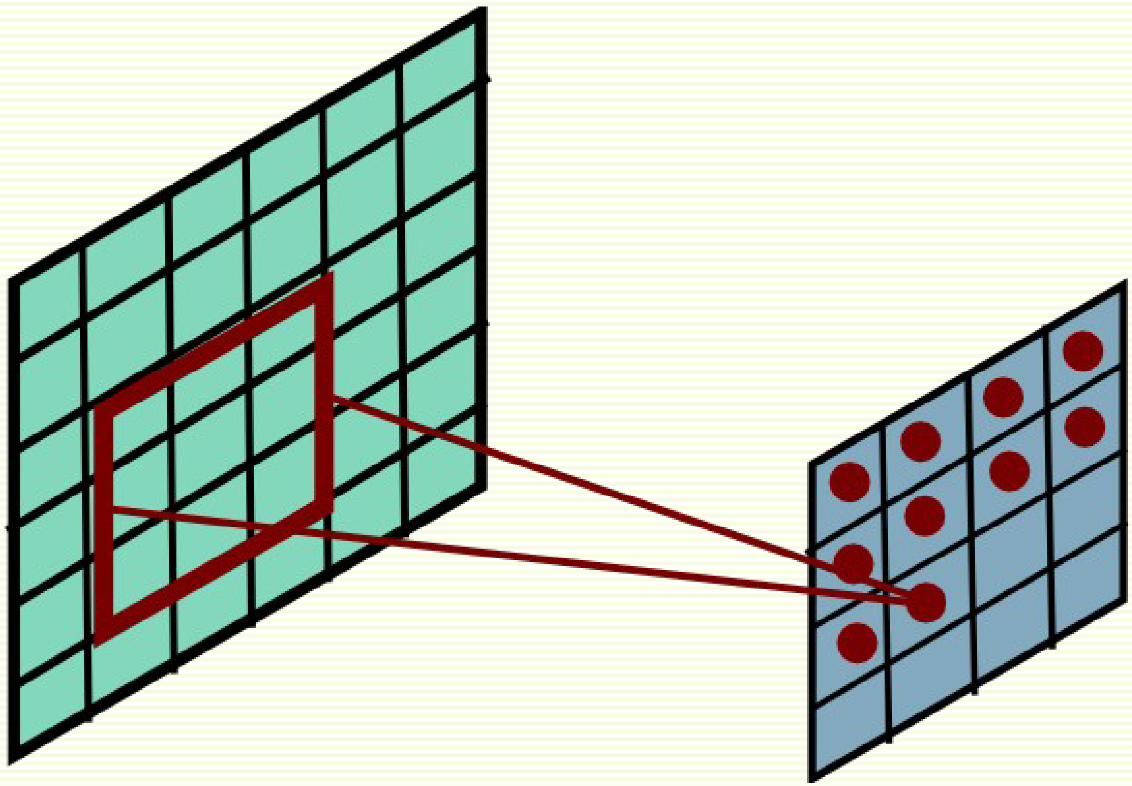
Convolutional Layer



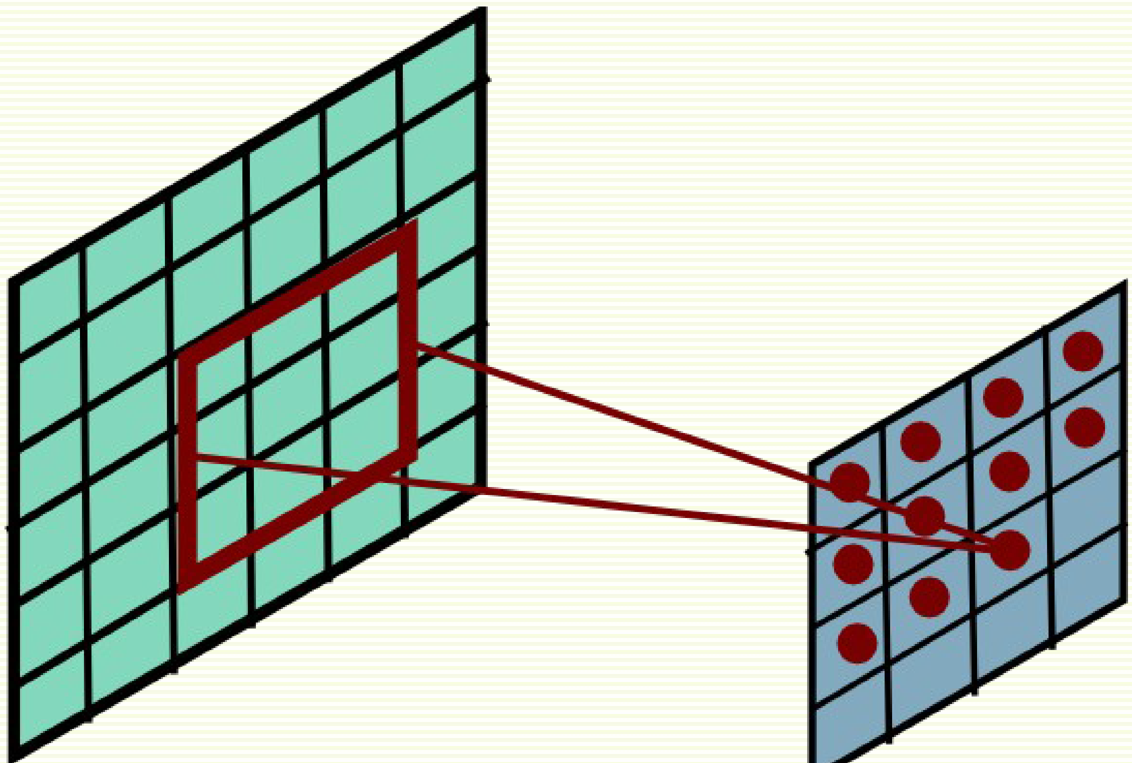
Convolutional Layer



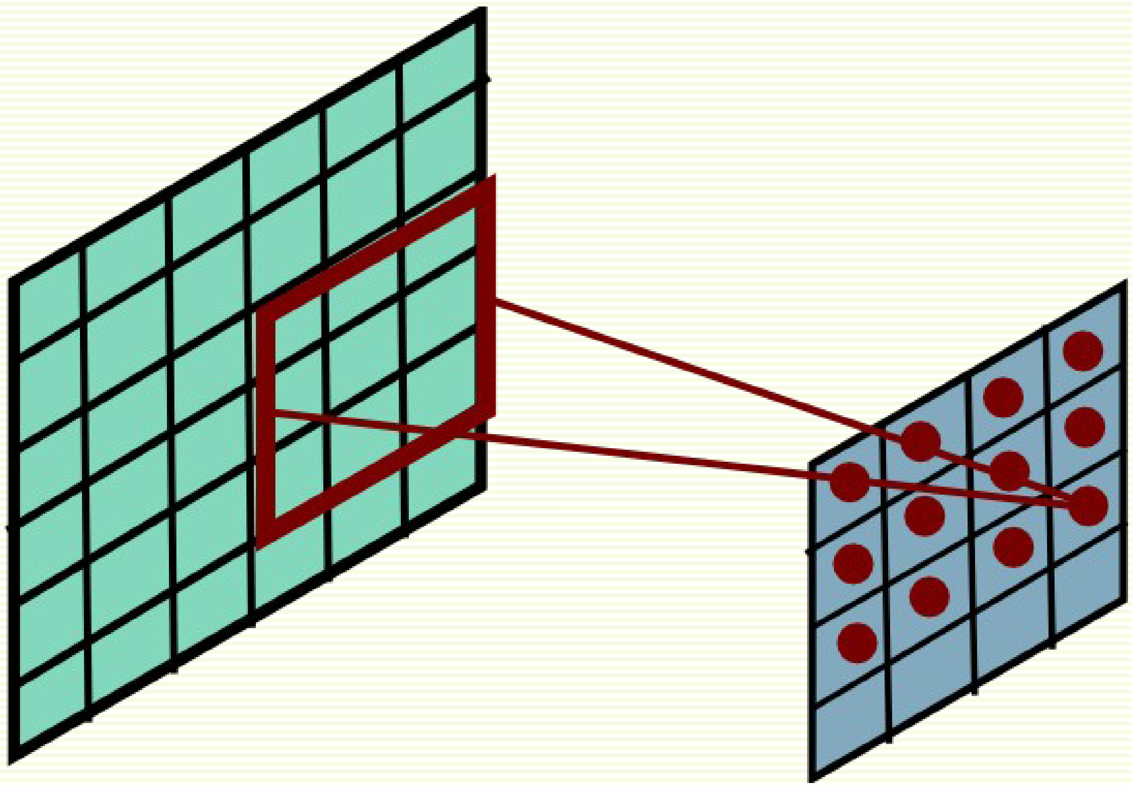
Convolutional Layer



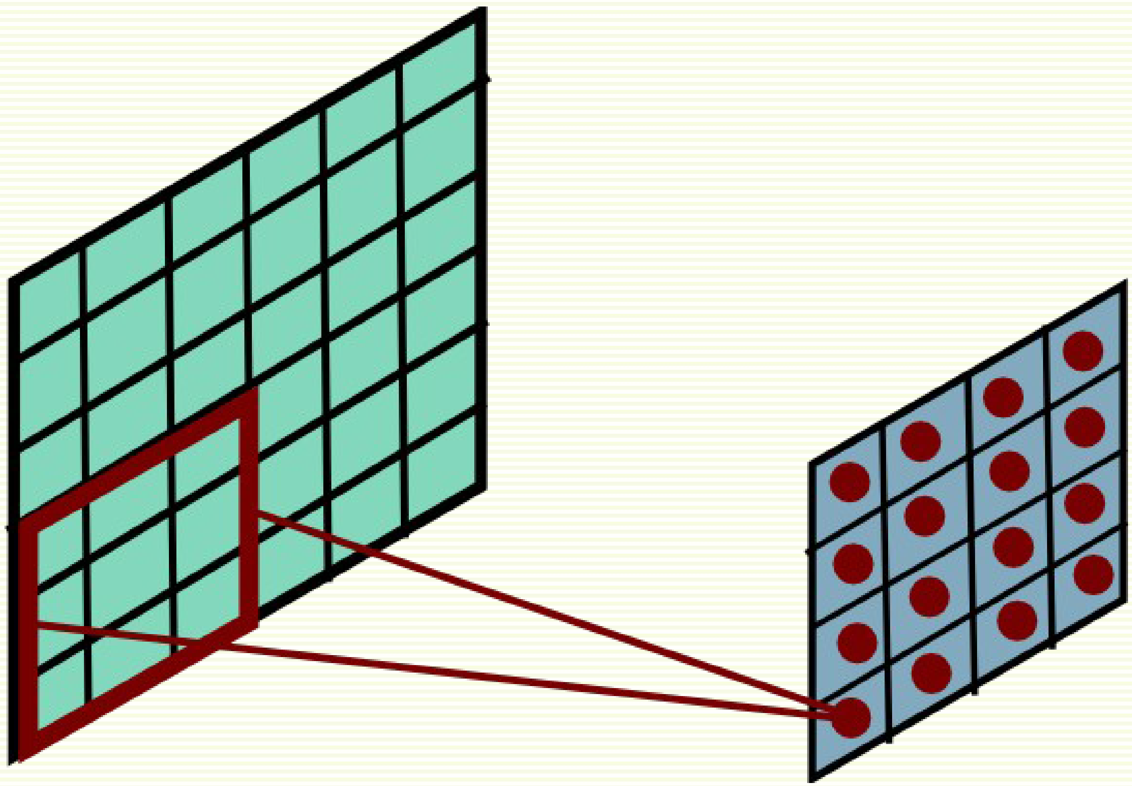
Convolutional Layer



Convolutional Layer

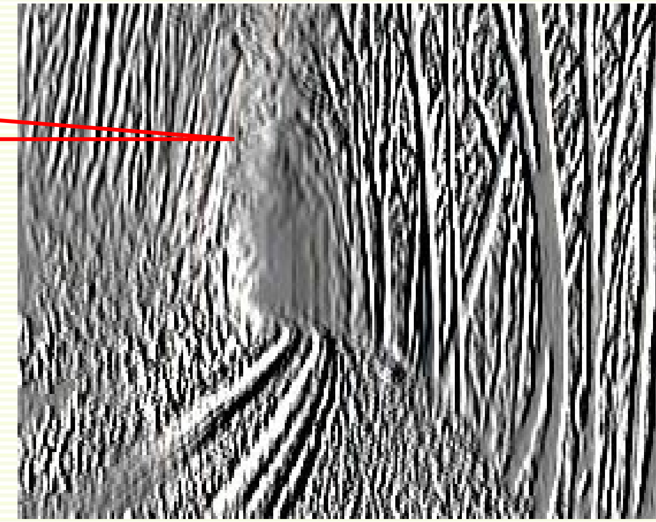


Convolutional Layer



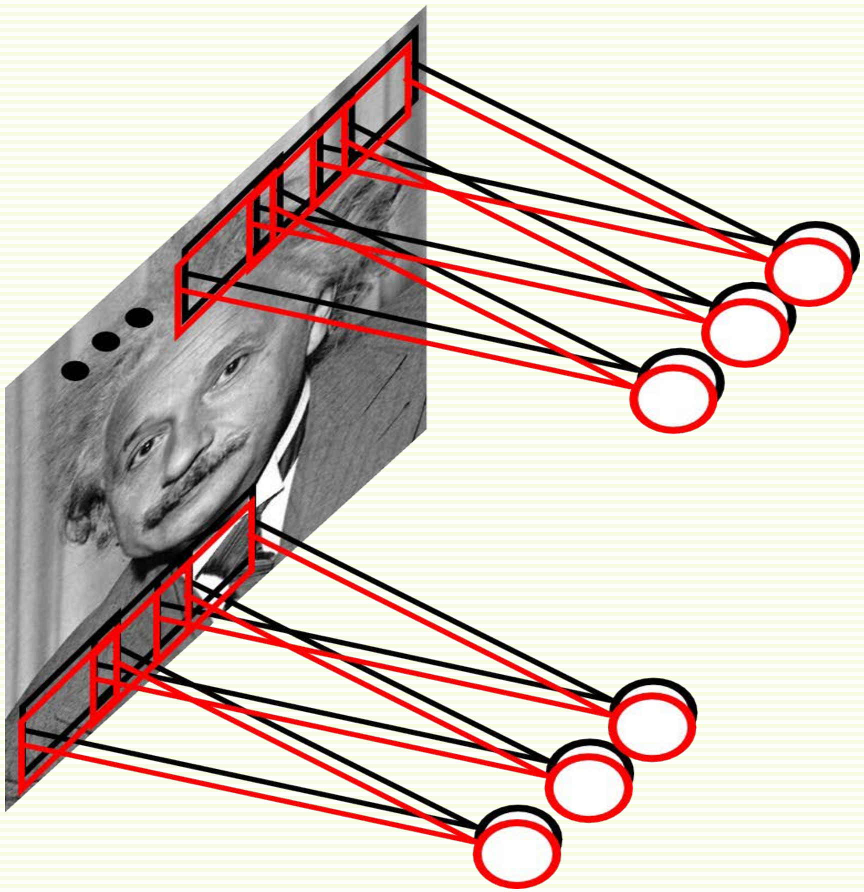
