

CS9840

Learning and Computer Vision

Prof. Olga Veksler

Lecture 11

Convolutional Neural Networks

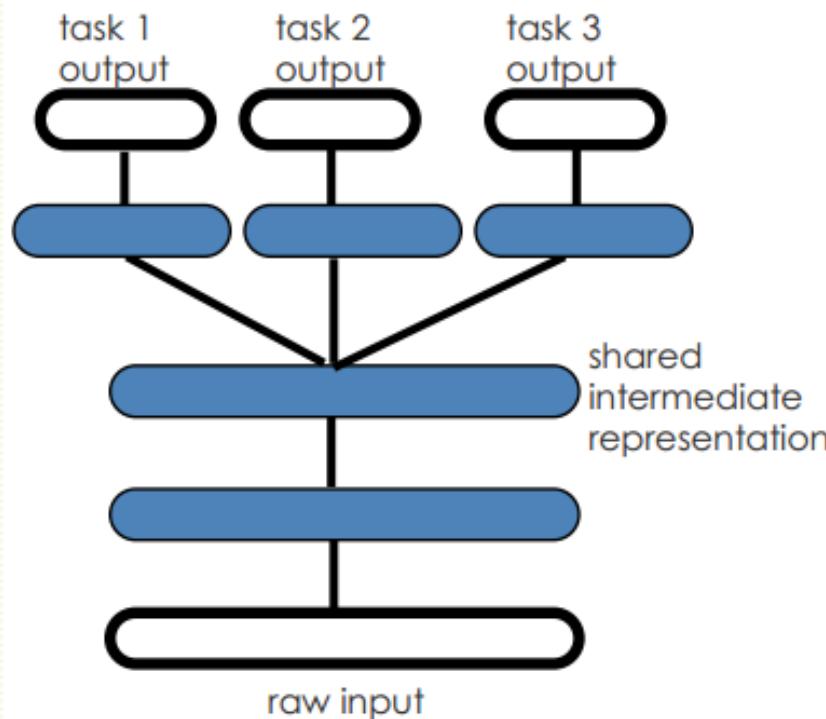
Many slides are from A. Ng, Y. LeCun, G. Hinton, A. Ranzato

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

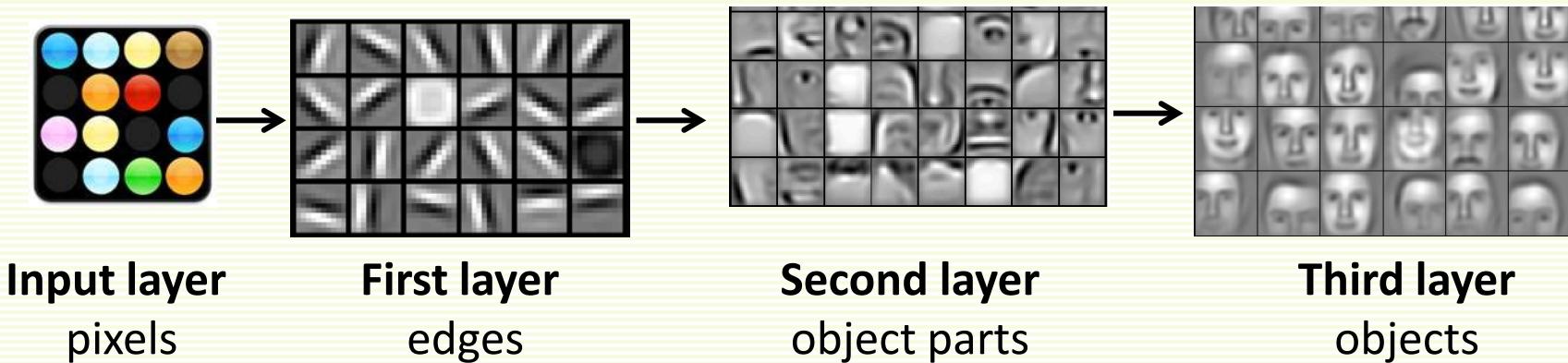
Why Deep Networks

- Sub-features created in deep architecture can potentially be shared between multiple tasks



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

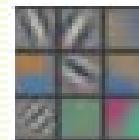


Why Deep Networks: Hierarchical Feature Extraction

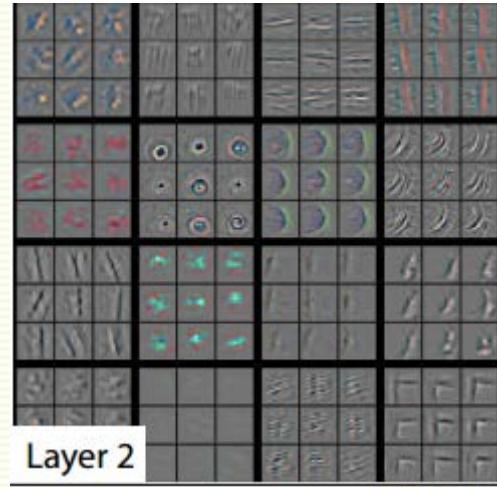
- Another example (from M. Zeiler'2013)

visualization of
learned features

Layer 1



Layer 2



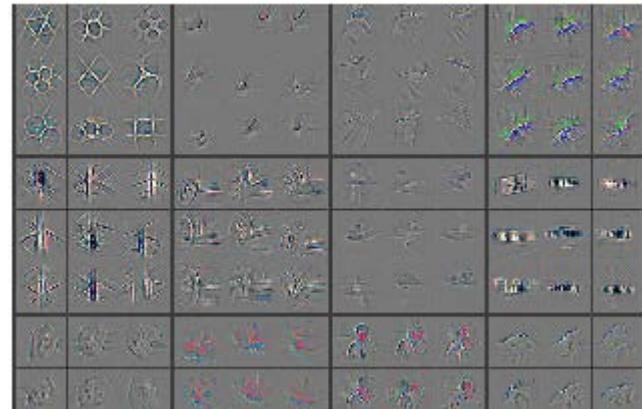
Patches that result in high
response



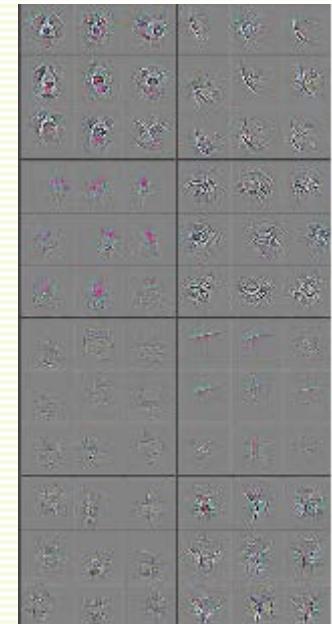
Why Deep Networks: Hierarchical Feature Extraction

visualization of learned features

Layer 3



Layer 4



Patches that result in high response



Early Work on Deep Networks

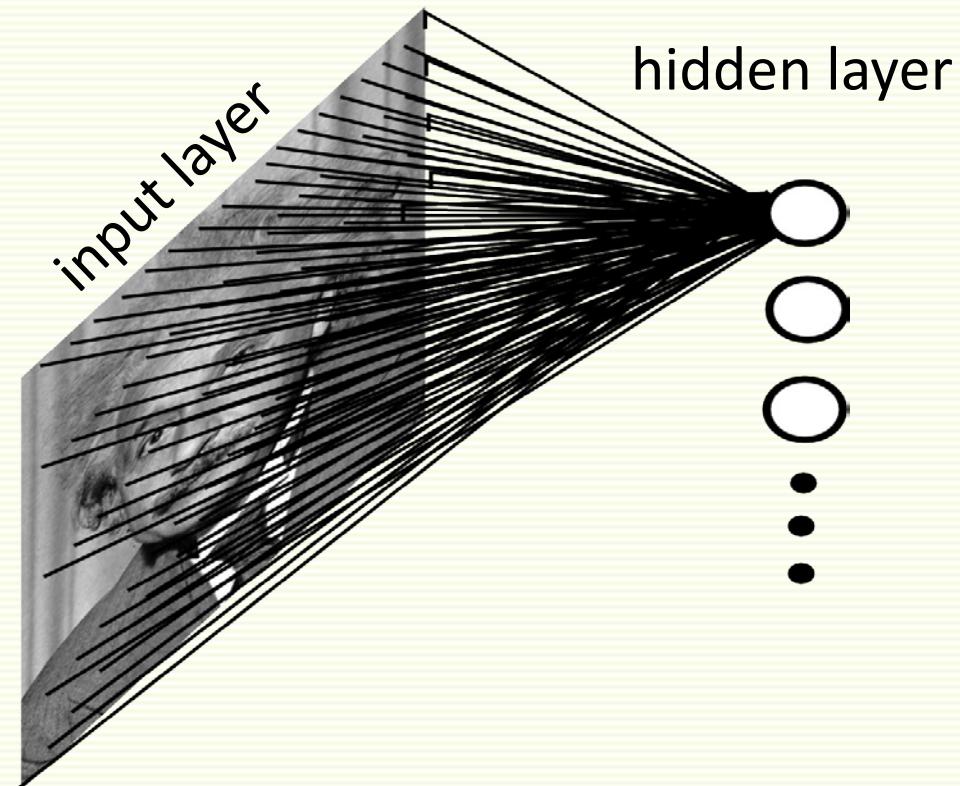
- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Networks (convnets)
 - Similarities to Neo-Cognitron
 - Success on character recognition
- Other attempts at deeply layered Networks trained with backpropagation
 - not much success
 - very slow
 - diffusion of gradient
 - recent work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

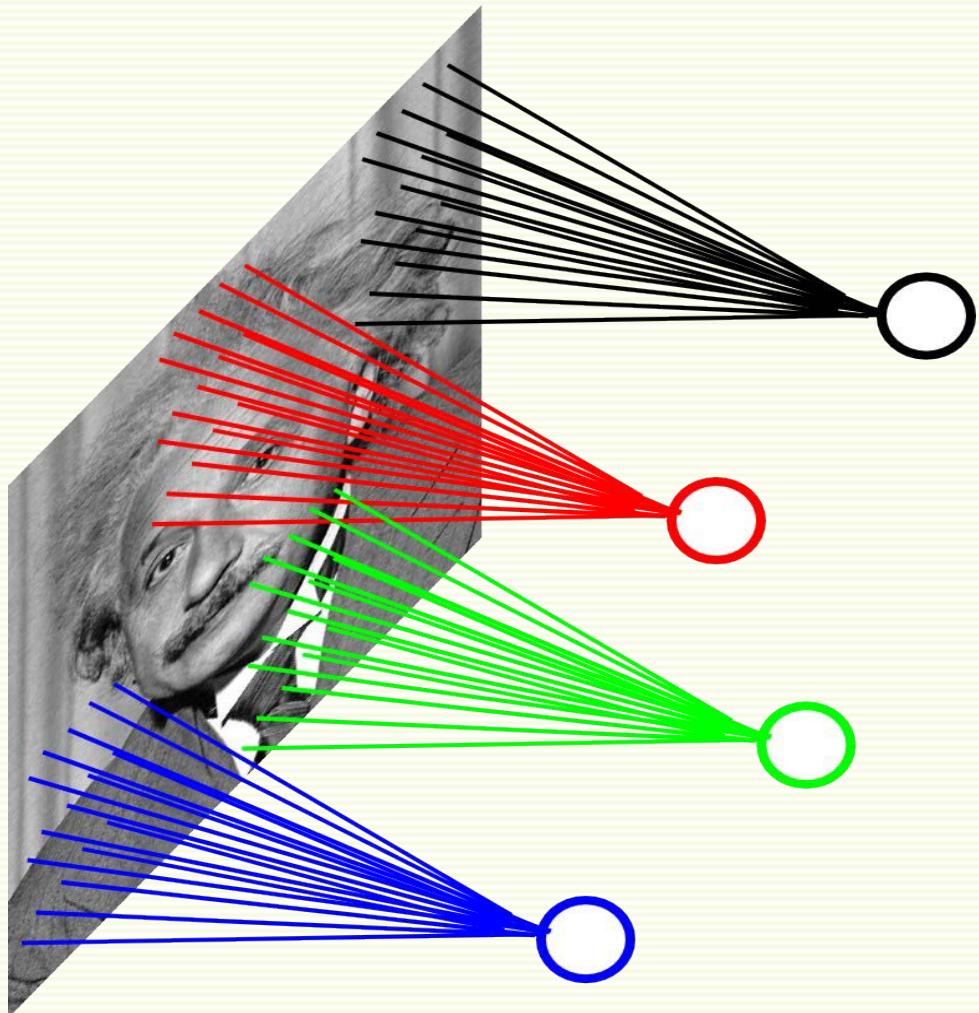
Convolutional Network: Motivation

- Consider a fully connected network
- Example: 200 by 200 image, 4×10^4 connections to one hidden unit
- For 10^5 hidden units $\rightarrow 4 \times 10^9$ connections
- But spatial correlations are mostly local
- Do not waste resources by connecting unrelated pixels



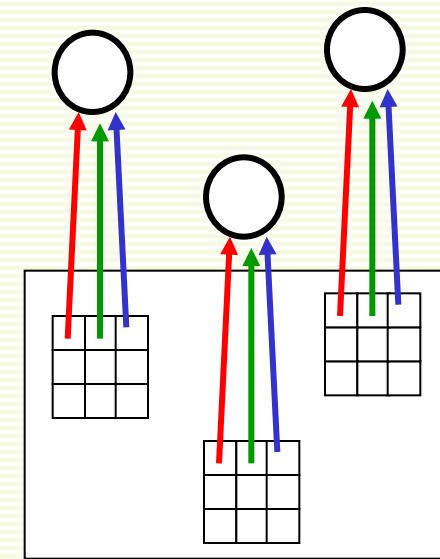
Convolutional Network: Motivation

- Connect only pixels in a local patch, say 10×10
- For 200×200 image, 10^2 connections to one hidden unit
- For 10^5 hidden units $\rightarrow 10^7$ connections
 - contrast with 4×10^9 for fully connected layer
 - 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



