CS4442/9542b
Artificial Intelligence II
prof. Olga Veksler

Lecture 12

Computer Vision

Object Recognition
Convolutional Neural Networks

Many slides are from A. Ng, Y. LeCun, G. Hinton, A. Ranzato

Outline

- Object Recognition
- Deep Neural Networks
    - Convolutional Neural Network

Traditional Object Classification

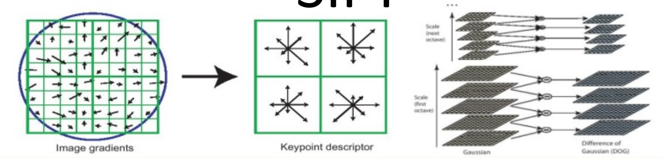
- Tradition Object Recognition system

training data → extract hand-crafted features → train classifier

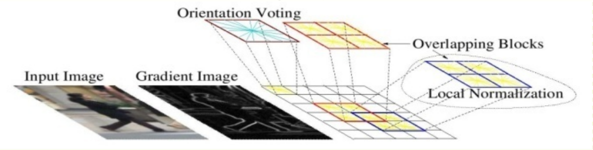
new data → extract hand-crafted features → apply classifier

- Most of the progress in Object Recognition in 1990's and early 2000's was due to designing good features by hand

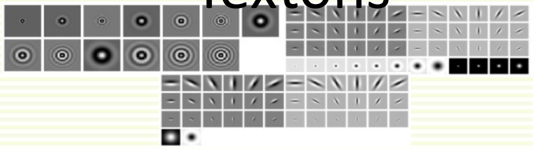
SIFT



HOG

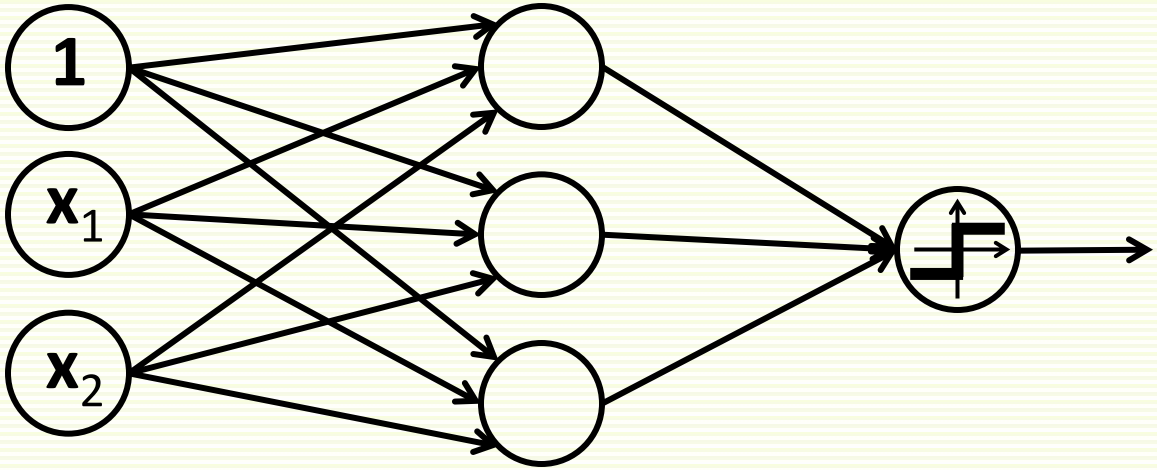


Textons



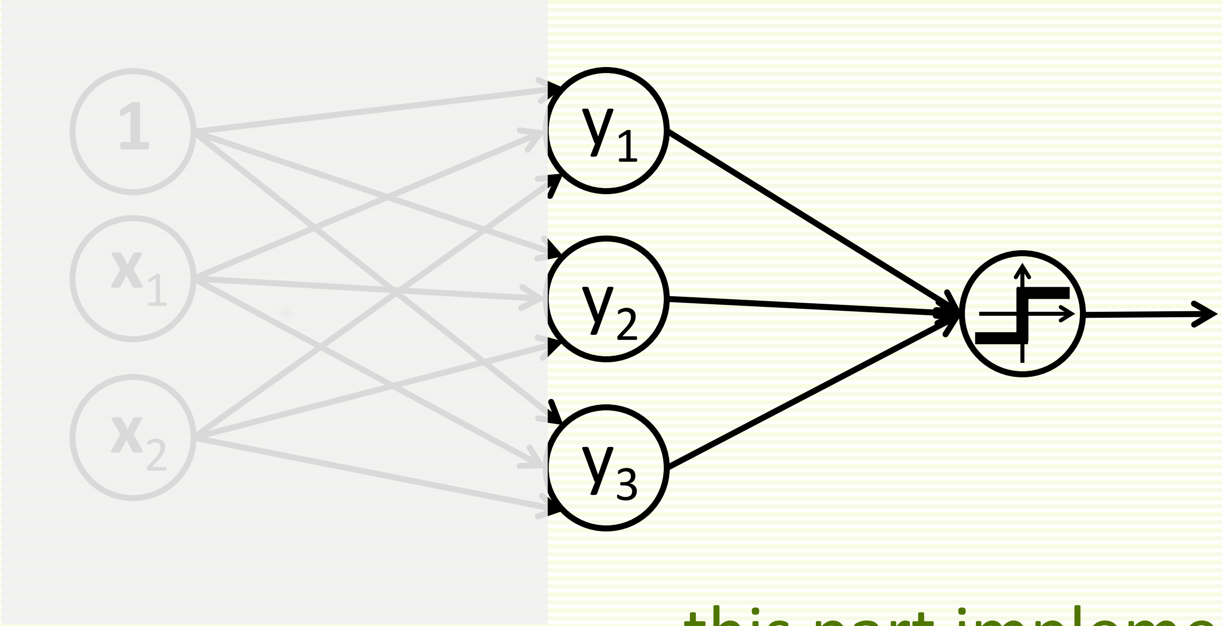
Patches

NN as Non-Linear Feature Mapping



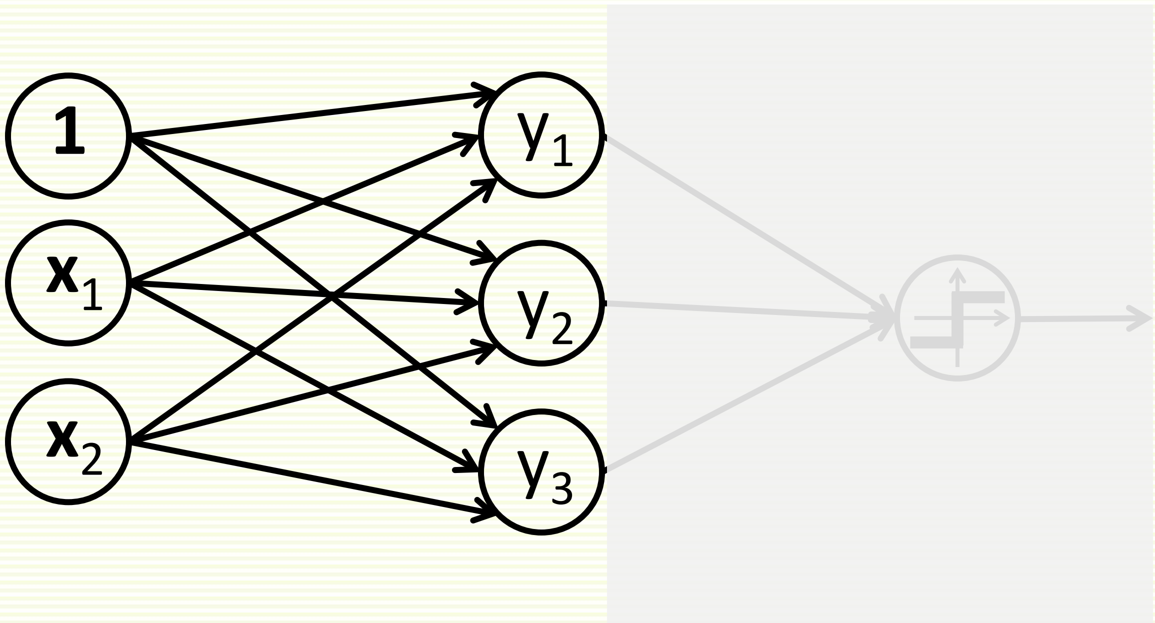
- 1 hidden layer NN can be interpreted as first mapping input features to new features
- Then applying (linear classifier) to the new features

NN as Non-Linear Feature Mapping



this part implements
Perceptron (liner classifier)

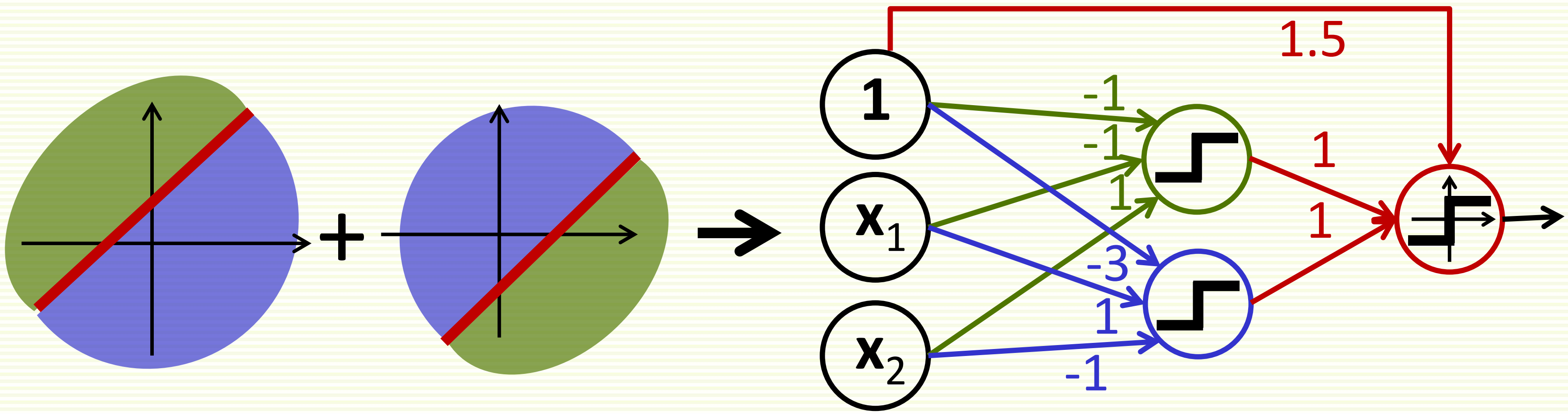
NN as Non-Linear Feature Mapping

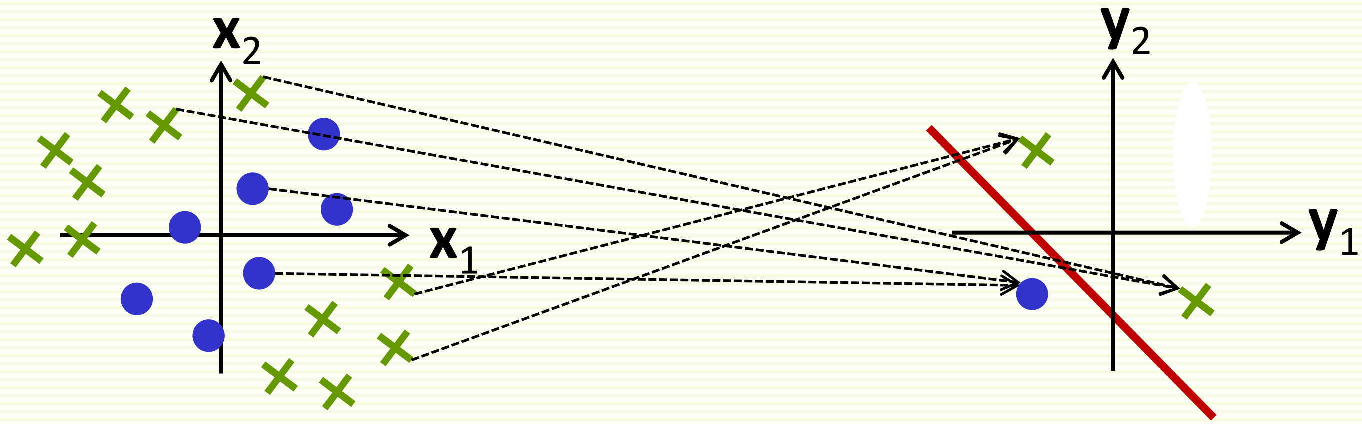


this part implements
mapping to new features **y**

NN as Nonlinear Feature Mapping

- Consider 3 layer NN example we saw previously:





non linearly separable in the original feature space

linearly separable in the new feature space

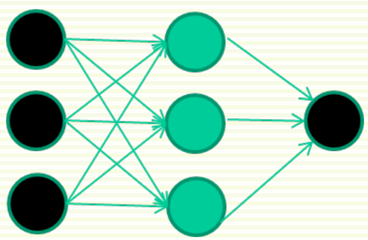
NN as Nonlinear Feature Mapping

- Designing hand-crafted features is time consuming
- With NN, change in paradigm
  - instead of hand-crafting , learn features automatically from data

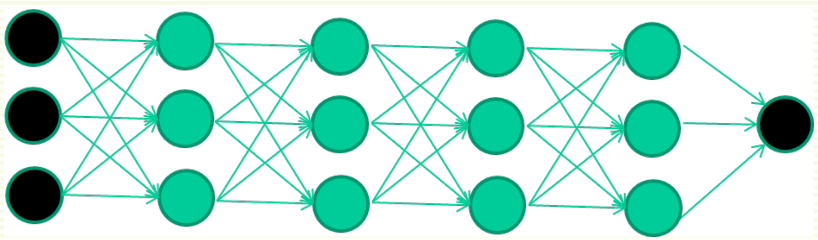
Shallow vs. Deep Architecture

- How many layers should we choose?

Shallow network



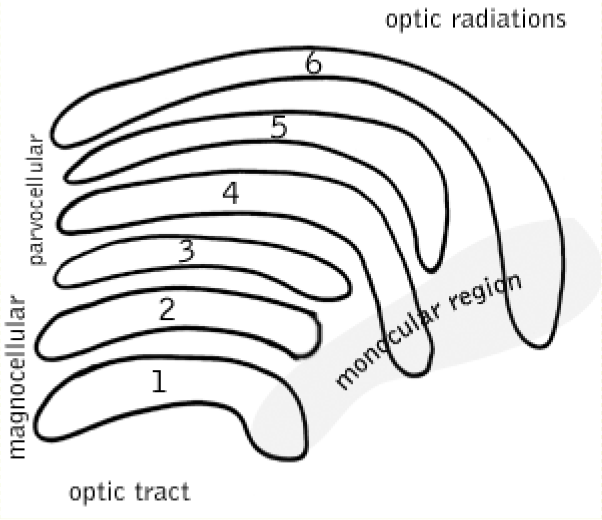
Deep network

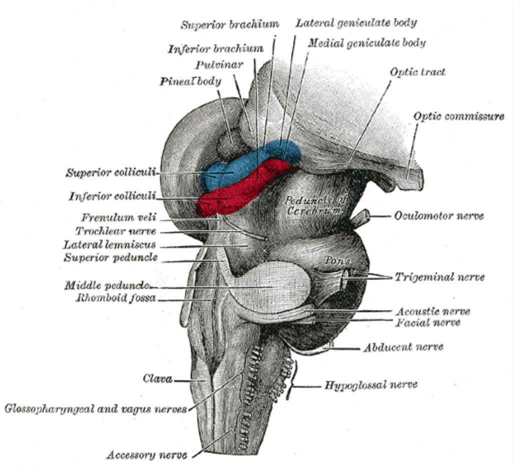


- Deep network lead to many successful applications recently

Why Deep Networks

- Evidence from biology



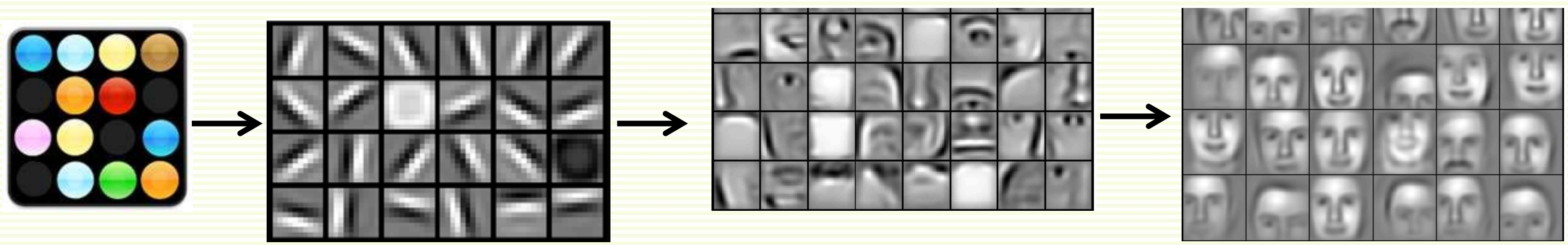


Why Deep Networks

- 2 layer networks can represent any function
- But deep architectures are more efficient for representing some functions
  - problems that can be represented with a polynomial number of nodes with *k* layers, may require an exponential number of nodes with *k*-1 layers
  - thus with deep architecture, less units might be needed overall
    - less weights, less parameter updates, more efficient

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

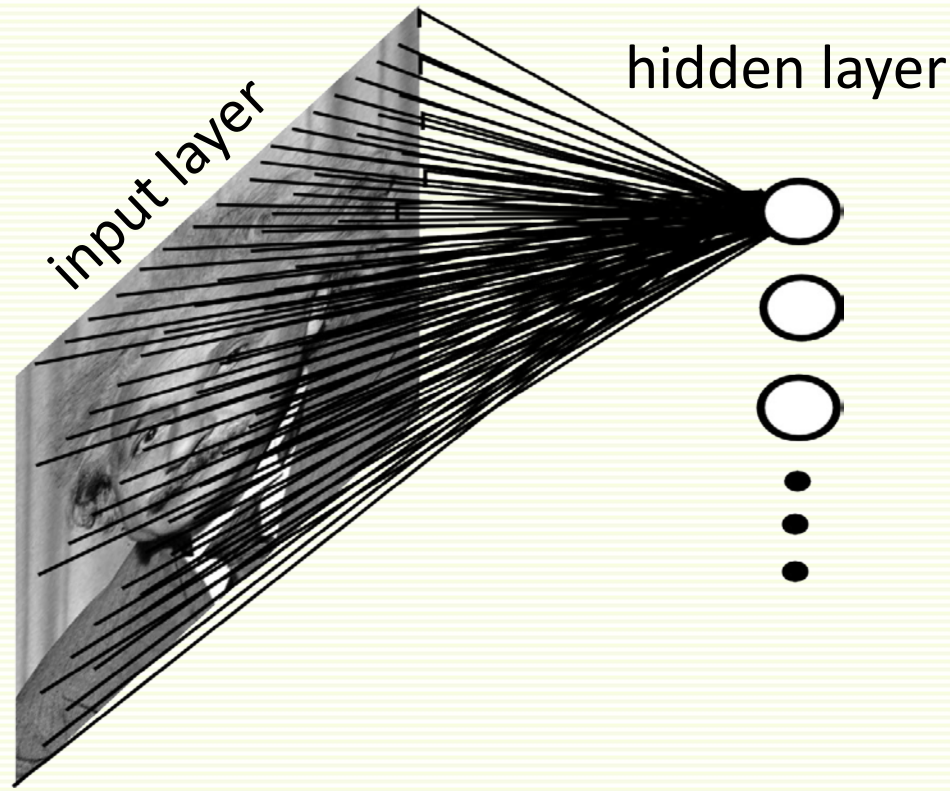
- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Networks (convnets)
  - Similarities to Neo-Cognitron
  - Success on character recognition
- Other attempts at deeply layered Networks trained with backpropagation
  - not much success
    - very slow
    - diffusion of gradient
  - recent 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