Convolutional Layer

$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$



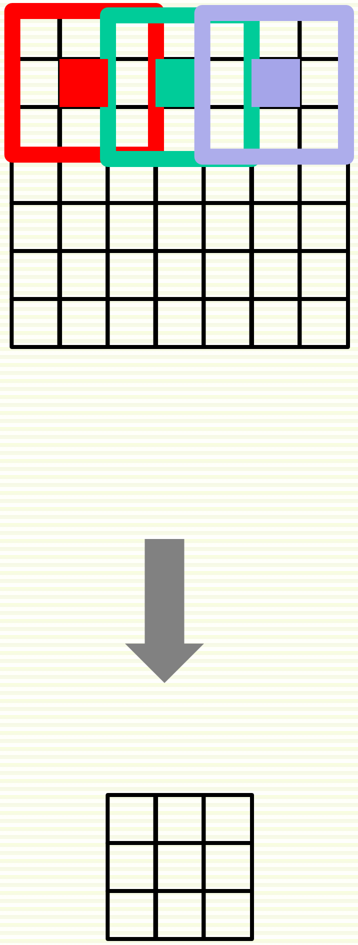
Convolutional Layer

- Each filter is responsible for one feature type
- Learn multiple filters
- Example:
  - 10x10 patch
  - 100 filters
  - only 10^4 parameters to learn
  - because parameters are shared between different locations



