# Lecture 7: Convolutional Neural Networks

Shuai Li

John Hopcroft Center, Shanghai Jiao Tong University

https://shuaili8.github.io

https://shuaili8.github.io/Teaching/CS410/index.html

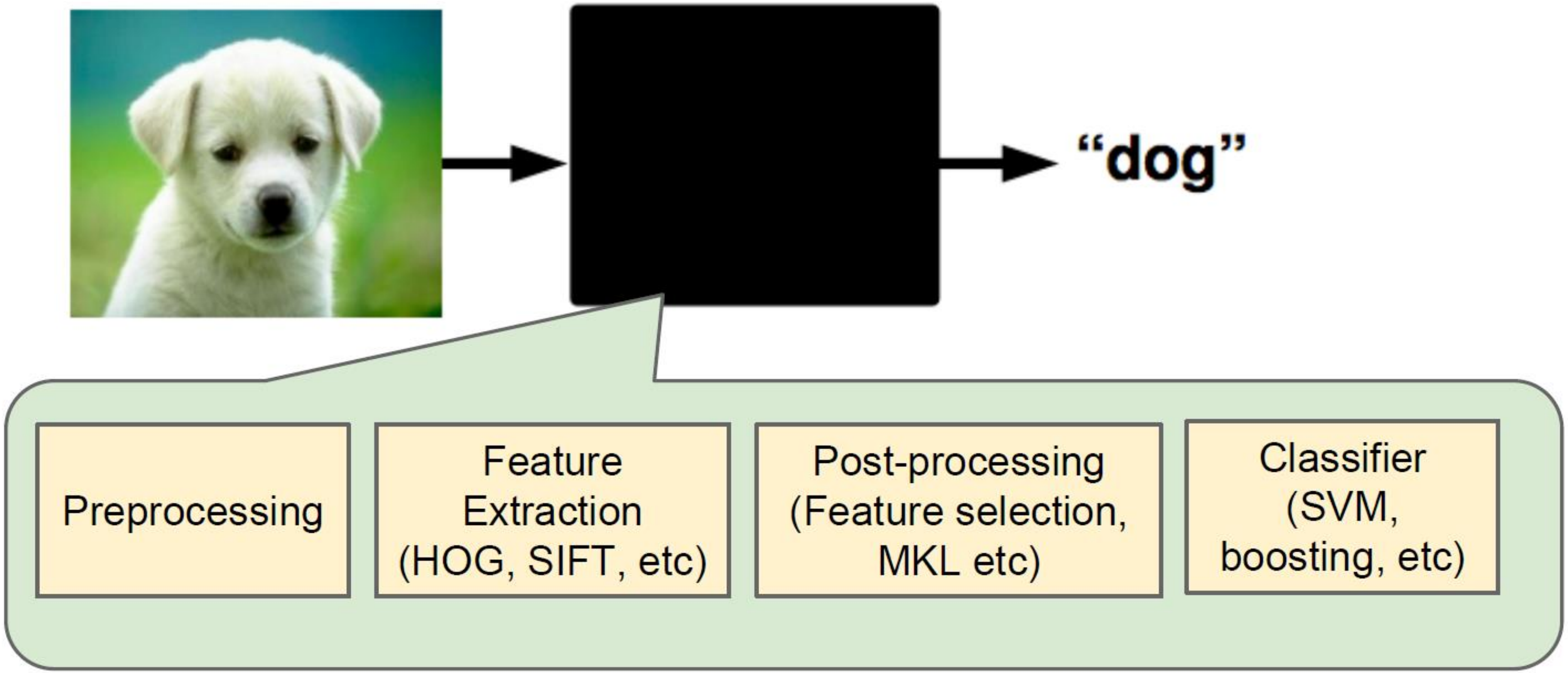
# Outline

- Motivation
- Convolution in math and NN
- Usage and parameters
- Training techniques
- Famous neural networks

# Motivation

# Previous pipeline of pattern recognition

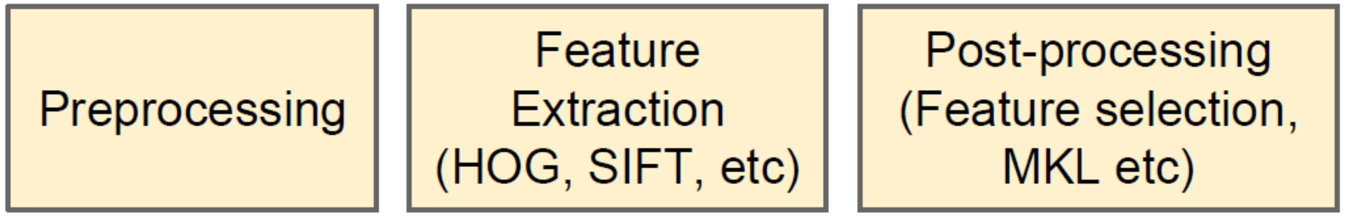
- The black box in a traditional pattern recognition problem



# Hand engineered features

- Feature is of critical importance in machine learning, and there are many things to consider when design the features manually:
  - How to design a feature?
  - What is the best feature?
  - Time and money cost in feature engineering.

- Question: Can feature be learned automatically?



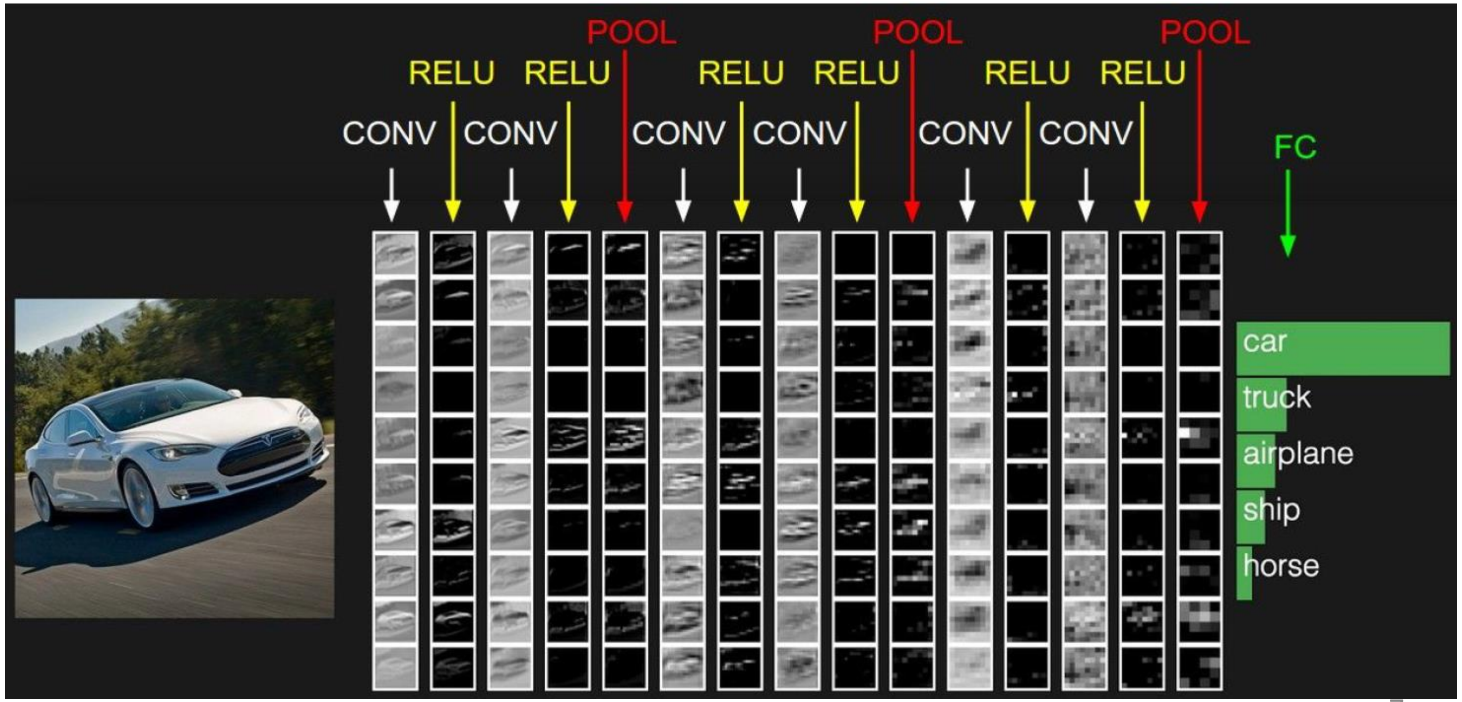
# Objective

- Learn features and classifier at the same time
- Learn an end-to-end recognition system
  - A non-linear map that takes raw pixels directly to labels

# Convolution neural networks

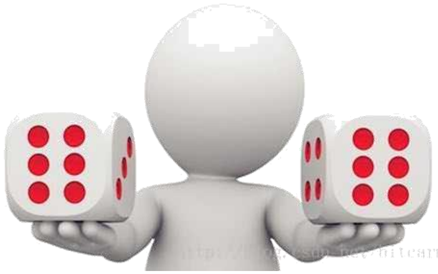
- Is an answer of an end-to-end recognition system
- Contains the following layers with flexible order and repetitions
  - Convolution layer
  - Activation layer (ReLU)
  - Pooling layer

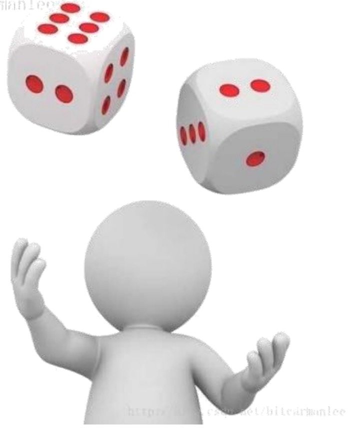
- Example of CNN:



# Convolution in Math and NN

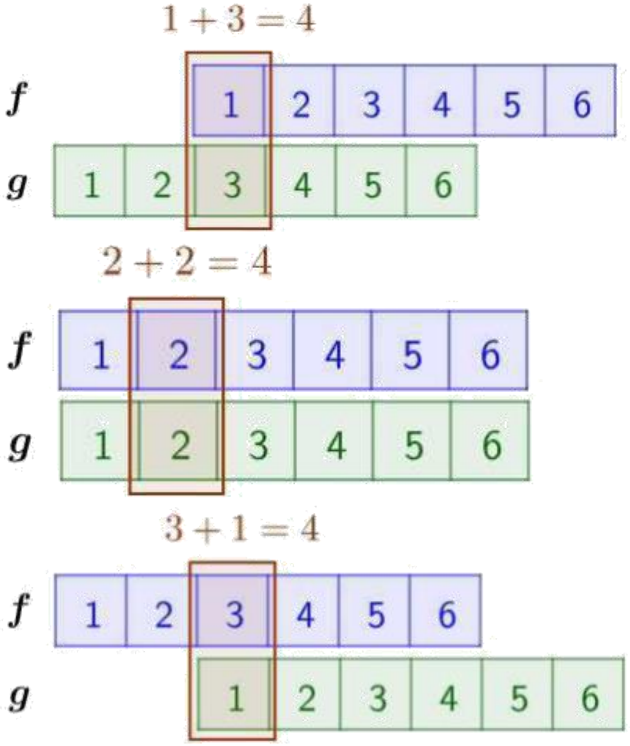
# Example of convolution in math

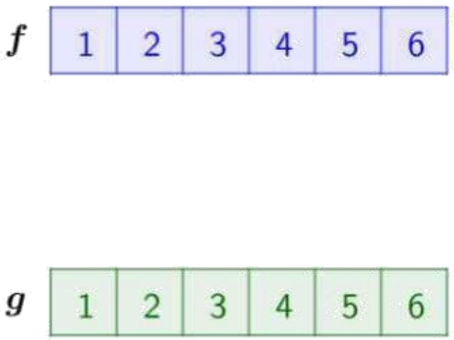




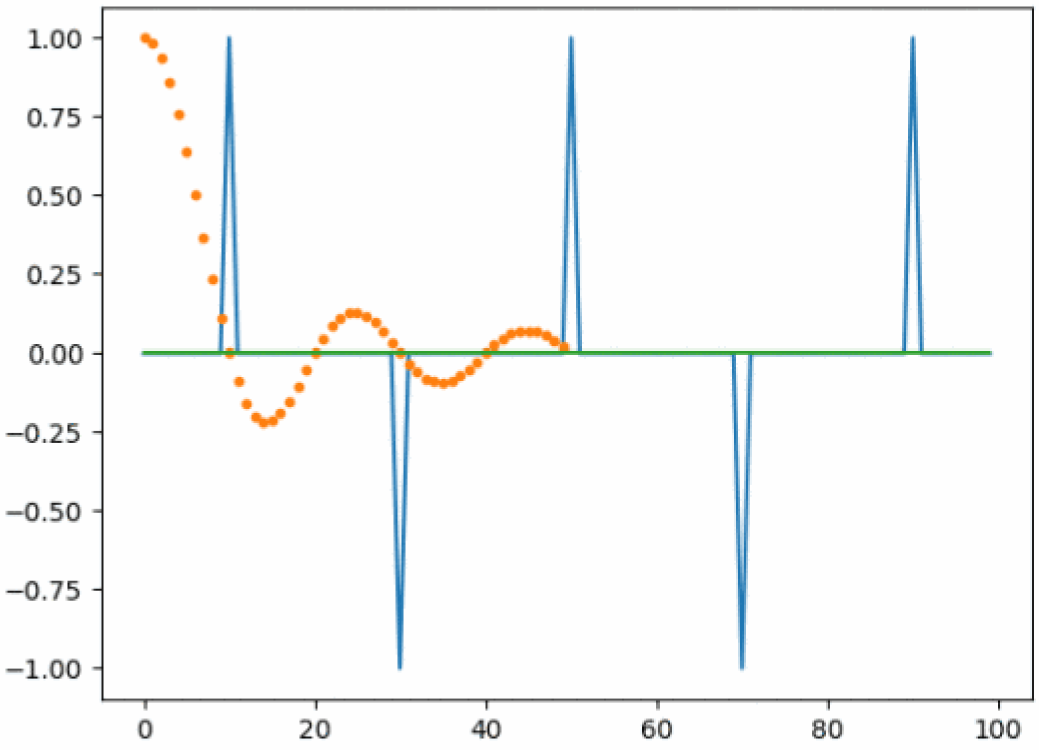
- Suppose we have two dice with probability over $1,2,\ldots,6$ are $f$ and $g$ respectively. We roll two dice. Then what is the probability that the sum of the two dice is $4$?