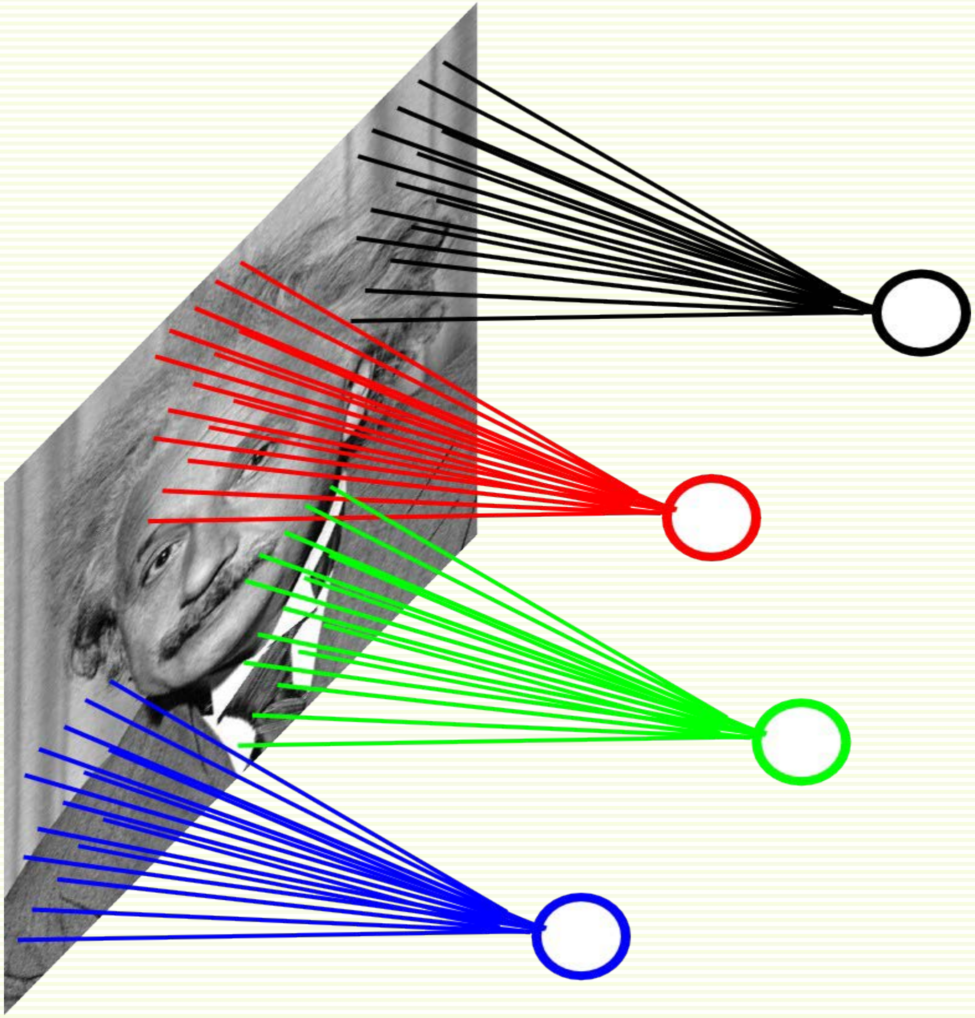
Convolutional Network: Motivation

- 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
- Do not waste resources by connecting unrelated pixels



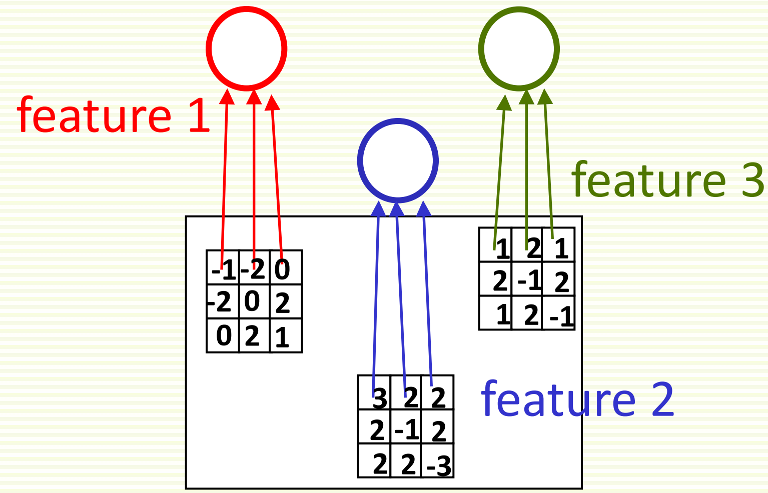
Convolutional Network: Motivation

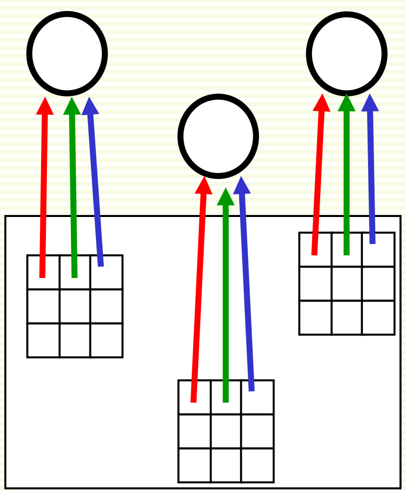
- Connect only pixels in a local patch, say 10x10
- For 200 by 200 image, 10^2 connections to one hidden unit
- For 10^5 hidden units → 10^7 connections
  - contrast with $4x10^9$ for fully connected layer
  - factor of 400 decrease



Convolutional Network: Motivation

- Intuitively, each neuron learns a good feature (a filter) in one particular location
- If a feature is useful in one image location, it should be useful in all other locations
  - *stationarity*: statistics is similar at different locations
- Idea: make all neurons detect **the same** feature at different positions
  - i.e. share parameters (network weights) across different locations
  - bias is usually not shared
  - greatly reduces the number of tunable parameters to learn

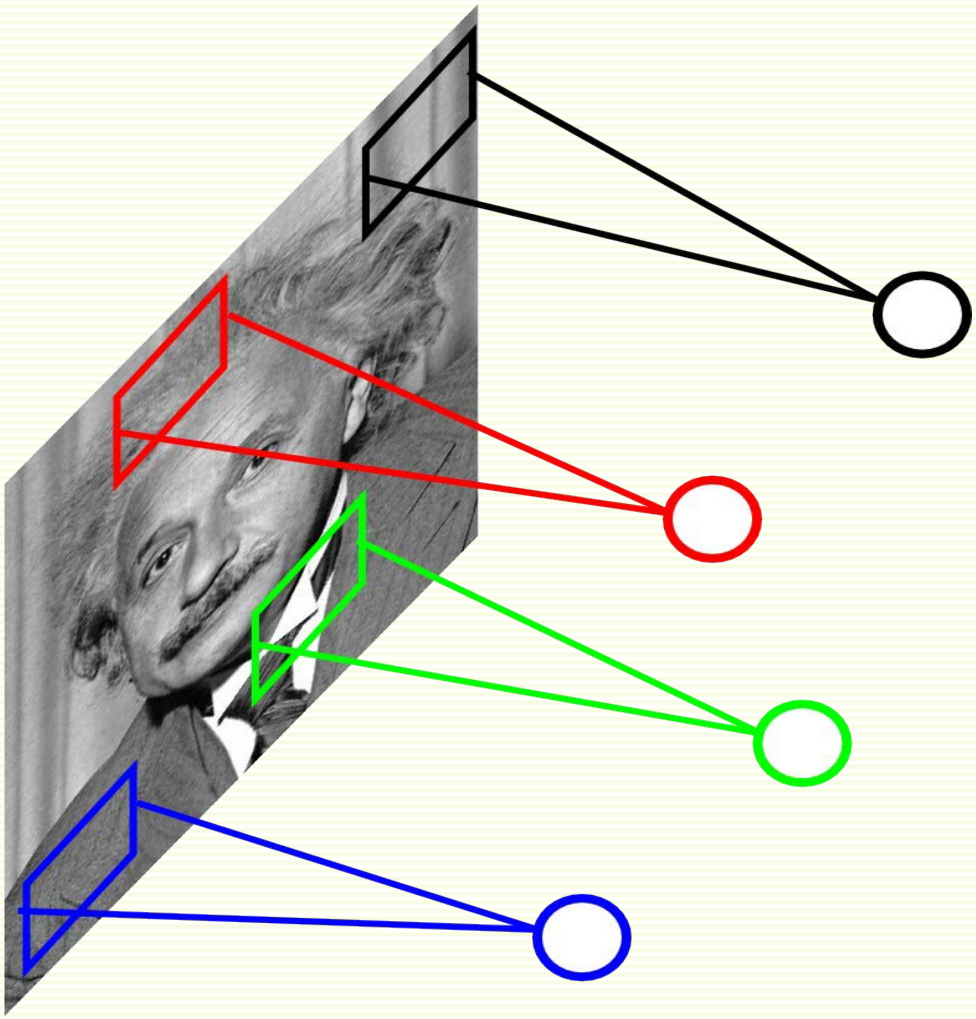




red connections have equal weight
green connections have equal weight
blue connections have equal weight

ConvNets: Weight Sharing

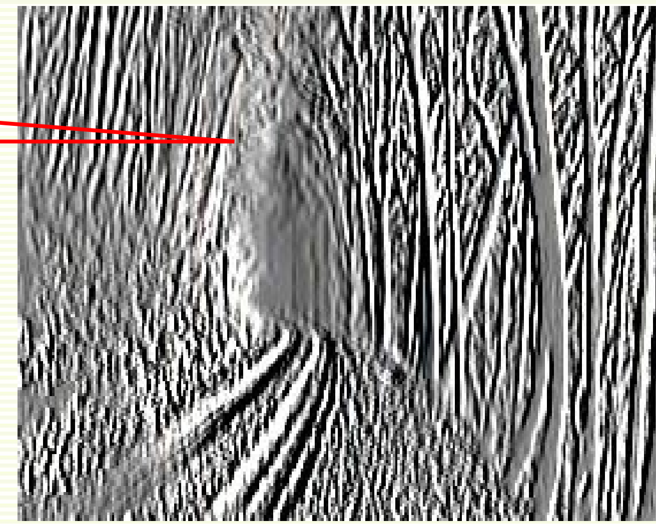
- Much fewer parameters to learn
- For 10^5 hidden units and 10x10 patch
  - 10^7 parameters to learn without sharing
  - 10^2 parameters to learn with sharing



Filtering via Convolution Recap

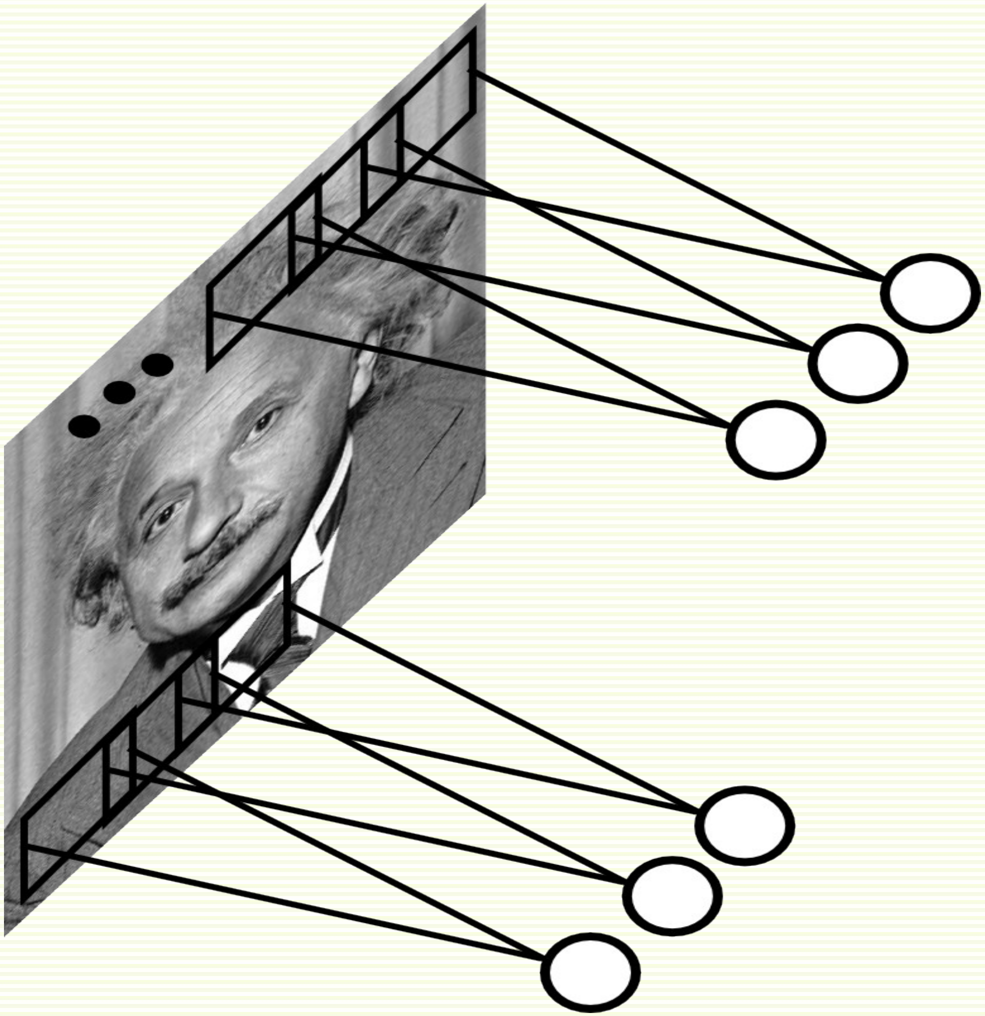
- Recall filtering with convolution for feature extraction

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

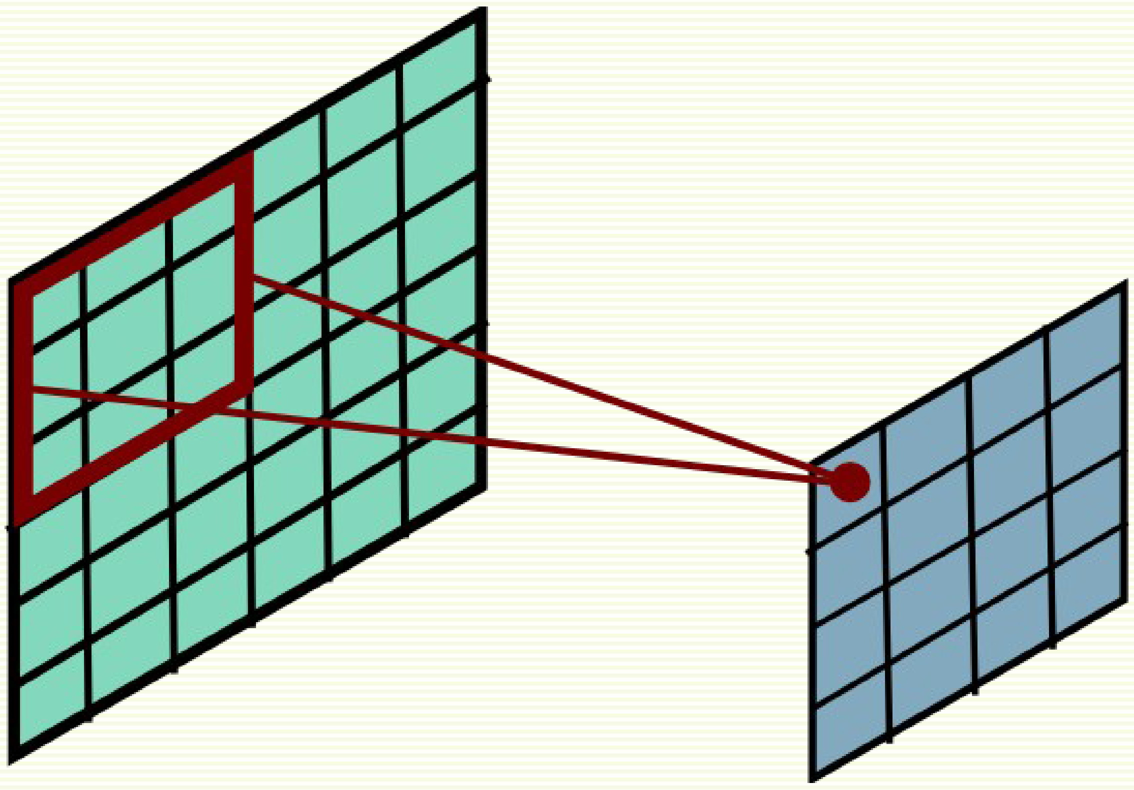


Convolutional Layer

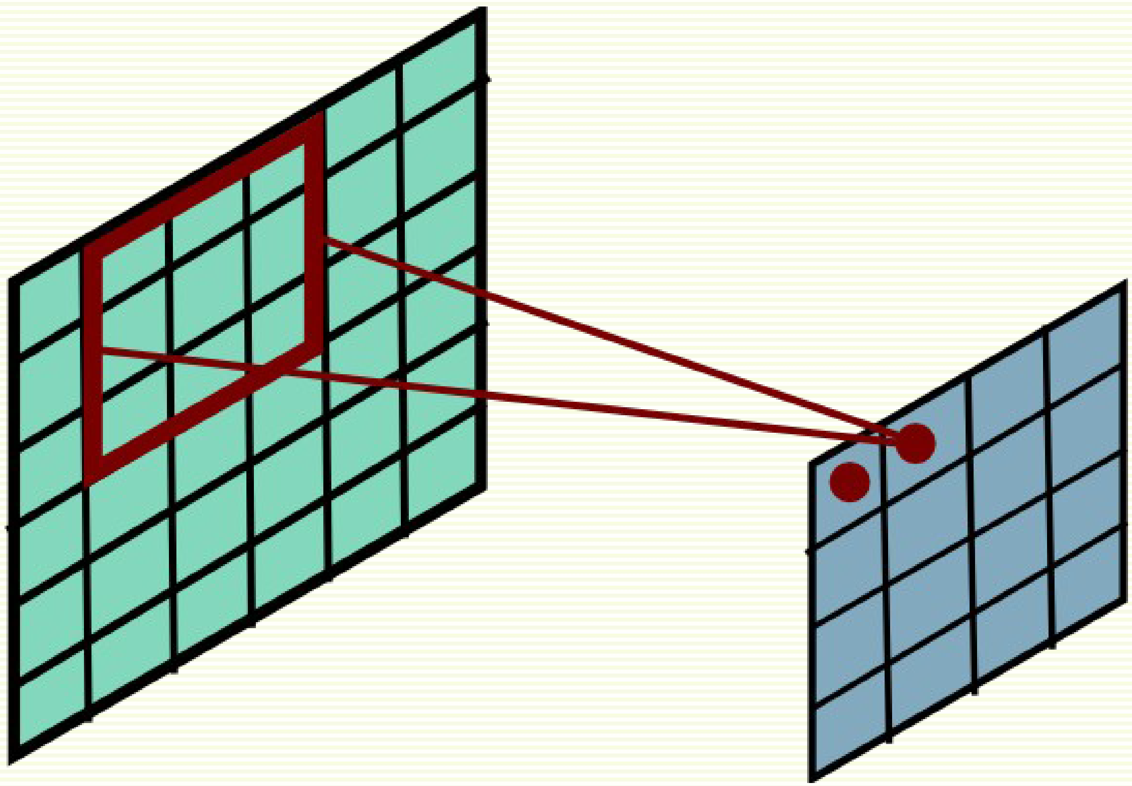
- Note similarity to convolution with some fixed filter
- But here the filter is learned