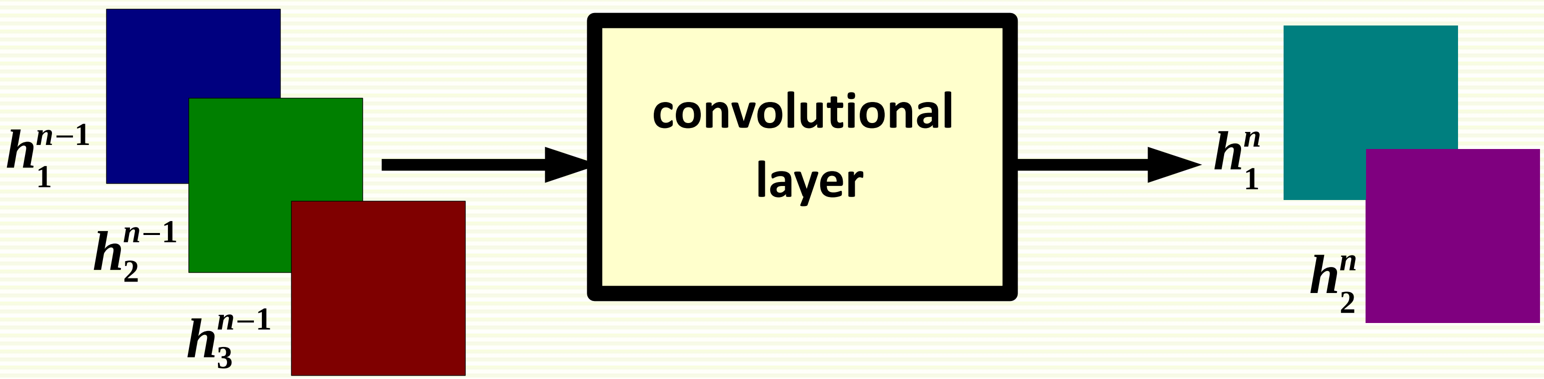
Convolutional Layer

- Can apply convolution only to some pixels (say every second)
  - output layer is smaller
  - less parameters to learn
- Example
  - stride = 2
  - apply convolution every second pixel
  - makes image approximately twice smaller in each dimension
    - there is also cropping of image border due to convolution



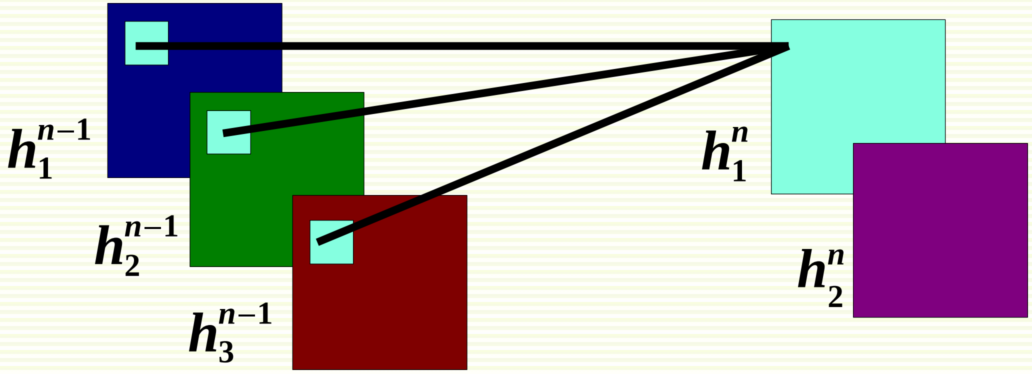
Convolutional Layer

- Each layer **h** is a **d**-dimensional *image* or *map* **r** x **c** x **d**
- Thus perform **d**-dimensional convolution
- If using **d'** filters, next layer is a map of size **r'** x **c'** x **d'**
- Example with **d** = 3 and **d'** = 2 (i.e. 2 filters)
- **r'** and **c'** depend on whether convolution crops image border and the stride of convolution



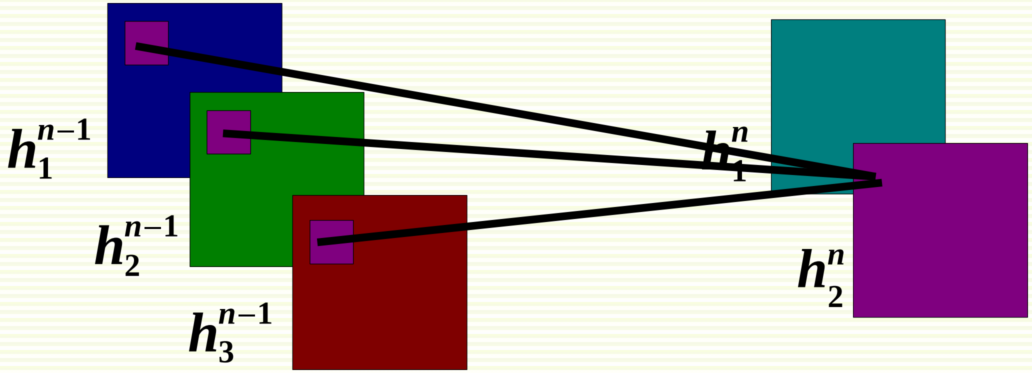
Convolutional Layer

- Example with **d** = 3 and **d'** = 2 (i.e. 2 filters)
- Applying the first filter



Convolutional Layer

- Example with **d** = 3 and **d'** = 2 (i.e. 2 filters)
- Applying the second filter



Convolutional Layer

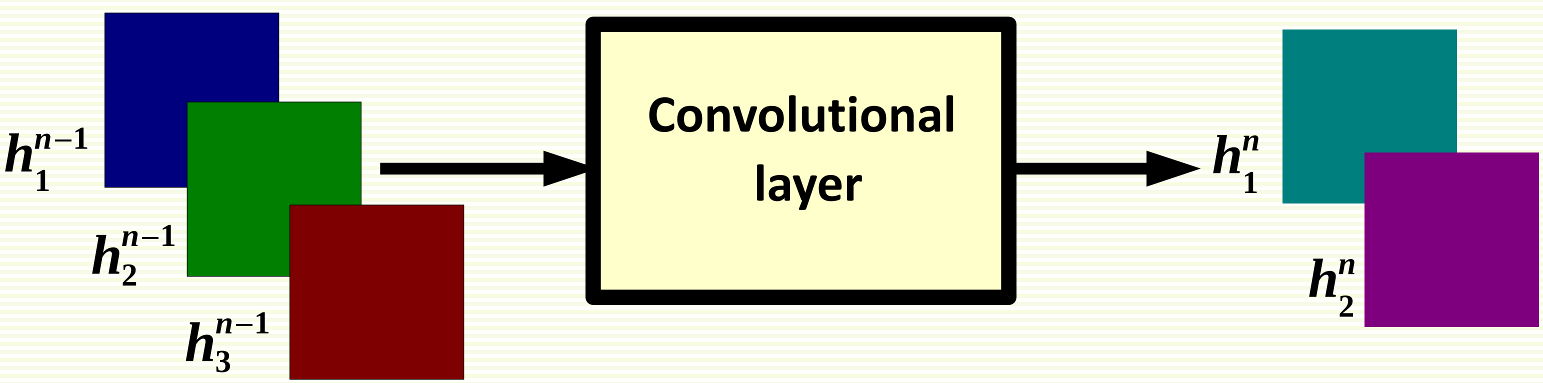
- Formula for convolution application to **K** dimensional layer $\mathbf{h}^{n-1}$
  - Also with application of ReLu activation function