- $f * g(n) = \sum_i f(i)g(n-i)$



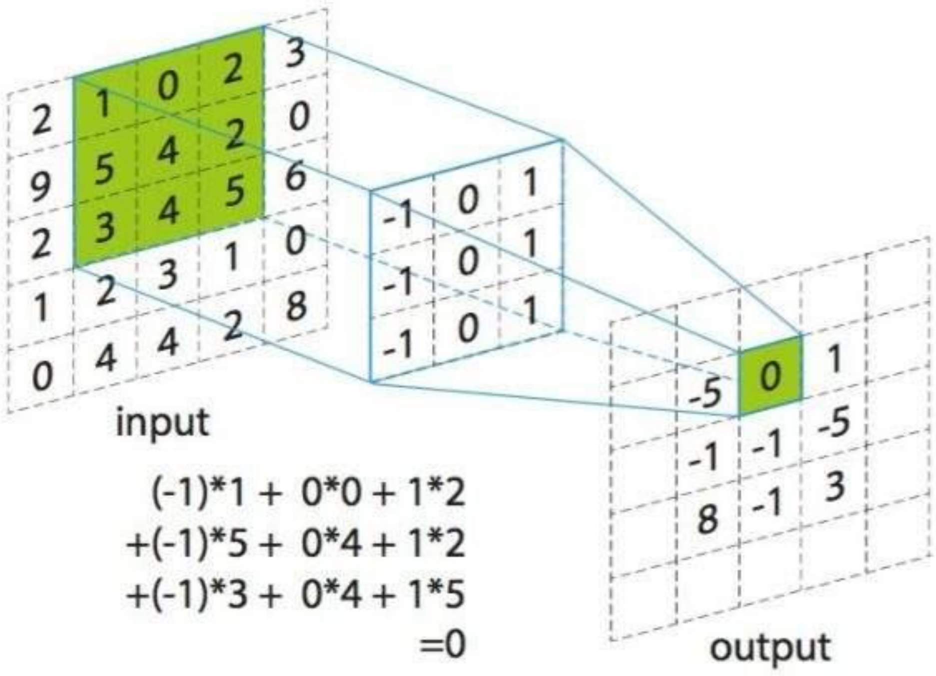
# Convolution in math – continuous case



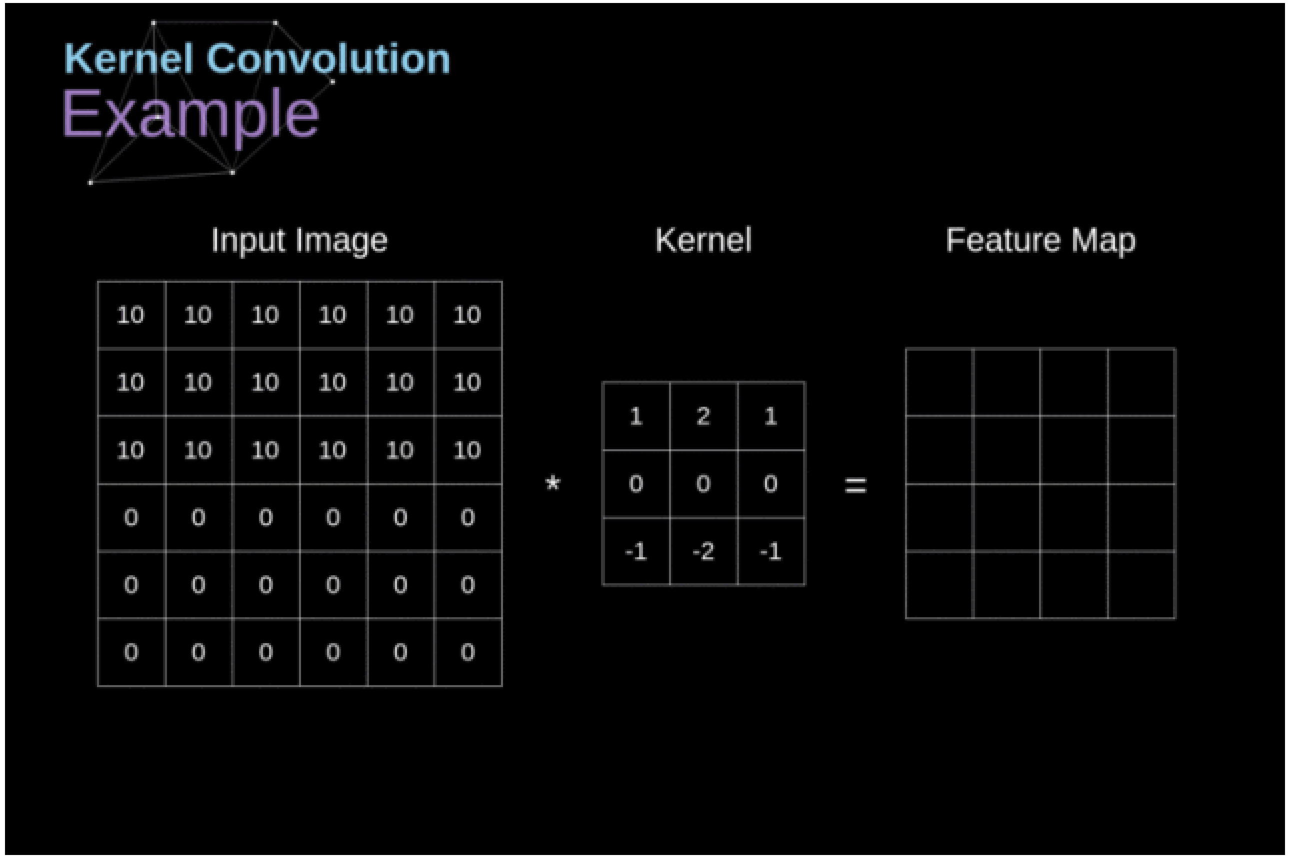

$$f * g(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau$$

# Convolution in neural networks

- Given an input matrix (e.g. an image)
- Use a small matrix (called filter or kernel) to screening the input at every position of the input matrix
- Put the convolution results at corresponding positions

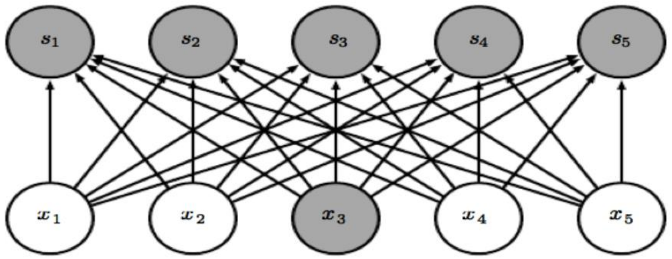


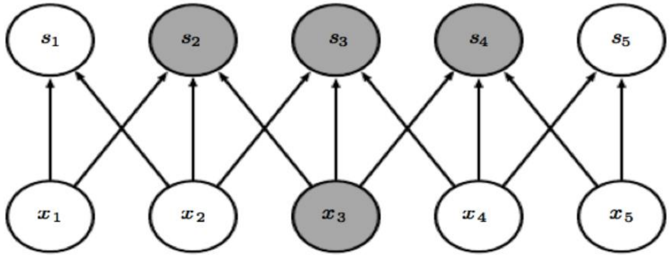
# An animation example



# Advantage – sparse connections

- Less computing burden
- In fully connected layer (top), every $s$ is linked to every input $x$, so there are $5 \times 5 = 25$ connecting edges
- In the convolution layer (bottom) with filter width 3, e.g. $s_2$ is a weighted sum of $x_1, x_2, x_3$, so there is no weight connecting $s_2$ and $x_4, x_5$. In this example, there are $13$ connecting edges



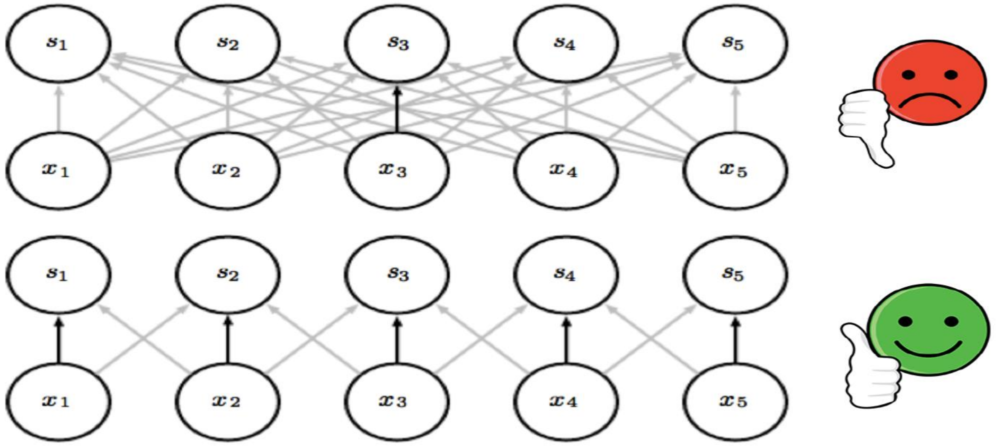



# Advantage – weight sharing

- When moving the filter, we don't change the weights inside the filter and these weights are shared at different connecting edges
- In fully connected layer (top), there are 25 connecting edges. The weights on different edges are different parameters.
- In convolution layer (bottom), there are 13 connecting edges. But since

$$s_2 = w_1 x_1 + w_2 x_2 + w_3 x_3$$
$$s_3 = w_1 x_2 + w_2 x_3 + w_3 x_4$$

  the number of different weights is even smaller. In this case, there are 3 different weights (just the size of the filter!). E.g. the weights on the black arrows are the same.

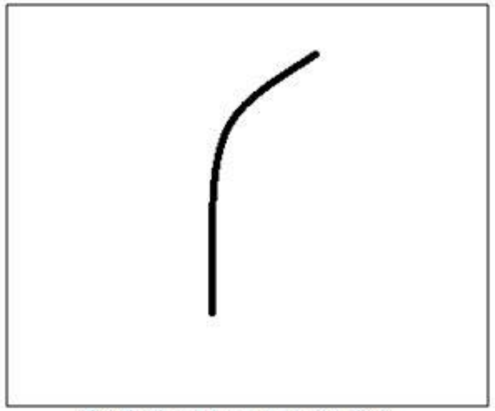


# Interpretation of convolution

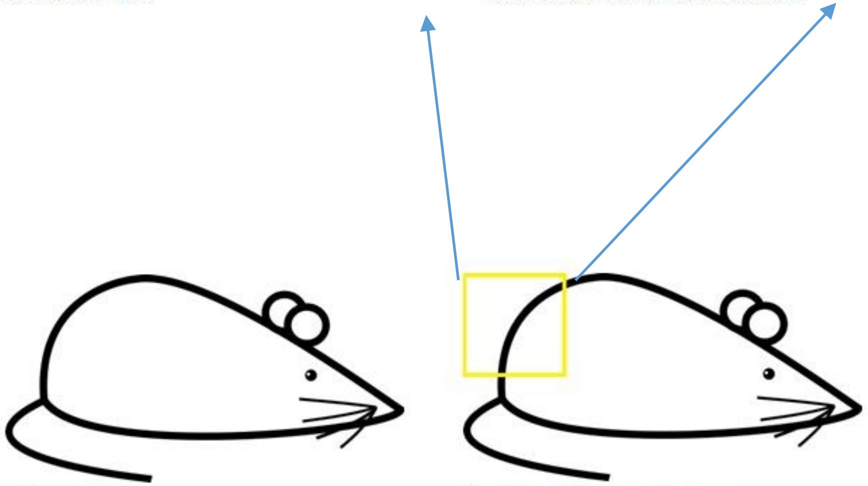
- Convolution can be used to find an area with particular patterns!
- Example:
  - The filter in the left represents the edge in the right, which is the back of a mouse

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Pixel representation of filter



Visualization of a curve detector filter



Original image

Visualization of the filter on the image

# Interpretation of convolution (cont.)

- When the filter moves to the back of the mouse, the convolution operation will generate a very large value

Visualization of the receptive field

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 30 |
| 0 | 0 | 0 | 0 | 50 | 50 | 50 |
| 0 | 0 | 0 | 20 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |

Pixel representation of the receptive field

*

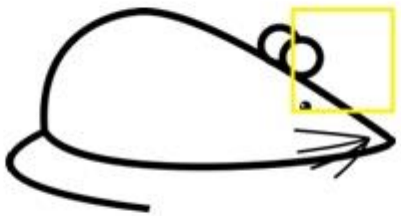
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

# Interpretation of convolution (cont.)

- When the filter moves to other positions, it will generate small values



Visualization of the filter on the image

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 40 | 0 | 0 | 0 | 0 | 0 |
| 40 | 0 | 40 | 0 | 0 | 0 | 0 |
| 40 | 20 | 0 | 0 | 0 | 0 | 0 |
| 0 | 50 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 50 | 0 | 0 | 0 | 0 |
| 25 | 25 | 0 | 50 | 0 | 0 | 0 |

Pixel representation of receptive field

*

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Pixel representation of filter

Multiplication and Summation = 0

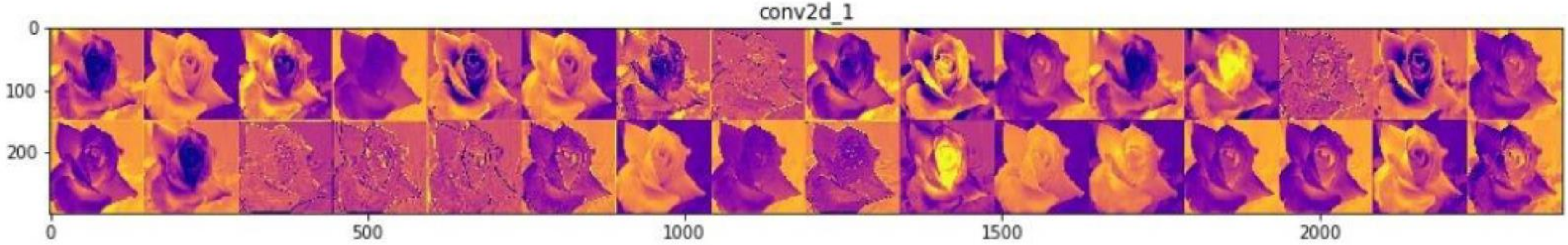
# Visualization

- Train the InceptionV3 model (by Google) on the ImageNet dataset
- Then test it on a flower image from the test dataset
  - Example input image:

- Then let's look at the outputs of different convolutional layers

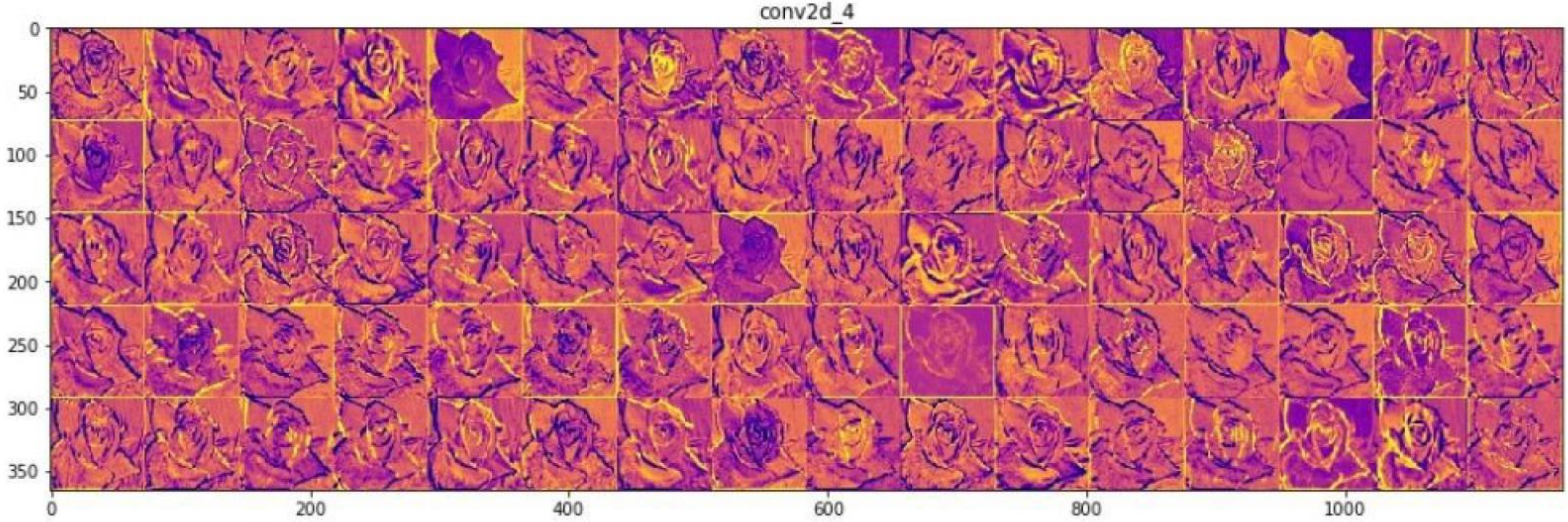
# Visualization (cont.)

- The outputs of the filters in the first layer



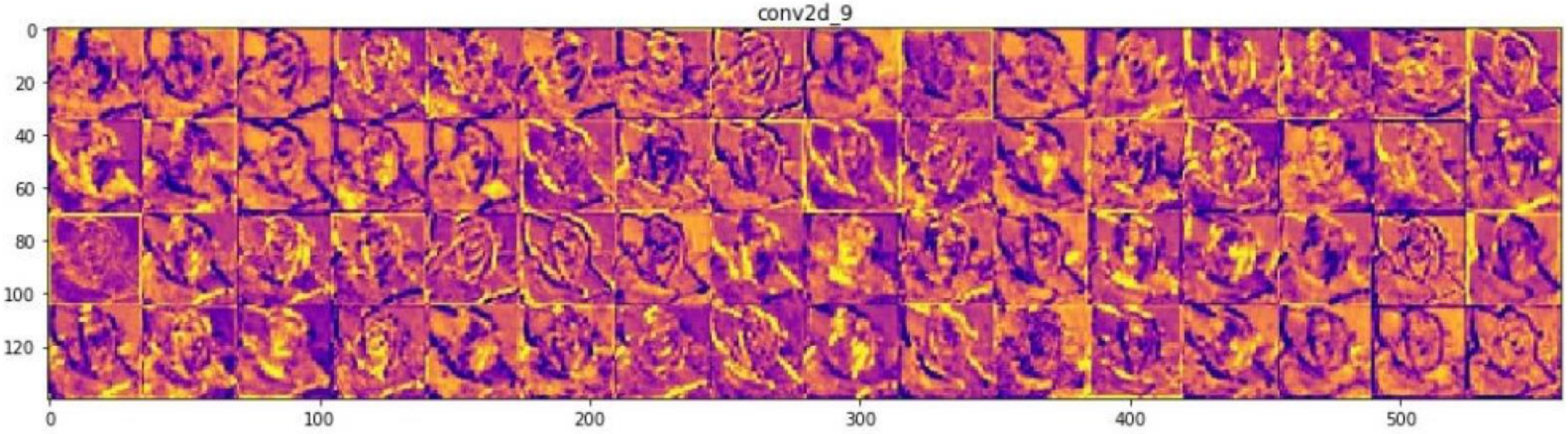
# Visualization (cont.)

- The outputs of the filters in the fourth layer



# Visualization (cont.)

- The outputs of the filters in the ninth layer



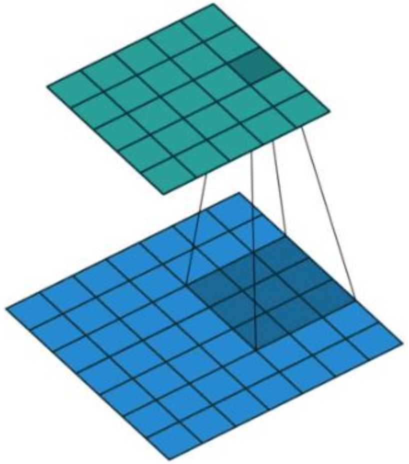
# Visualization (cont.)

- Summary
  - The filters in the deeper layers characterize more abstract patterns
  - Layers that are deeper in the network visualize more training data specific features
  - While the earlier layers tend to visualize general patterns like edges, texture, background

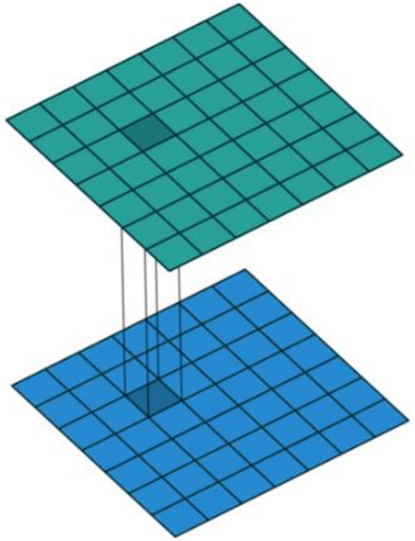
# Usage and parameters

# $1 \times 1$ convolution

- Here we introduce a special filter, which as a size of $1 \times 1$



Convolution with large kernel

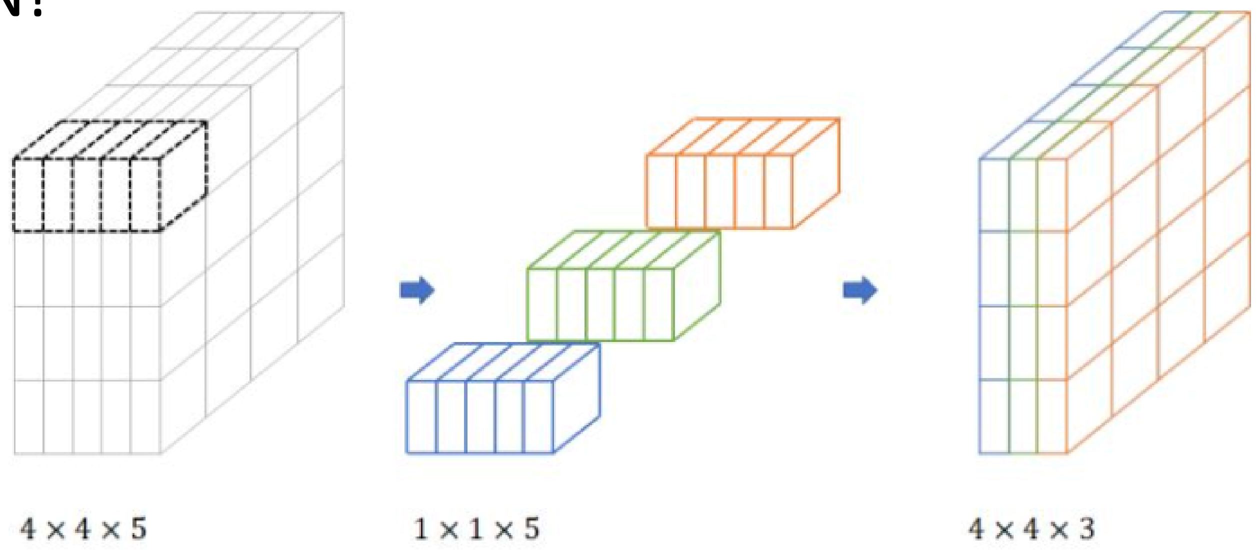


Convolution with 1*1 kernel

- The $1 \times 1$ convolution cannot detect edges with any shape, so is it really useful? Or is it redundant?

# $1 \times 1$ convolution (cont.)

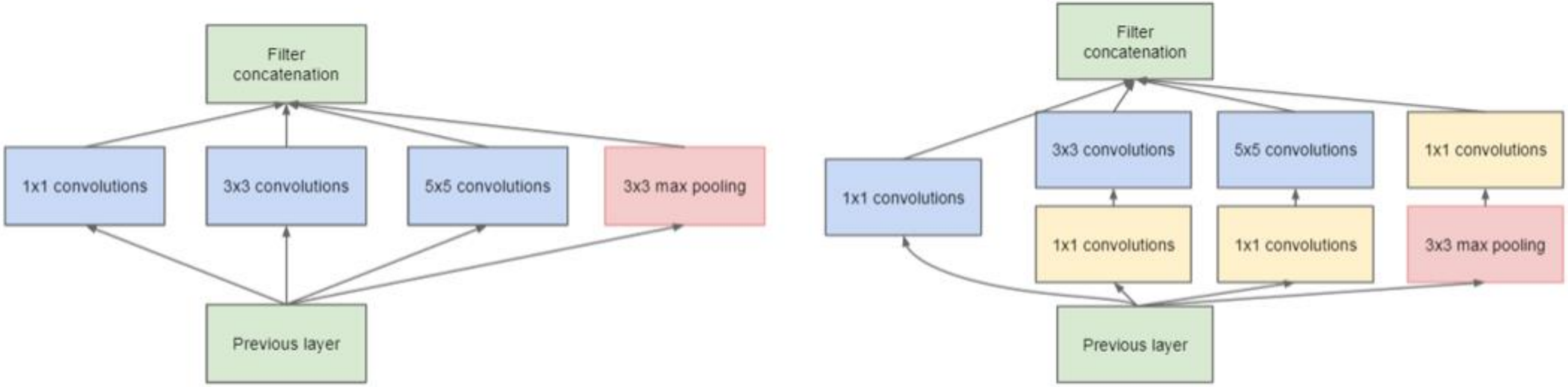
- The $1 \times 1$ convolution is very useful as it can reduce the computation complexity in CNN!




- Help reduce the number of channels
- In the example above, the size of the input data is reduced from $4 \times 4 \times 5$ to $4 \times 4 \times 3$
- Usually we assume the depth of the filter is the same with depth of data

# $1 \times 1$ convolution (cont.)

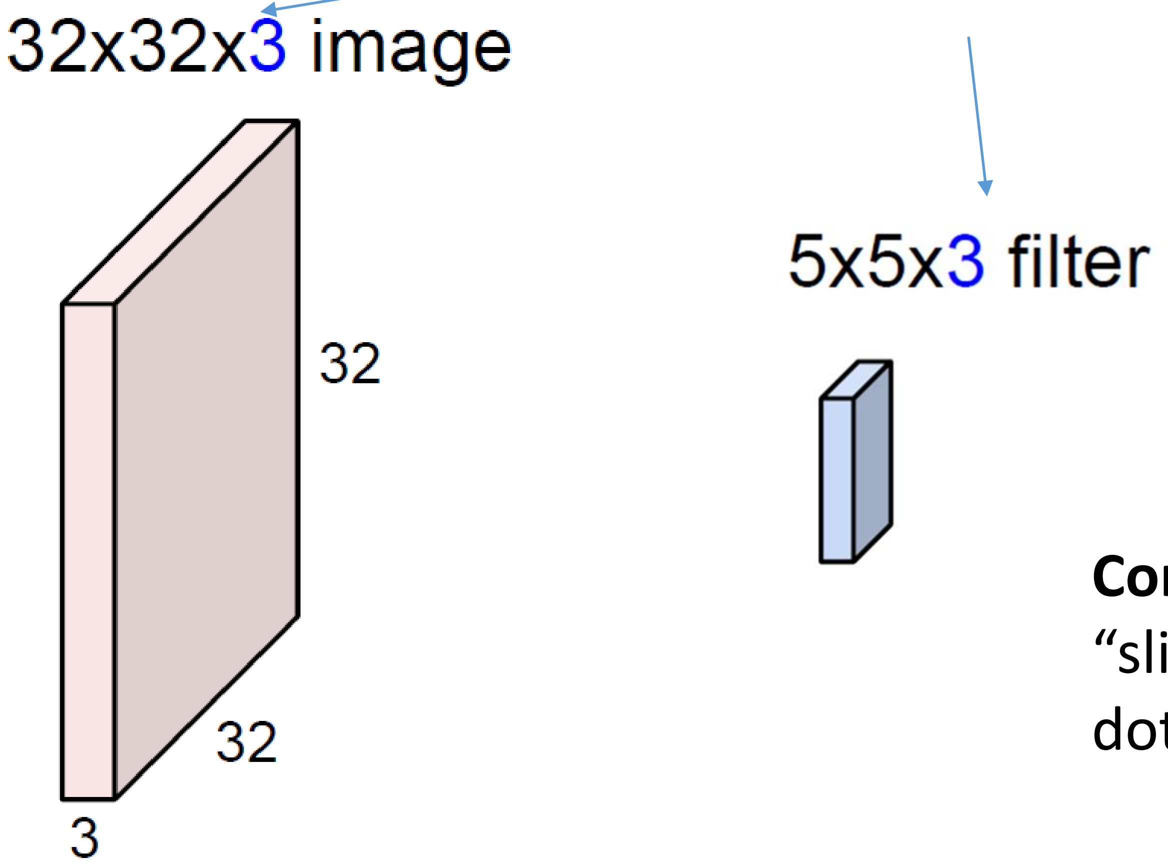
- $1 \times 1$ convolution filter is usually followed by $3 \times 3$ or other bigger filters. In this way, the computational complexity is greatly reduced
- This architecture is used in Google's inception model




(a) Inception module, naïve version (b) Inception module with dimension reductions