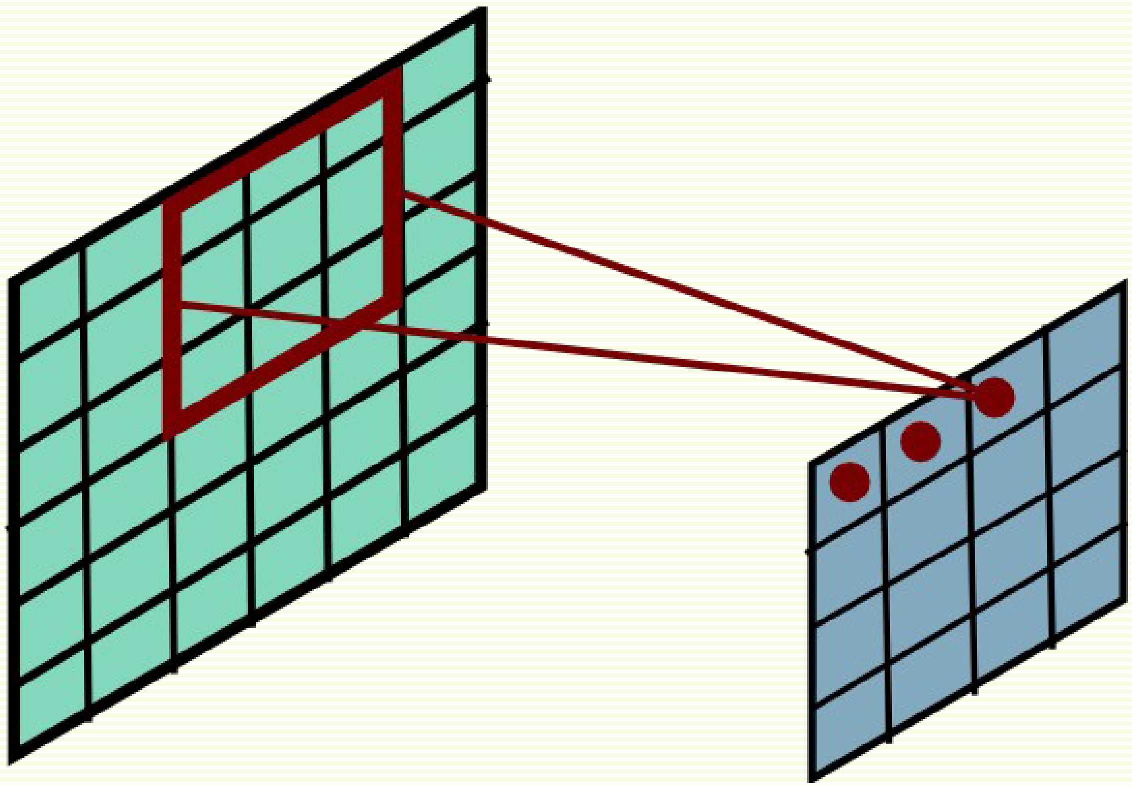
Convolutional Layer



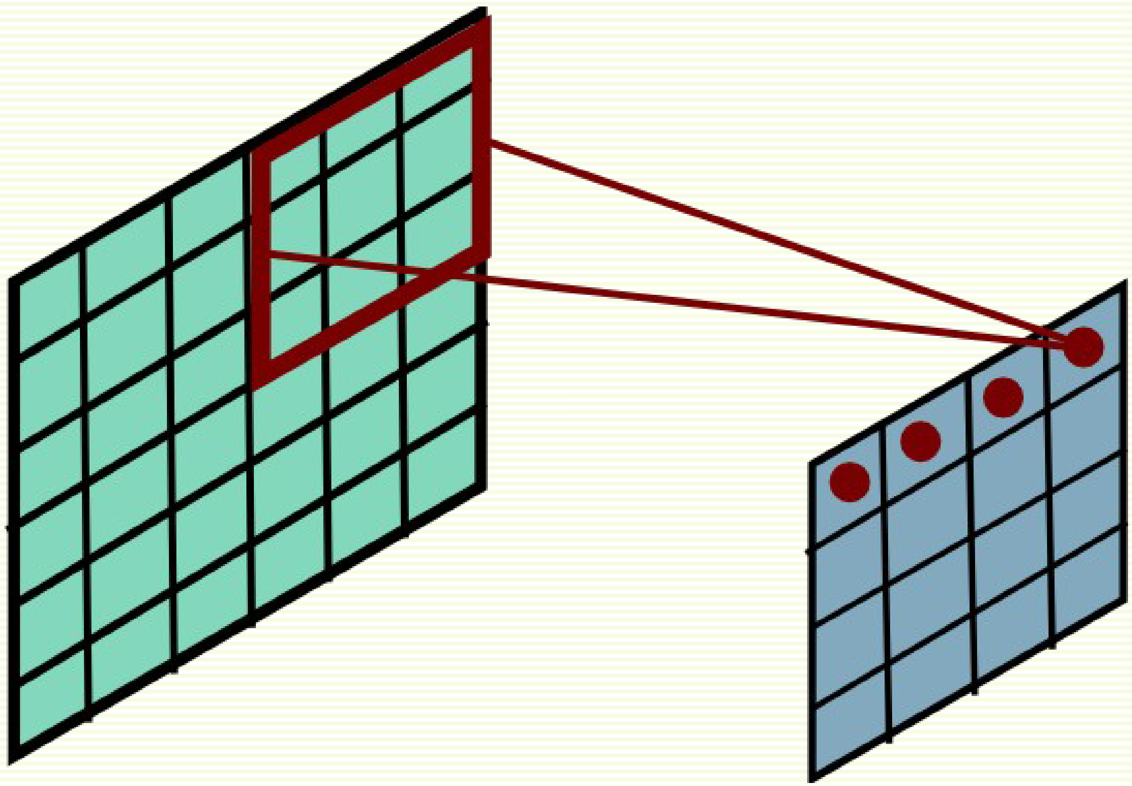
Convolutional Layer



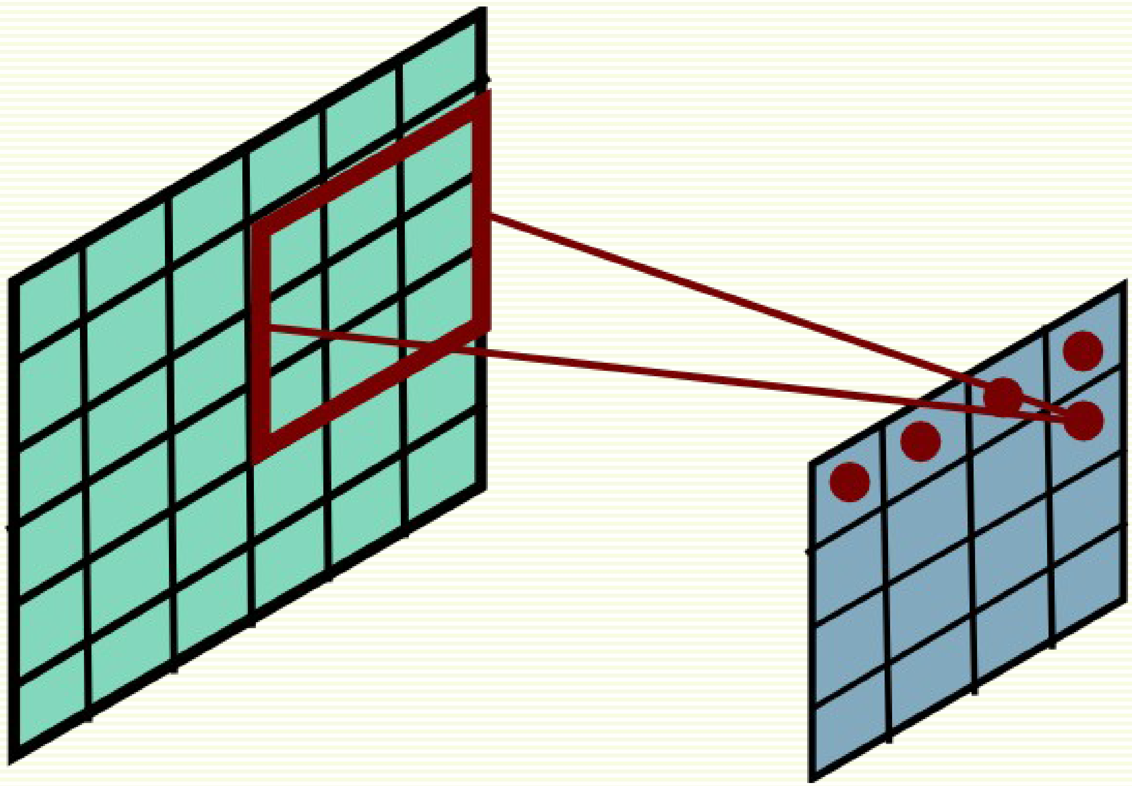
Convolutional Layer



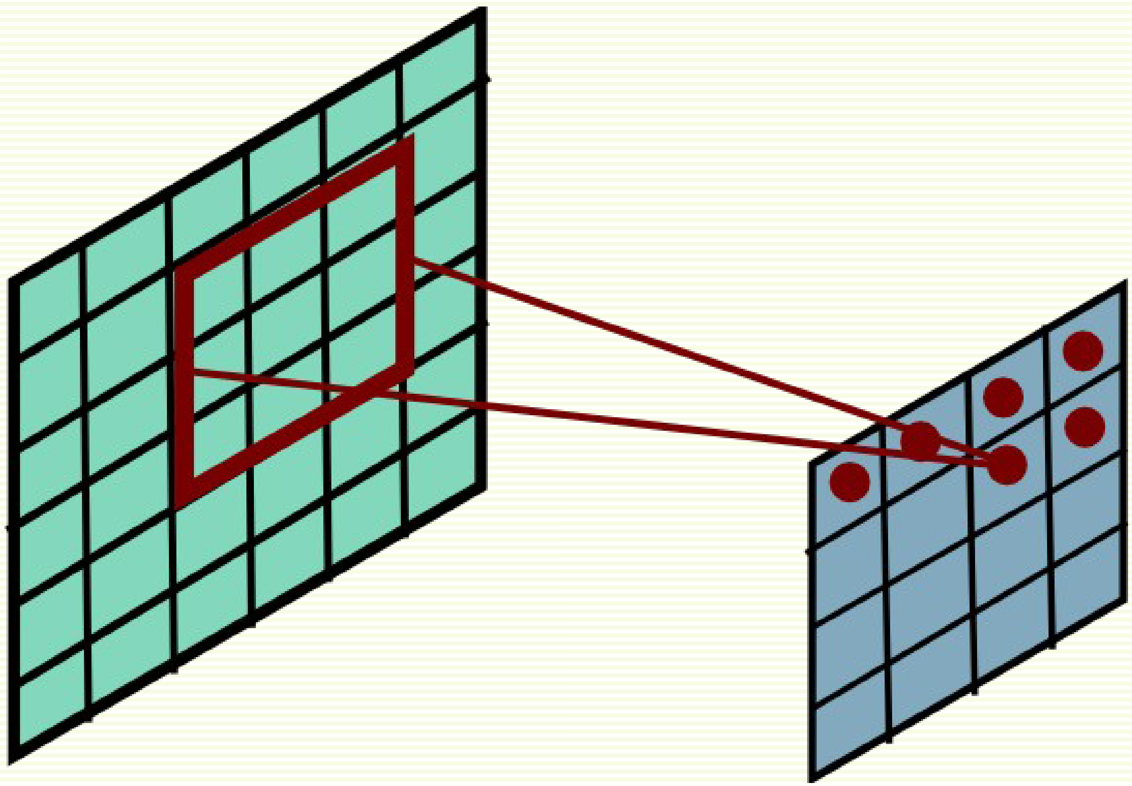
Convolutional Layer



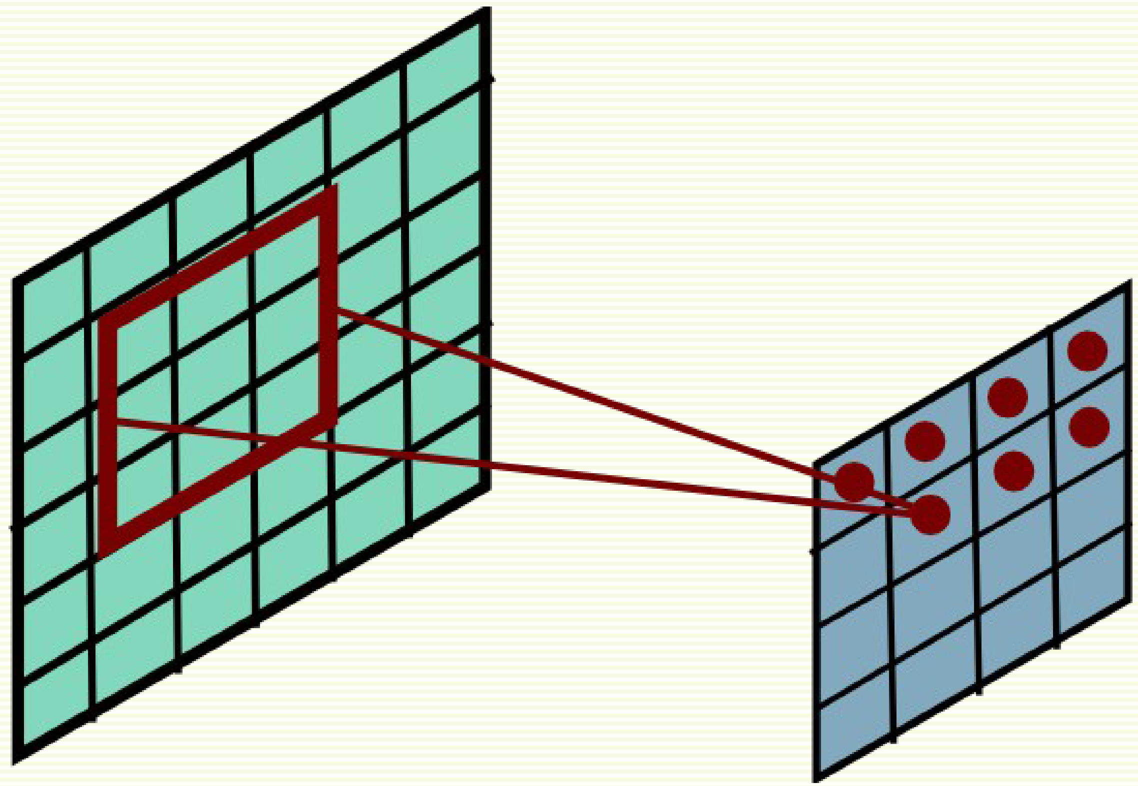
Convolutional Layer



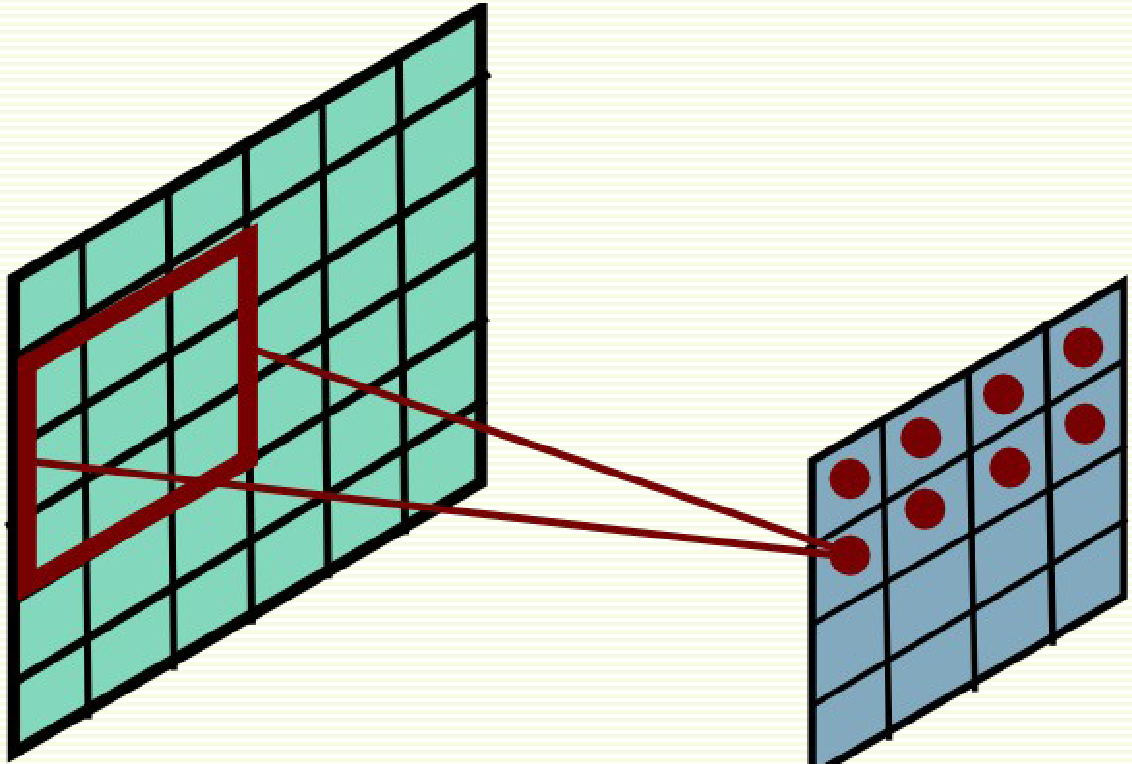
Convolutional Layer



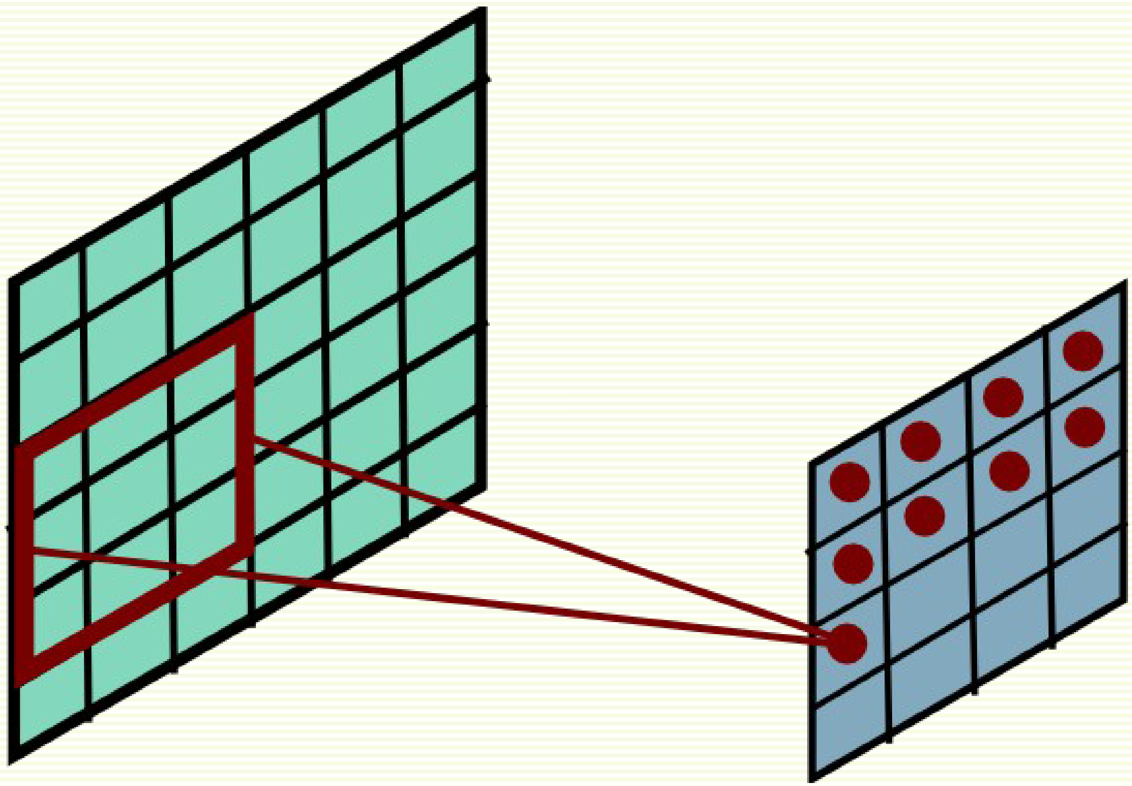
Convolutional Layer



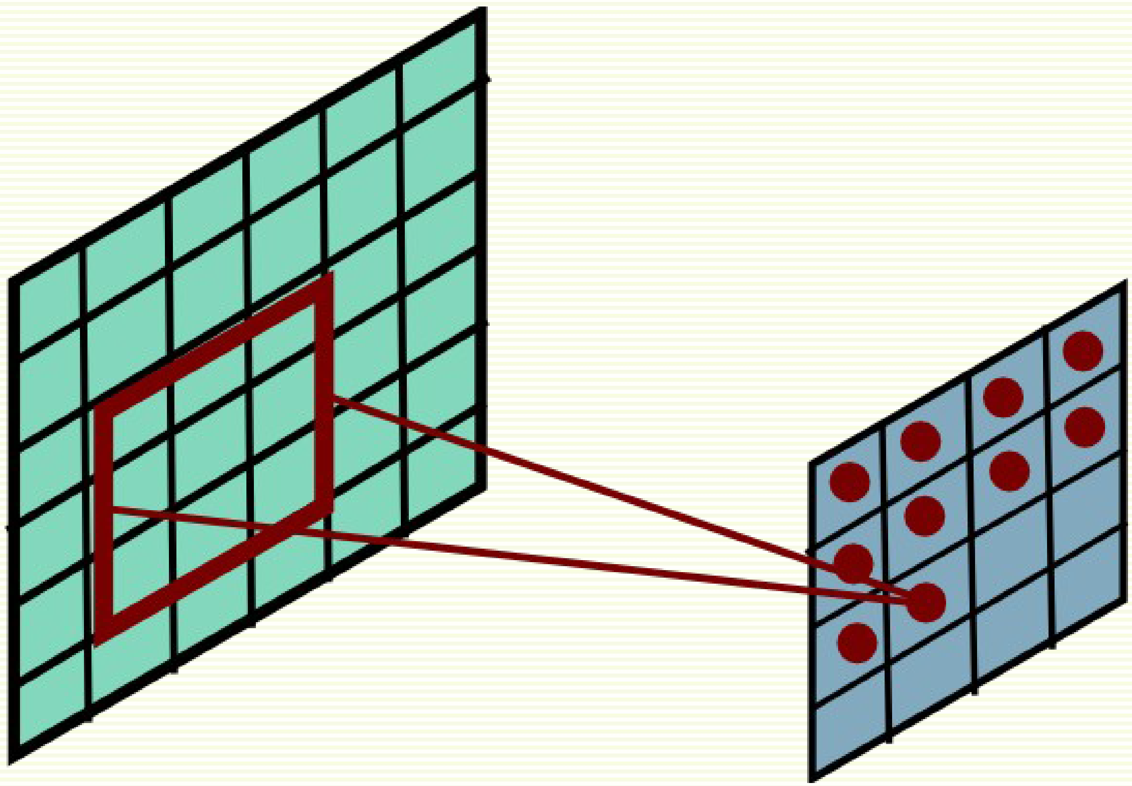
Convolutional Layer



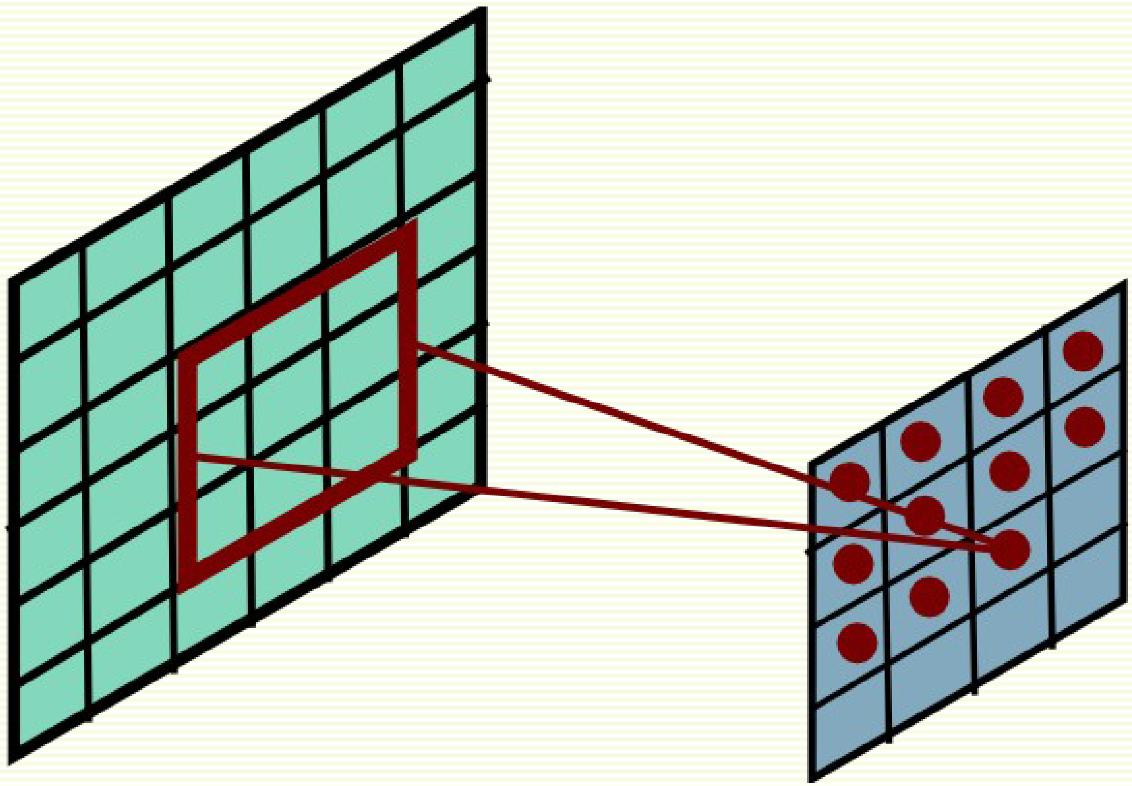
Convolutional Layer



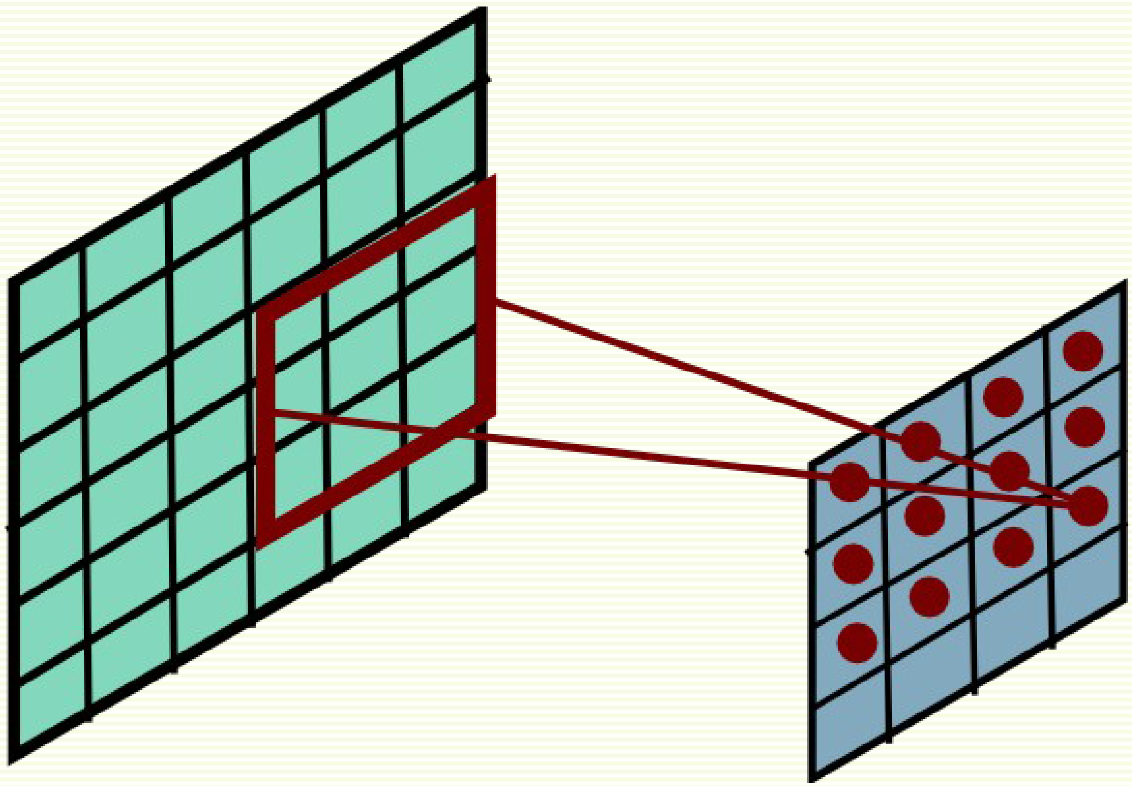
Convolutional Layer



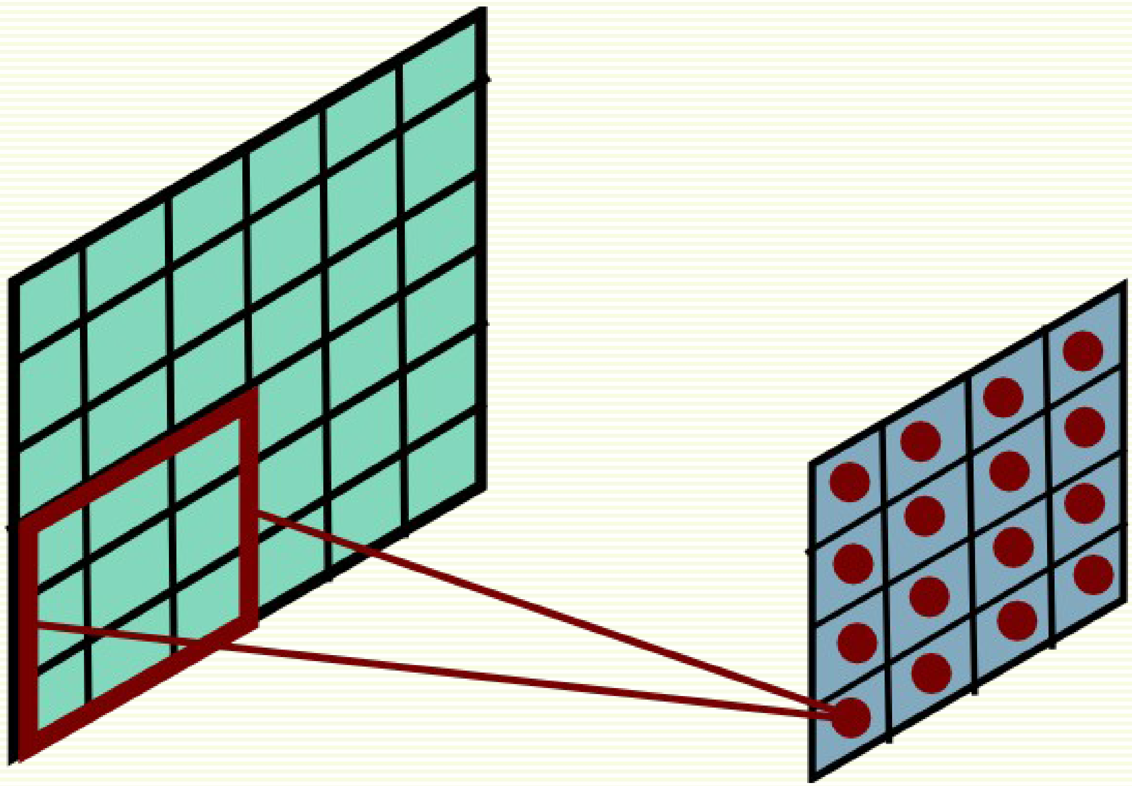
Convolutional Layer



Convolutional Layer

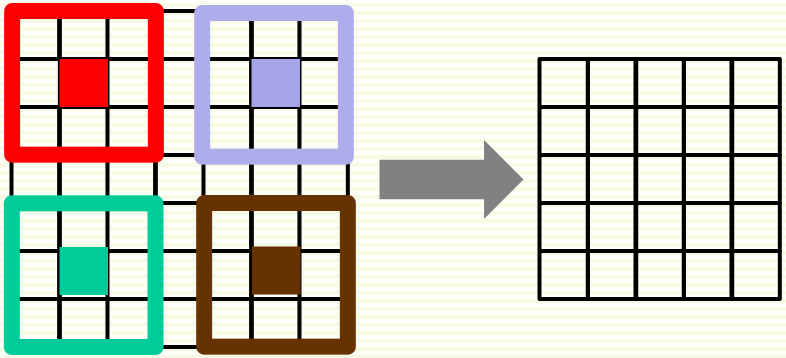


Convolutional Layer

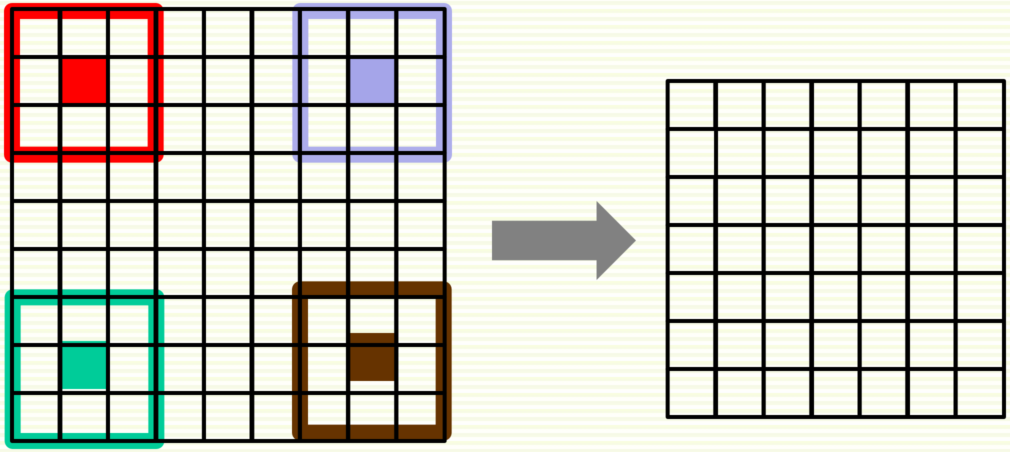


Convolutional Layer

- Output is usually slightly smaller because the borders of the image are left out