kernel

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

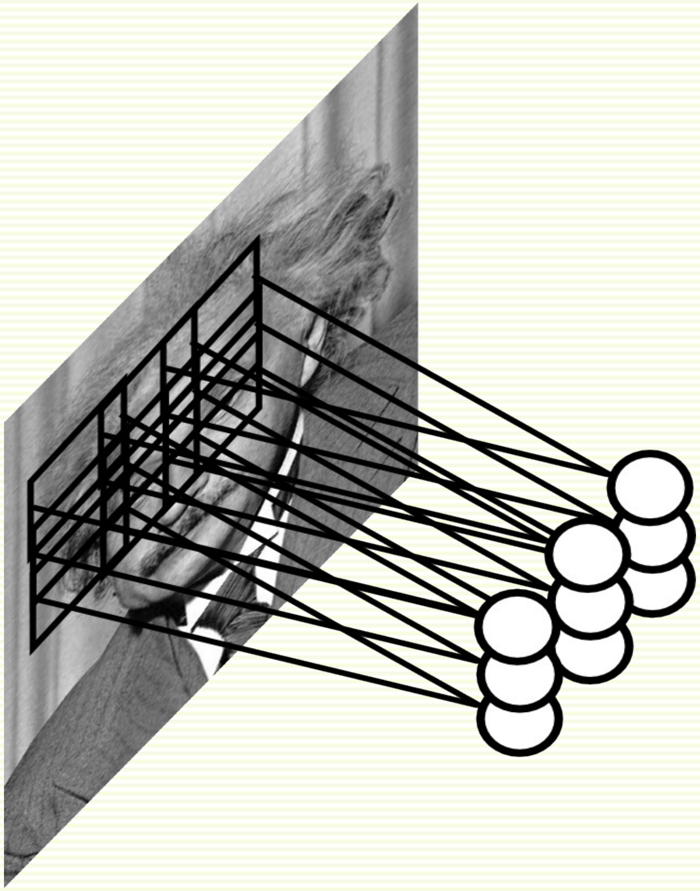
output feature map

input feature map



Pooling Layer

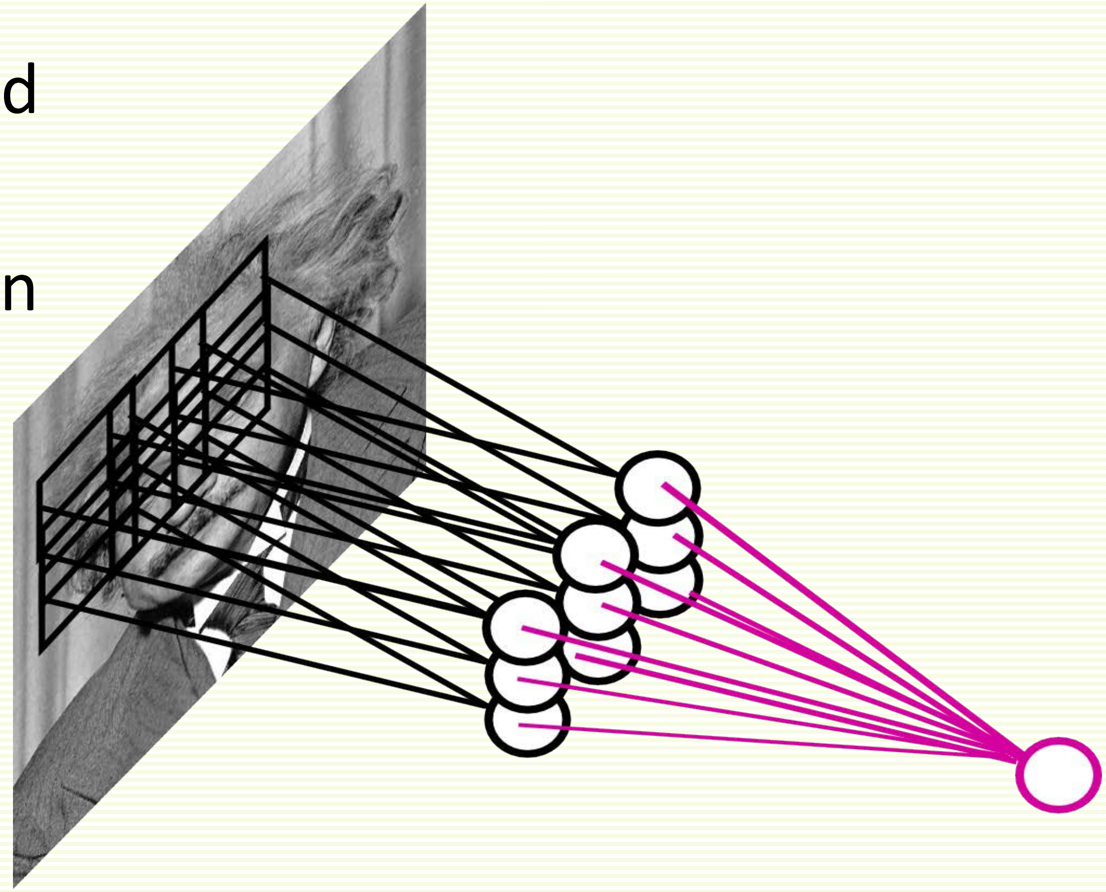
- Say a filter is an eye detector
- Want to detection to be robust to precise eye location



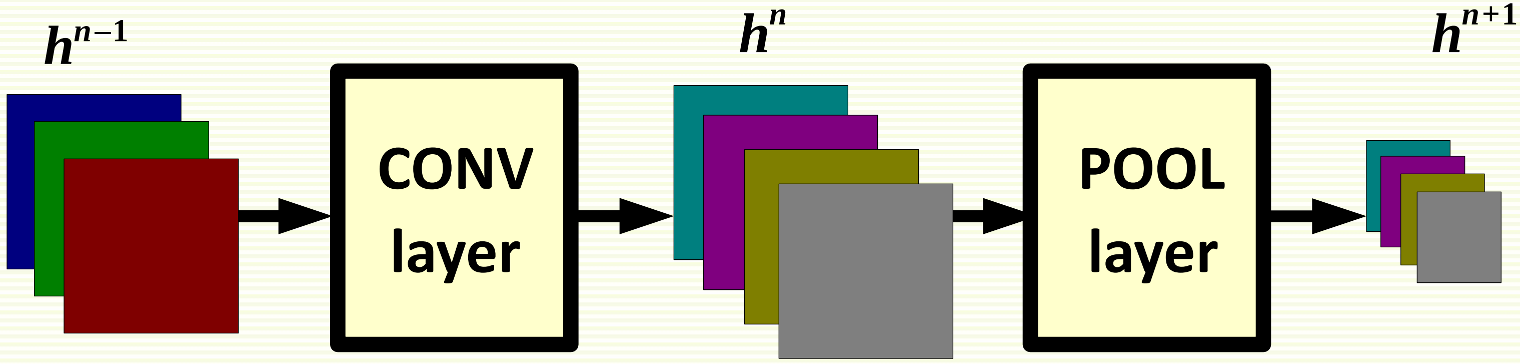
Pooling Layer

- *Pool* filter responses at different locations gain robustness to exact spatial location
  - pooling could be taking max, average, etc.

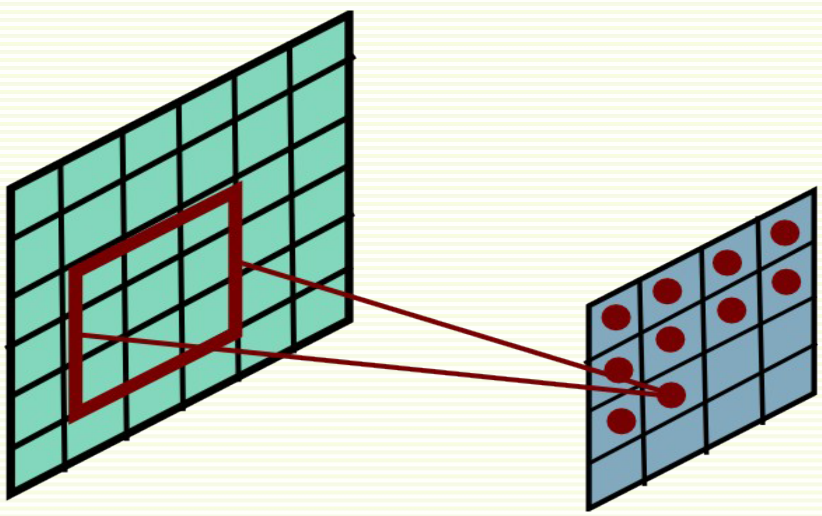
- Usually pooling applied with stride > 1
- This reduces resolution of output map
- But we already lost resolution (precision) by pooling