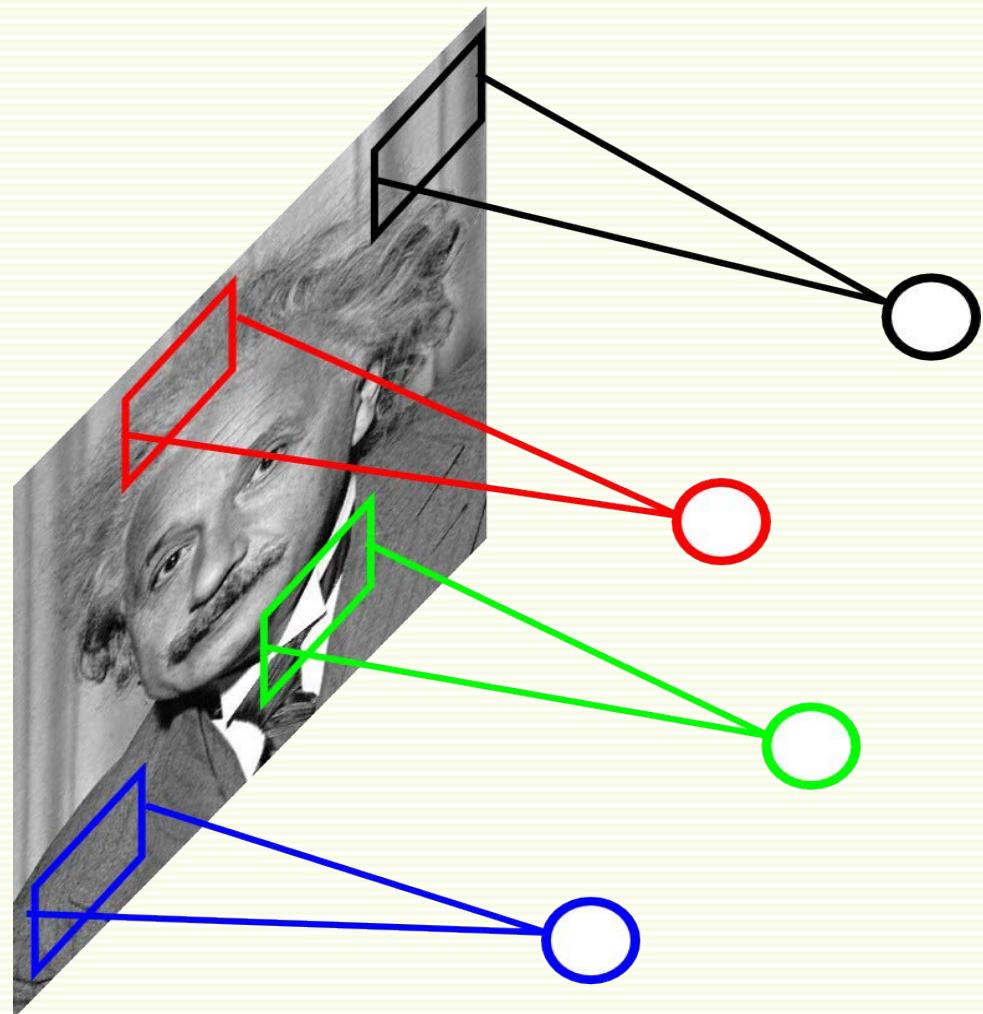
all red connections
have the same weight

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 10×10 patch
 - 10^7 parameters to learn without sharing
 - 10^2 parameters to learn with sharing

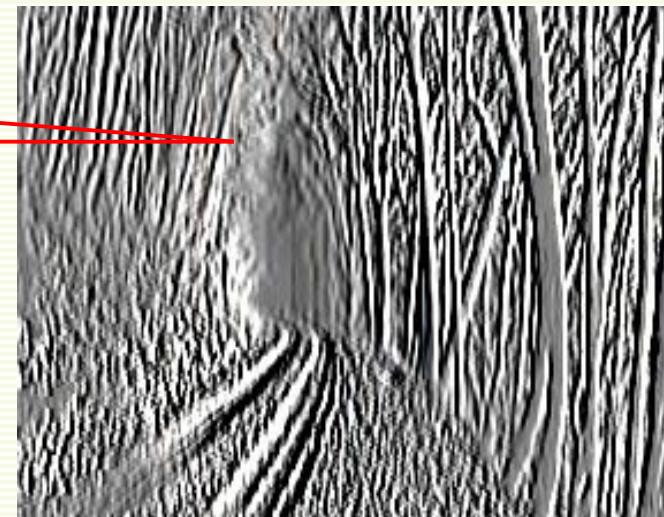


Filtering via Convolution Recap

- Recall filtering with convolution for feature extraction

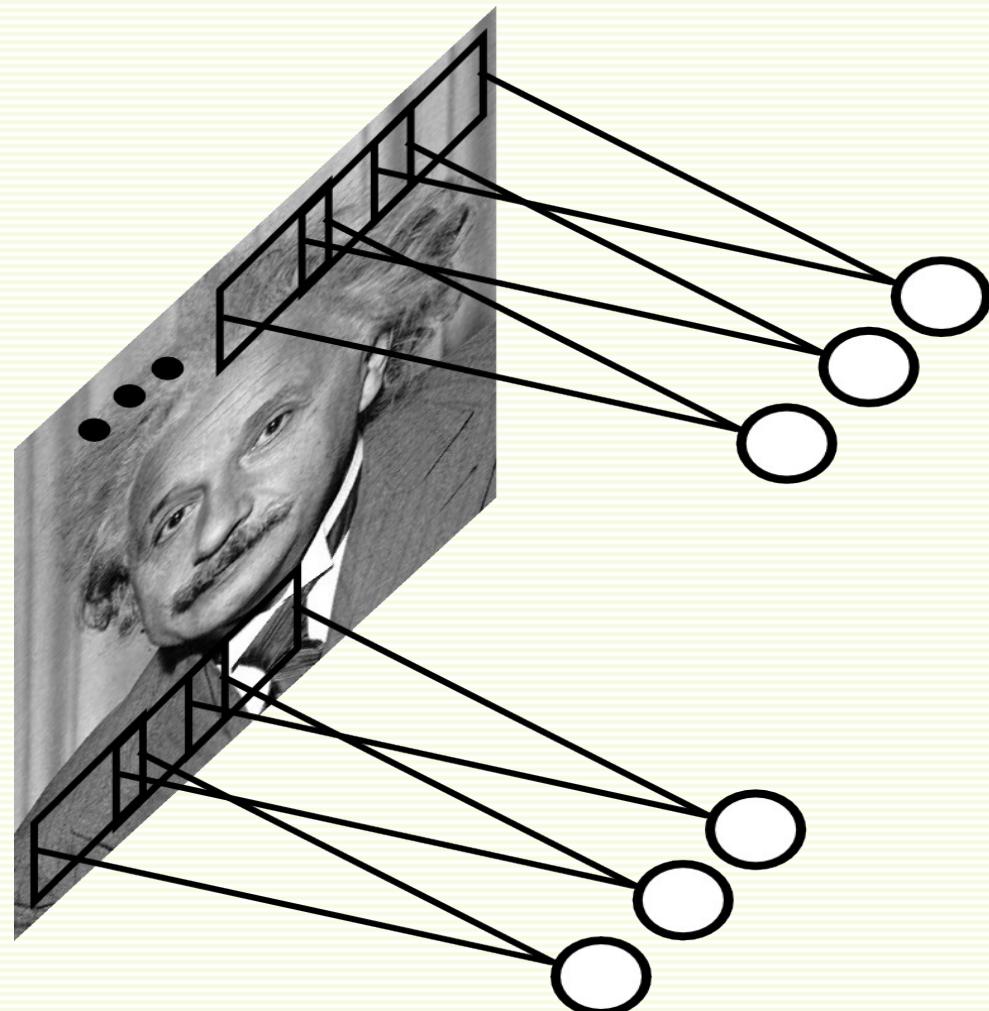


$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$

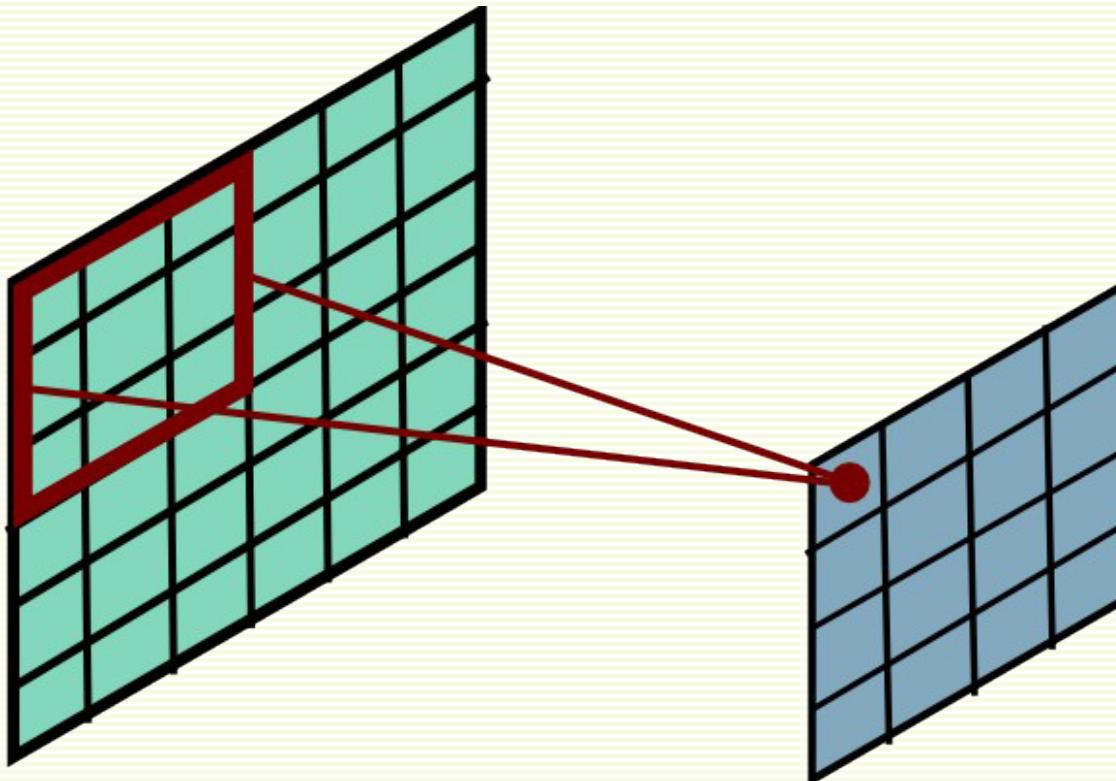


Convolutional Layer

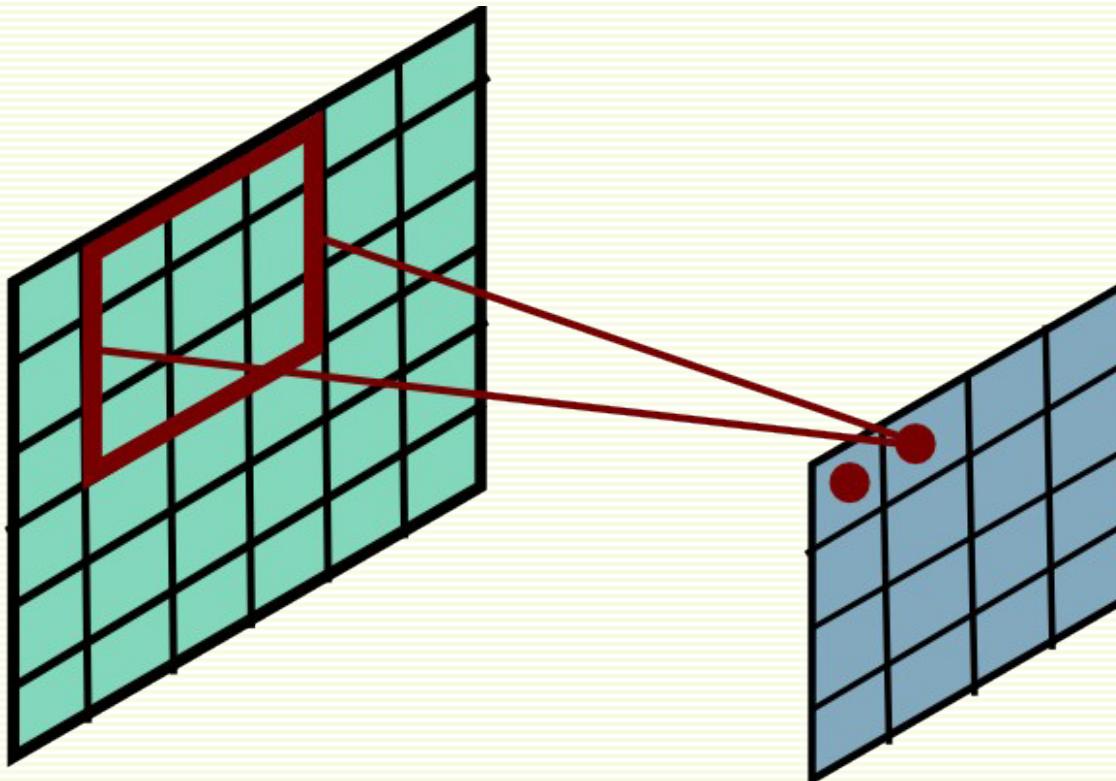
- Note similarity to convolution with some fixed filter
- But here the filter is learned