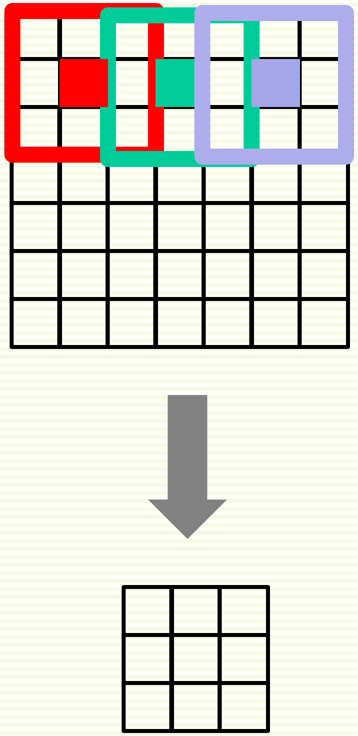


- If want output to be the same size, zero-pad the image appropriately



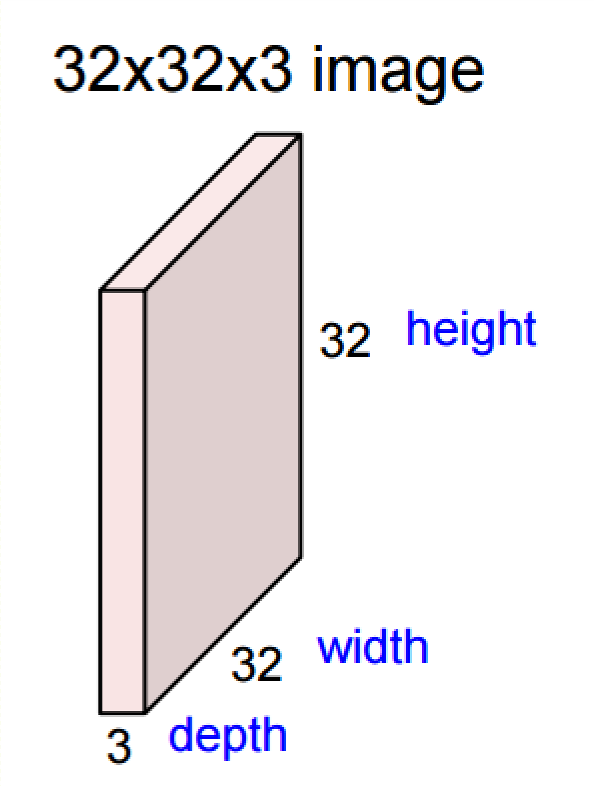
Convolutional Layer

- 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
    - image not zero-padded in this example



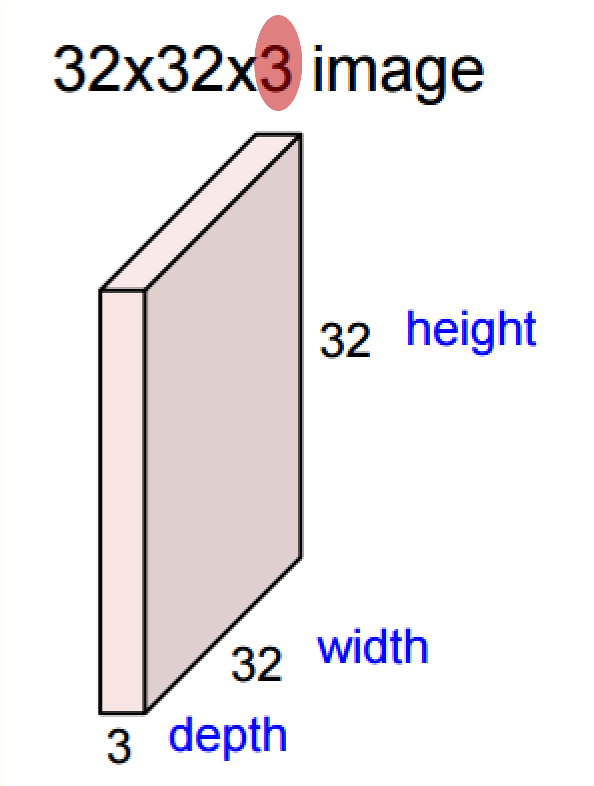
Convolutional Layer

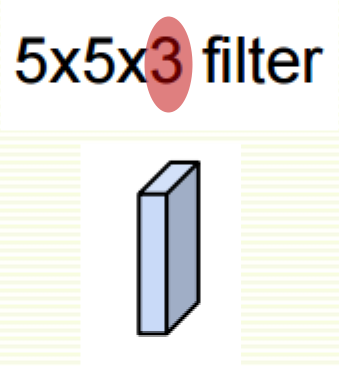
- Input image is usually color, has 3 channels or depth 3



Convolutional Layer

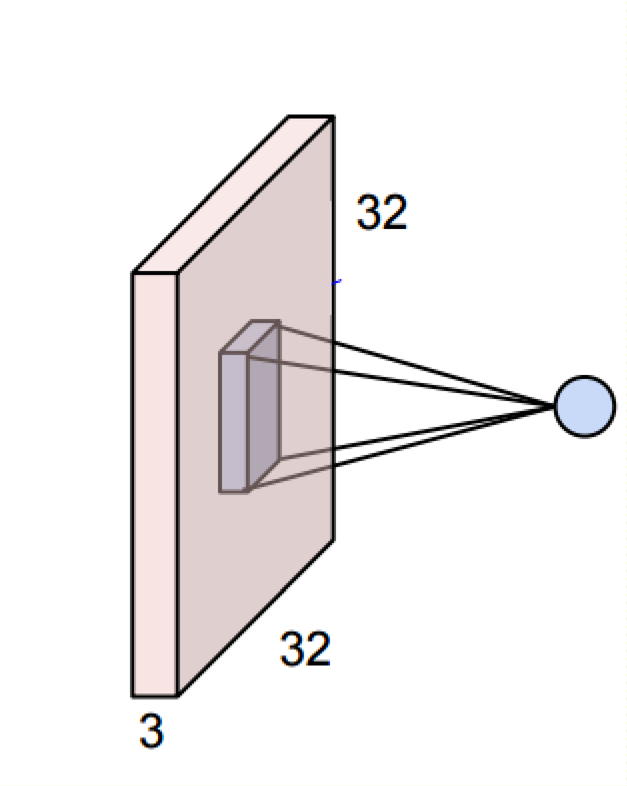
- Convolve 3D image with 3D filter





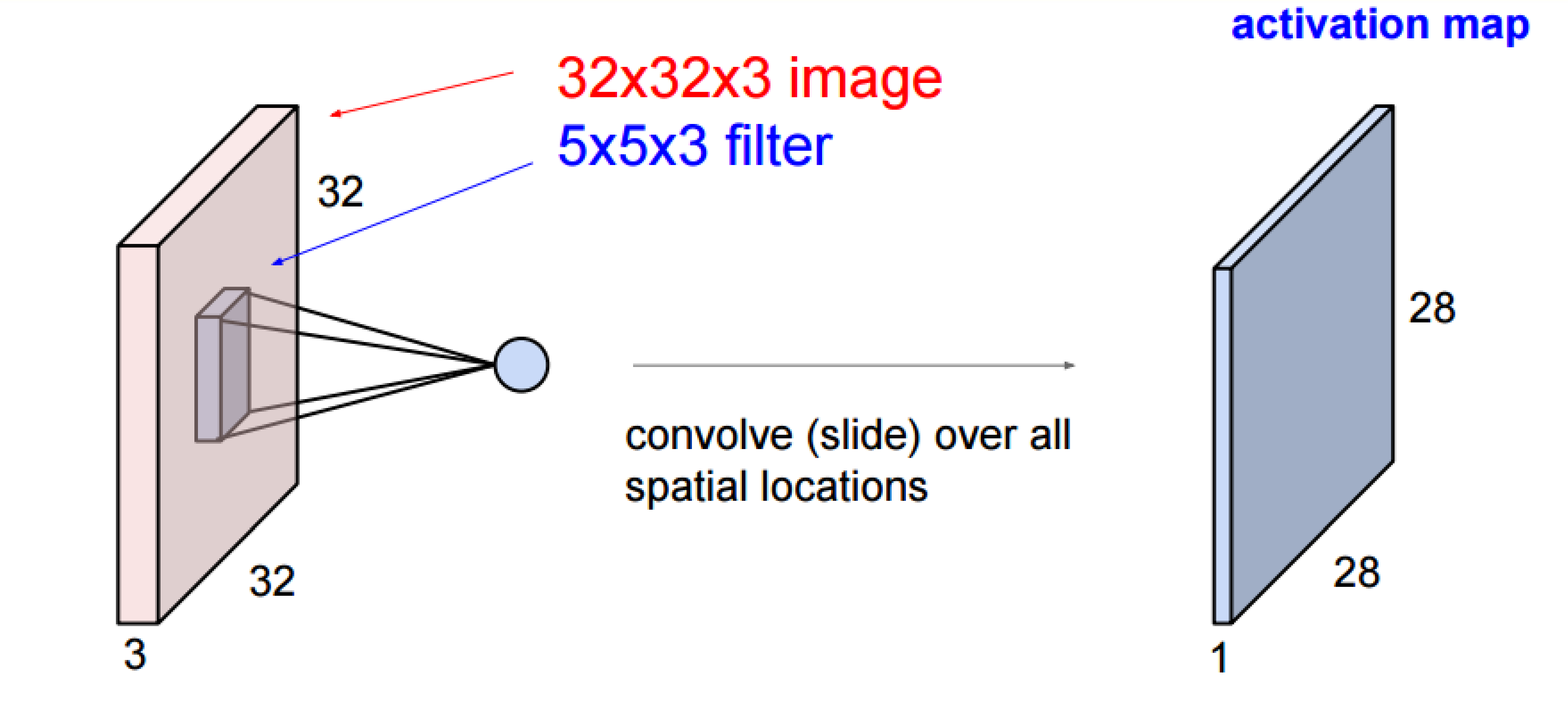
Convolutional Layer

- One convolution step is a 75 dimensional dot product between the 5x5x3 filter and a piece of image of size 5x5x3
- Can be expressed as $\mathbf{w}^t\mathbf{x}$, 75 parameters to learn ($\mathbf{w}$)
- Can add bias $\mathbf{w}^t\mathbf{x} + b$, 76 parameters to learn ($\mathbf{w}$,b)