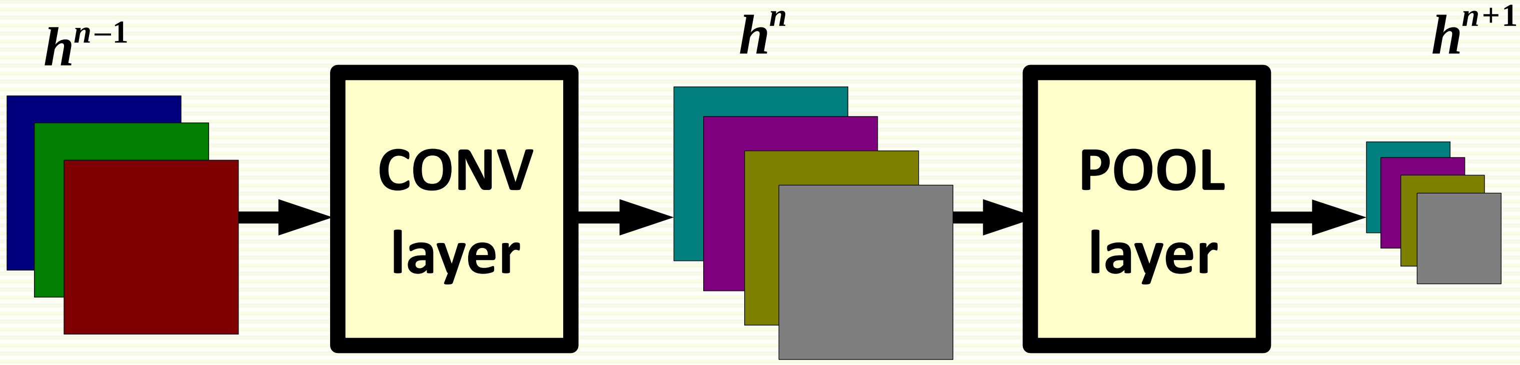
Pooling Layer: Receptive Field Size



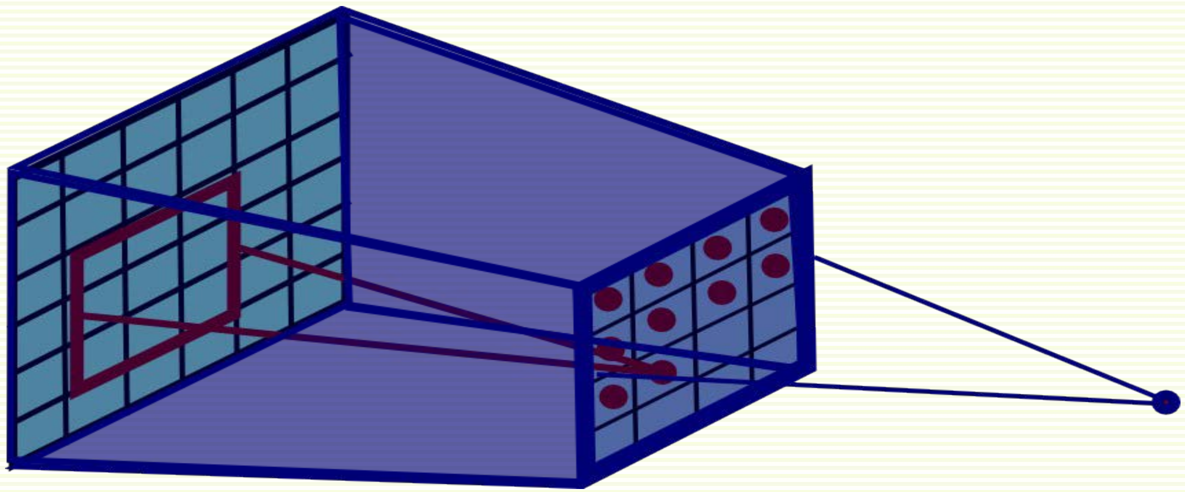
- If convolution filters have size **K** x **K** and stride 1, and pooling layer has pools of size **P** x **P**, then each unit in pooling layer depends on patch (in preceding convolution layer) of size (**P**+**K**-1) x (**P**+**K**-1)



Pooling Layer: Receptive Field Size



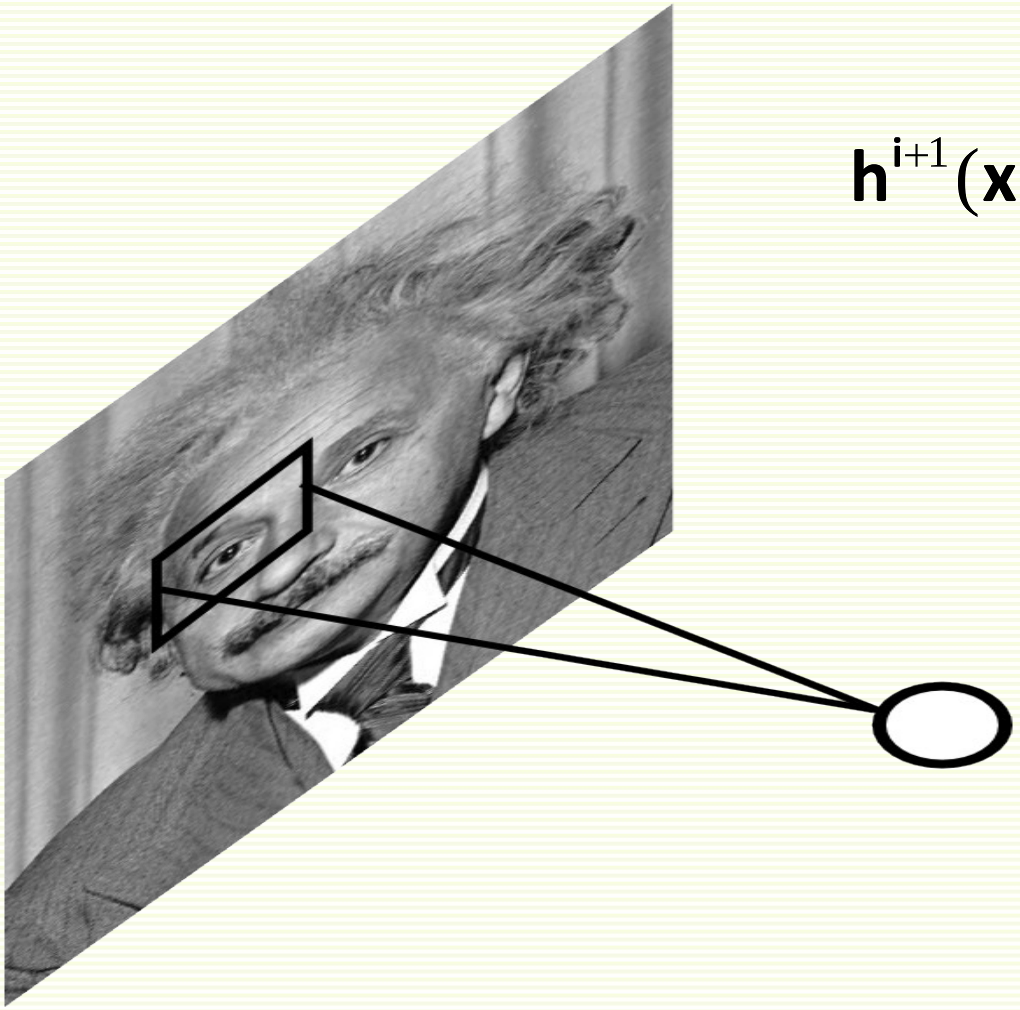
- If convolution filters have size **K** x **K** and stride 1, and pooling layer has pools of size **P** x **P**, then each unit in pooling layer depends on patch (in preceding convolution layer) of size (**P**+**K**-1) x (**P**+**K**-1)



Problem with Pooling

- After several levels of pooling, we have lost information about the precise positions of things
- This makes it impossible to use the precise spatial relationships between high-level parts for recognition.