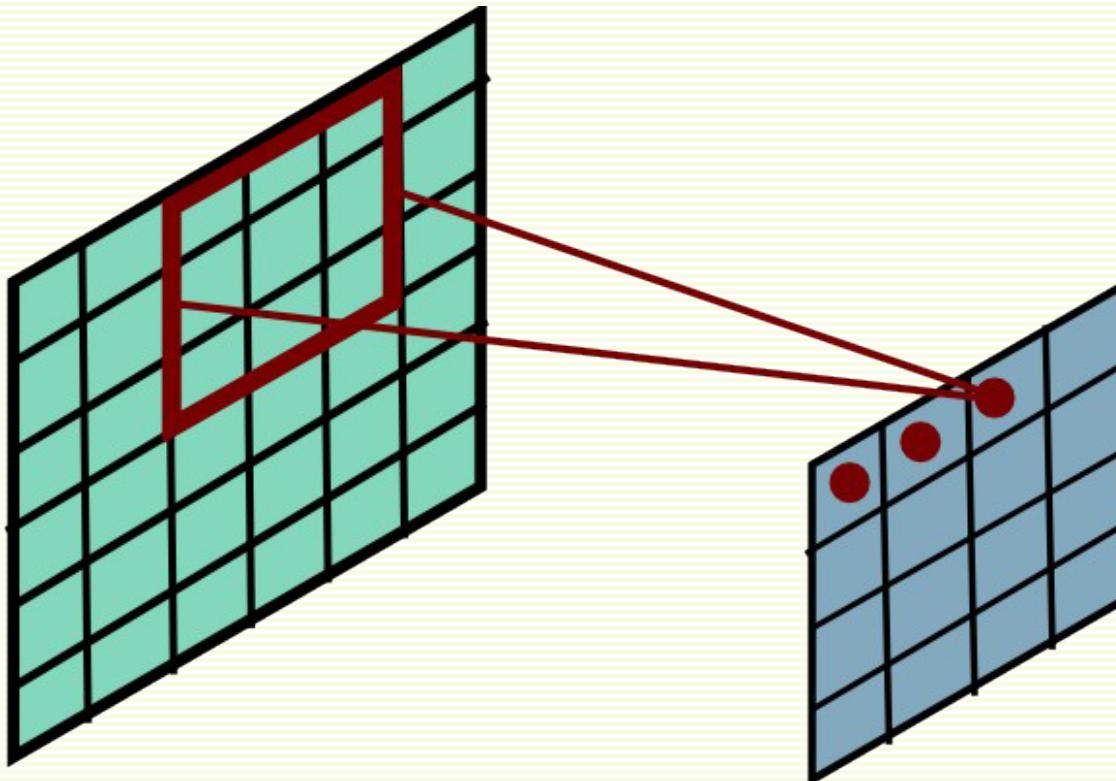
Convolutional Layer



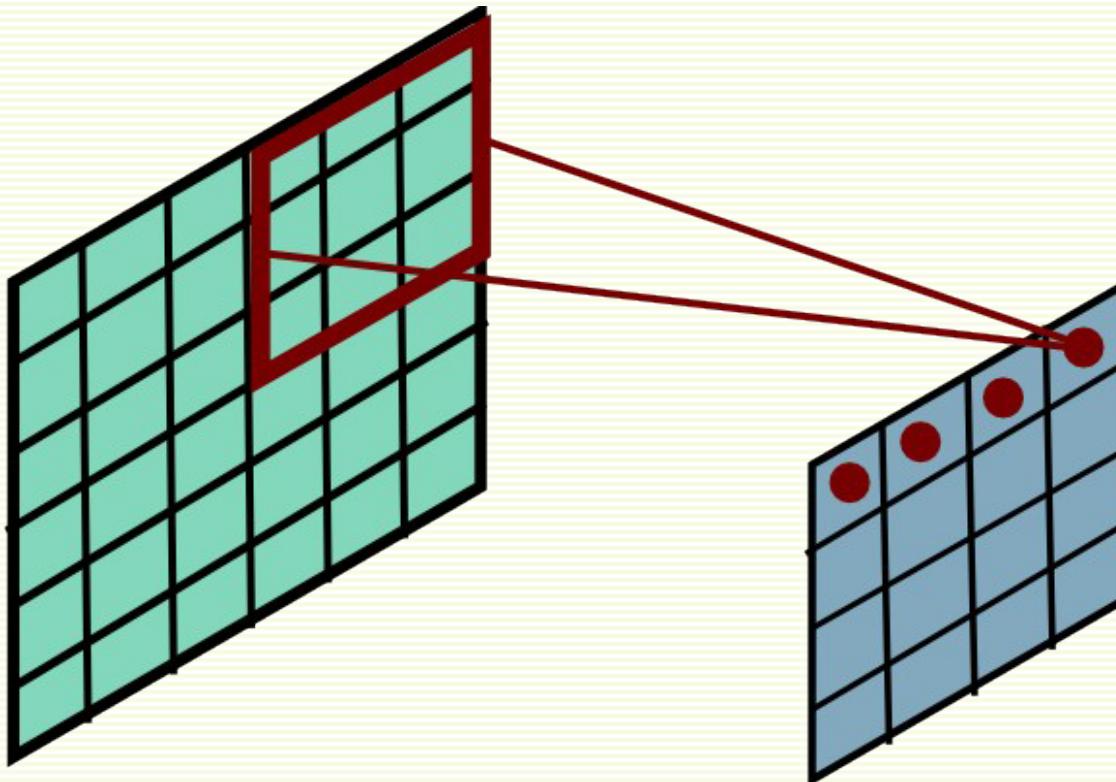
Convolutional Layer



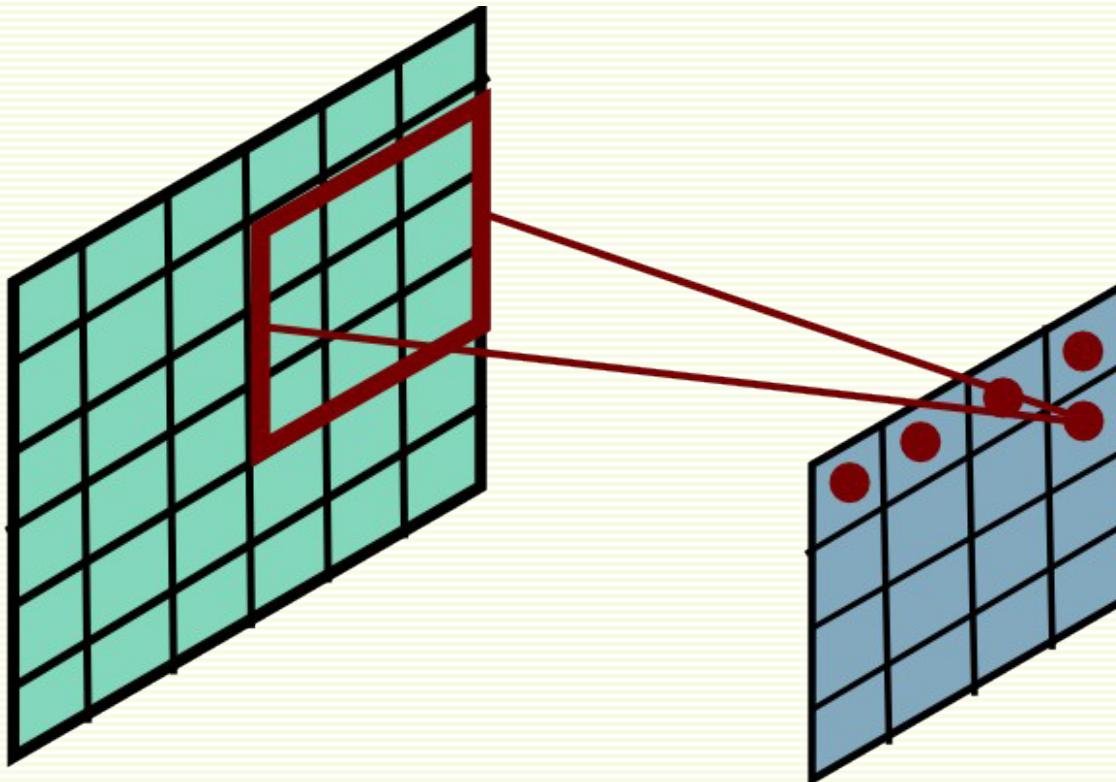
Convolutional Layer



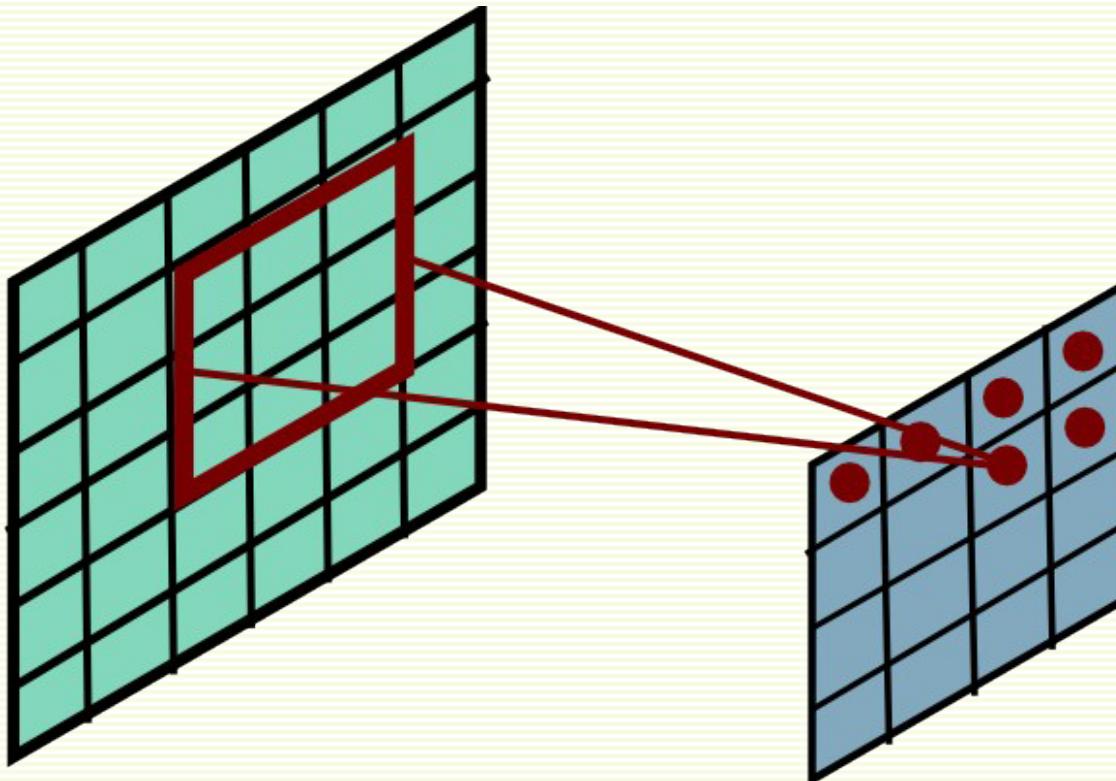
Convolutional Layer



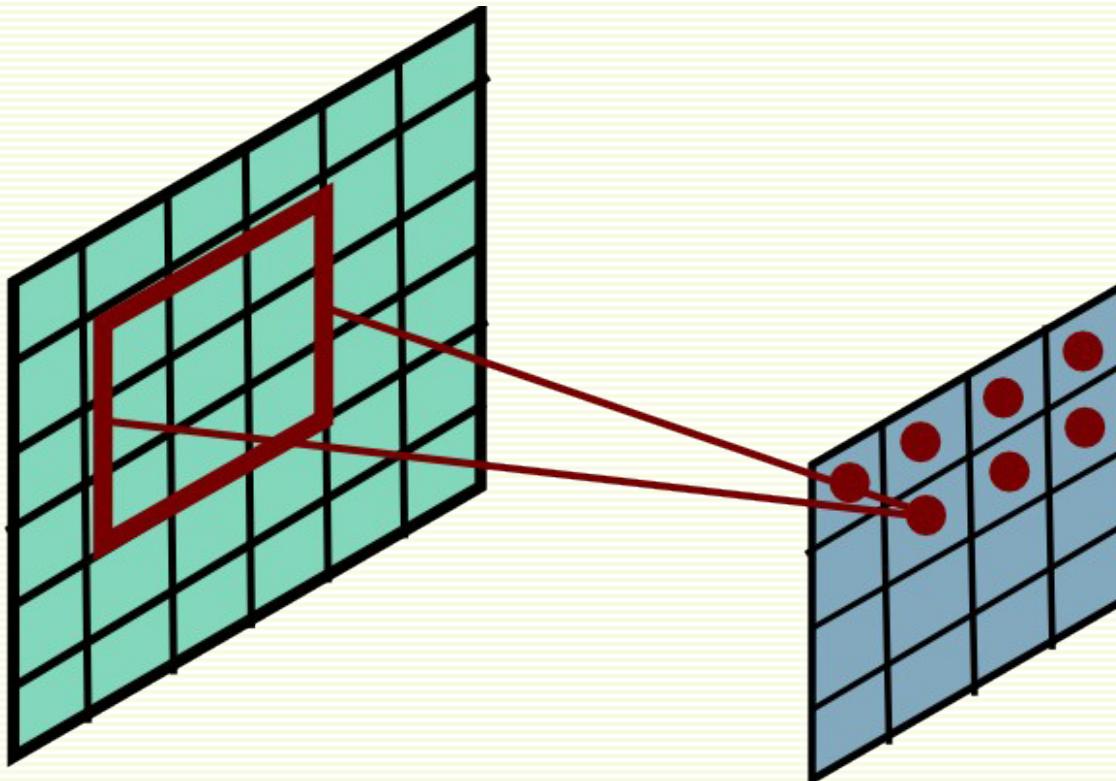
Convolutional Layer



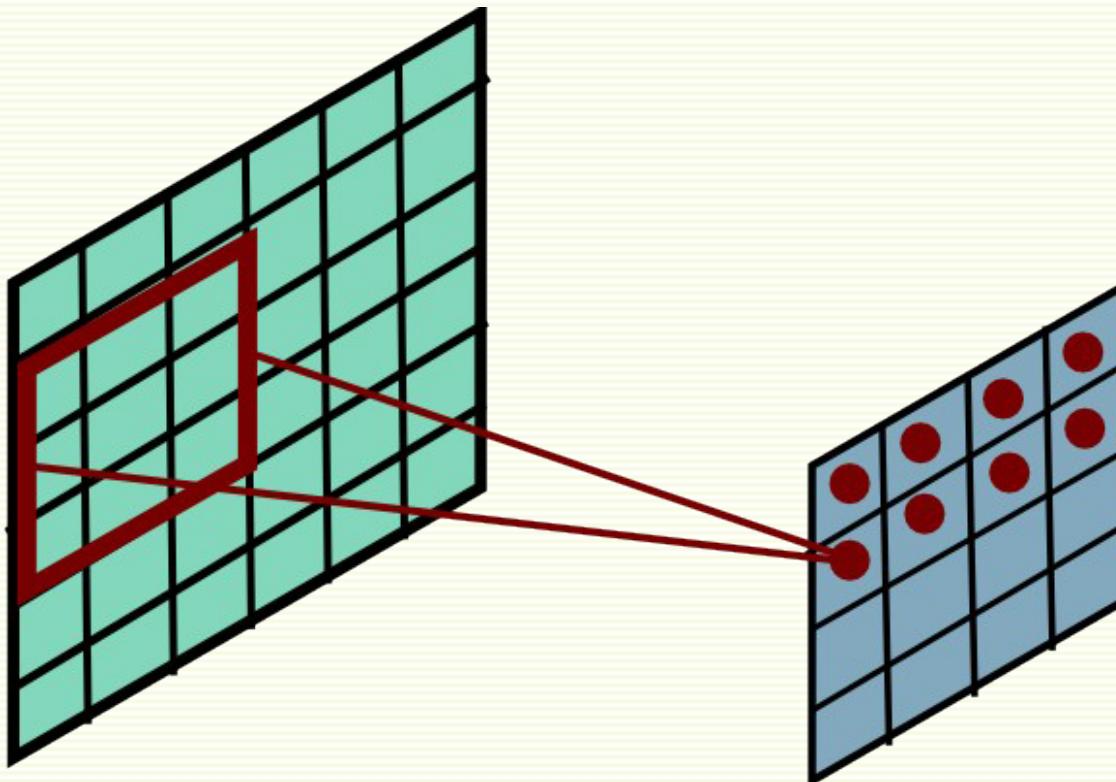
Convolutional Layer



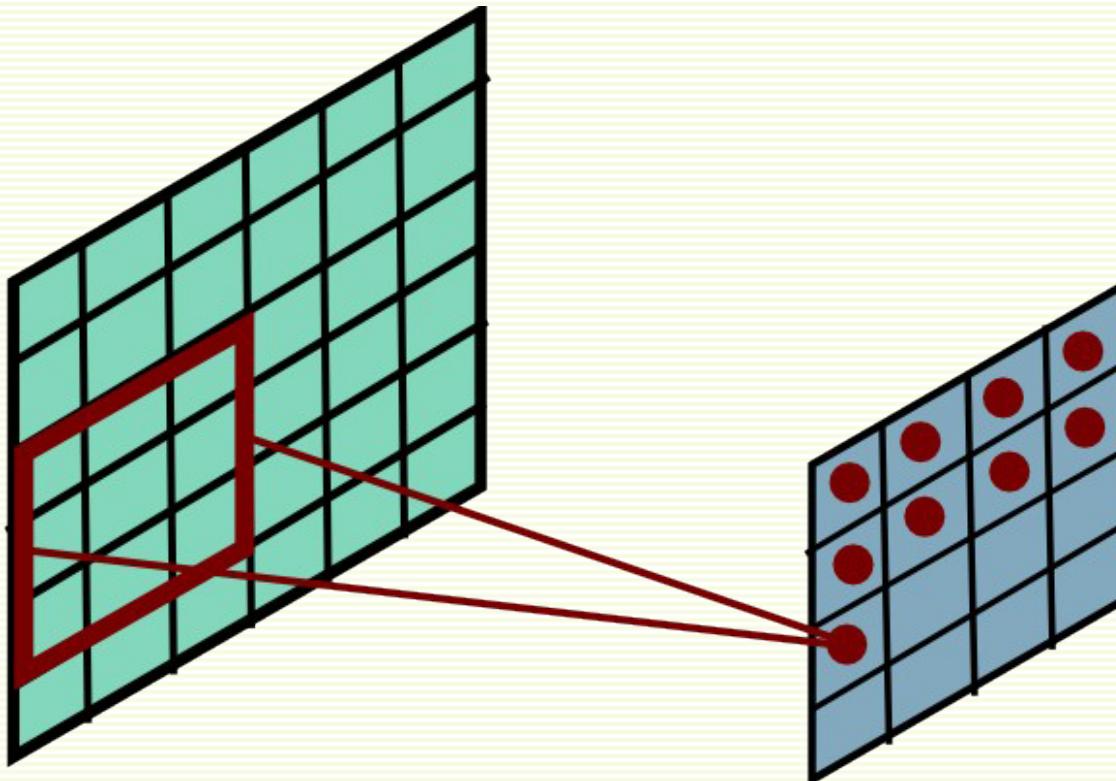
Convolutional Layer



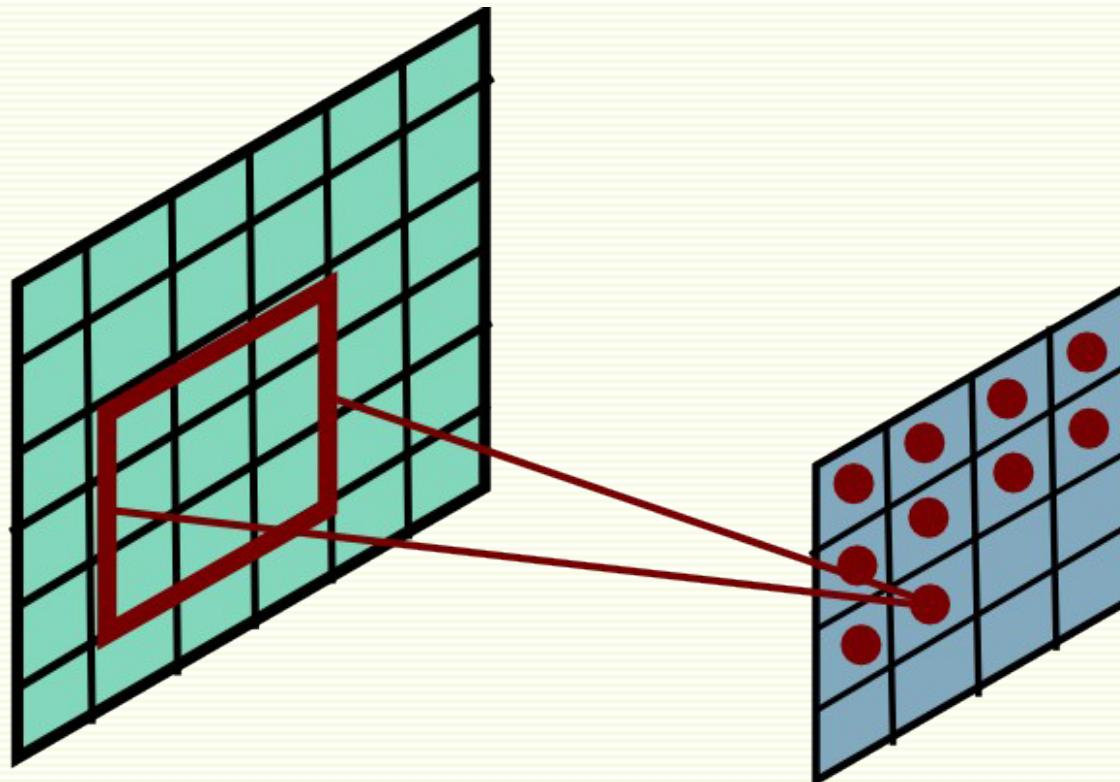
Convolutional Layer



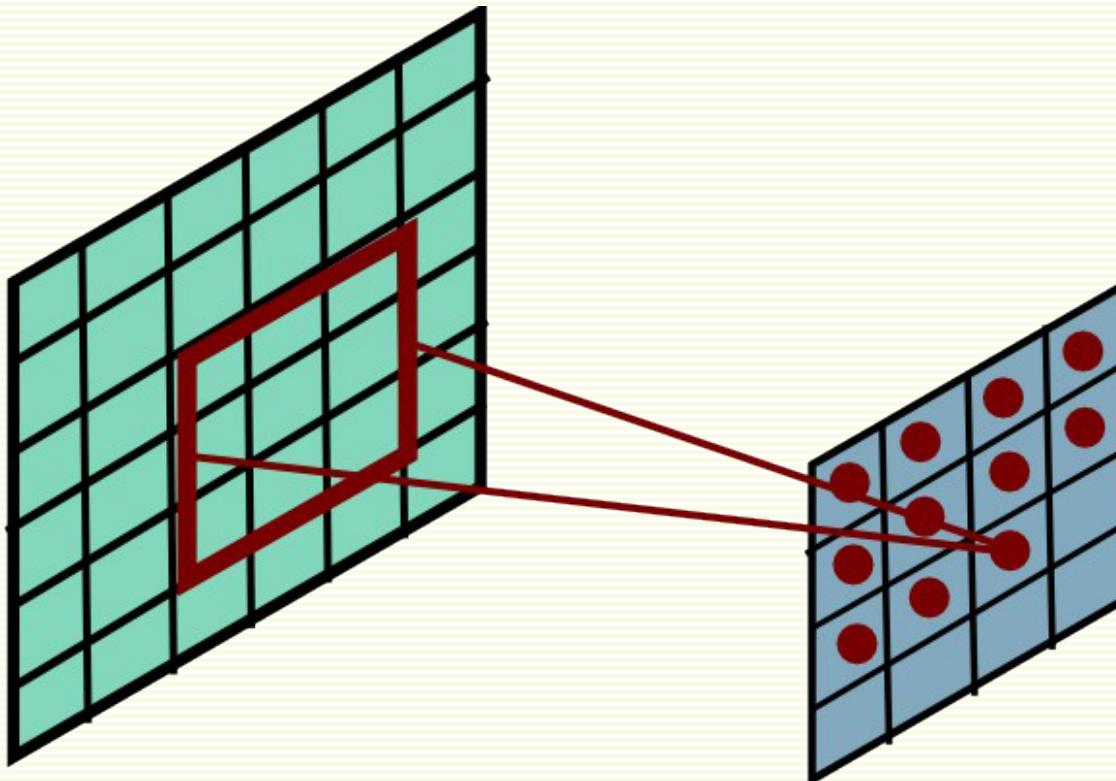
Convolutional Layer



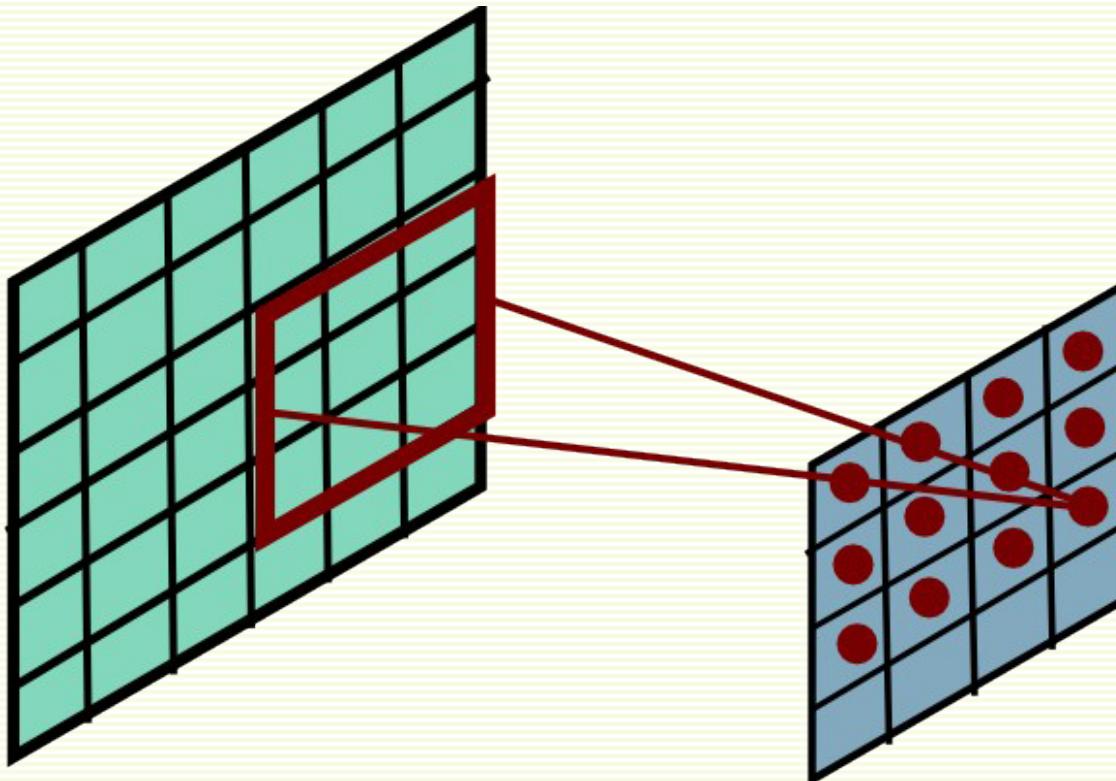
Convolutional Layer



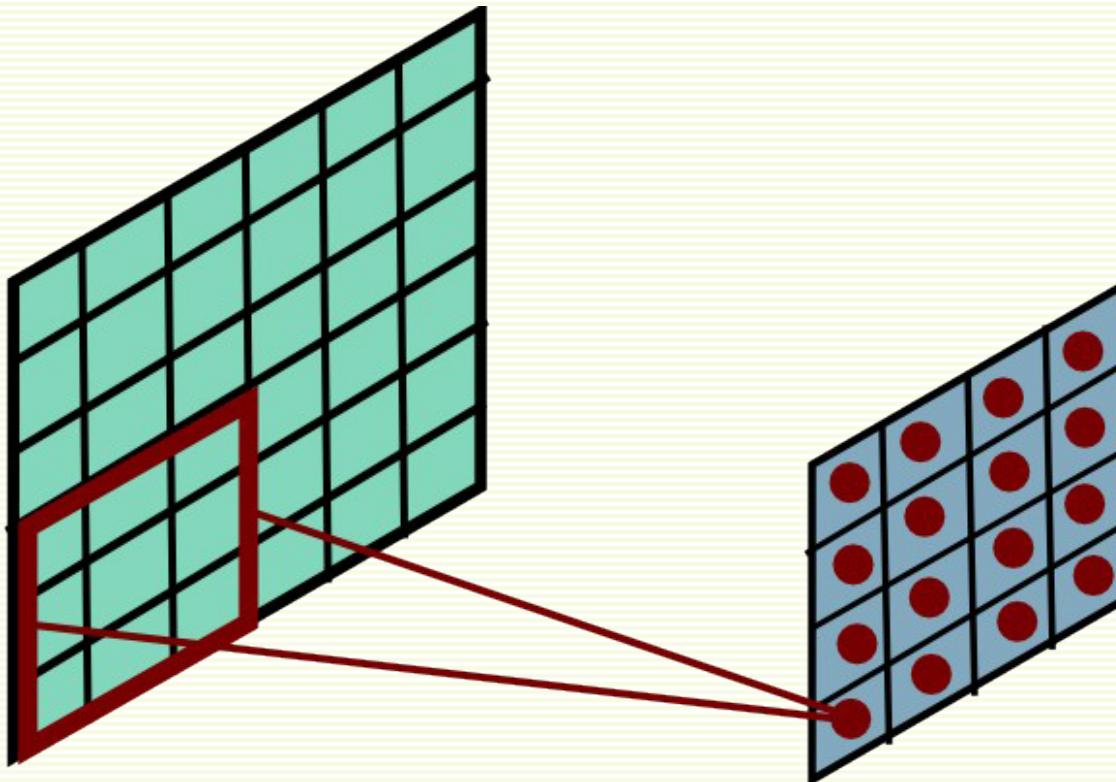
Convolutional Layer



Convolutional Layer

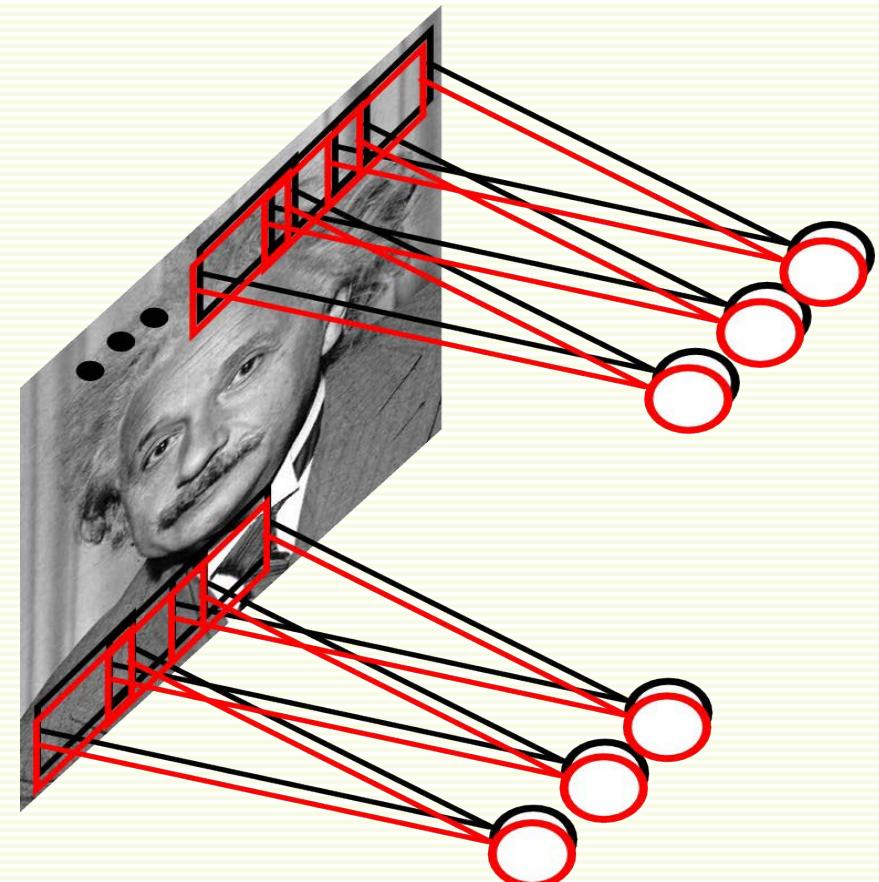