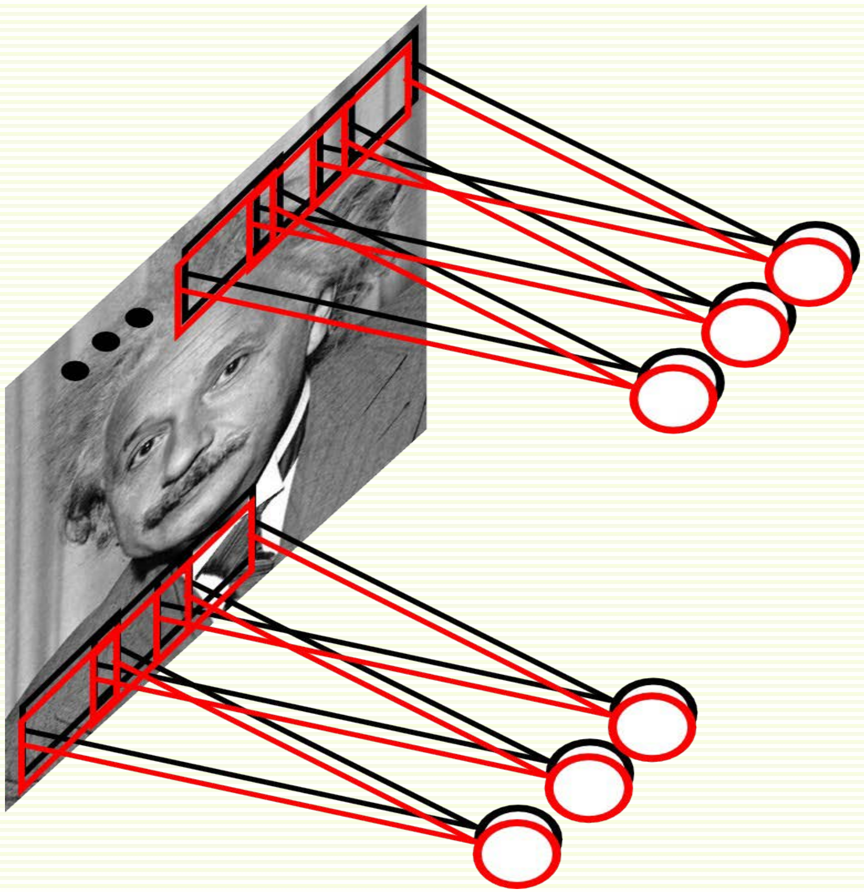
Convolutional Layer

- Convolve 3D image with 3D filter
  - result is a 28x28x1 activation map, no zero padding used
  - 76 parameters to learn



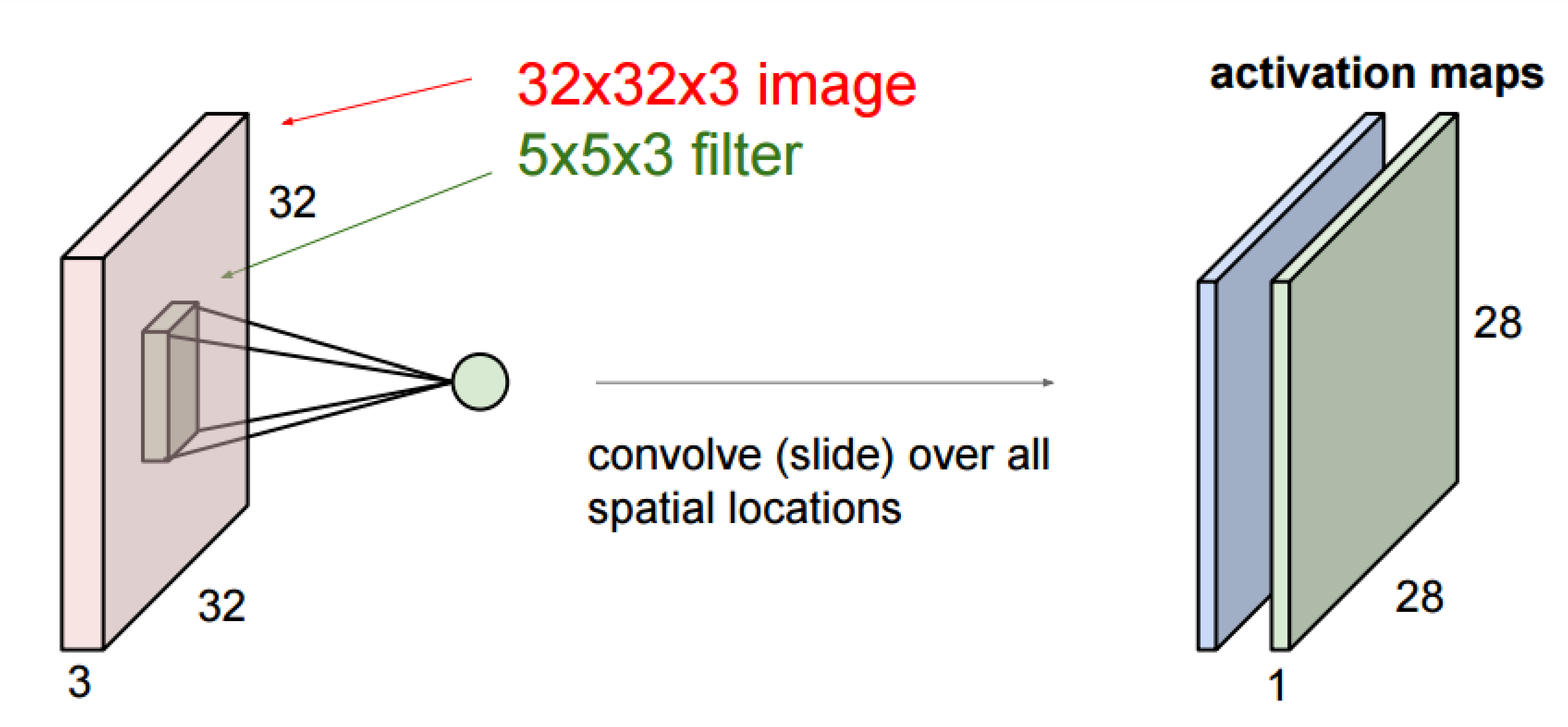
Convolutional Layer

- 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



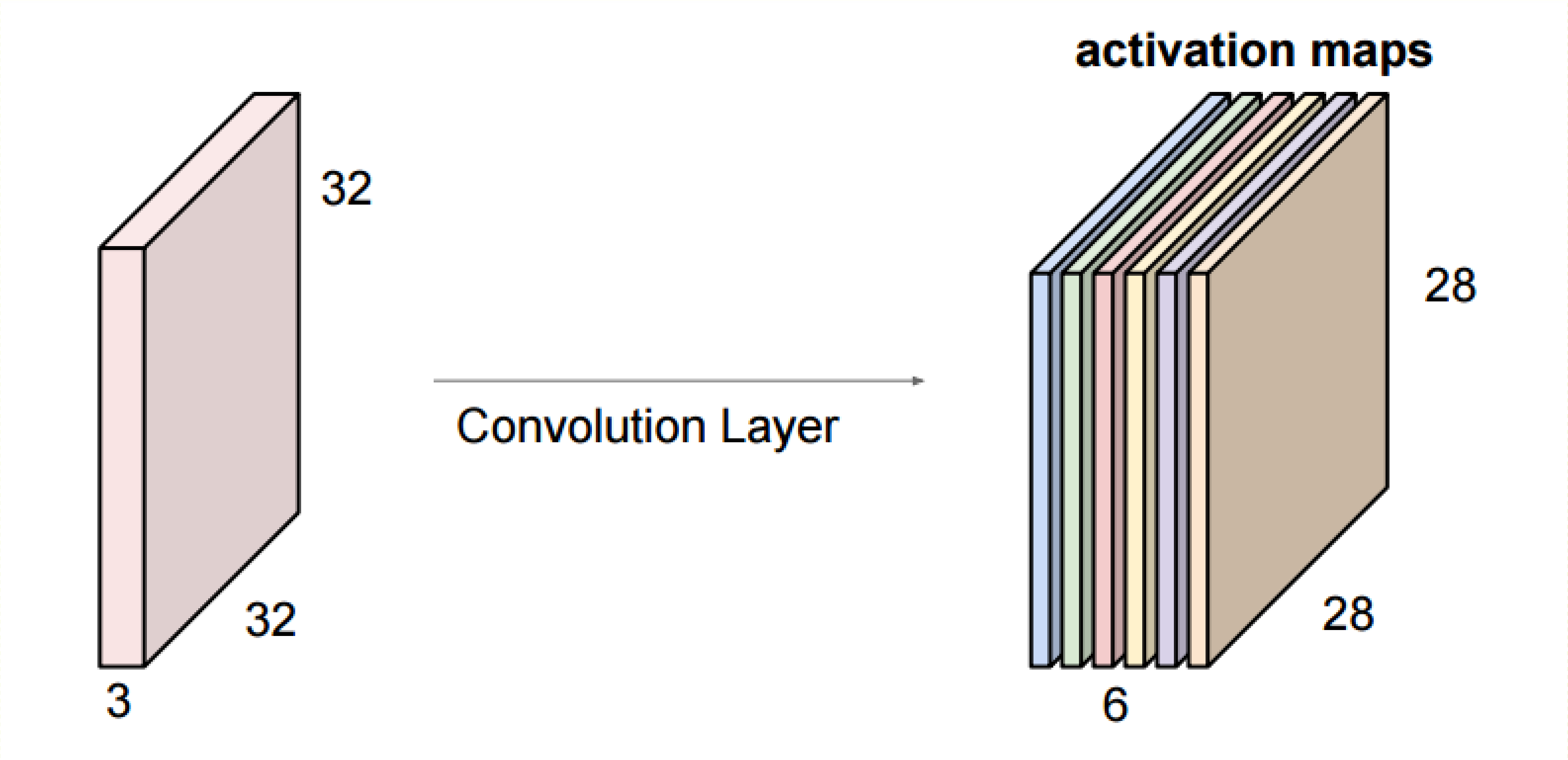
Convolutional Layer

- Consider a second, green filter



Convolutional Layer

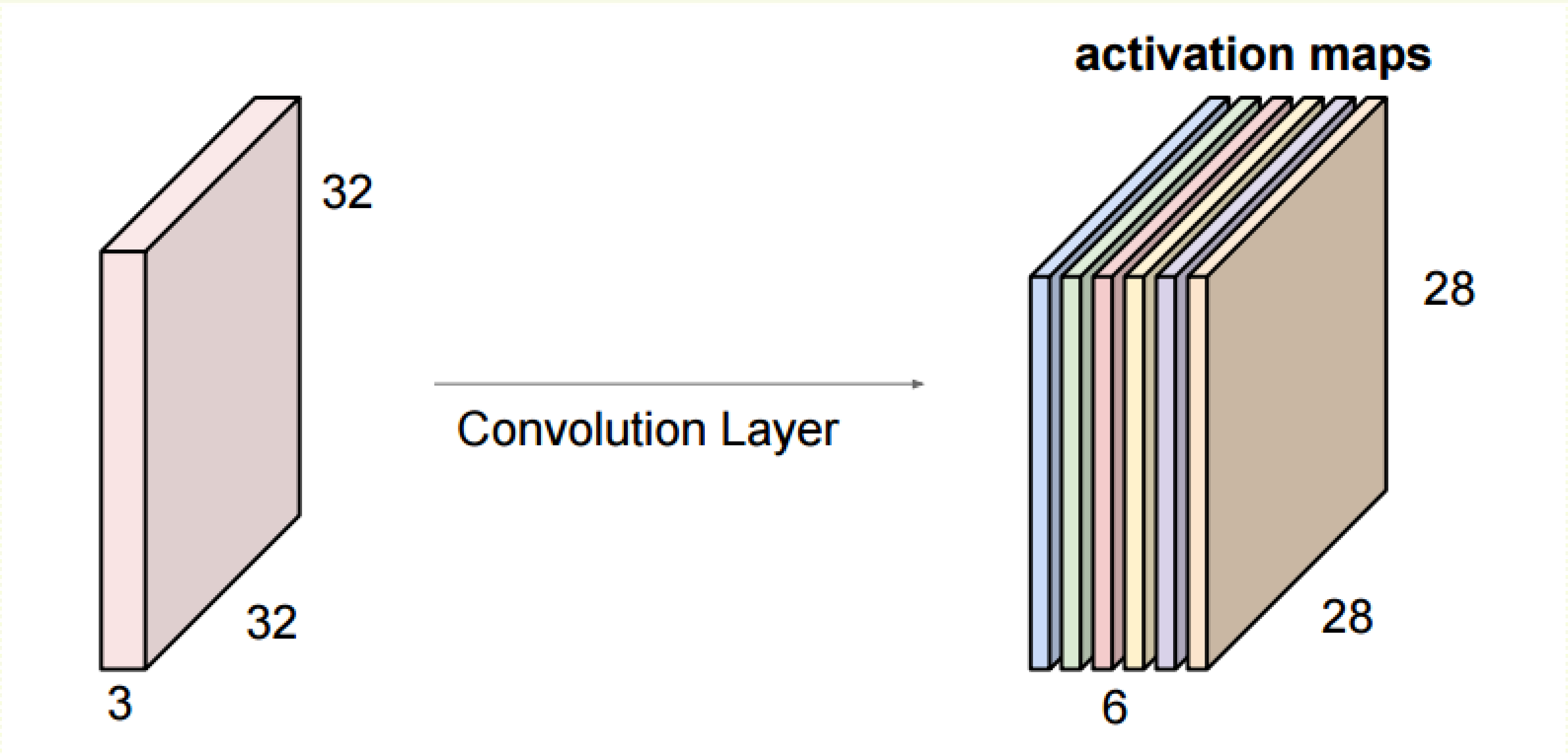
- If have 6 filters (each of size 5x5x3) get 6 activation maps, 28x28 each



- Stack them to get a new 28x28x6 “image”
- 76x6 = 456 parameters to learn

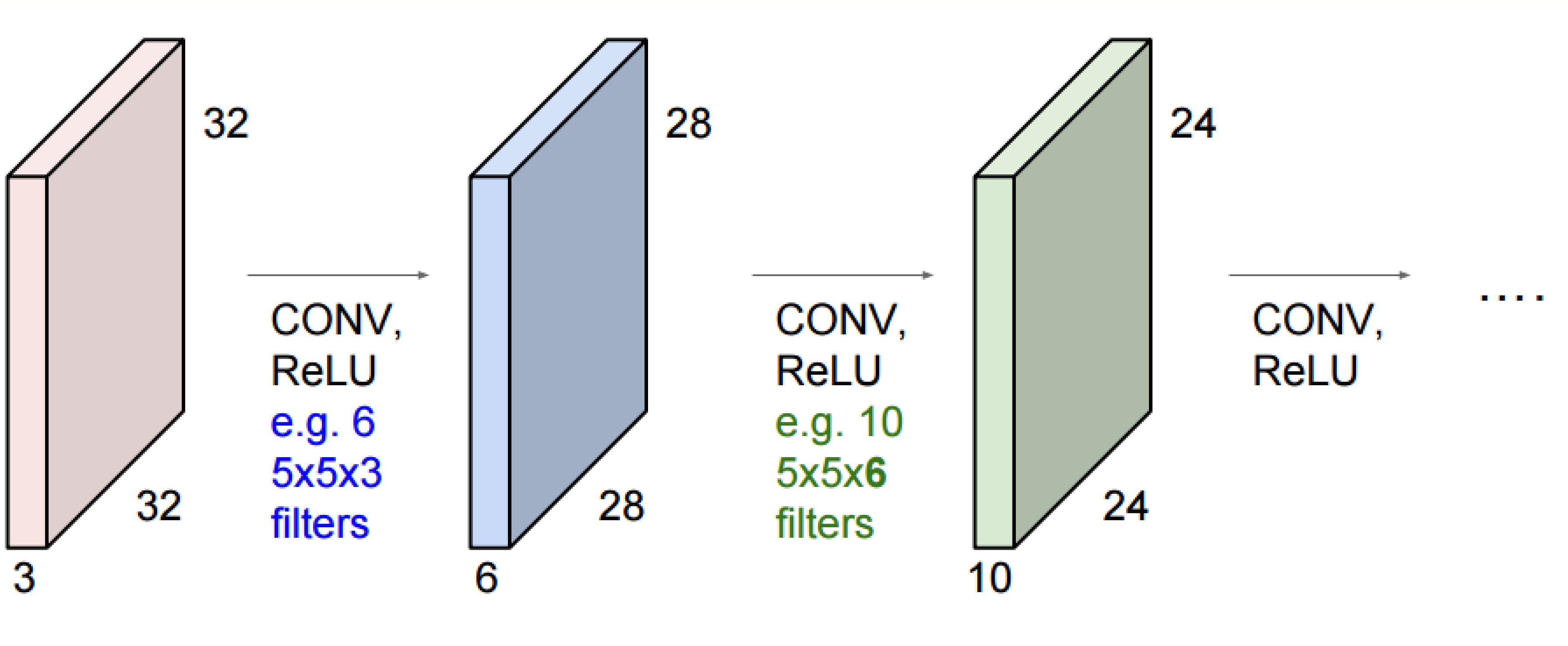
Convolutional Layer

- Apply activation function (say ReLu) to the activation map



Several Convolution Layers

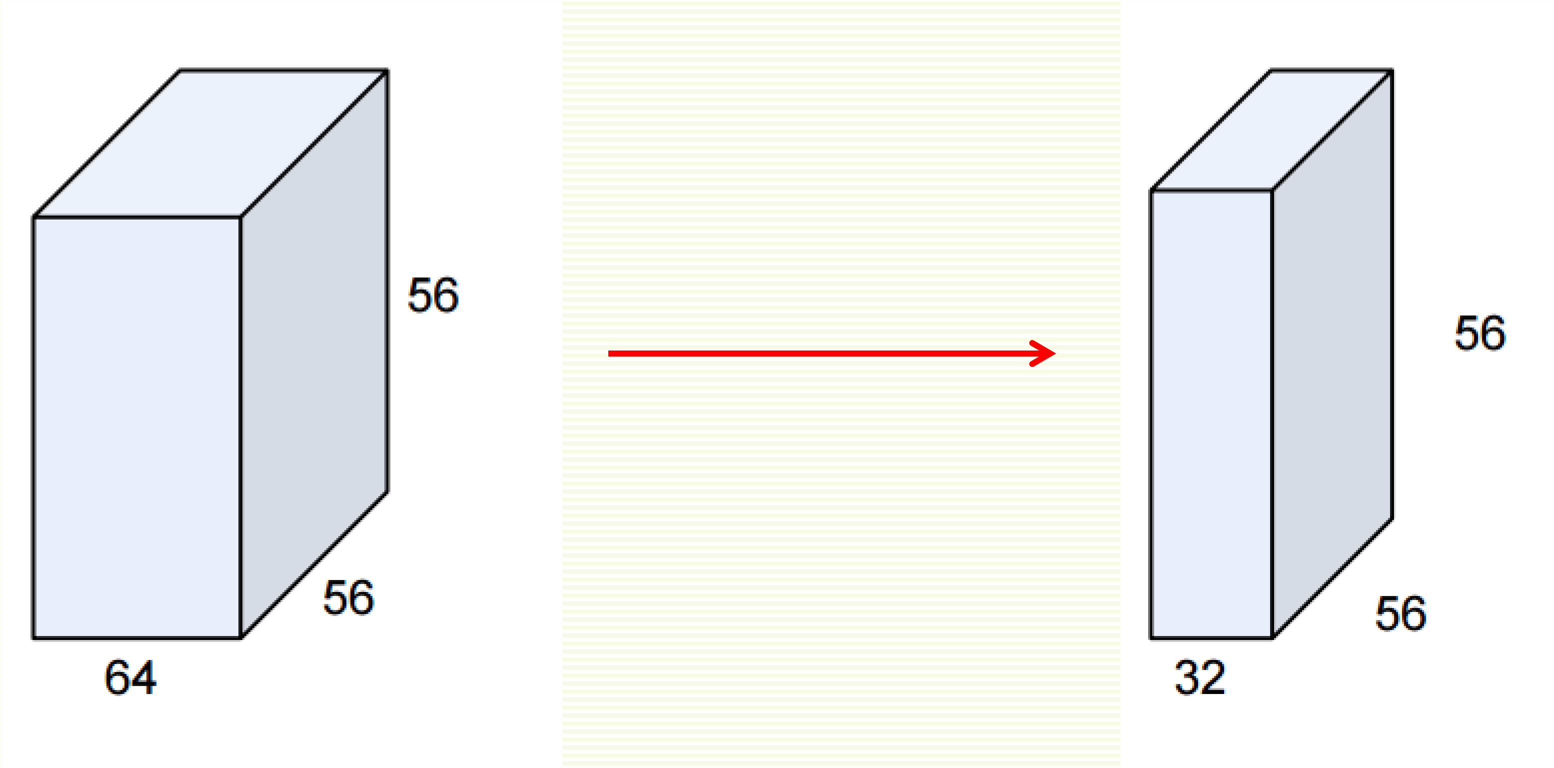
- Construct a sequence of convolution layers interspersed with activation functions