# Filter depth

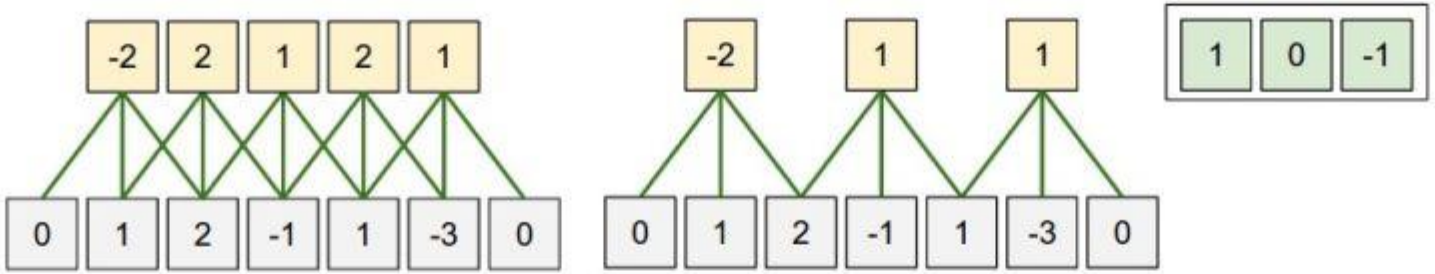
filters always extend the full depth of the input volume




**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”

# Stride

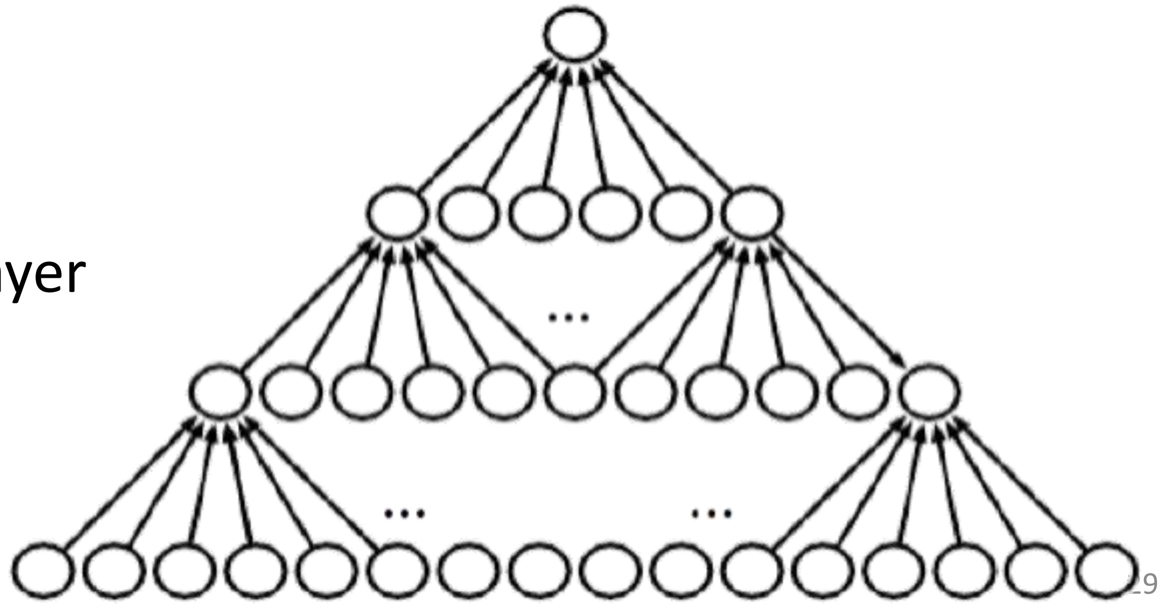
- The distance that the filter is moved in each step
- Examples of stride=1 and stride=2



# Padding

- A solution to the problem of data shrinking
- Data shrinking
  - As convolution can only happen within the border of the input data, the size of the data will become smaller as the network becomes deeper

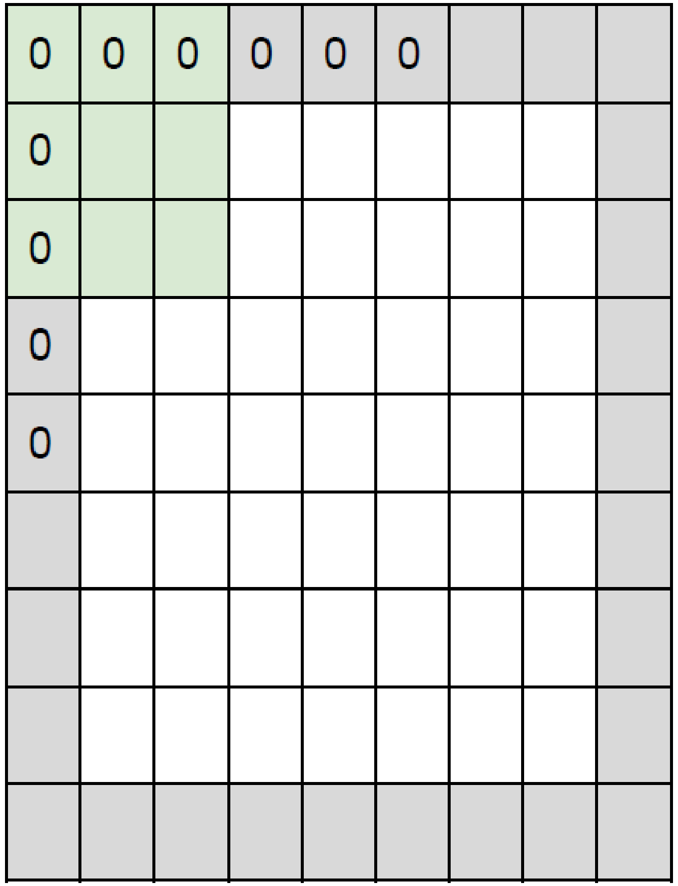
- 1D Example:
  - suppose the filter width is 6,
    the data will shrink 5 pixel each layer



# Padding (cont.)

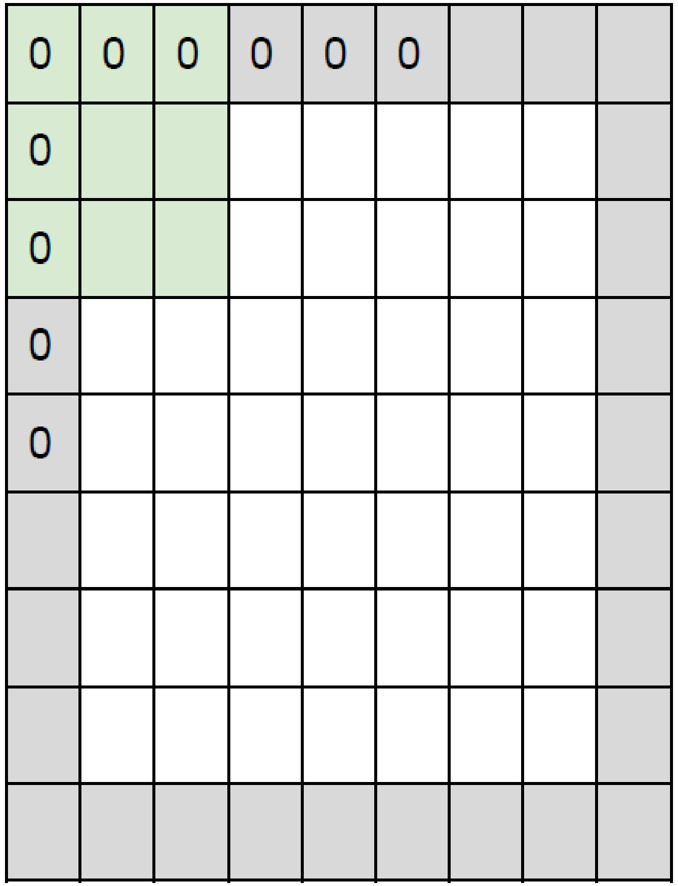
- Add numbers (usually zero, called zero padding) around the input data to make sure that the size of the output data is the same as that of the input data
- Example: Left padding = 3, right padding = 2
- Usually left padding + right padding = filter width – 1

# Example



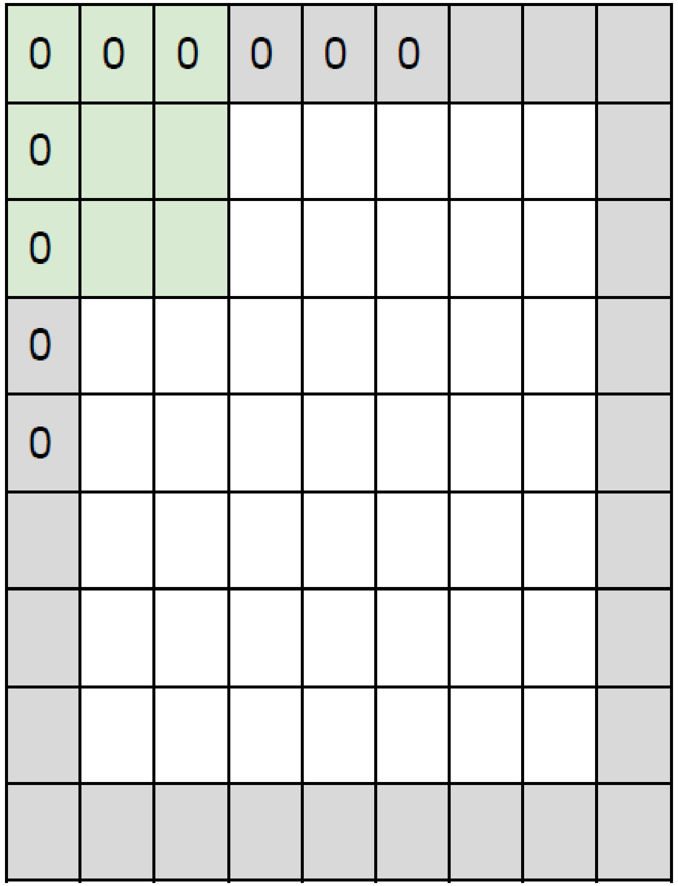

- Input 7x7 (white area)
- **3x3** filter, applied with **stride 1**
- **pad with 1 pixel** border (dark area)
- => what is the size of the output?

# Example (cont.)




- Input 7x7 (white area)
- **3x3** filter, applied with **stride 1**
- **pad with 1 pixel** border (dark area)
- => what is the size of the output?
- **7x7 output!**

# Example (cont.)




- Input 7x7 (white area)
- **3x3** filter, applied with **stride 1**
- **pad with 1 pixel** border (dark area)
- => what is the size of the output?
- **7x7 output!**
- In general, convolution layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
- e.g. F = 3 => zero pad with 1

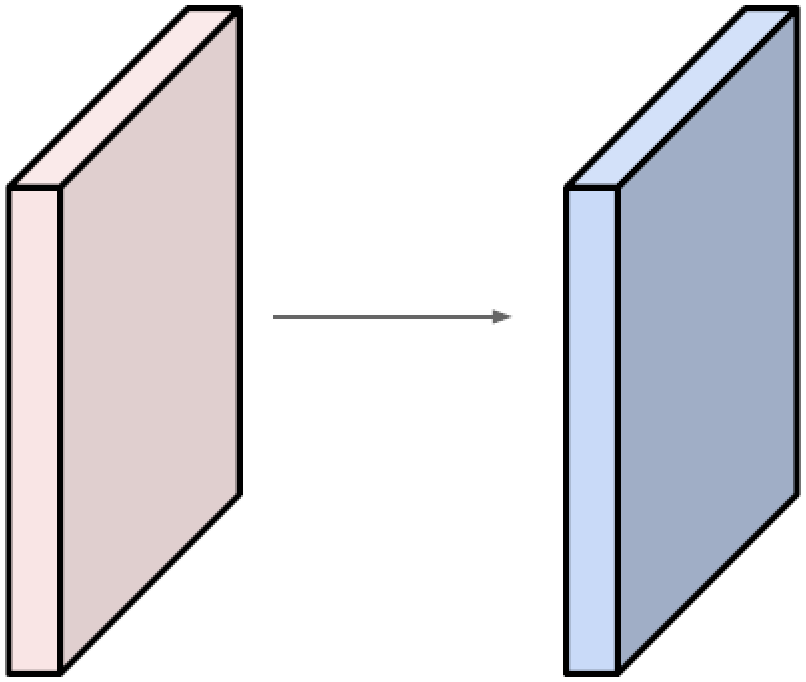
  F = 5 => zero pad with 2

  F = 7 => zero pad with 3

# Example 2

- Input volume: $32 \times 32 \times 3$
- 10 $5 \times 5$ filters with stride 1, pad 2

- Output volume size: ?