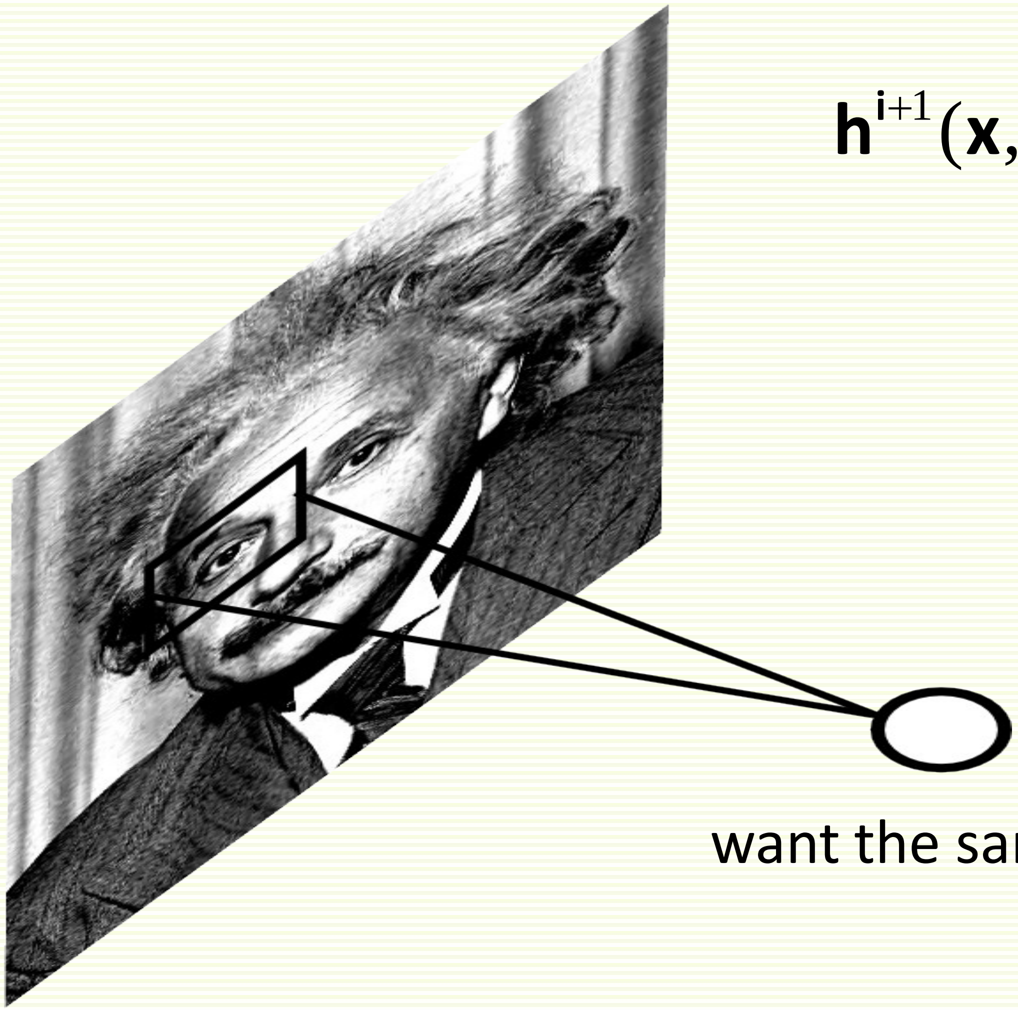
Local Contrast Normalization



$$\mathbf{h}^{\mathbf{i}+1}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{h}^{\mathbf{i}}(\mathbf{x},\mathbf{y}) - \mu^{\mathbf{i}}(\mathbf{N}(\mathbf{x},\mathbf{y}))}{\sigma^{\mathbf{i}}(\mathbf{N}(\mathbf{x},\mathbf{y}))}$$

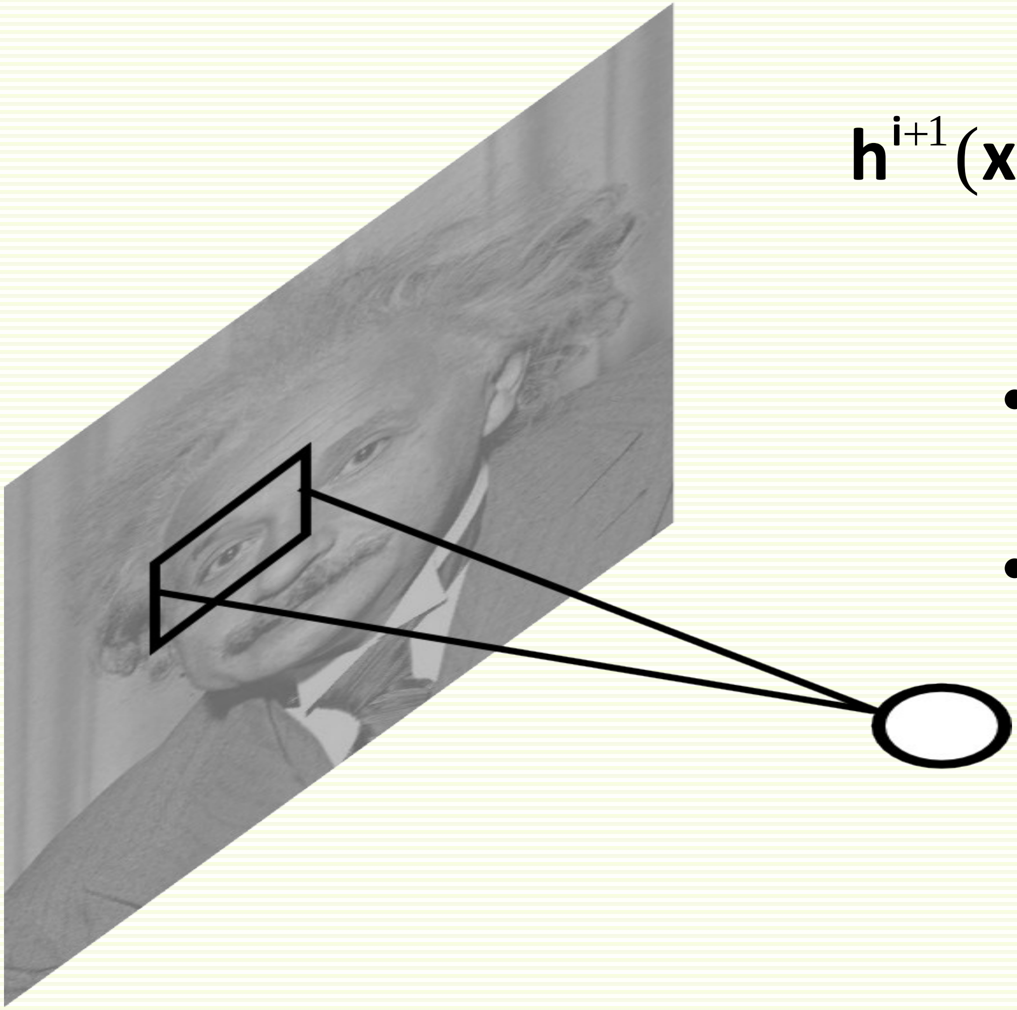
Local Contrast Normalization

$$\mathbf{h}^{\mathbf{i}+1}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{h}^{\mathbf{i}}(\mathbf{x},\mathbf{y}) - \mu^{\mathbf{i}}(\mathbf{N}(\mathbf{x},\mathbf{y}))}{\sigma^{\mathbf{i}}(\mathbf{N}(\mathbf{x},\mathbf{y}))}$$