Convolutional Layer



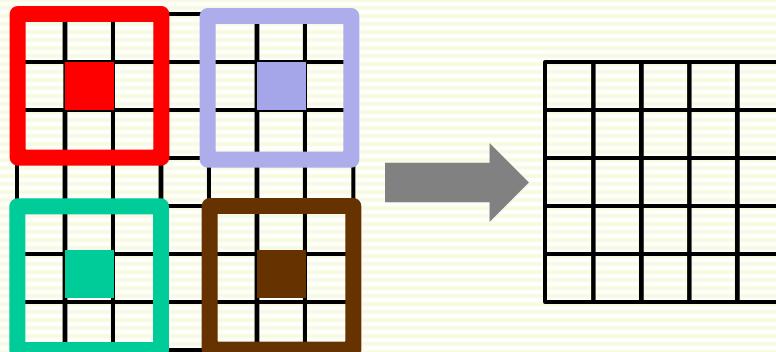
Convolutional Layer

- 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

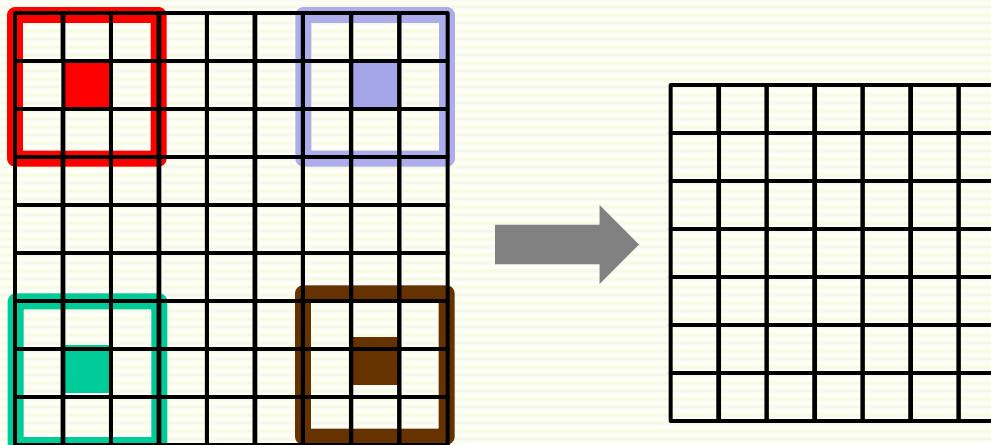


Convolutional Layer

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

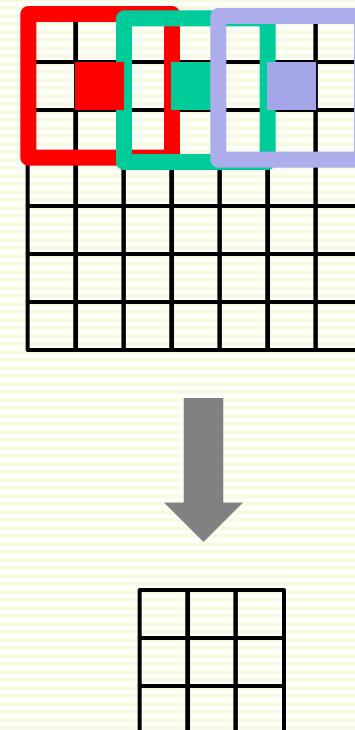


- If want output to be the same size, zero-pad the image appropriately



Convolutional Layer

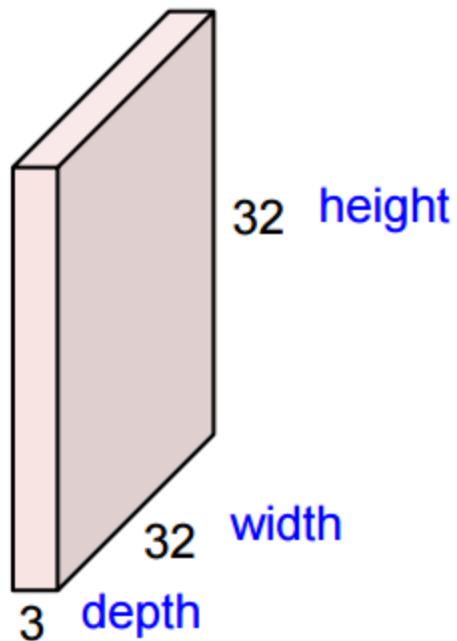
- Can apply convolution only to some pixels (say every second)
 - output layer is smaller
 - less parameters to learn
- Example
 - stride = 2
 - apply convolution every second pixel
 - makes image approximately twice smaller in each dimension
 - image not zero-padded in this example



Convolutional Layer