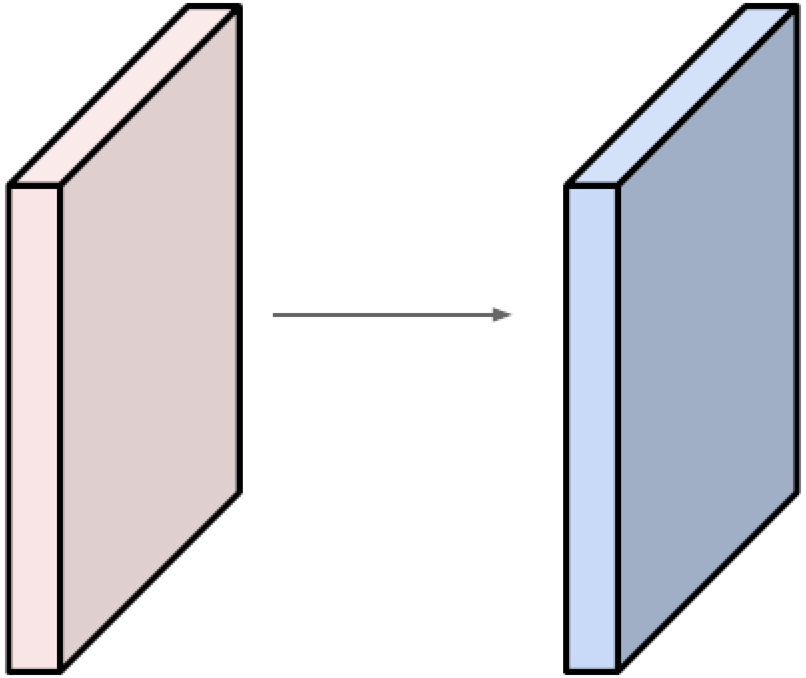


# Example 2 (cont.)

- Input volume: $32 \times 32 \times 3$
- $10\ 5 \times 5$ filters with stride $1$, pad $2$

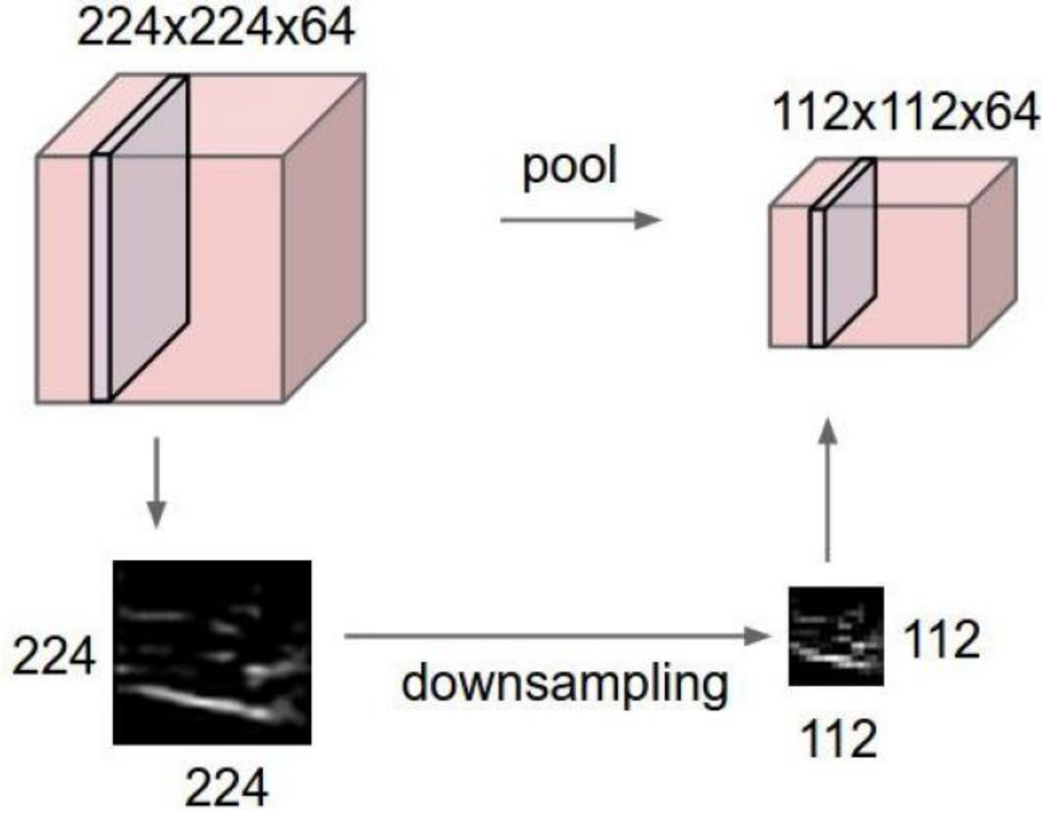
- Output volume size: ?

- Filter size $5$, pad = $(F - 1)/2$,
  so keep the size $32 \times 32$ spatially
- $10\ 5 \times 5$ filters means $10\ 5 \times 5 \times 3$ filters
- So $32 \times 32 \times 10$



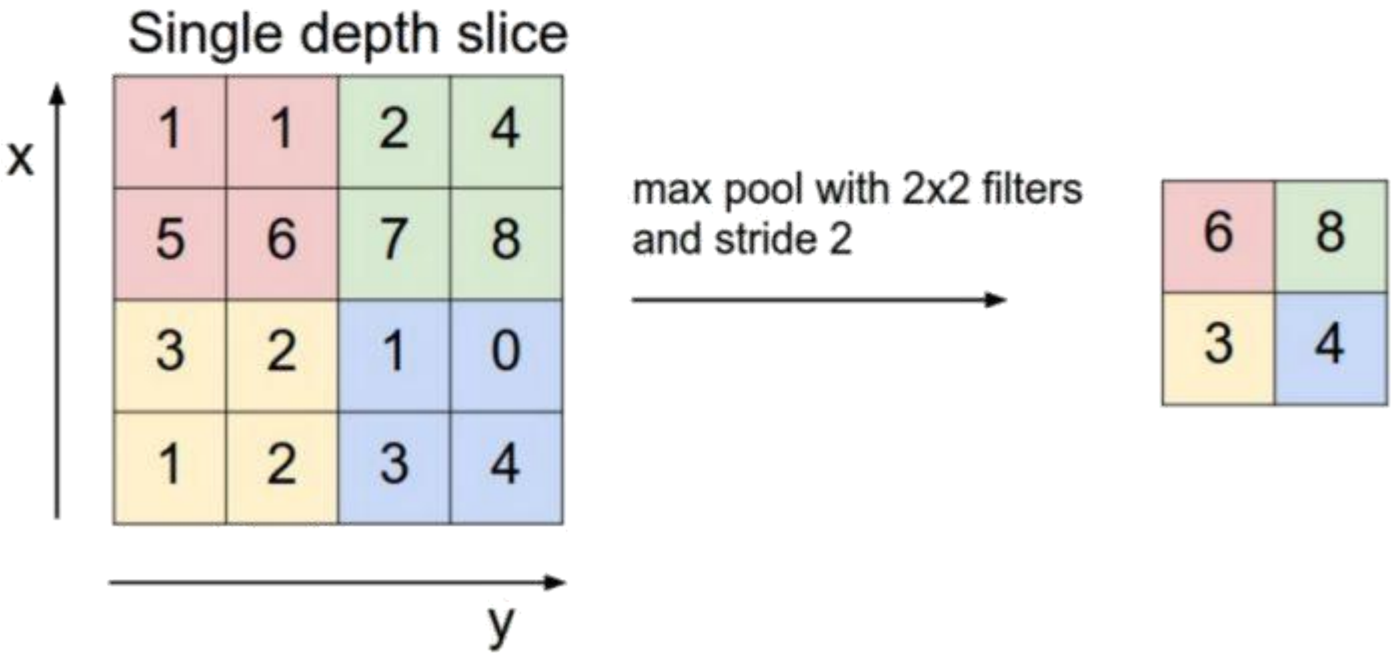
# Pooling layer

- Make the representations denser and more manageable
- Operate over each activation map independently:



# Example of pooling layer

- Pooling of size $2 \times 2$ with stride 2

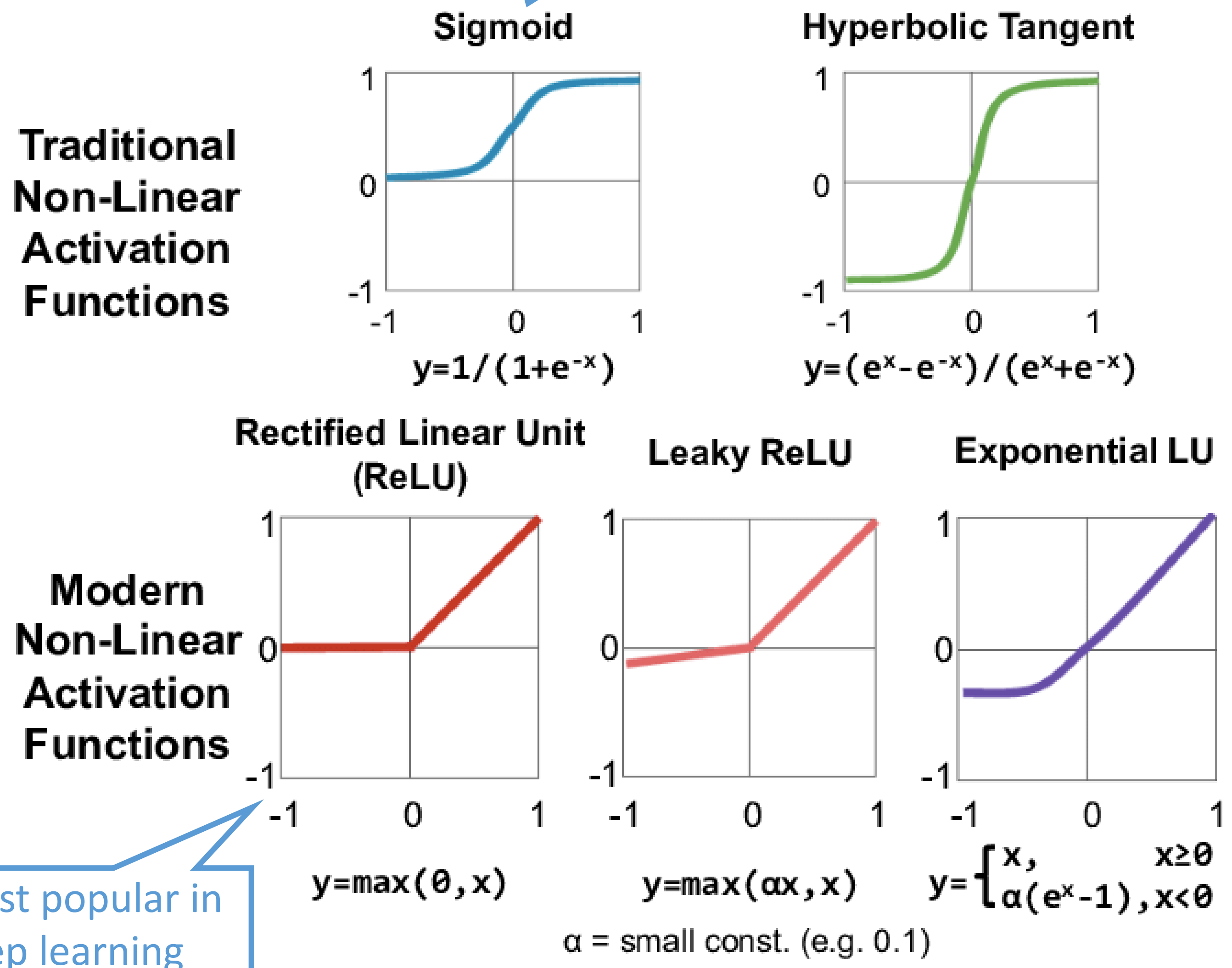



- Most common pooling operations are max and average

# Activation functions (Review)

Most popular in fully connected neural network

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
- Tanh: $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
- ReLU (Rectified Linear Unity):
  $\mathrm{ReLU}(z) = \max(0, z)$

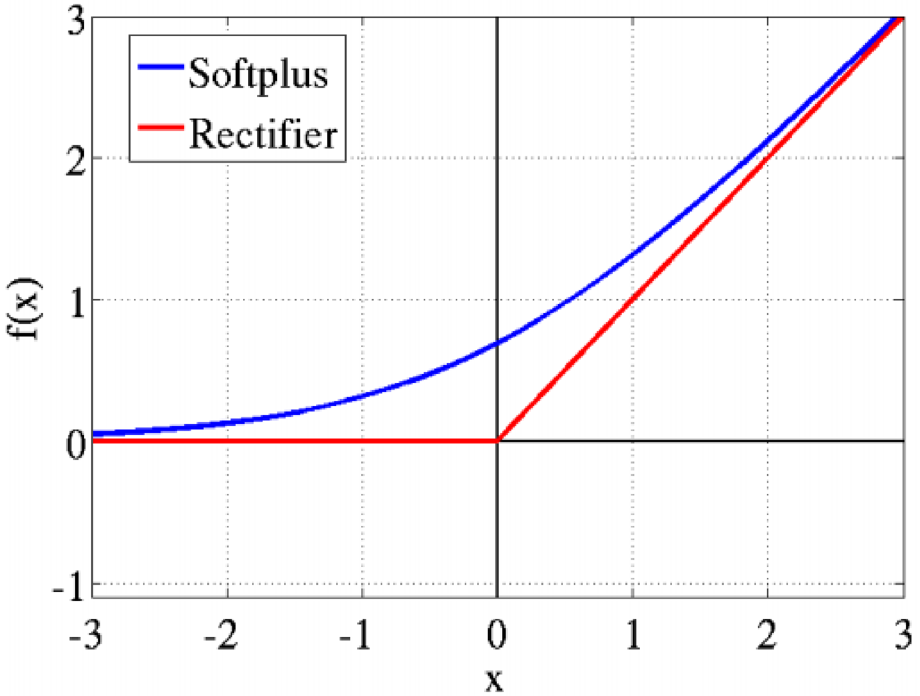



Most popular in deep learning

# ReLU activation function

- ReLU (Rectified linear unity) function

$$\mathrm{ReLU}(z) = \max(0, z)$$




- Its derivative

$$\mathrm{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

- ReLU can be approximated by softplus function

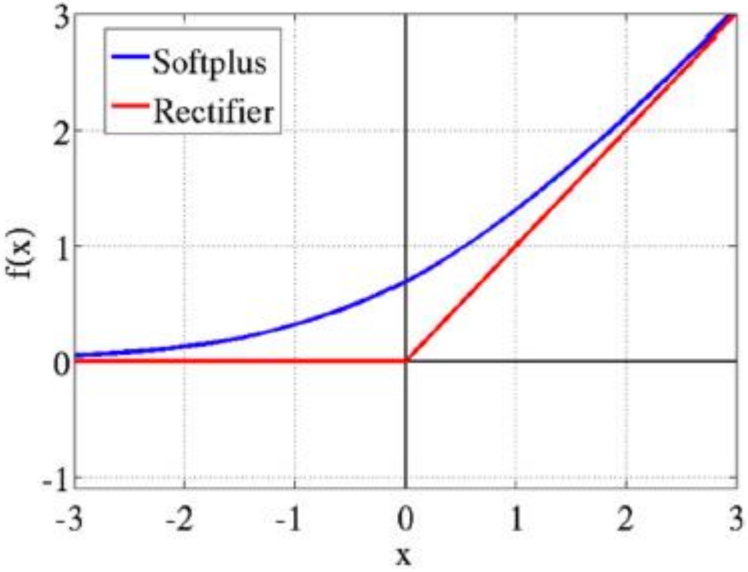
$$f_{\text{Softplus}}(x) = \log(1 + e^x)$$

- ReLU's gradient doesn't vanish as x increases
- Speed up training of neural networks
  - Since the gradient computation is very simple
  - The computational step is simple, no exponentials, no multiplication or division operations (compared to others)
- The gradient on positive portion is larger than sigmoid or tanh functions
  - Update more rapidly
  - The left "dead neuron" part can be ameliorated by Leaky ReLU

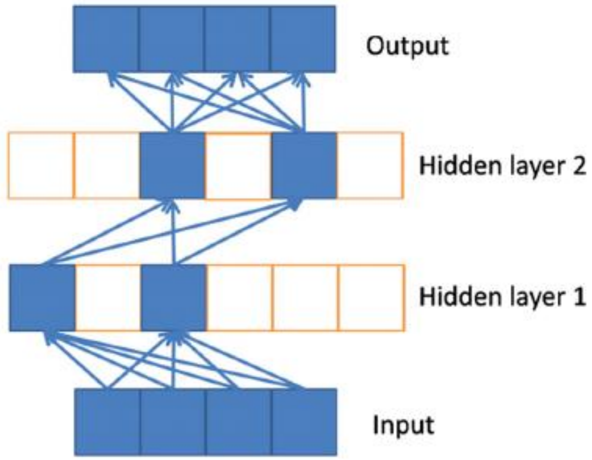
# ReLU activation function (cont.)