want the same response

Local Contrast Normalization

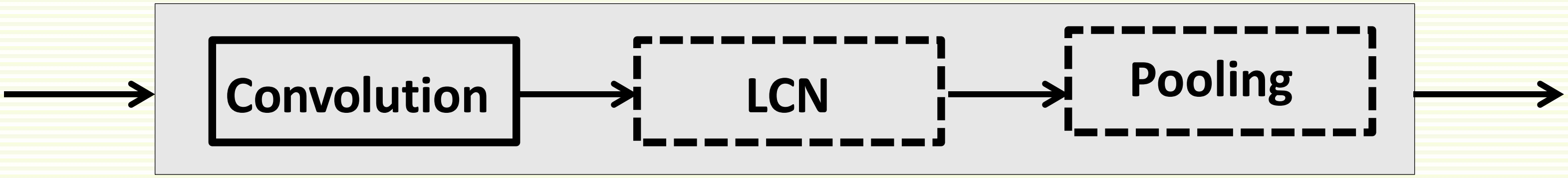


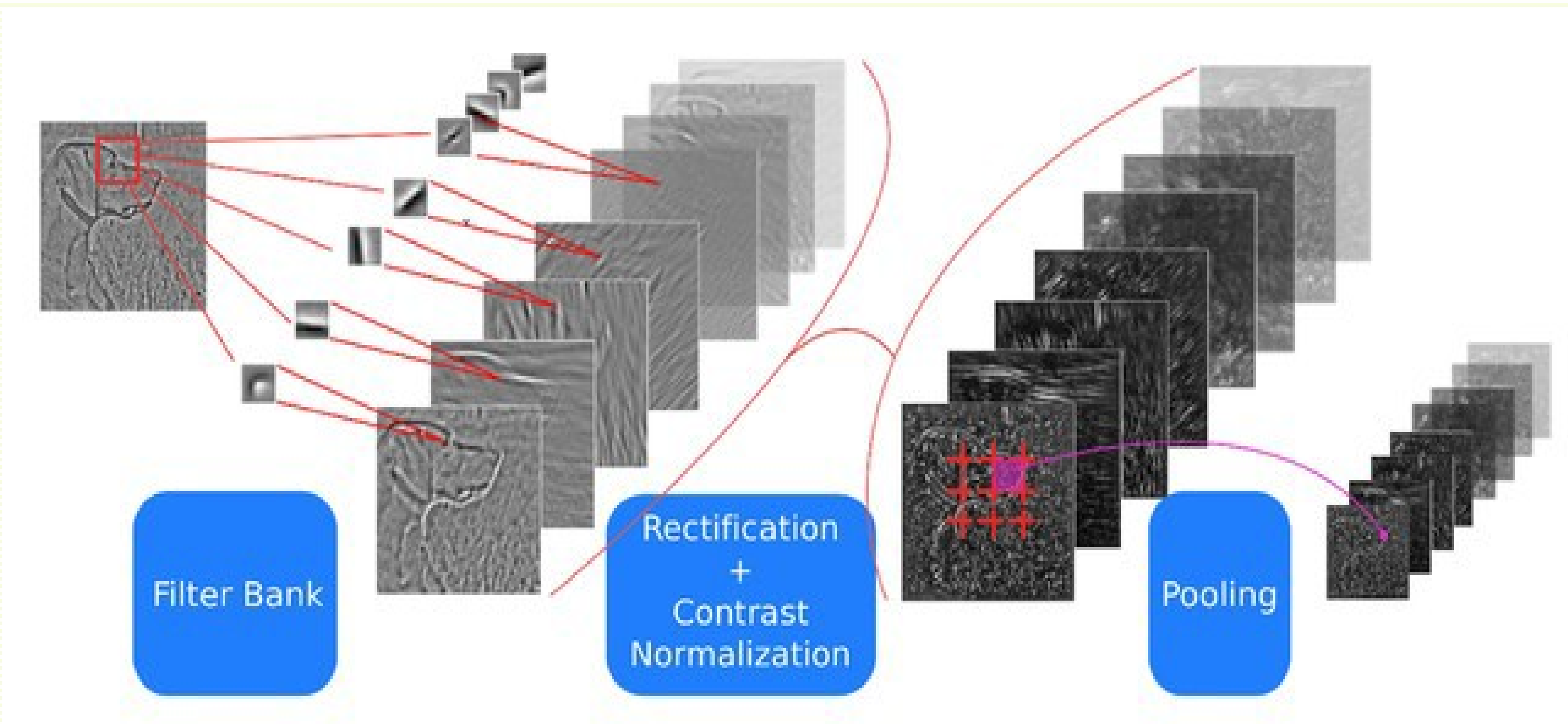
$$\mathbf{h}^{\mathbf{i}+1}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{h}^{\mathbf{i}}(\mathbf{x},\mathbf{y}) - \mu^{\mathbf{i}}(\mathbf{N}(\mathbf{x},\mathbf{y}))}{\sigma^{\mathbf{i}}(\mathbf{N}(\mathbf{x},\mathbf{y}))}$$

- Performed also across features and in higher layers
- Effects
  - Improves invariance
  - Improves optimization

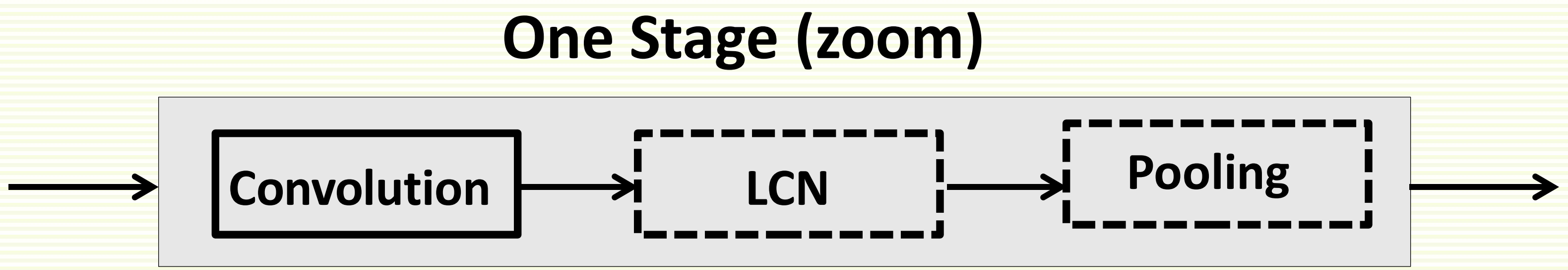
ConvNets: Typical Stage

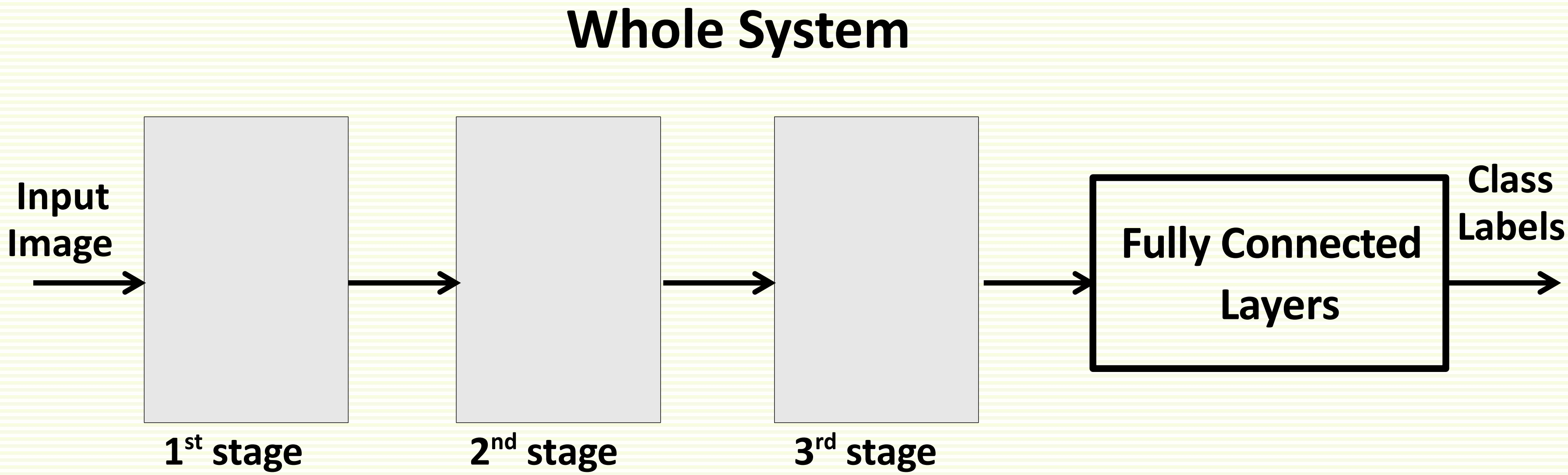
One Stage (zoom)