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

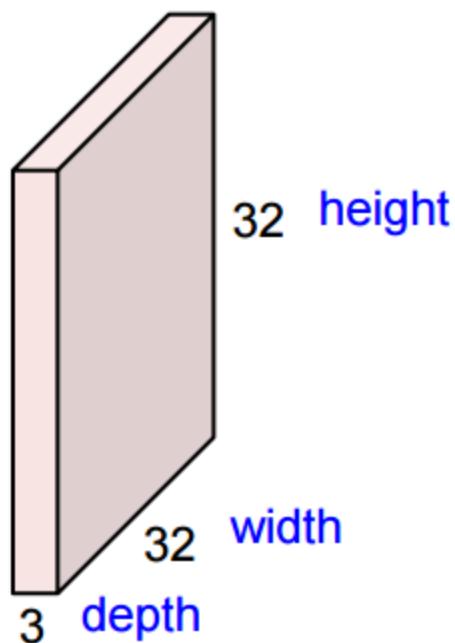
32x32x3 image



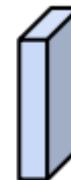
Convolutional Layer

- Convolve 3D image with 3D filter

$32 \times 32 \times 3$ image

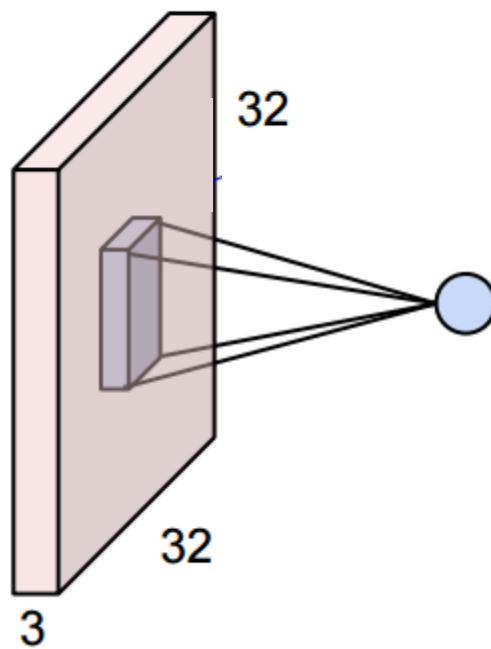


$5 \times 5 \times 3$ filter



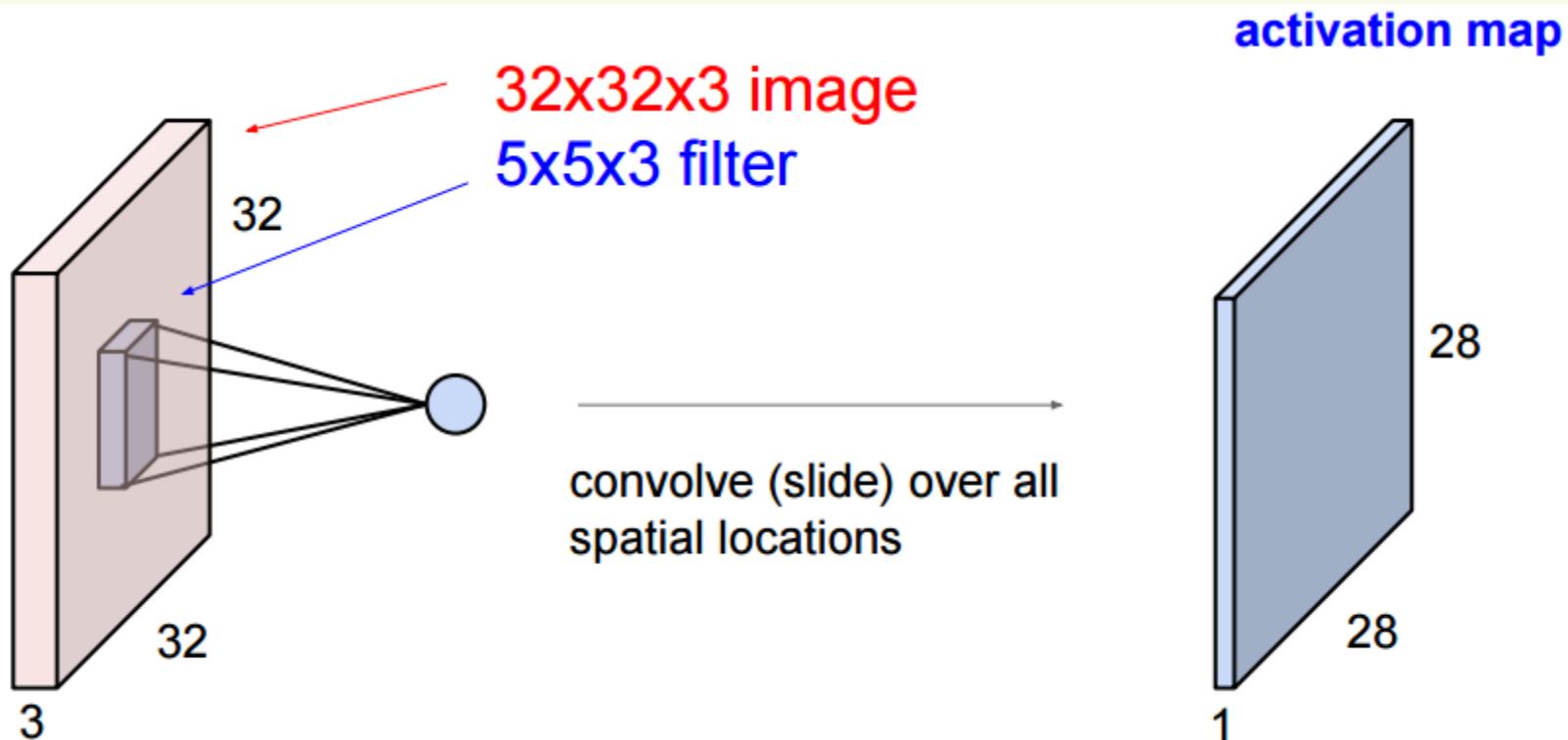
Convolutional Layer

- One convolution step is a 75 dimensional dot product between the $5 \times 5 \times 3$ filter and a piece of image of size $5 \times 5 \times 3$
- 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)



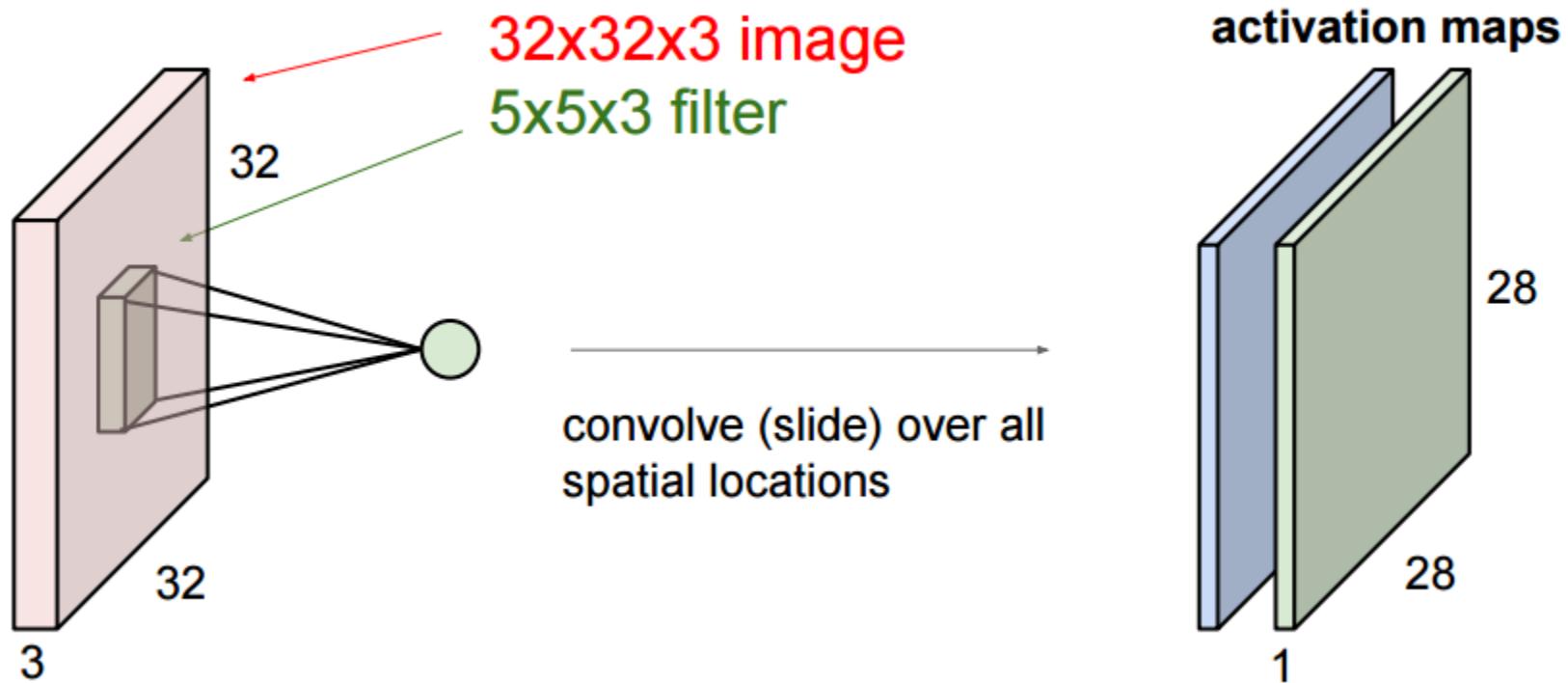
Convolutional Layer

- Convolve 3D image with 3D filter
 - result is a $28 \times 28 \times 1$ activation map, no zero padding used
 - 76 parameters to learn