- ReLU function

$$\mathrm{ReLU}(z) = \max(0, z)$$



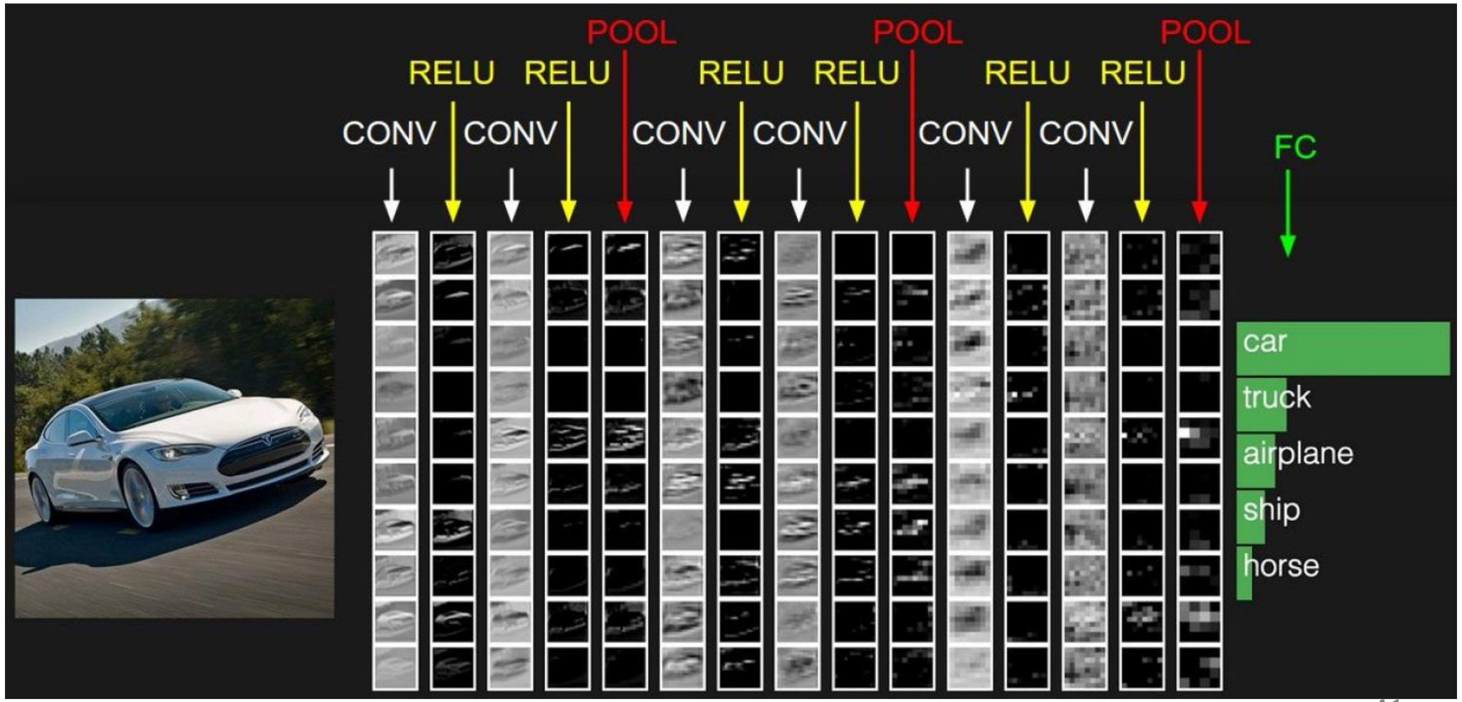

- The only non-linearity comes from the path selection with individual neurons being active or not
- It allows sparse representations:
  - for a given input only a subset of neurons are active




Sparse propagation of activations and gradients

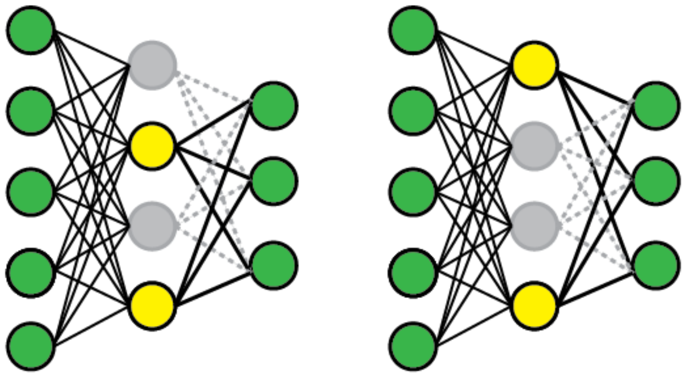
# A typical CNN structure

- Convolution/Activation (ReLU)/Pooling layers appear in flexible order and flexible repetitions

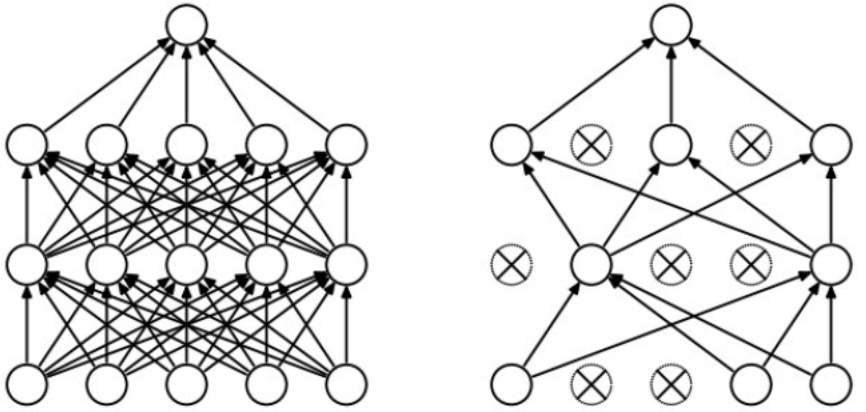


# Training Techniques

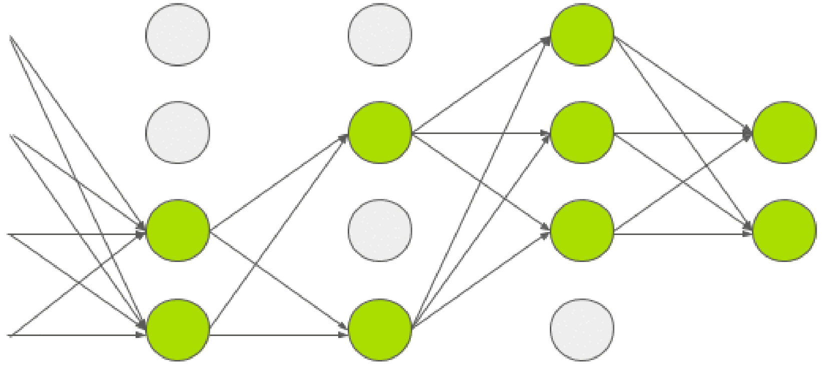
# Dropout



- Dropout randomly ‘drops’ units from a layer on each training step, creating ‘sub-architectures’ within the model
- It can be viewed as a type of sampling a small network within a large network
- Prevent neural networks from overfitting



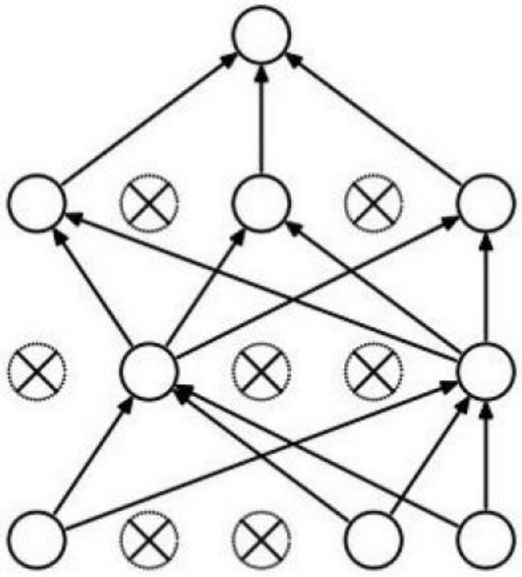
(a) Standard Neural Net (b) After applying dropout.



Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

# Dropout (cont.)

- Forces the network to have a redundant representation
- Increase robustness in prediction power



# Weights initialization

- If the weights in a network start too small,
  - then the signal shrinks as it passes through each layer until it’s too tiny to be useful
- If the weights in a network start too large,
  - then the signal grows as it passes through each layer until it’s too massive to be useful

# Weights initialization (cont.)

- All zero initialization

- Small random numbers

- Draw weights from a Gaussian distribution
  - with the standard deviation of $\sqrt{\frac{2}{n}}$
    - $n$ is the number of inputs to the ending neuron

# Batch normalization

- Batch training:
  - Given a set of data, each time a small portion of data are put into the model for training
- Extreme example
  - Suppose we are going to learn some pattern of people, and the input data are people's weights and heights
  - Unluckily, women and men are divided into two batches when we randomly split the data
  - As the weights and heights of women are very different from these of men, the neural network have to make huge changes to the weight when we switch the batch during training, which will cause slow convergence or even divergence

# Batch normalization (cont.)

- The problem in the example is called *Internal Covariate Shift*
- The solution to Internal Covariate Shift is batch normalization
- Suppose $Z_j^{(i)}$ is the i[th] input for the j[th] neuron in the input layer

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} Z_j^{(i)}$$

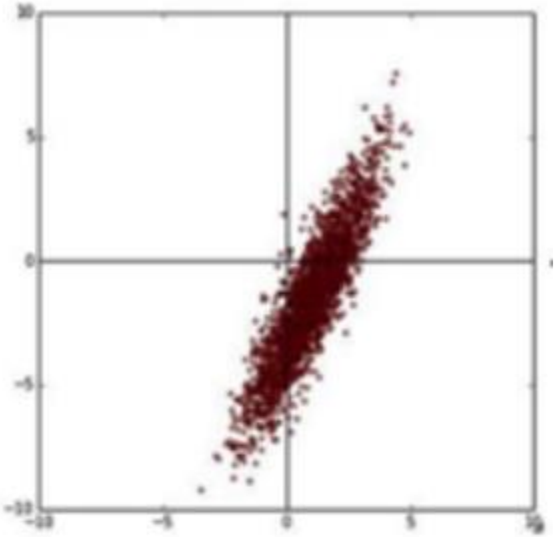
$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(Z_j^{(i)} - \mu_j)^2$$

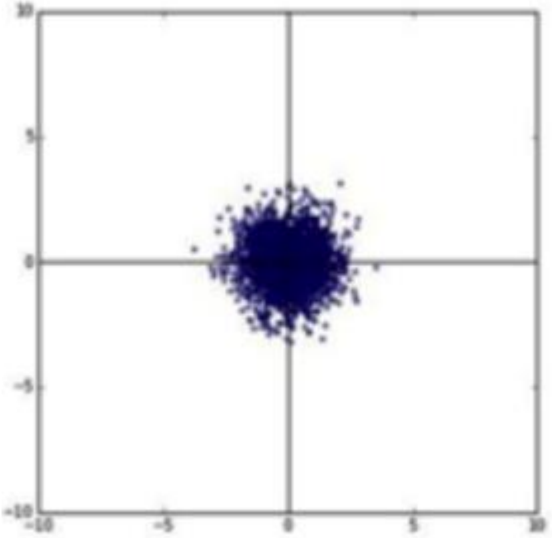
$$\hat{Z}_j = \frac{Z_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