- Use zero-padding if don't want output layers to shrink

Convolutional Layer

- 1x1 convolutions make perfect sense
- Example
  - Input image of size 56x56x64
  - Convolve with 32 filters, each of size 1x1x64



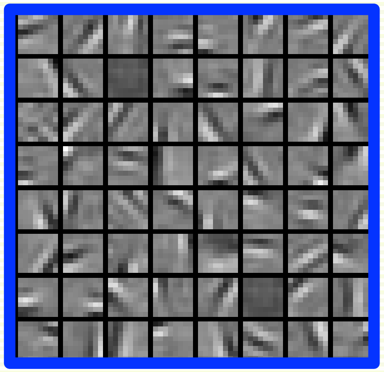
Weight Sharing Constraints

- 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 $\frac{\partial L}{\partial w_1}$ to update $\mathbf{w_1}$ and $\frac{\partial L}{\partial w_2}$ to update $\mathbf{w_2}$
- Now use $\frac{\partial L}{\partial w_1} + \frac{\partial L}{\partial w_2}$ to update $\mathbf{w_1}$ and $\mathbf{w_2}$, use

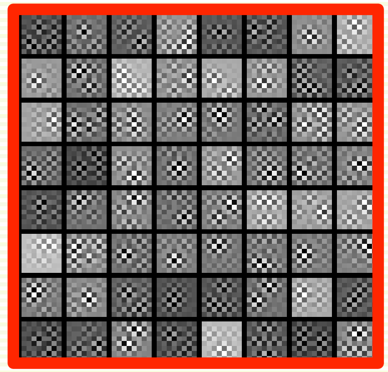
Check Learned Convolutions

- Good training: learned filters exhibit structure and are uncorrelated

GOOD

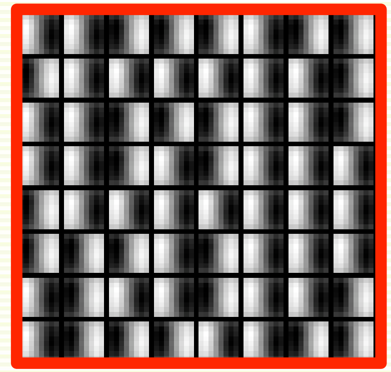


BAD



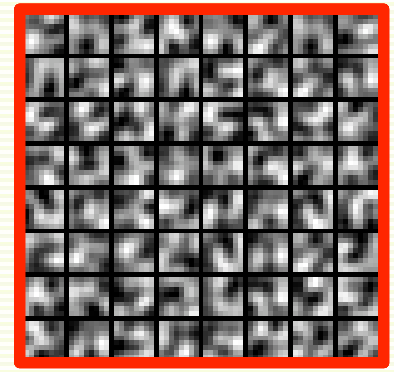
too noisy

BAD



too
correlated

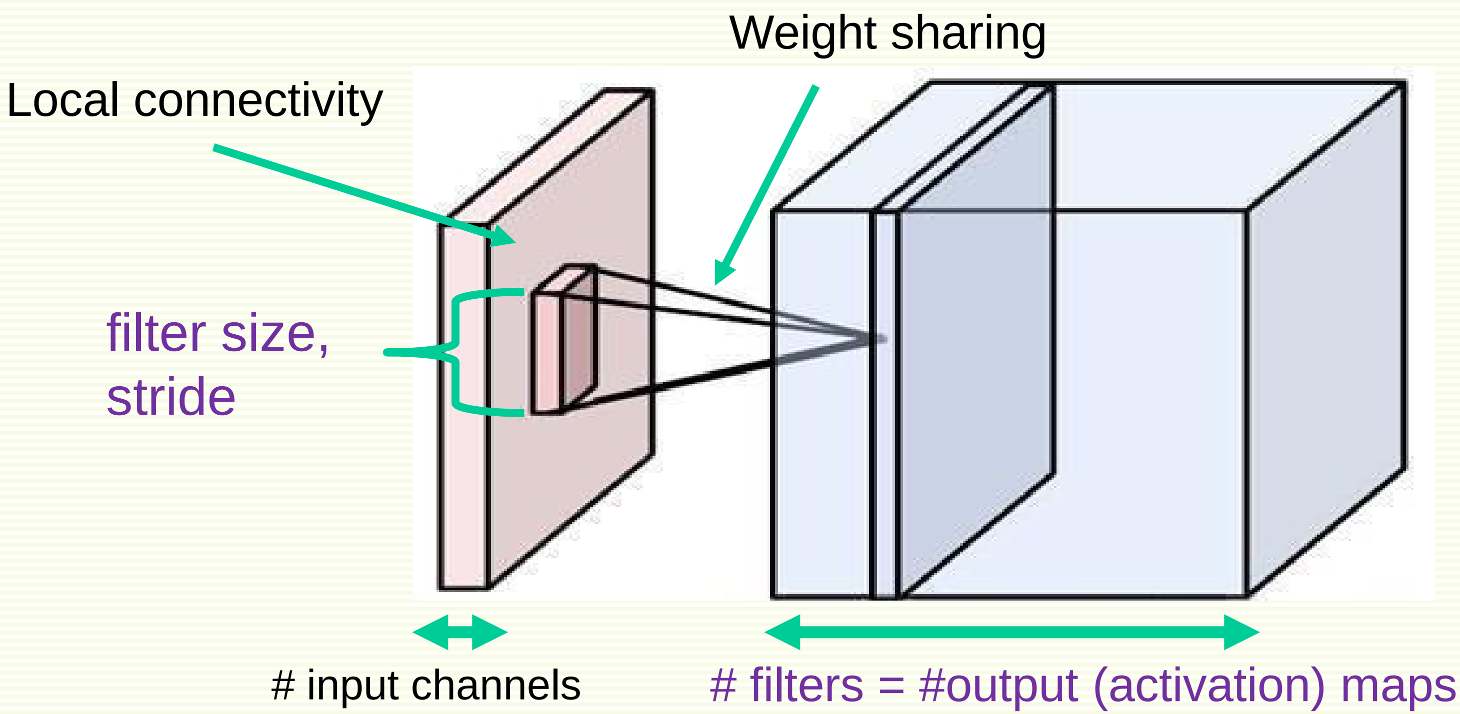
BAD



lack
structure

Convolutional Layer Summary

- Local connectivity
- Weight sharing
- Handling multiple input/output channels
- Retains location associations

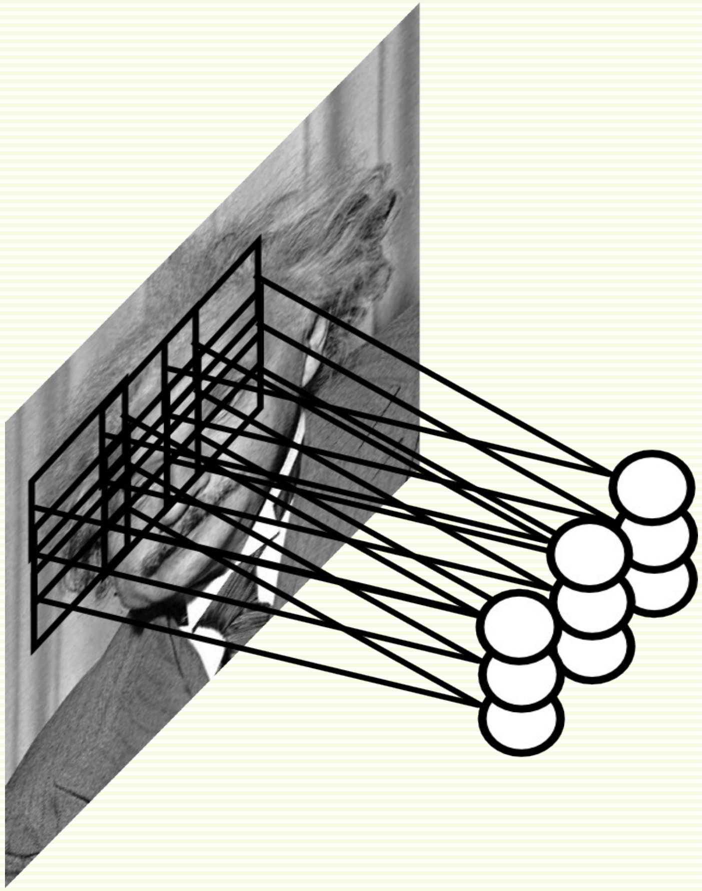


Convolutional Layer Summary

- Takes as input volume **W** x **H** x **D**
- Requires four hyperparameters
  - number of filters **K**
    - usually try powers of 2 (32, 64, 128, etc)
  - their spatial extent **F**
    - smaller size is more popular, 3, 5, 7
  - stride **S**
    - 1 or 2
  - amount of zero padding **P**
    - as fits
- Produces volume of size **W’** x **H’** x **D’** where
  - W’ = (**W** – **F** +2**P**)/**S** +1
  - **H’** = (**H** – **F** +2**P**)/**S** +1
  - **D’** = **K**
- With parameter sharing, introduces **F*F*D** weights per filter, for a total of (**F*F*D**)***K** weights and **K** biases

Pooling Layer

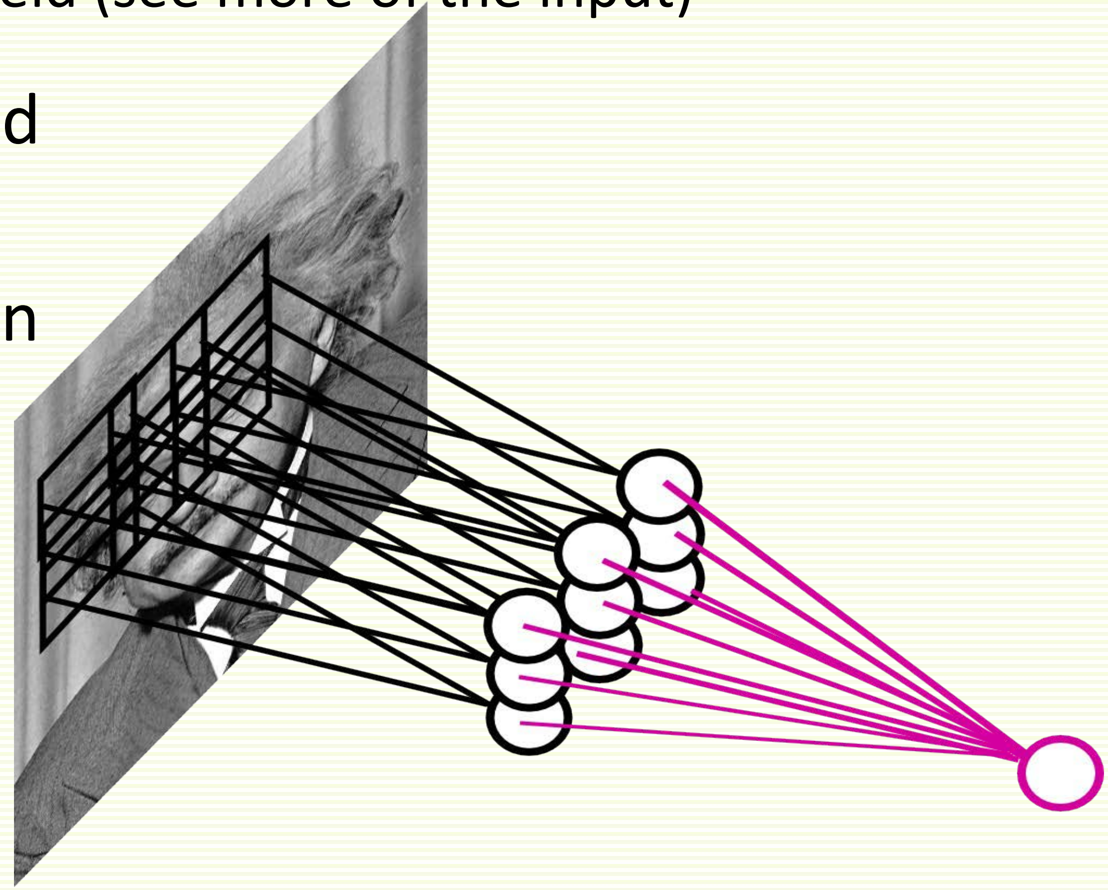
- Say a filter is an eye detector
- Want to detection to be robust to precise eye location



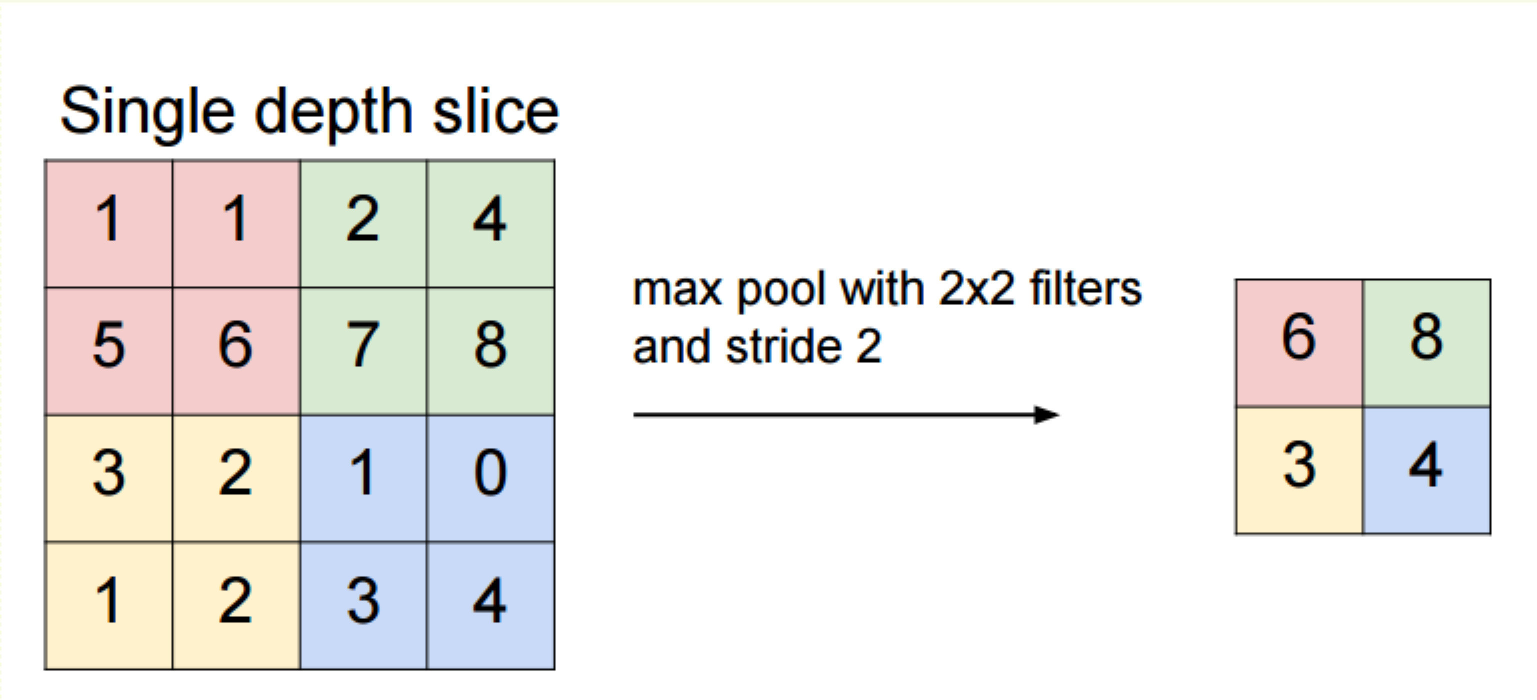
Pooling Layer

- *Pool* responses at different locations
  - by taking max, average, etc.
  - robustness to exact spatial location
  - also larger receptive field (see more of the input)

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

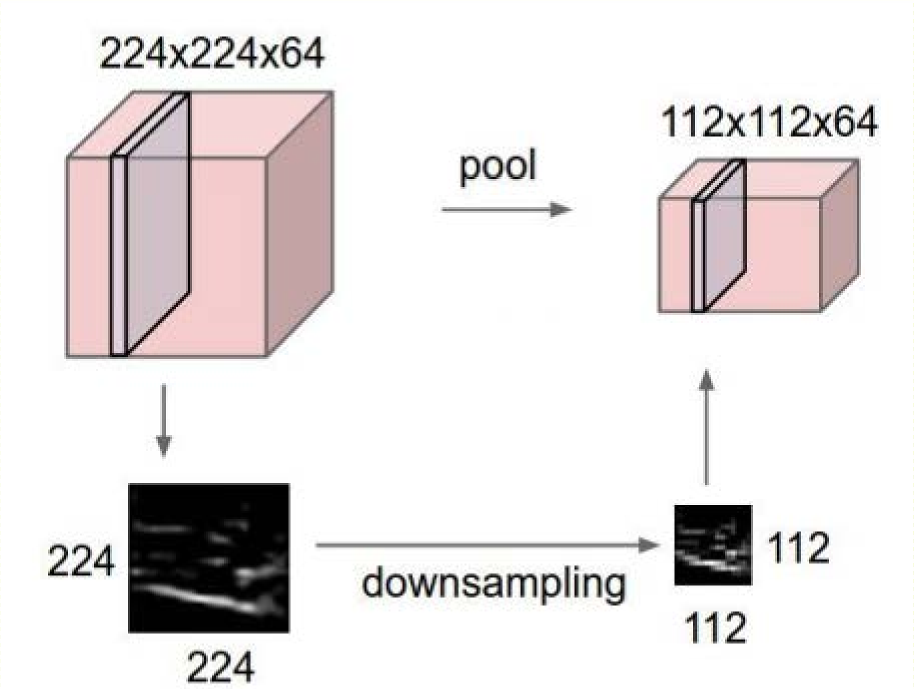


Pooling Layer: Max Pooling Example



Pooling Layer

- Pooling usually applied to each activation map separately



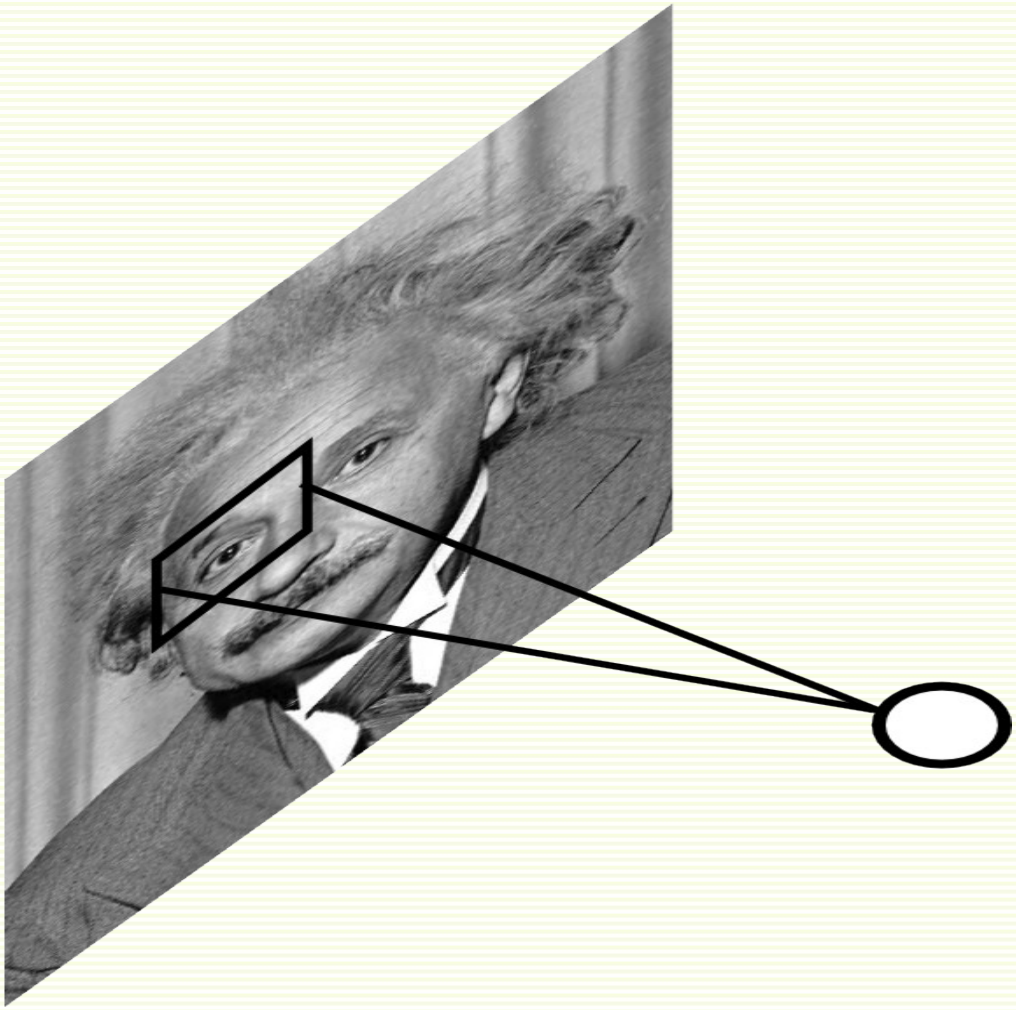
Pooling Layer Summary

- Takes volume of size **W** x **H** x **D**
- Introduces no parameters to learn
- Hyperparameters
  - stride **S**
    - common settings: 2
  - spatial extent **F**
    - common settings: 2,3
  - padding is not common to use with pooling
- Produces a volume of size **W'** x **H'** x **D'**
  - **W'** = (**W** - **F**)/**S** + 1
  - **H'** = (**H** - **F**)/**S**+1
  - **D'** = **D**