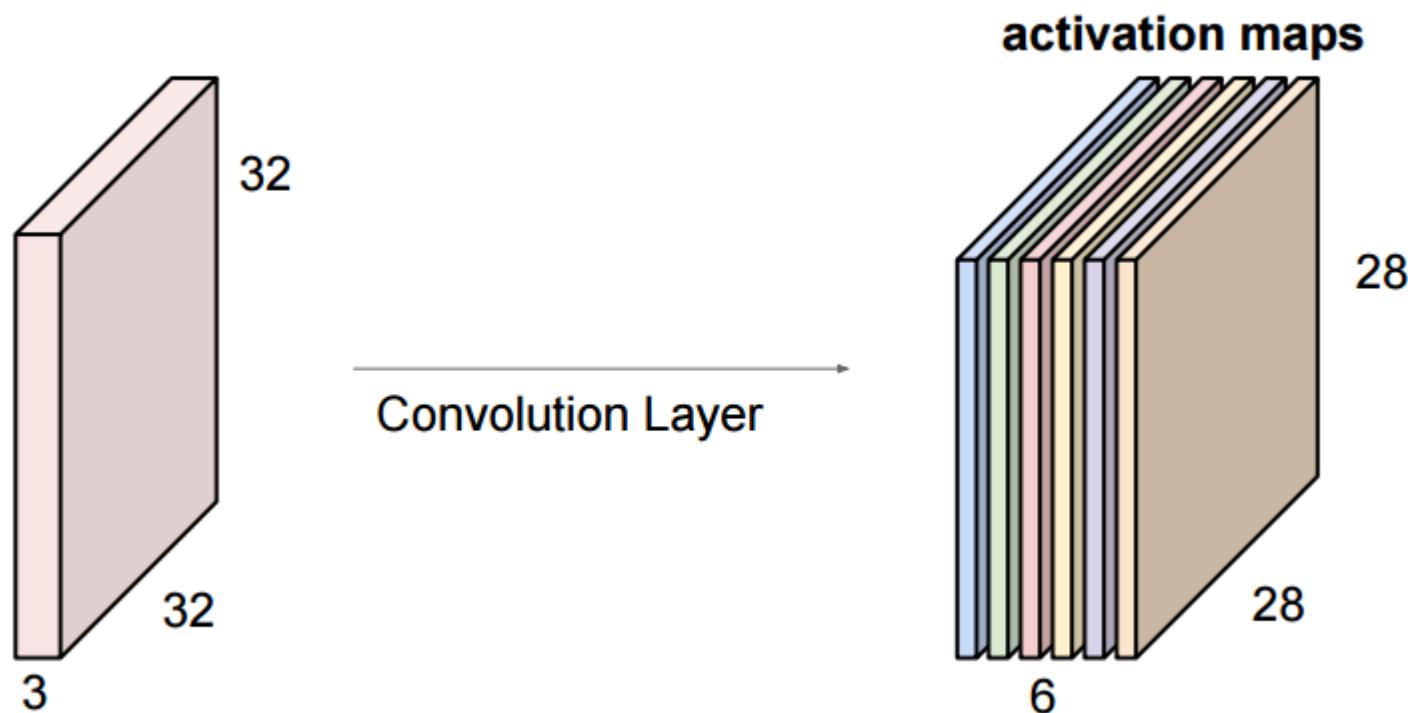
Convolutional Layer

- Consider a second, green filter



Convolutional Layer

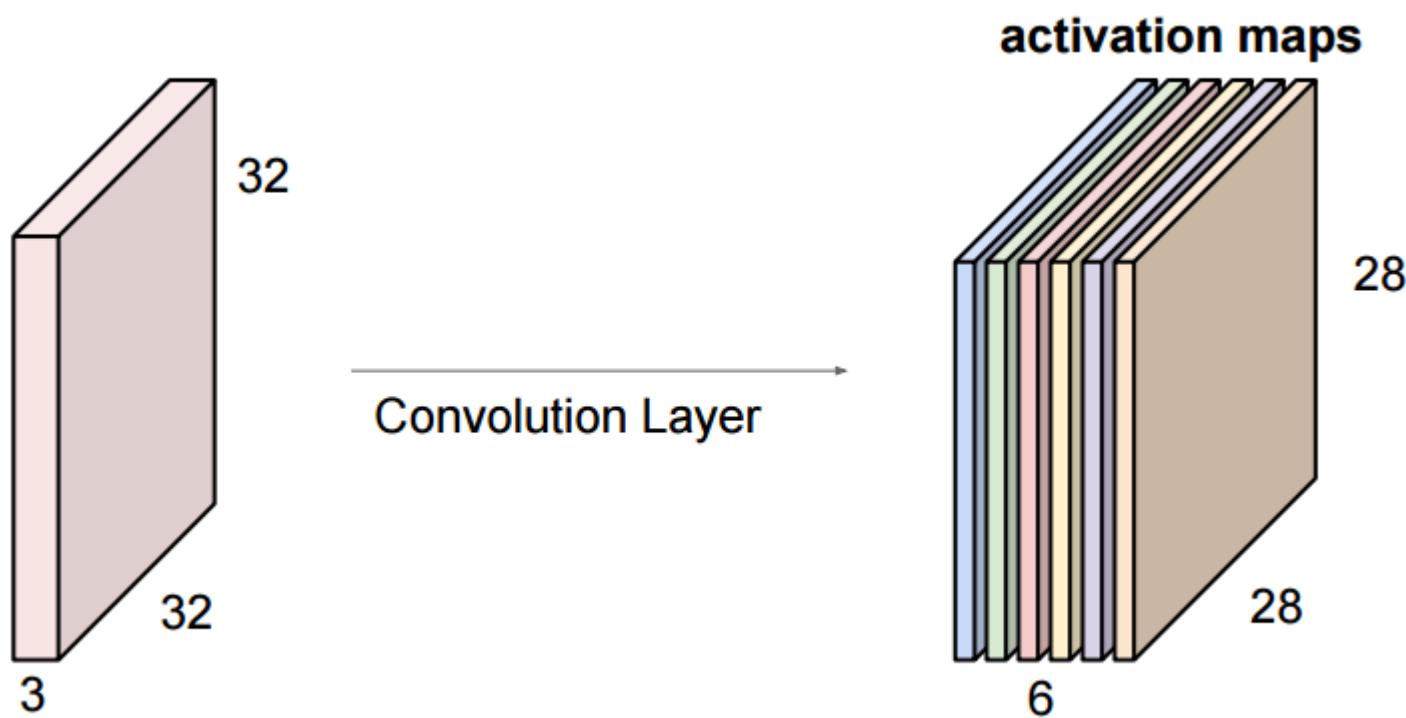
- If have 6 filters (each of size 5x5x3) get 6 activation maps, 28x28 each



- Stack them to get a new $28 \times 28 \times 6$ “image”
- $76 \times 6 = 456$ parameters to learn

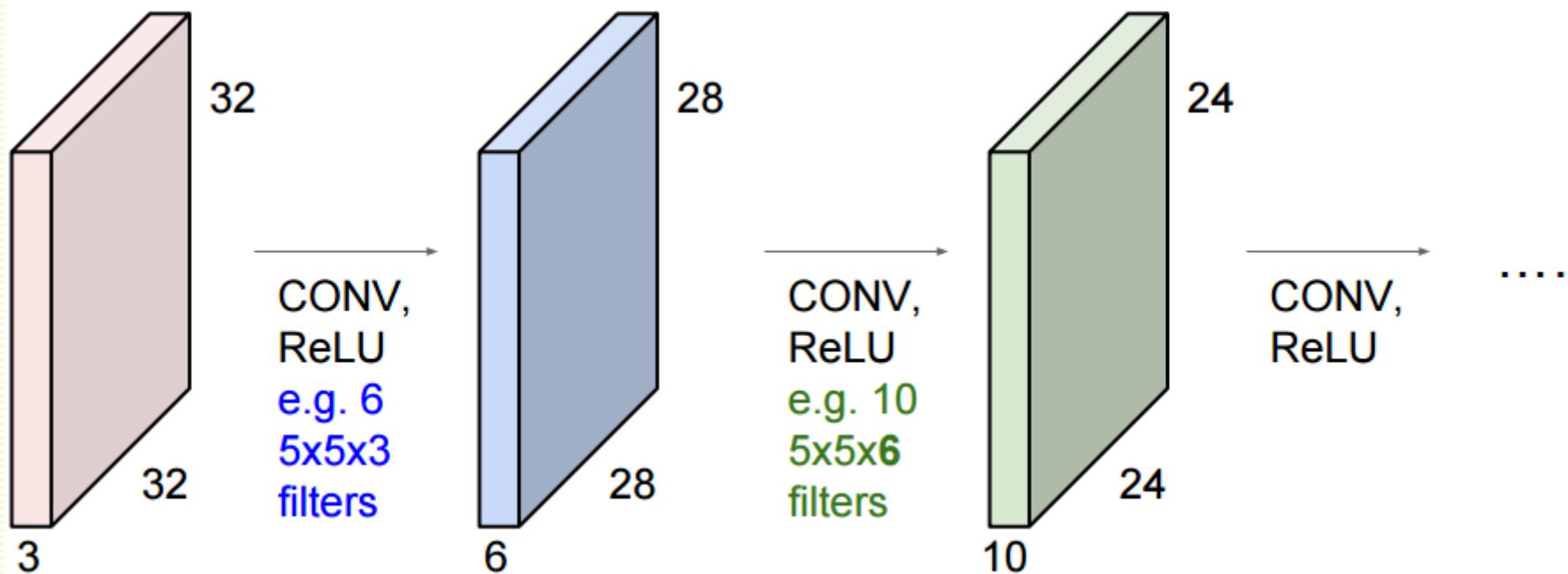
Convolutional Layer

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



Several Convolution Layers

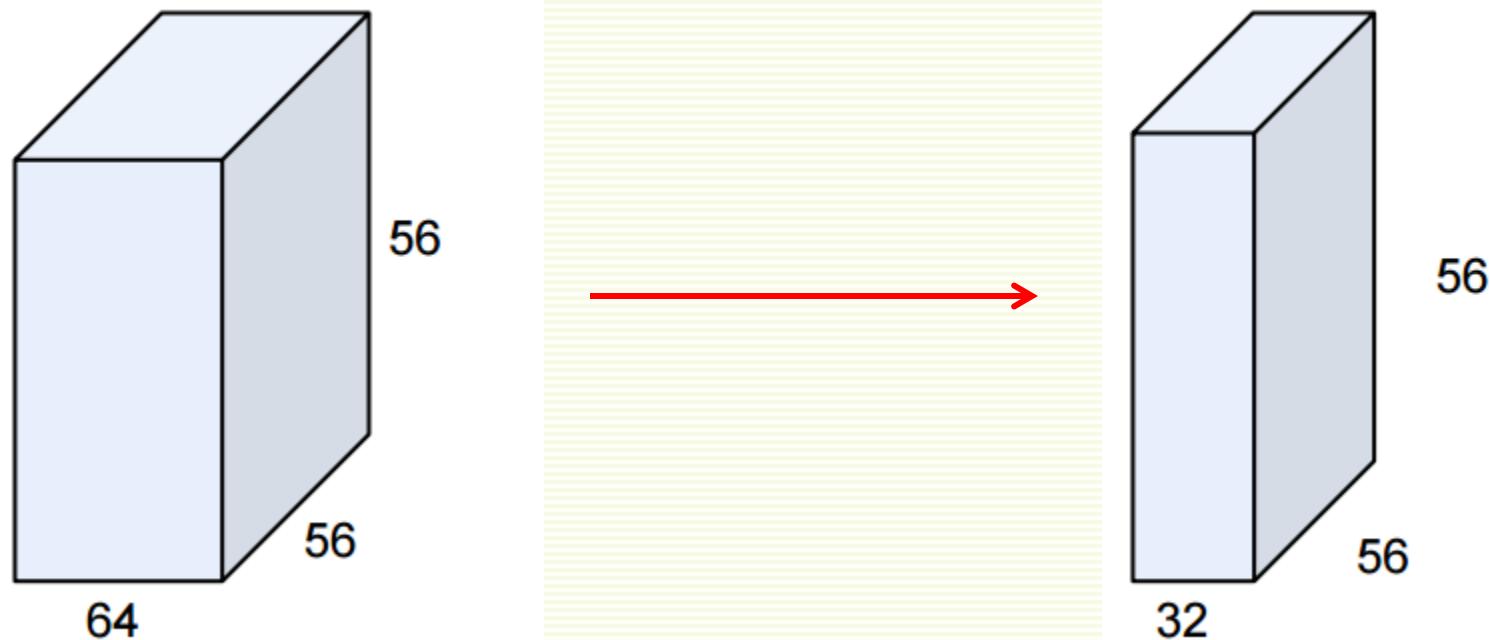
- Construct a sequence of convolution layers interspersed with activation functions



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

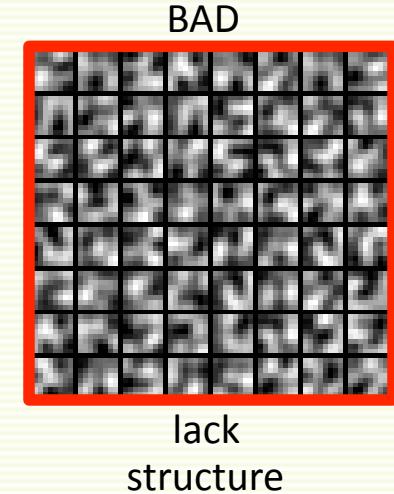
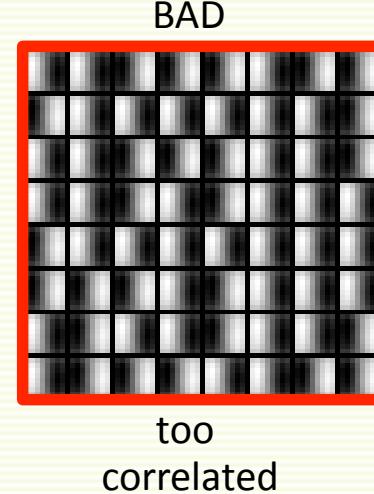
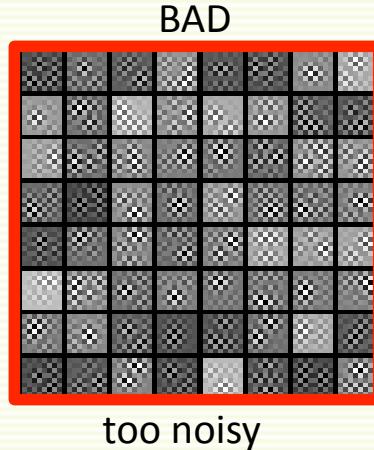
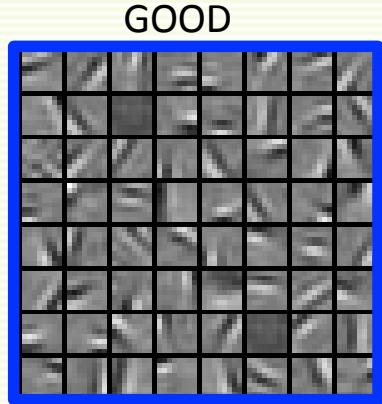
Convolutional Layer