- Or normalize the whole input layer together

# Batch normalization (cont.)

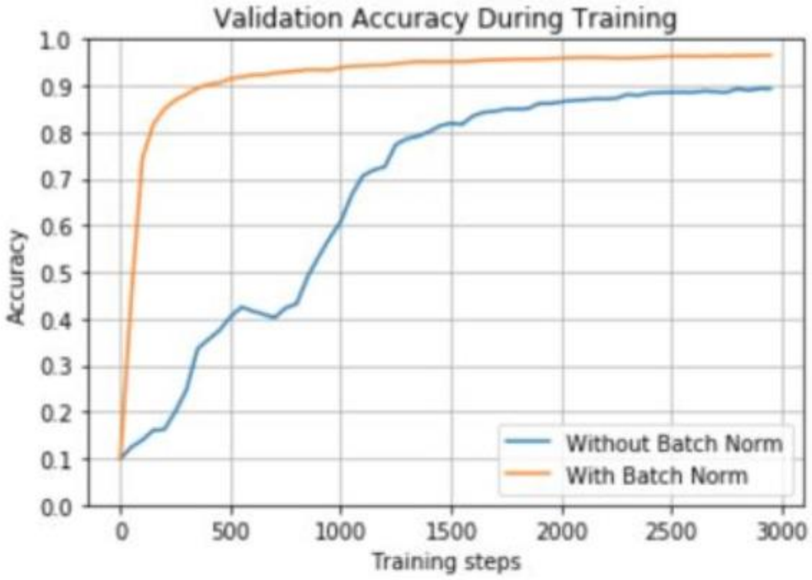
- 2D example

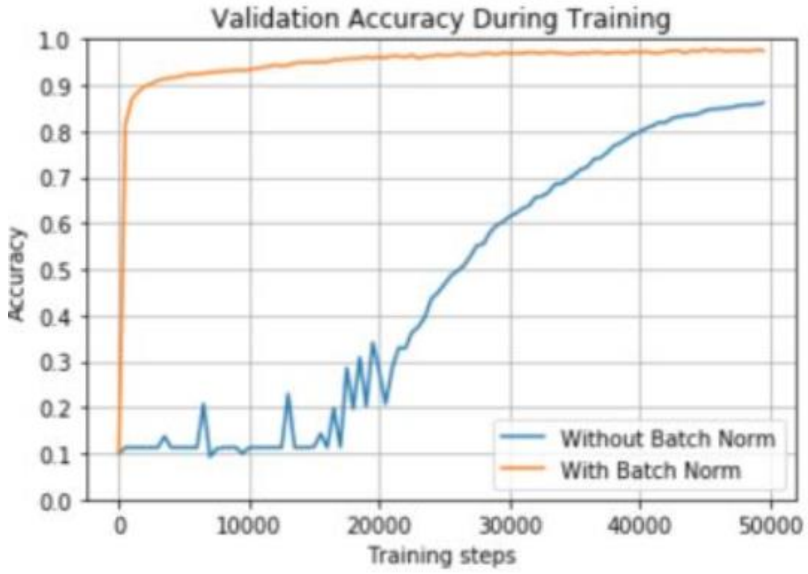




# Example of batch normalization

- Without batch normalization
  - Slow convergence and fluctuation

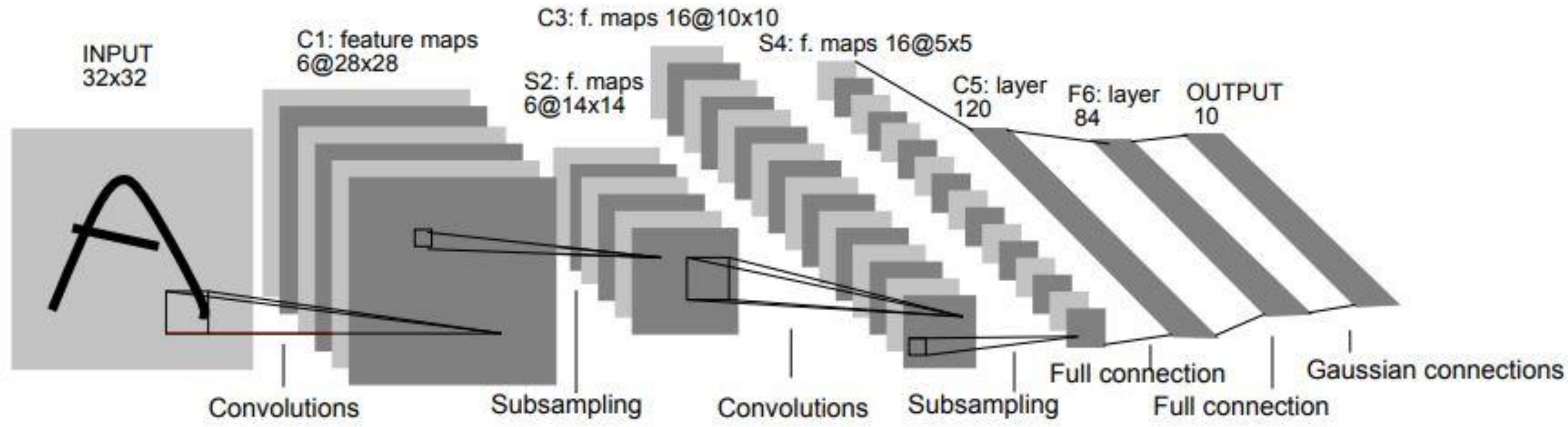


# Famous Neural Networks

# LeNet

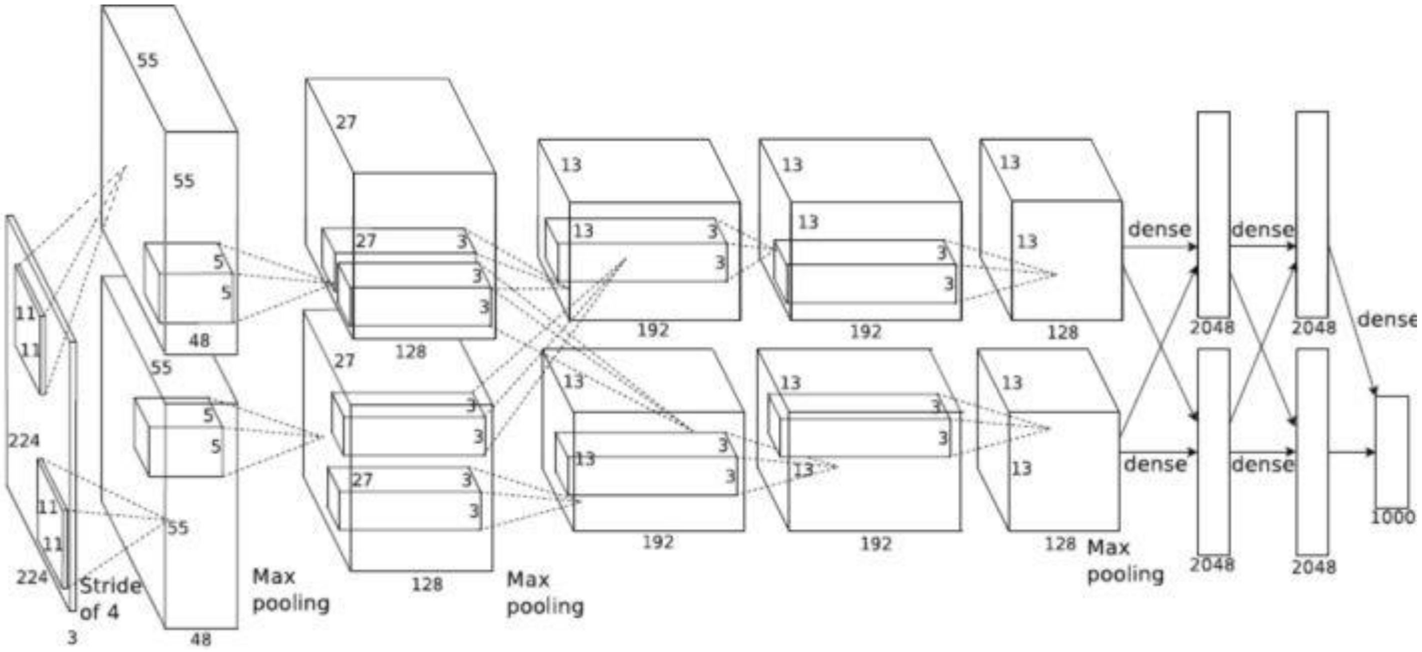
- LeNet [LeCun et al., 1998]




Input: 28*28*1 image, Conv filters of size 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

# AlexNet [Krizhevsky et al. 2012]

- AlexNet:



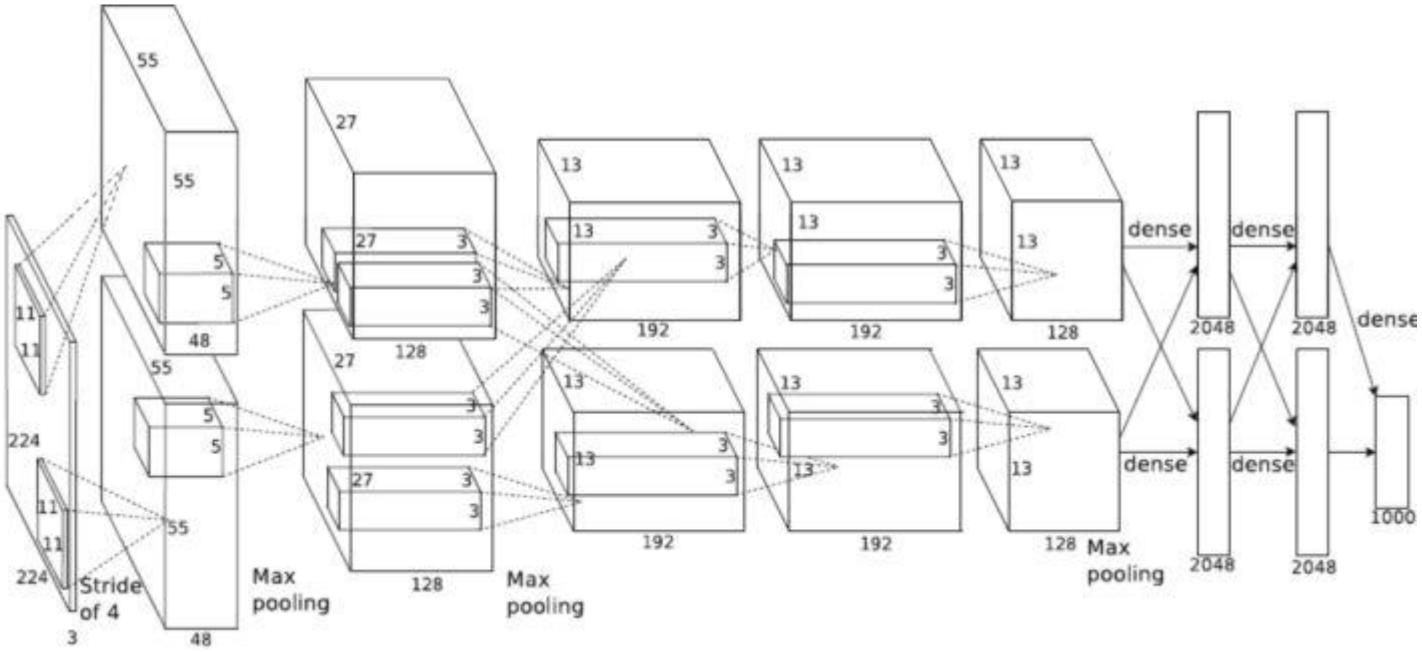

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 kernels applied at stride 4

Q: what is the output volume size?

# AlexNet (cont.)

- AlexNet:



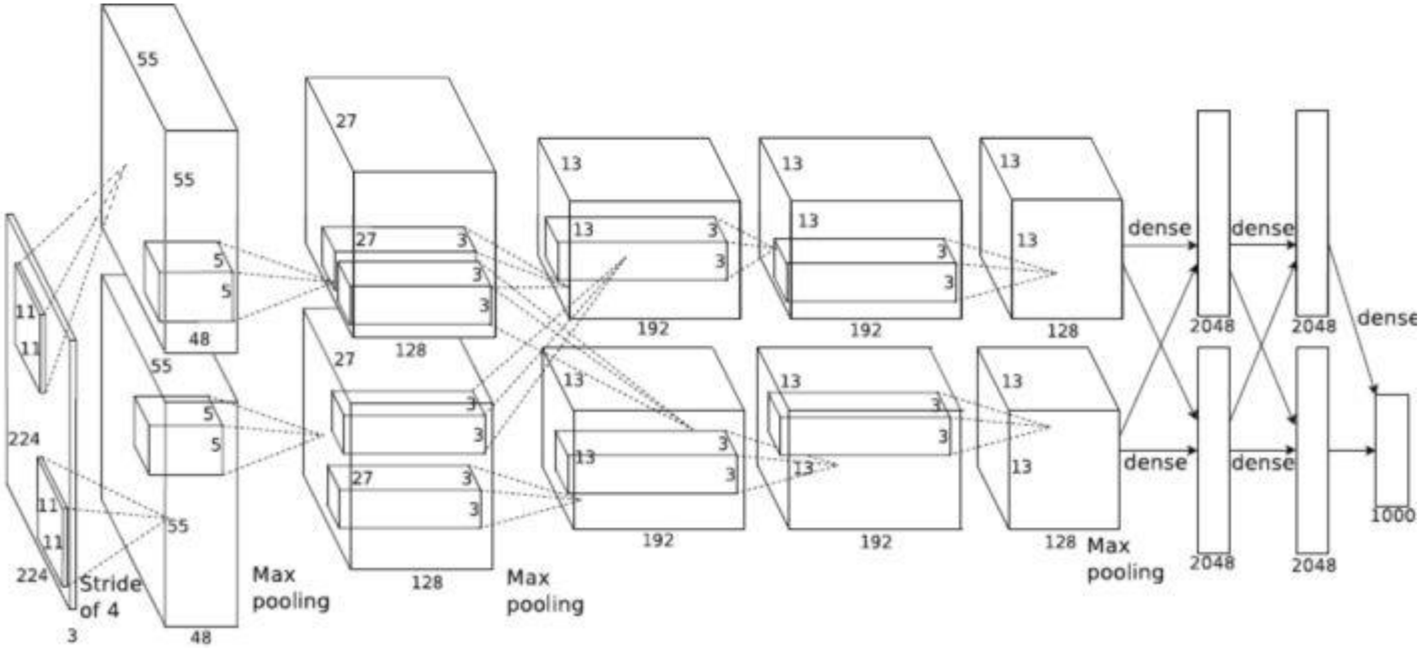

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 kernels applied at stride 4

Q: what is the output volume size? Hint: (227-11)/4+1 = 55 **[55x55x96]**

# AlexNet (cont.)

- AlexNet:




Input: 227x227x3 images

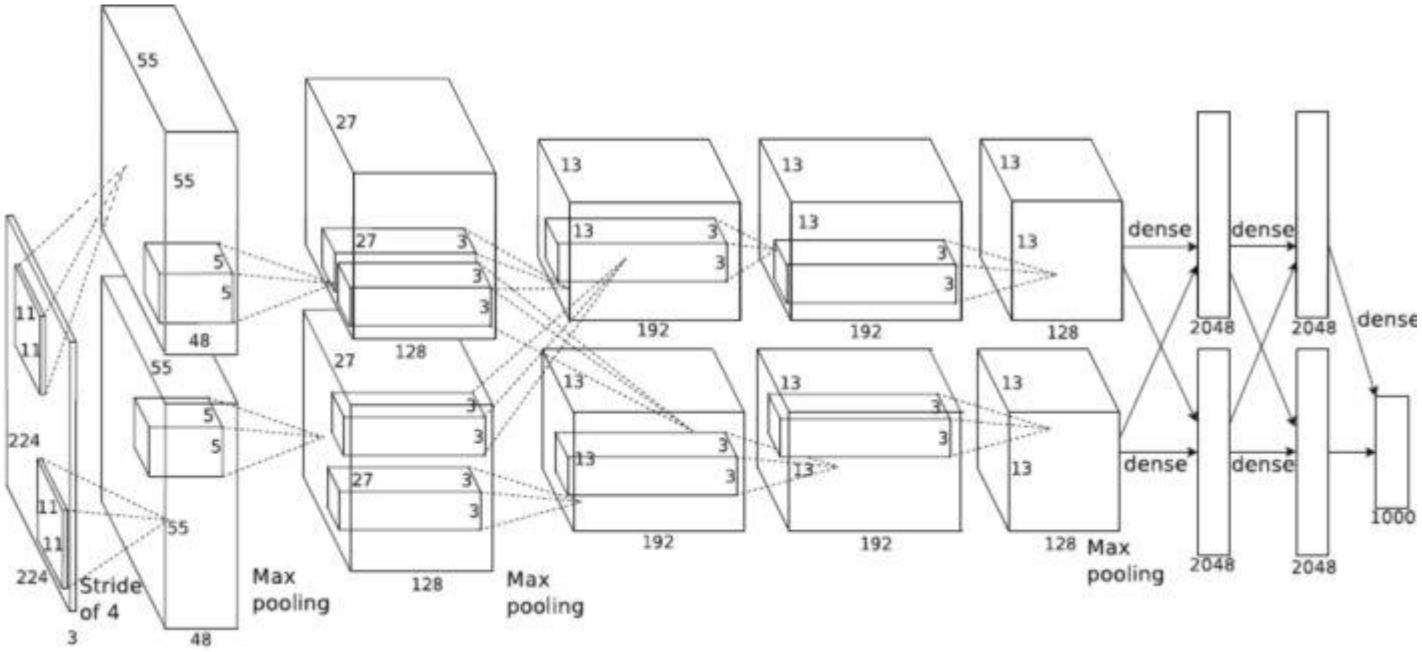
**First layer** (CONV1): 96 11x11 kernels applied at stride 4

Q: what is the output volume size? Hint: (227-11)/4+1 = 55 **[55x55x96]**

Q: What is the total number of parameters in this layer?