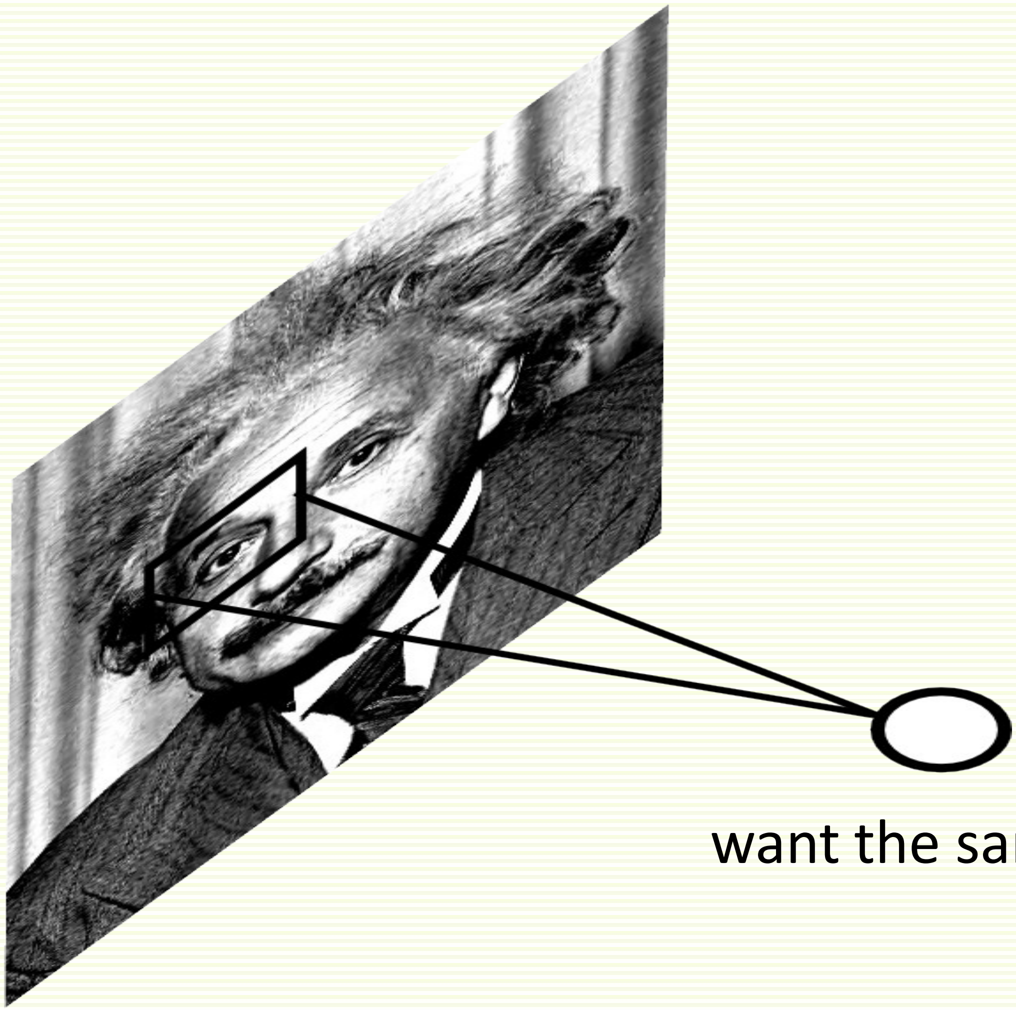
Issues with Pooling

- After several levels of pooling, we 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

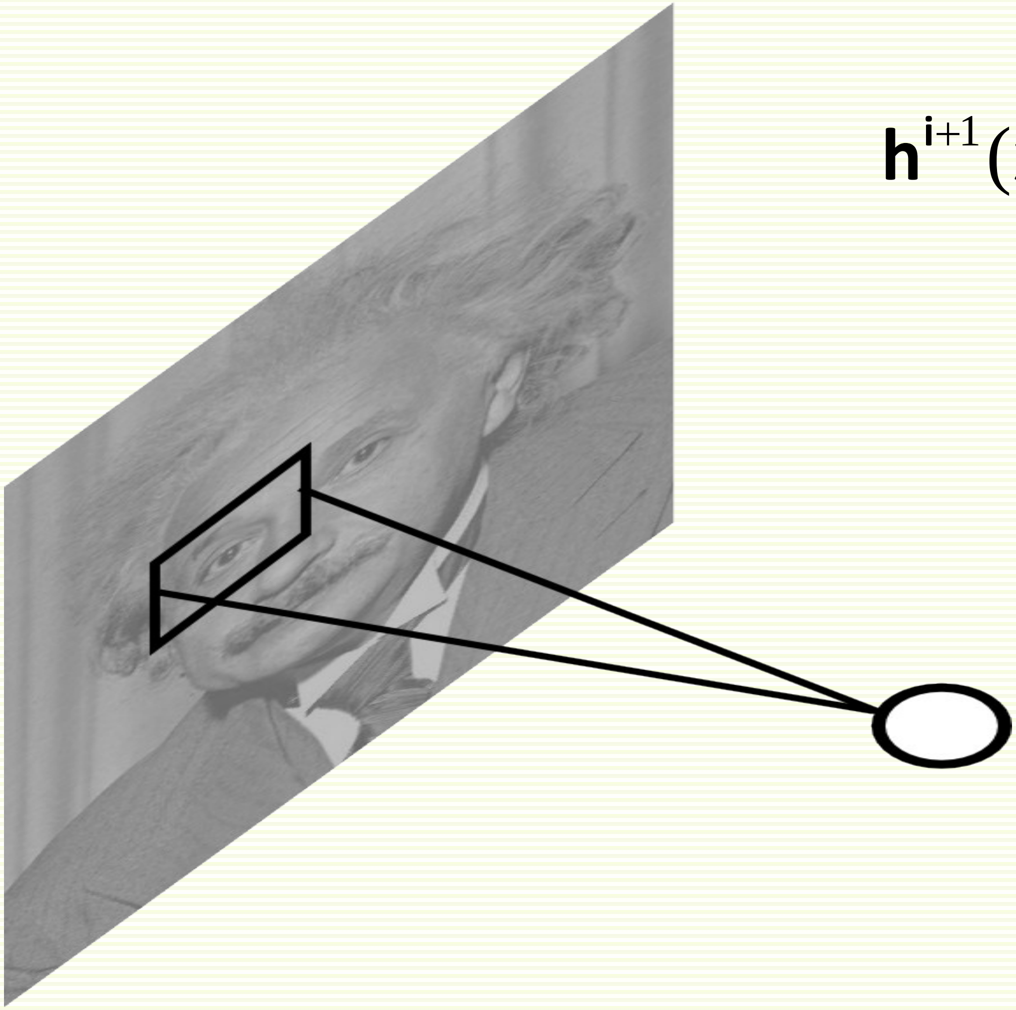


Local Contrast Normalization



want the same response

Local Contrast Normalization

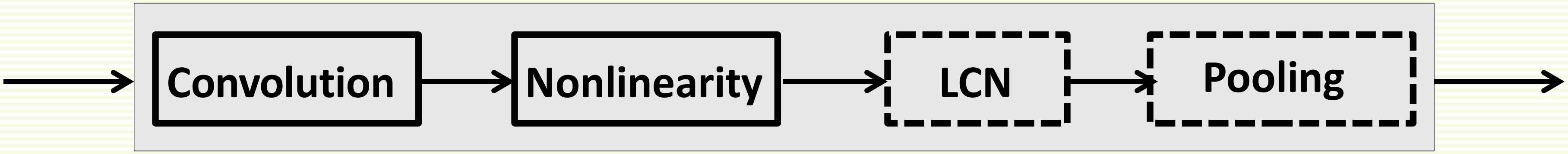


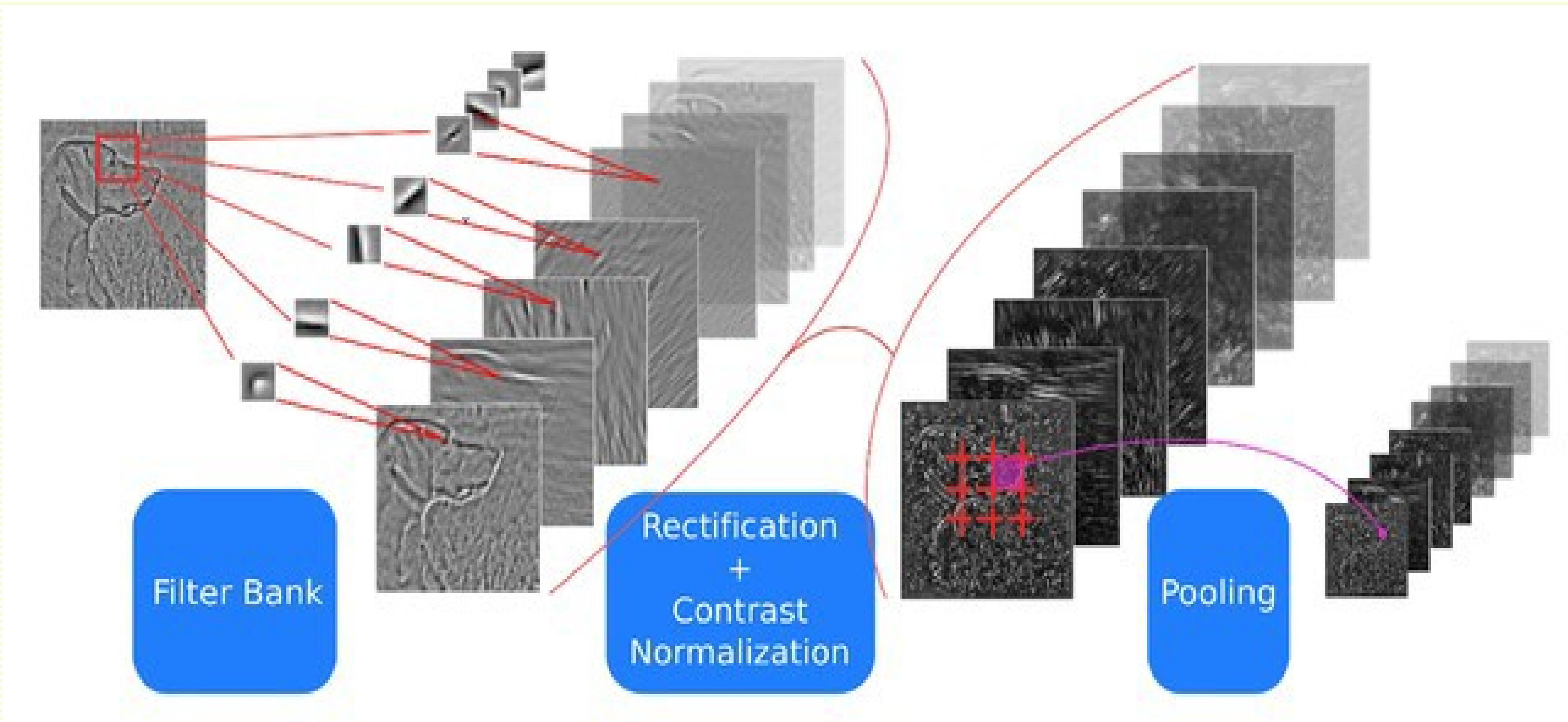
$$\mathbf{h}^{i+1}(\mathbf{x},\mathbf{y}) = \frac{\mathbf{h}^{i}(\mathbf{x},\mathbf{y}) - \mu^{i}(\mathbf{N}(\mathbf{x},\mathbf{y}))}{\sigma^{i}(\mathbf{N}(\mathbf{x},\mathbf{y}))}$$

- Normalize each patch (say 7x7) to be zero mean unit variance
- Effects
  - Improves invariance
  - Improves optimization by making activation layer on the same scale
  - Usually improves classification rate

ConvNets: Typical Stage

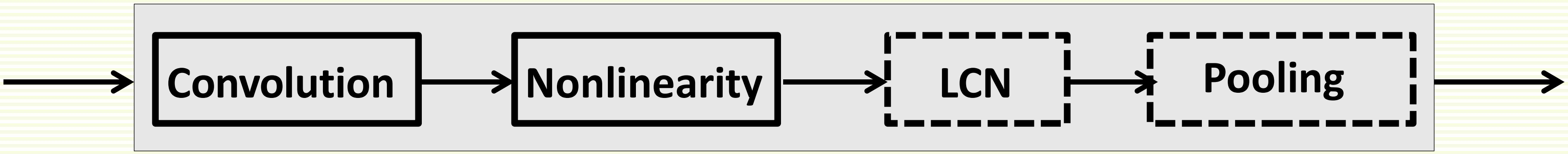
One Stage



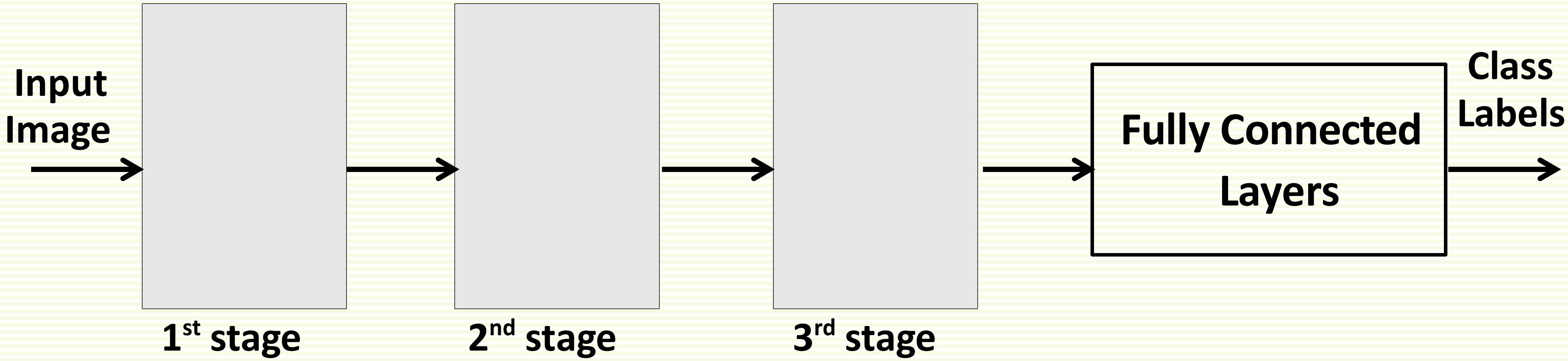


Typical Architecture

One Stage

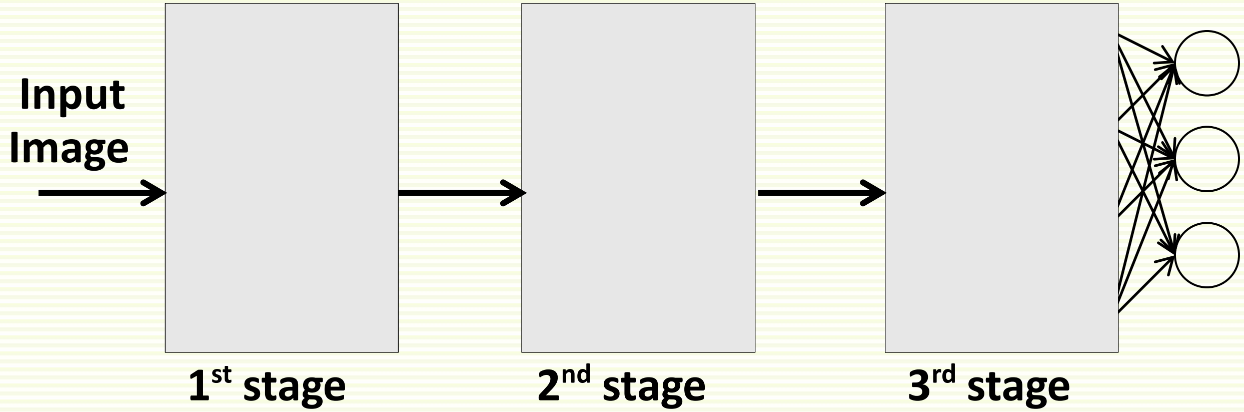


Whole System

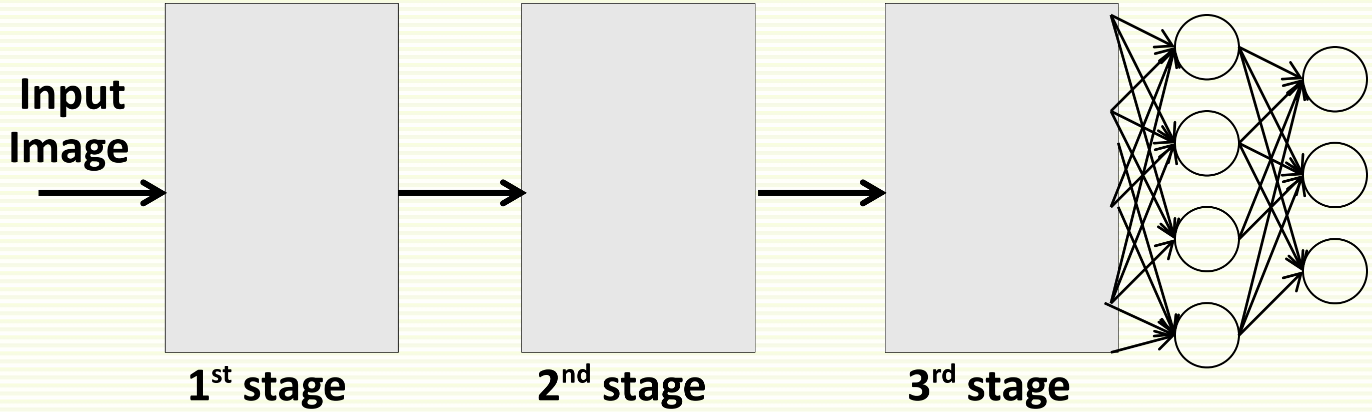


Fully Connected Layer

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

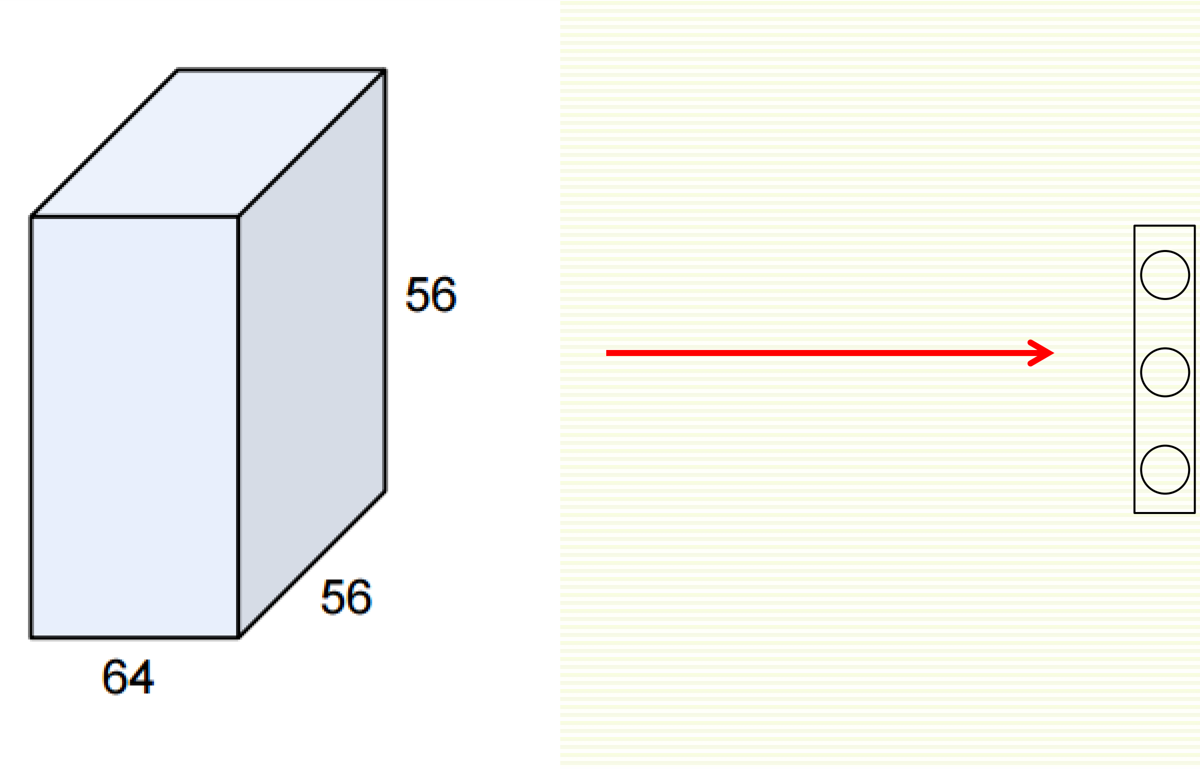


- Can have more than one fully connected layer



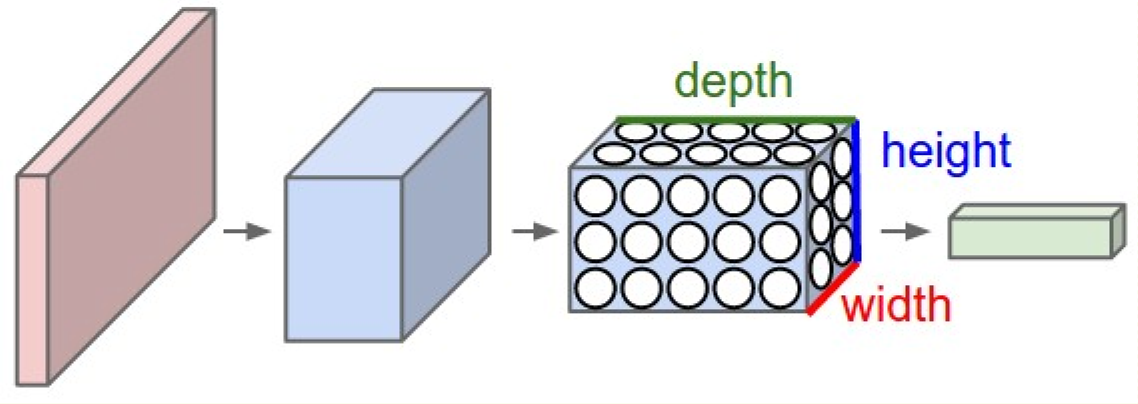
Fully Connected Layer

- Can implement as a convolutional layer
  - input of size 56x56x64
  - say 3 class problem
  - convolve with 3 filters, each of size 56x56x64