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



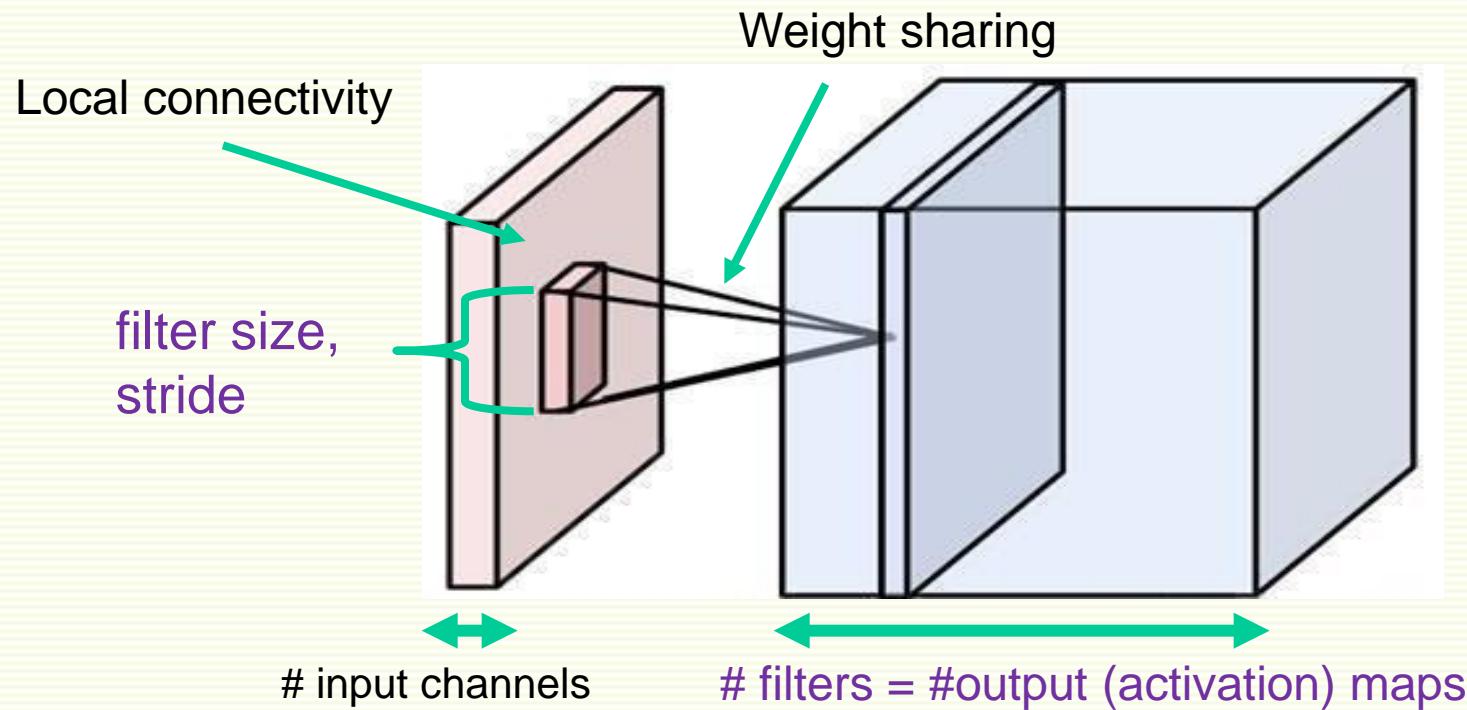
Check Learned Convolutions

- Good training: learned filters exhibit structure and are uncorrelated



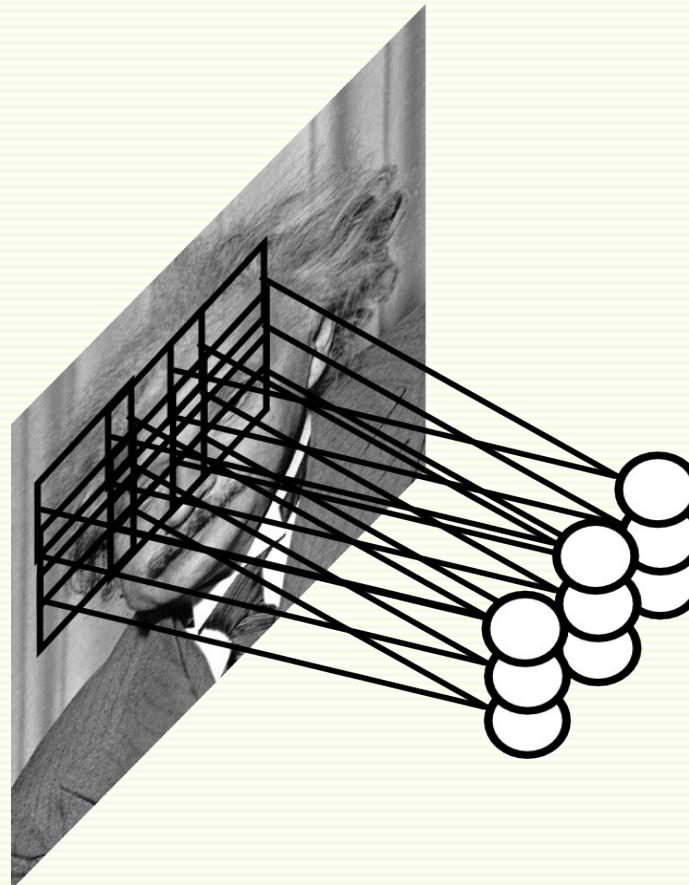
Convolutional Layer Summary

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



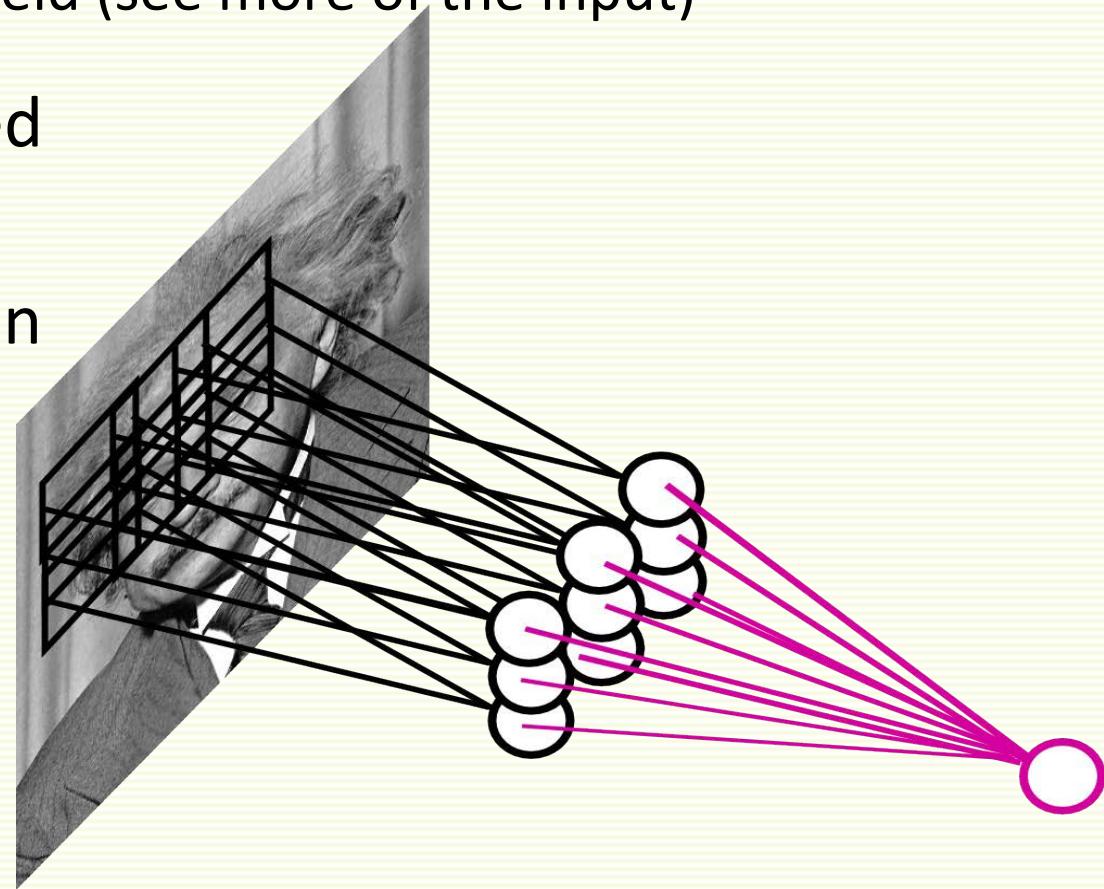
Pooling Layer

- 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

Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

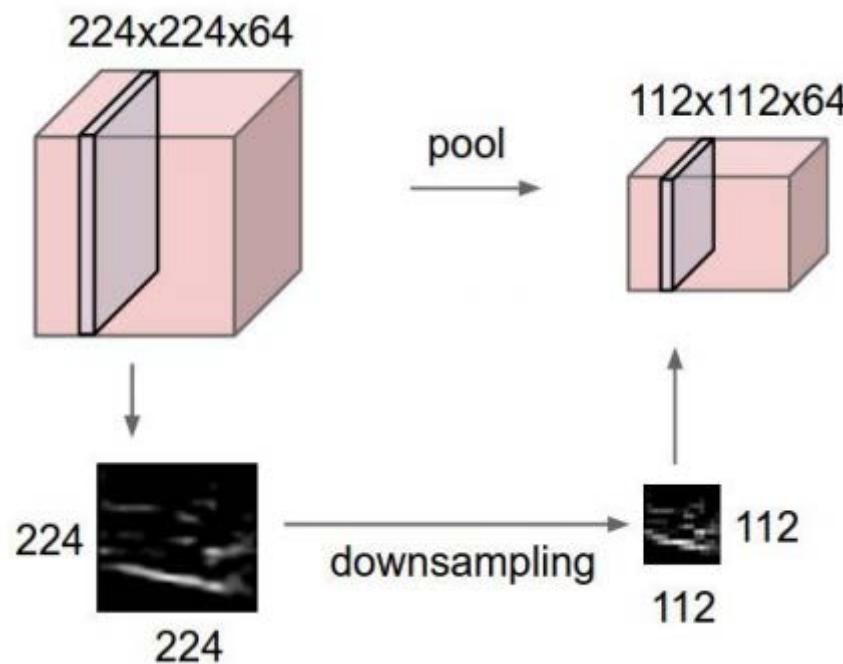
max pool with 2x2 filters
and stride 2



6	8
3	4

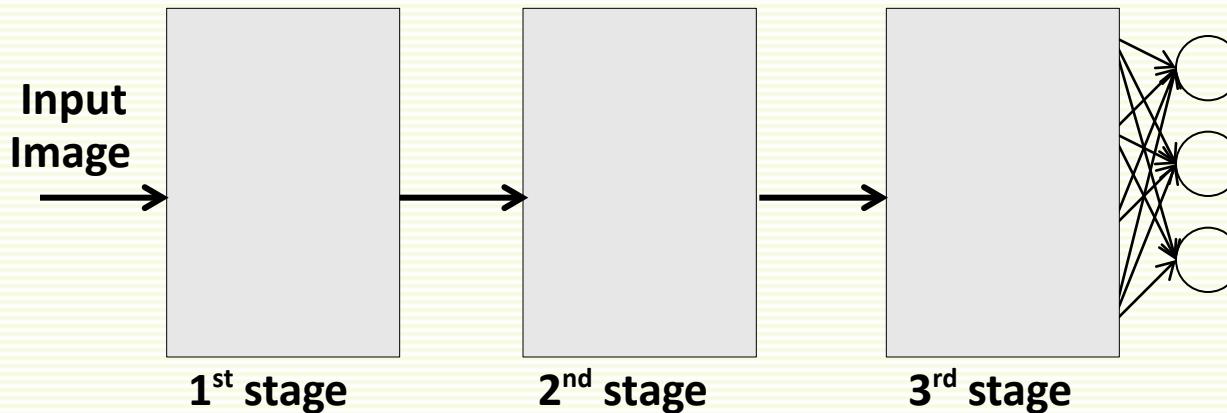
Pooling Layer

- Pooling usually applied to each activation map separately

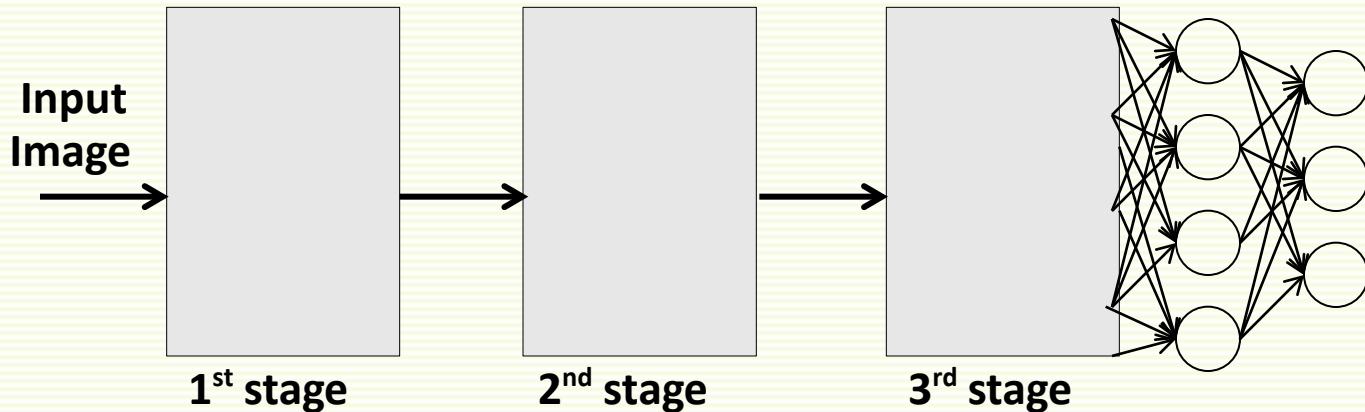


Fully Connected Layer

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

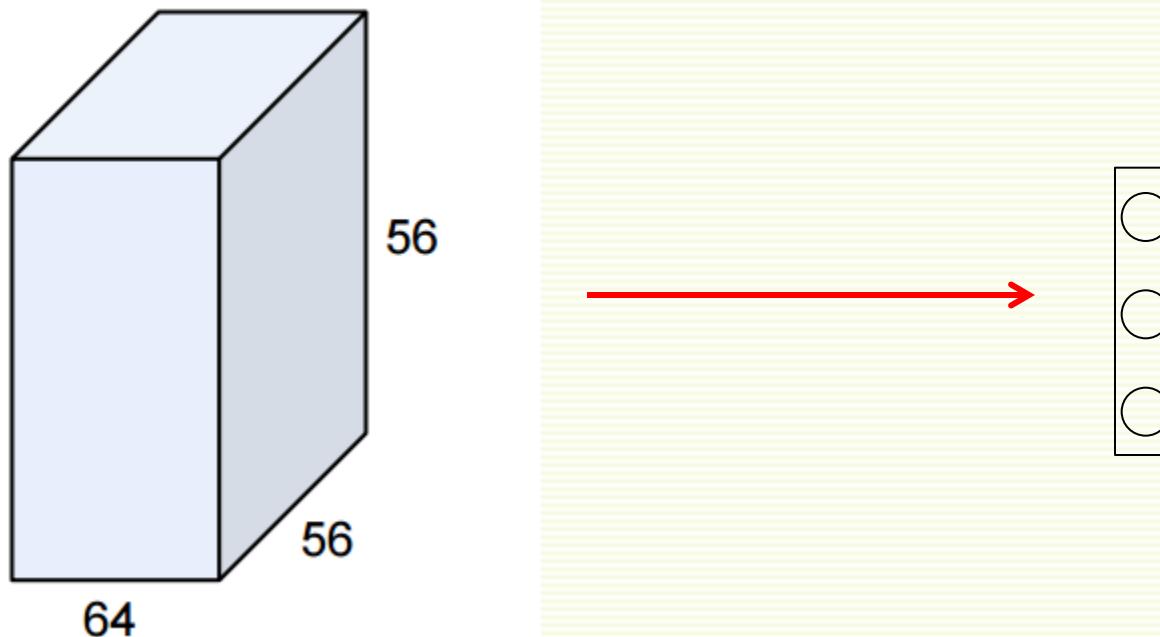


- Can have more than one fully connected layer



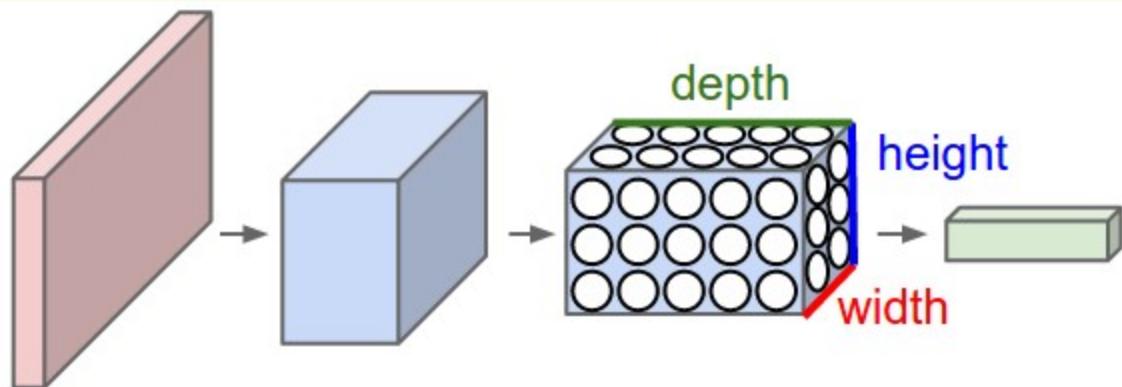
Fully Connected Layer

- Can implement as a convolutional layer
 - input of size $56 \times 56 \times 64$
 - say 3 class problem
 - convolve with 3 filters, each of size $56 \times 56 \times 64$