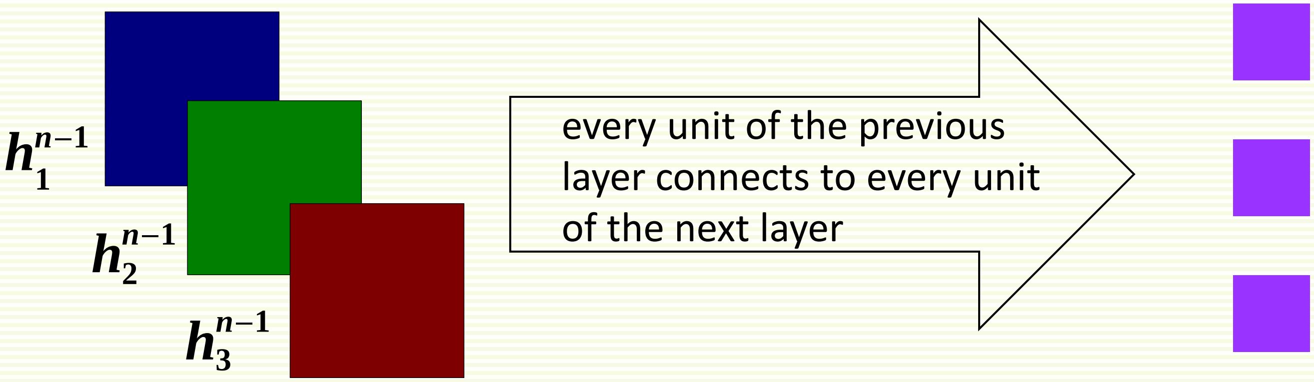
Typical Architecture



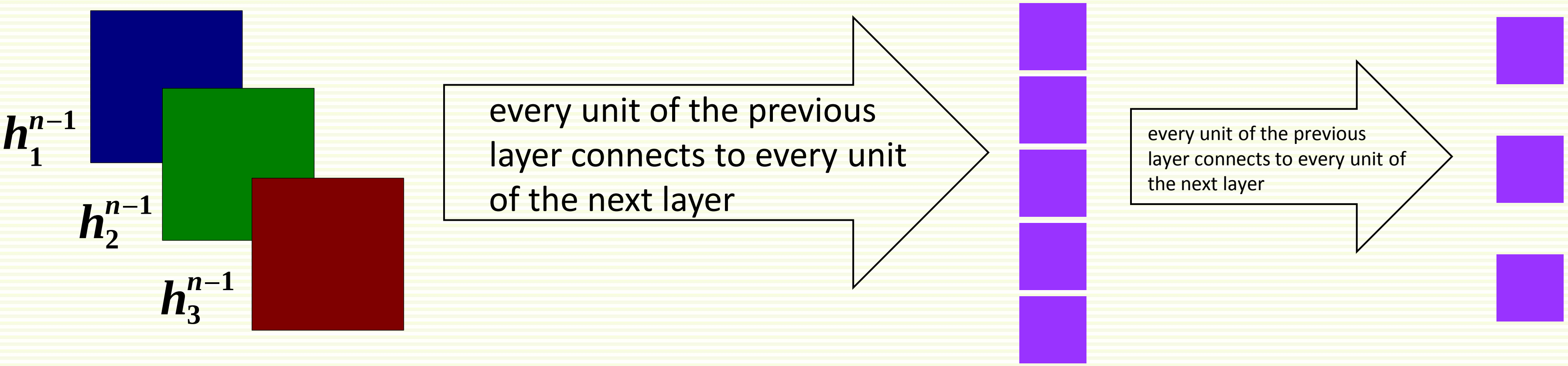


Fully Connected Layer

- Can have just one fully connected layer
- Example for 3-class classification problem



- Can have many fully connected layer
- Example for 3-class classification problem

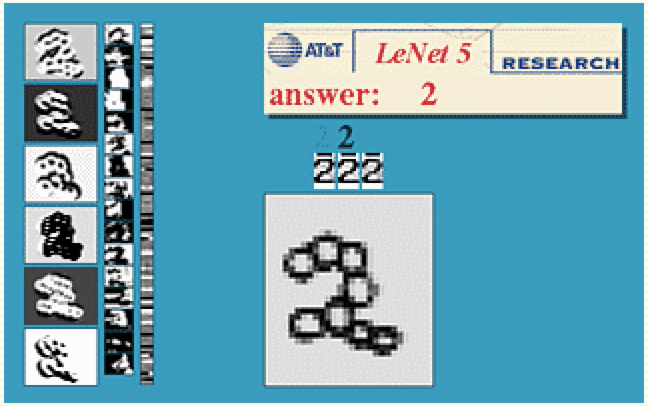


ConvNets: Training

- All Layers are differentiable
- Use standard back-propagation (gradient descent)
- At test time, run only in forward mode

Conv Nets: Character Recognition

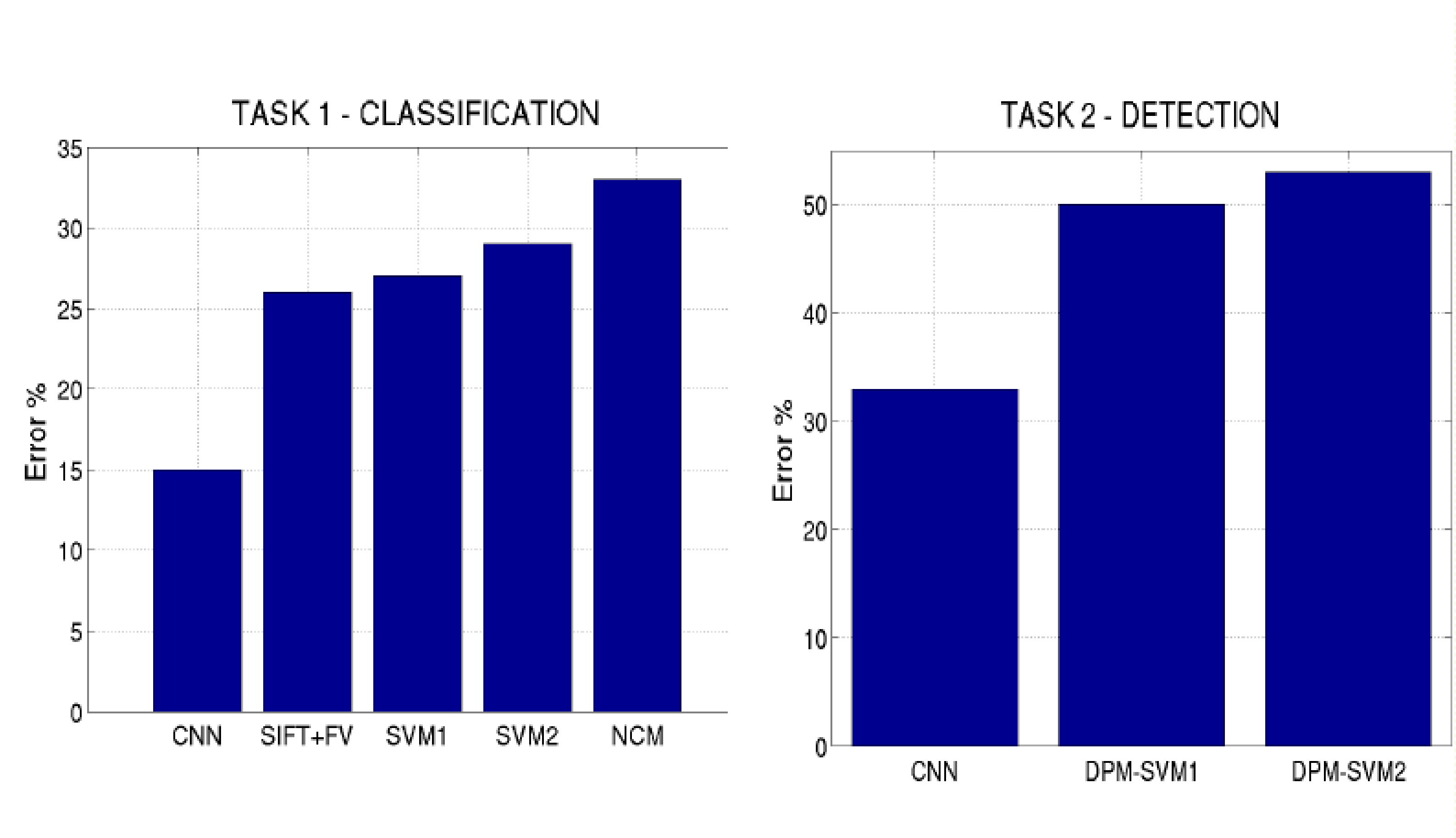
- http://yann.lecun.com/exdb/lenet/index.html

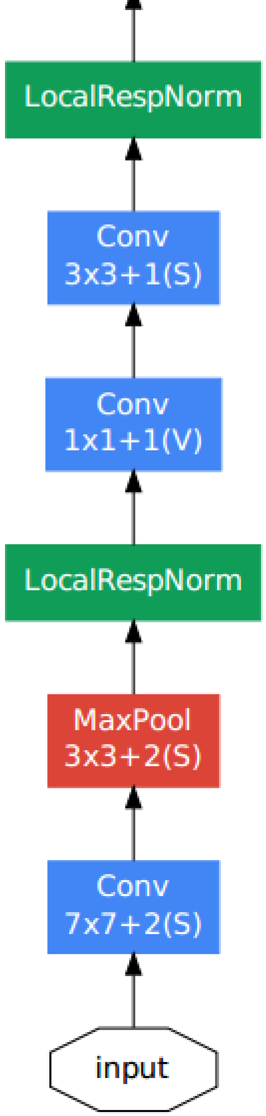


ConvNet for ImageNet

- Krizhevsky et.al.(NIPS 2012) developed deep convolutional neural net of the type pioneered by Yann LeCun
- Architecture:
  - 7 hidden layers not counting some max pooling layers
  - the early layers were convolutional
  - the last two layers were globally connected
- Activation function:
  - rectified linear units in every hidden layer
  - train much faster and are more expressive than logistic unit

Results: ILSVRC 2012





Going Deeper with Convolutions
http://arxiv.org/abs/1409.4842