Overview of CNN

- Made up of Layers
- Every Layer has a simple API
  - transforms an input 3D volume to an output 3D volume with some differentiable function
  - may or may not have parameters
  - may or may not have hyperparameters

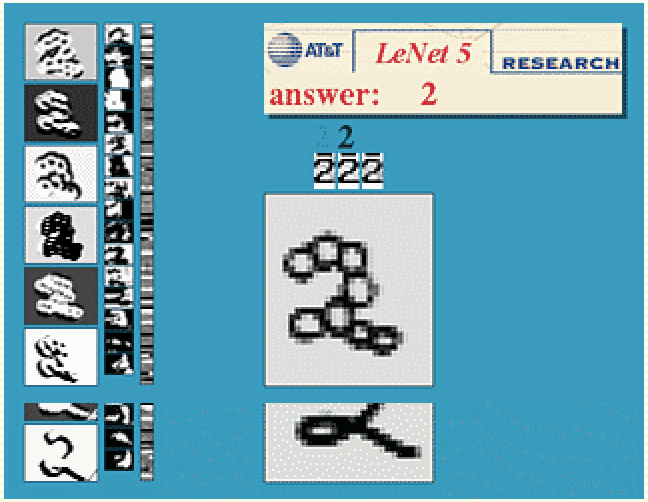


ConvNets: Training

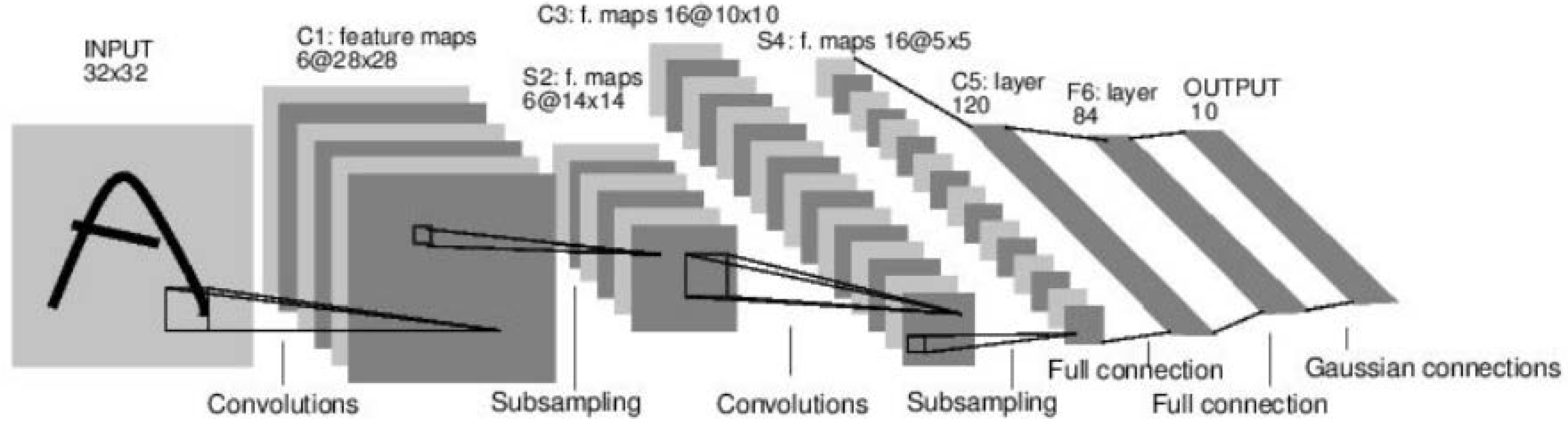
- All Layers are differentiable
- Use standard back-propagation (gradient descent)
- At test time, run only in forward mode

Conv Nets: Character Recognition

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



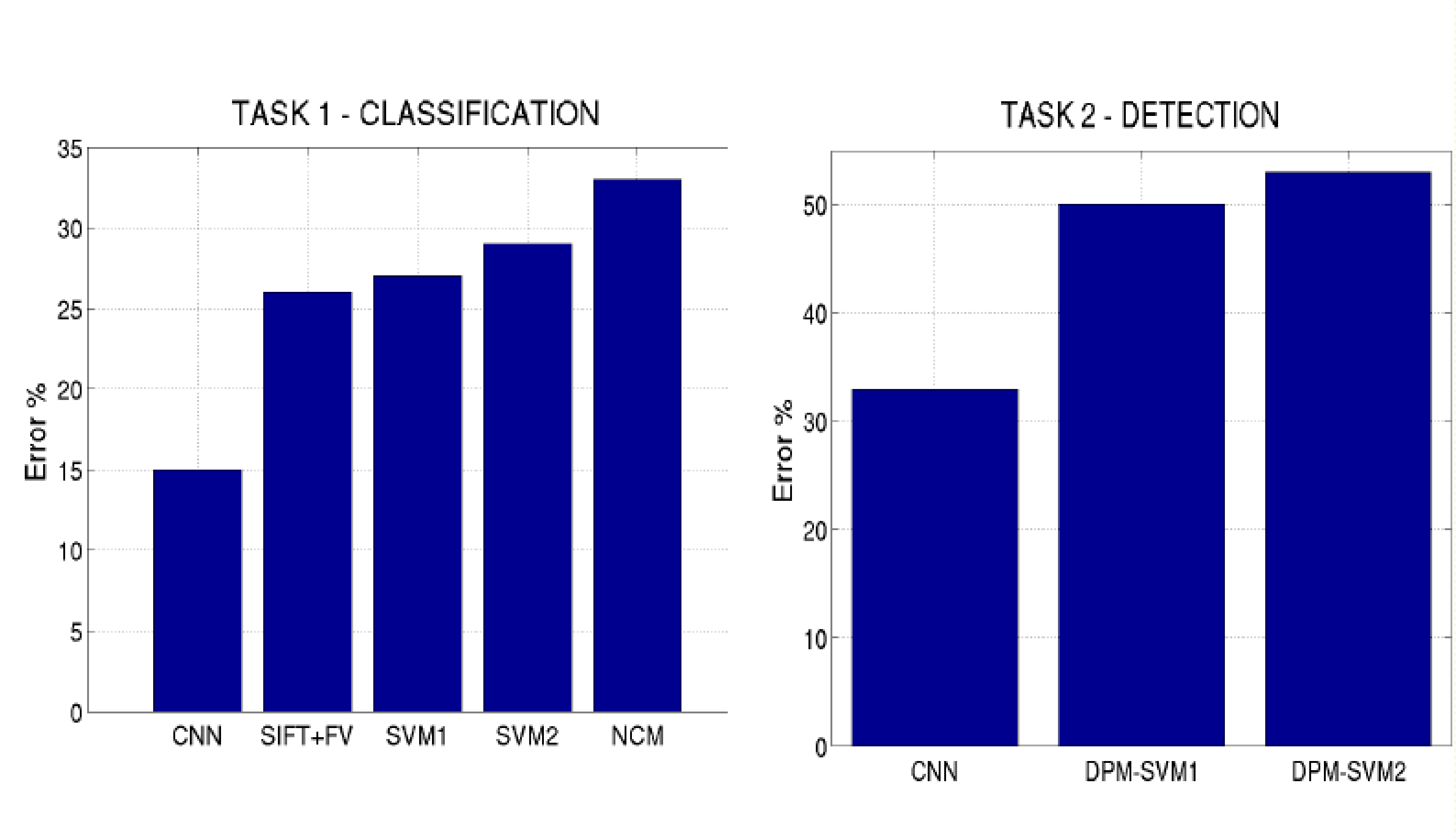
[LeCun et al., 1998]



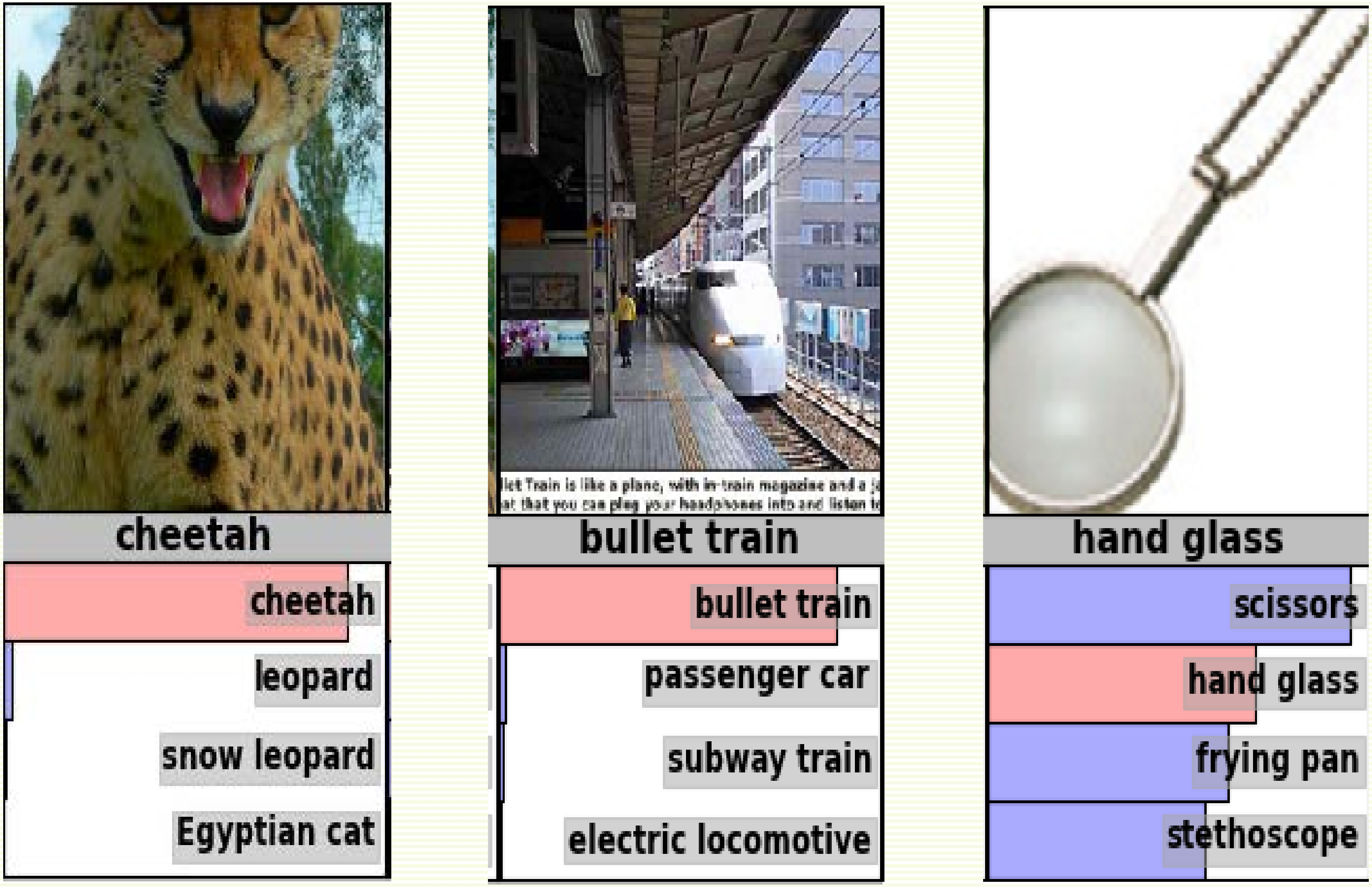
ConvNet for ImageNet

- 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

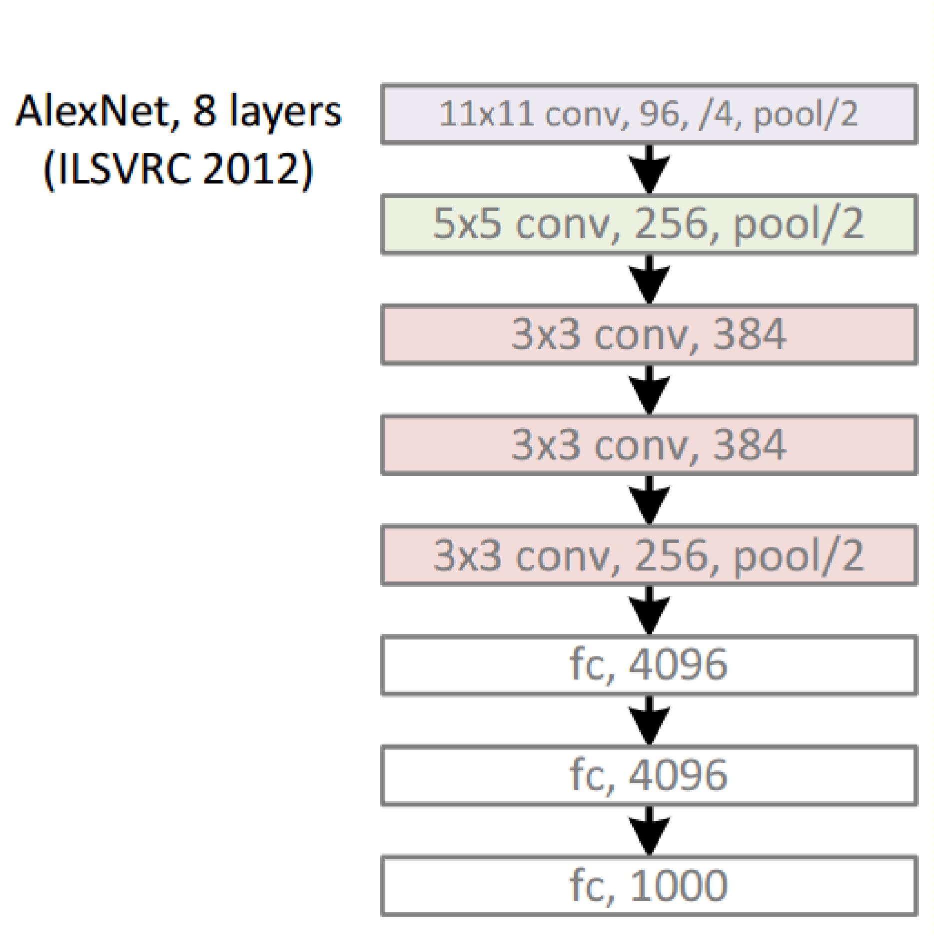
Results: ILSVRC 2012



ConvNet on Image Classification



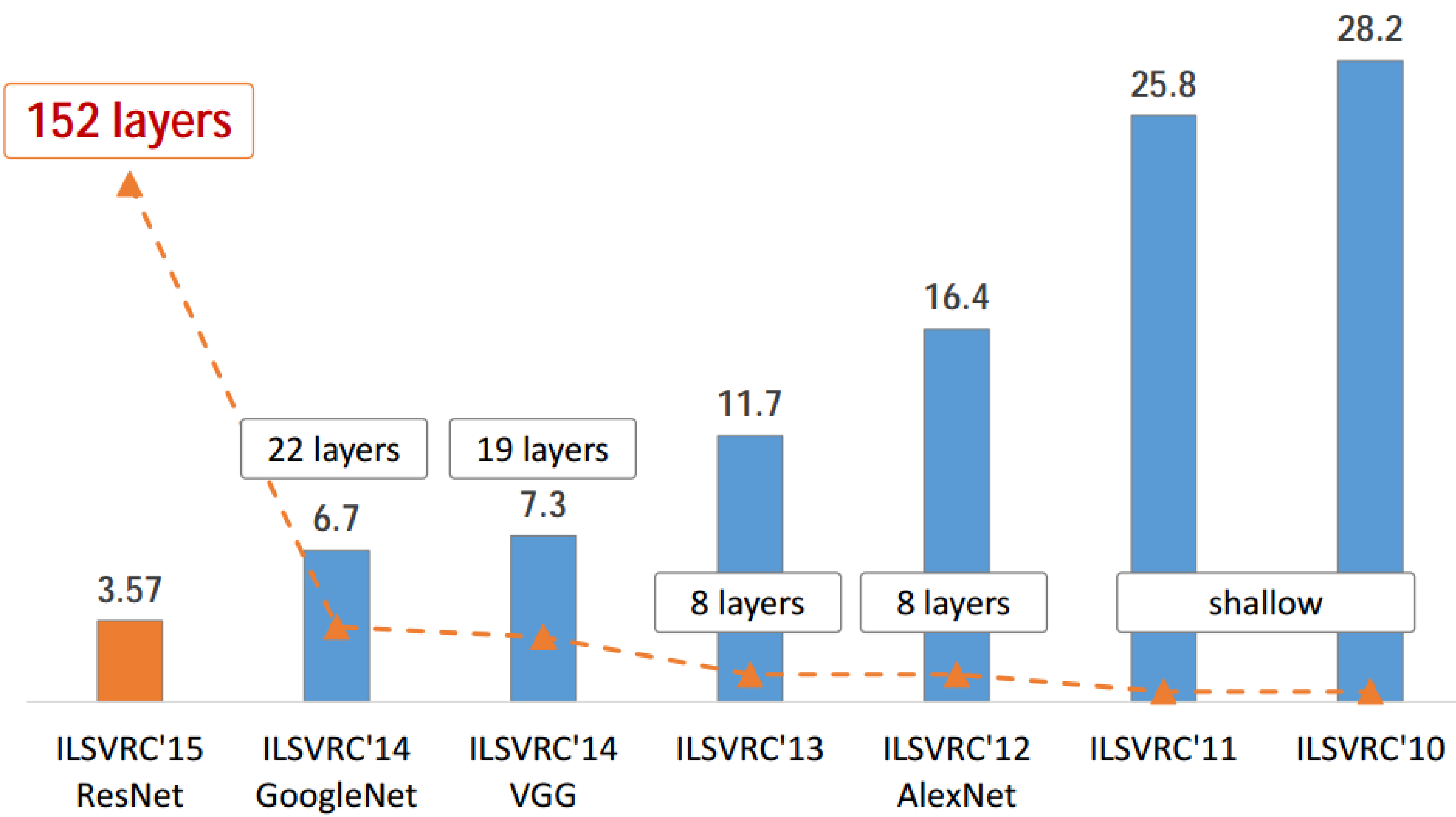
Krizhevsky et.al. Architecture



Tricks to Improve Generalization

- To get more data
  - Use left-right reflections of the images
  - Train on random 224x224 patches from the 256x256 images
- At test time
  - combine the opinions from ten different patches:
    - four 224x224 corner patches plus the central 224x224 patch
    - the reflections of those five patches
- Use *dropout* to regularize weights in the fully connected layers
  - half of the hidden units in a layer are randomly removed for each training example

ImageNet Experiments



ImageNet Classification top-5 error (%)