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

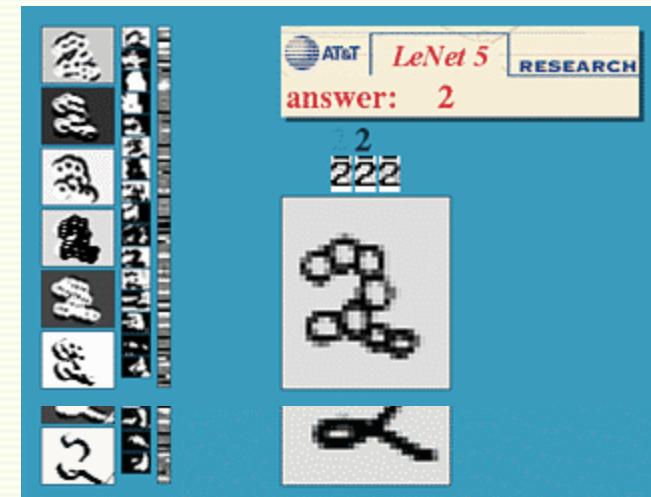


ConvNets: Training

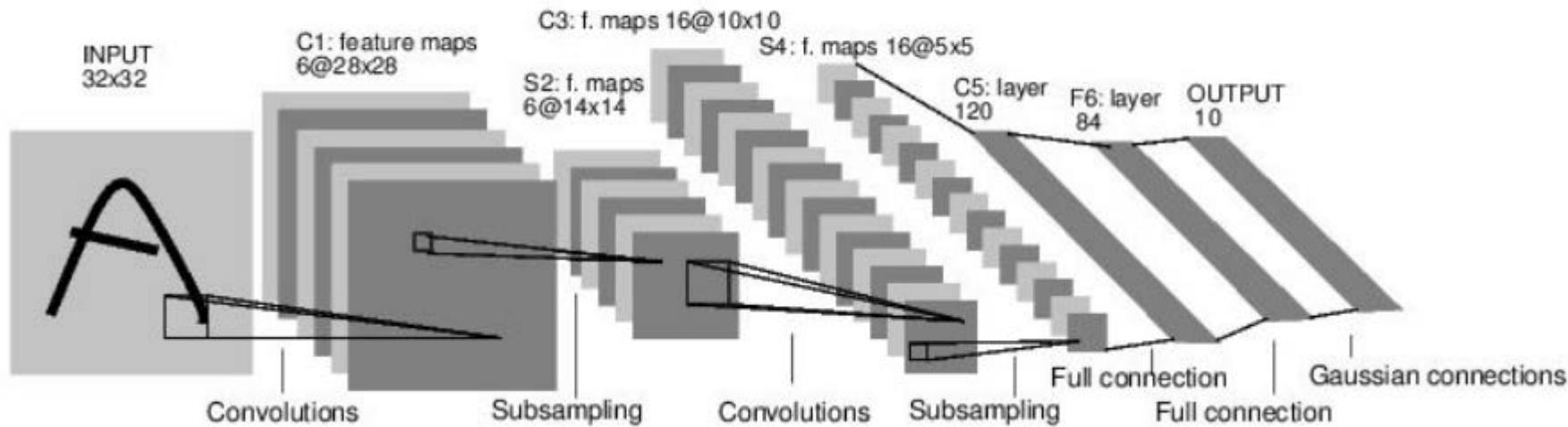
- All Layers are differentiable
- Use standard back-propagation (gradient descent)
- At test time, run only in forward mode

Conv Nets: Character Recognition

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



[LeCun et al., 1998]

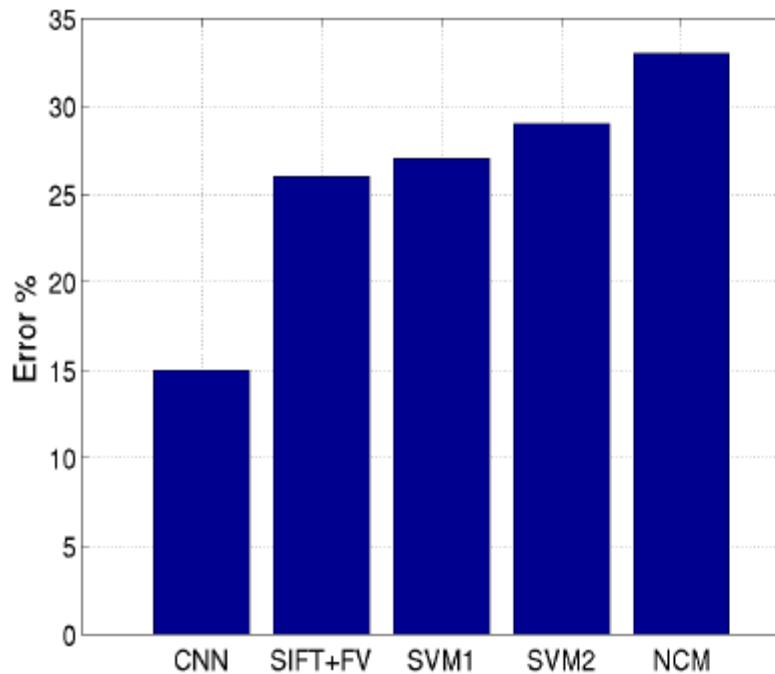


ConvNet for ImageNet

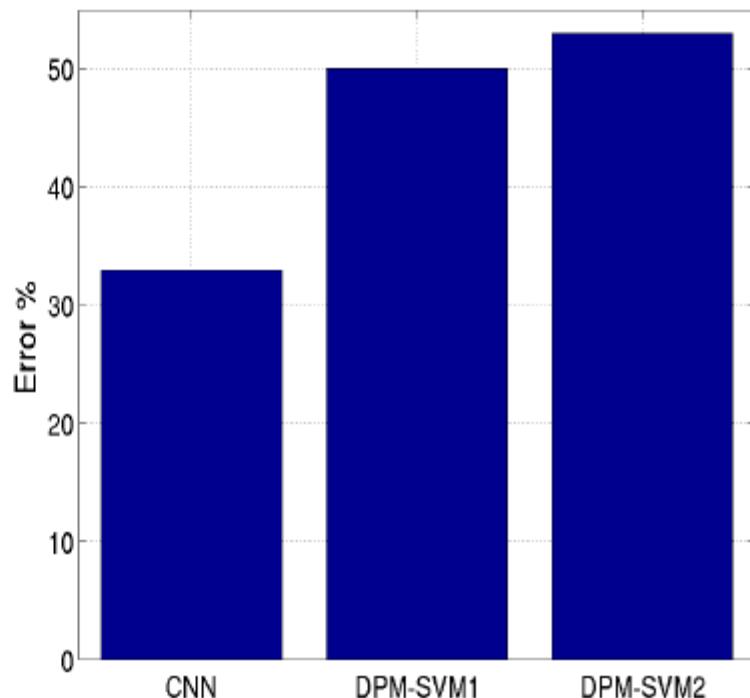
- Krizhevsky et.al.(NIPS 2012) developed deep convolutional neural net of the type pioneered by Yann LeCun
- Architecture
 - 7 hidden layers not counting some max pooling layers
 - the early layers were convolutional
 - the last two layers were globally connected
- Activation function
 - rectified linear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

Results: ILSVRC 2012

TASK 1 - CLASSIFICATION



TASK 2 - DETECTION



ConvNet on Image Classification



cheetah

cheetah

leopard

snow leopard

Egyptian cat



bullet train

bullet train

passenger car

subway train

electric locomotive



hand glass

scissors

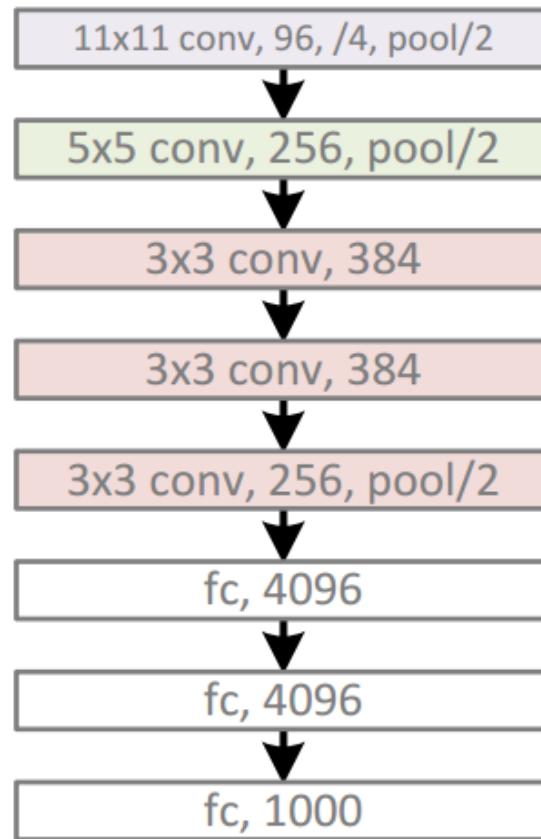
hand glass

frying pan

stethoscope

Krizhevsky et.al. Architecture

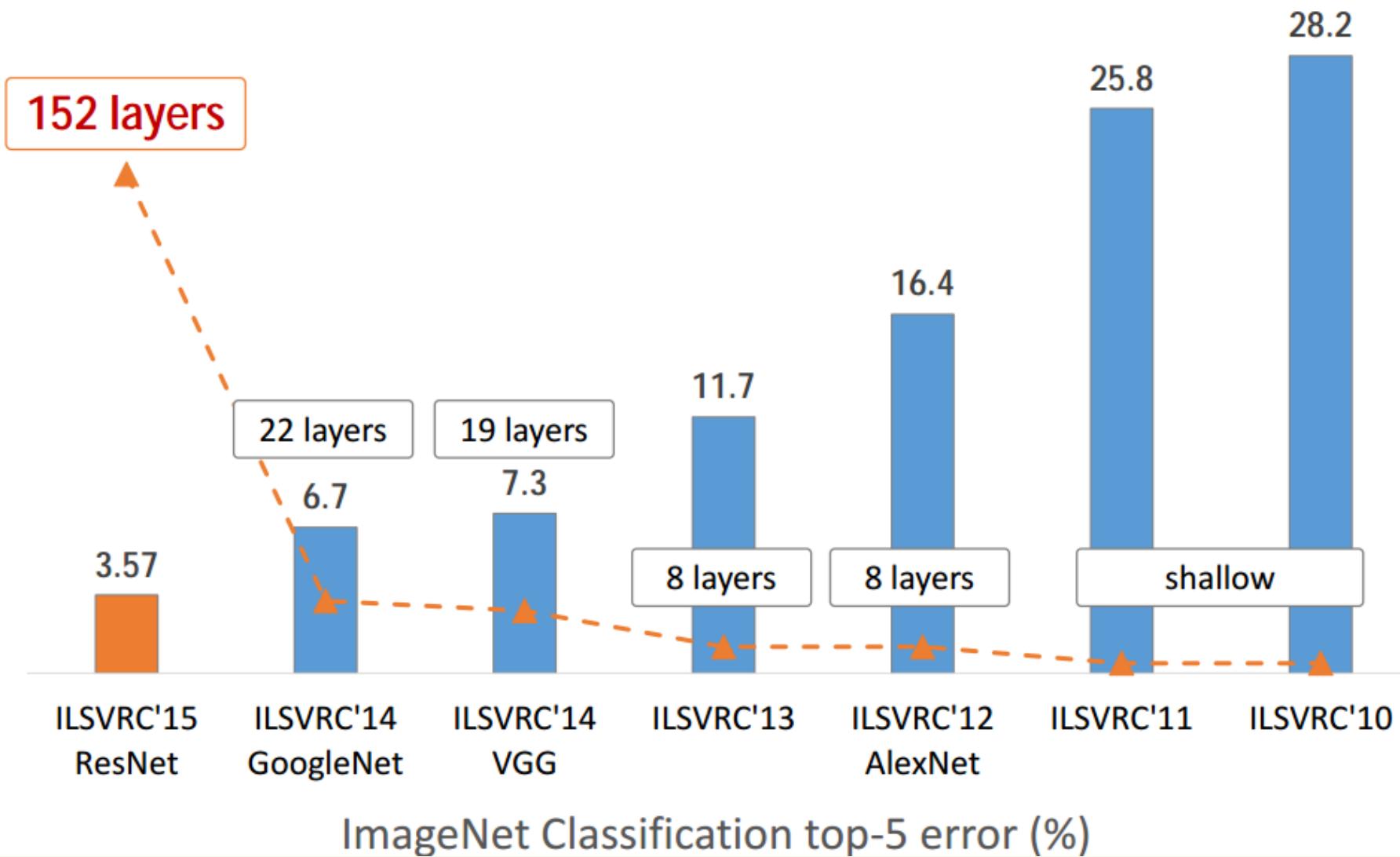
AlexNet, 8 layers
(ILSVRC 2012)

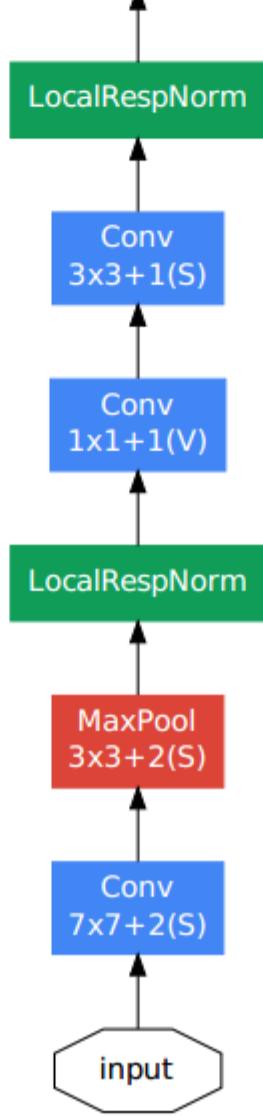


Tricks to Improve Generalization

- 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

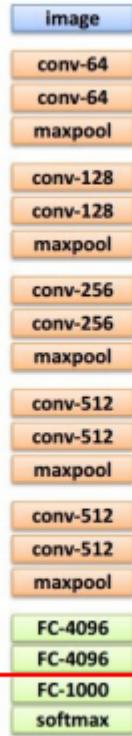




Transfer Learning with CNN



1. Train on ImageNet



2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, “**finetune**” instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.