# AlexNet (cont.)

- AlexNet:



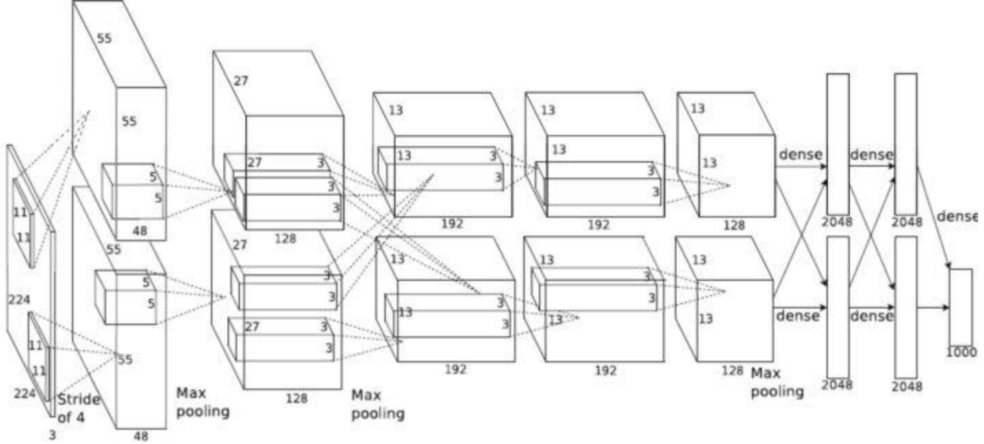

Input: 227x227x3 images

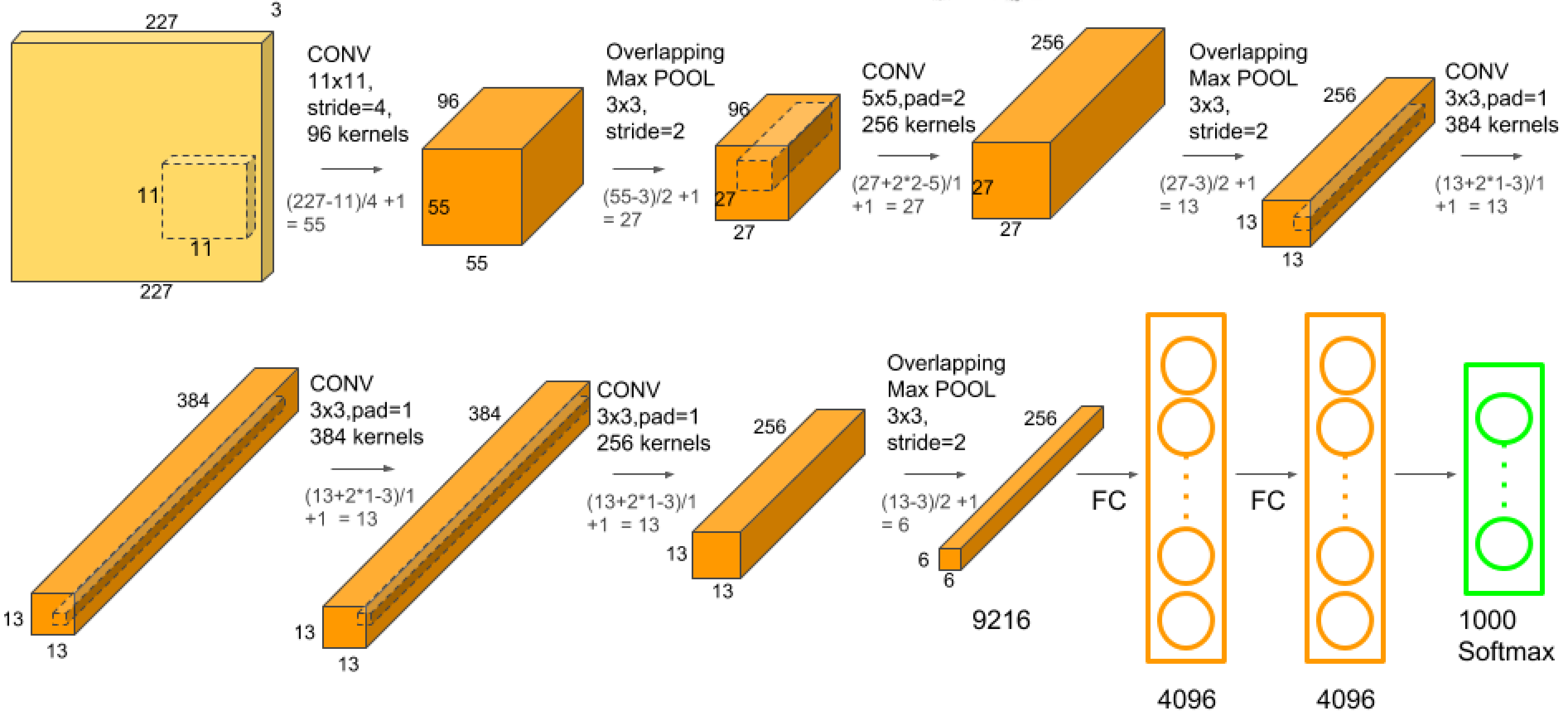
**First layer** (CONV1): 96 11x11 kernels applied at stride 4

Q: what is the output volume size? Hint: (227-11)/4+1 = 55 **[55x55x96]**

Q: What is the total number of parameters in this layer? (11*11*3)*96 = **35K**

# AlexNet (cont.)



# VGGNet

- VGGNet: [Simonyan and Zisserman, 2014]
- Only 3x3 CONV stride 1, pad 1
  2x2 MAX POOL stride 2

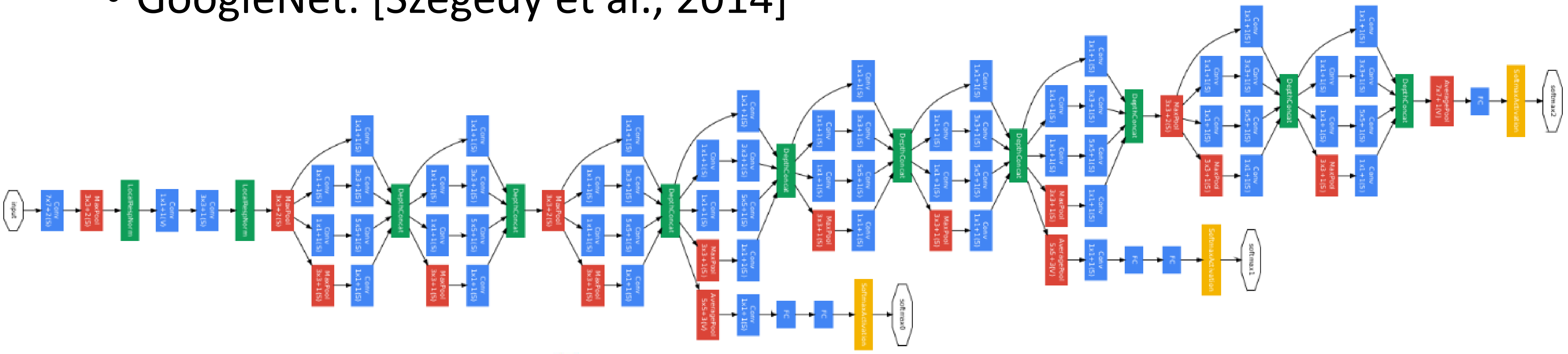
Best model

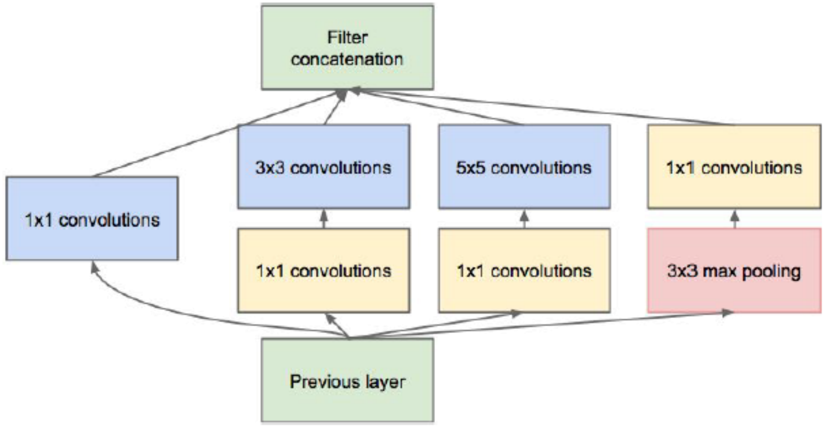
- 11.2% top 5 error in ILSVRC 2013
  - ImageNet Large Scale Visual Recognition Challenge 2013
- 7.3% top 5 error in ILSVRC 2014
  - Not champion, champion is GoogleNet

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

# GoogleNet

- GoogleNet: [Szegedy et al., 2014]






ILSVRC 2014 winner (6.7% top 5 error)

# GoogleNet (cont.)

- GoogleNet

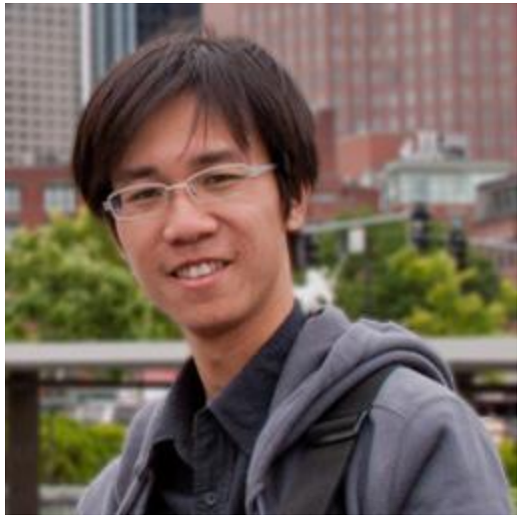
| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

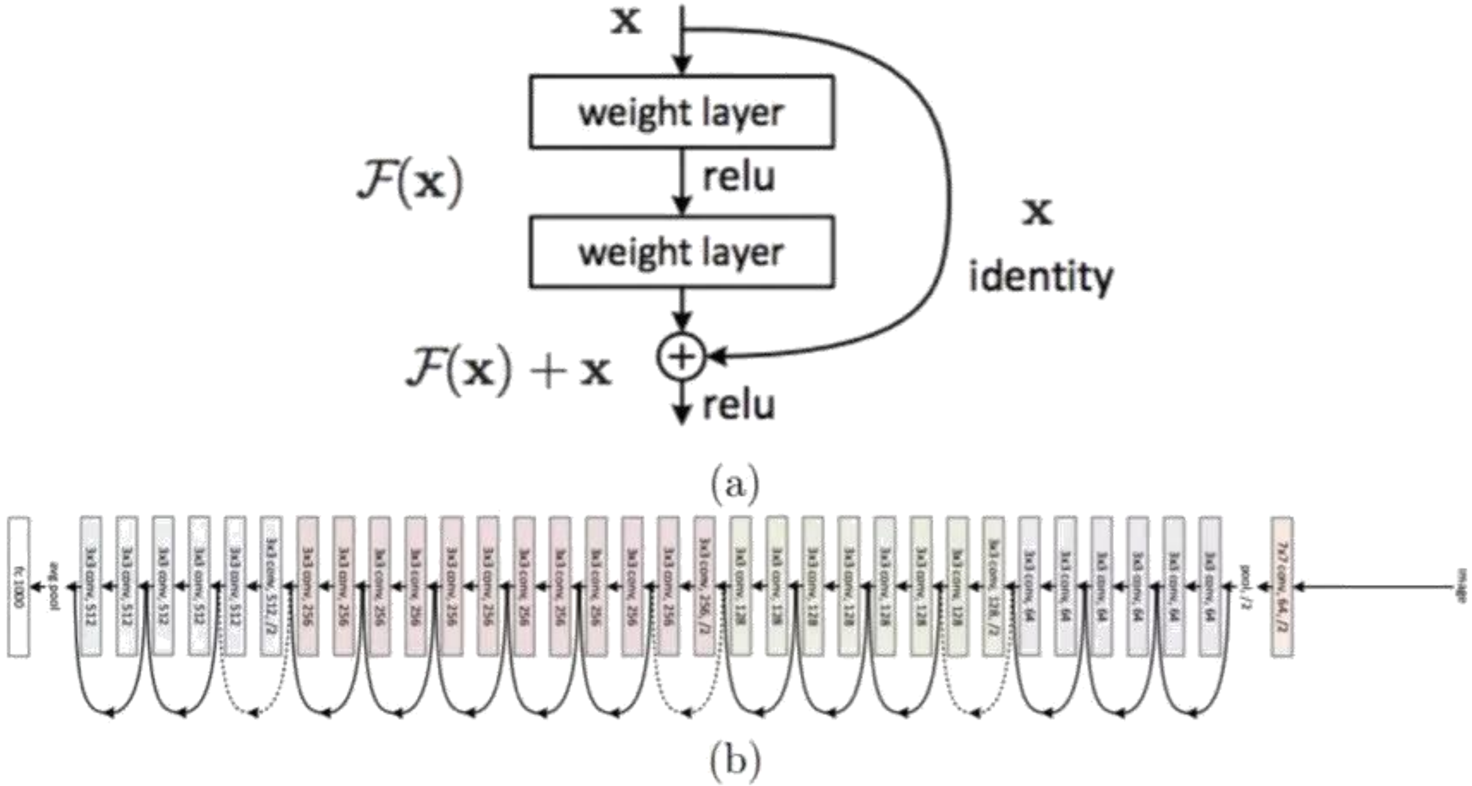
Fun features:
- Only 5 million parameters! (Removes FC layers completely)

Compared to AlexNet:
- 12X less parameters
- 2x running speed
- top-5 error rate 6.67%

# ResNet

- ResNet [Kaiming He et al., 2015]
  - Residual networks
  - Solves the problem of drifting by adding the original input to later layers





# ResNet (cont.)

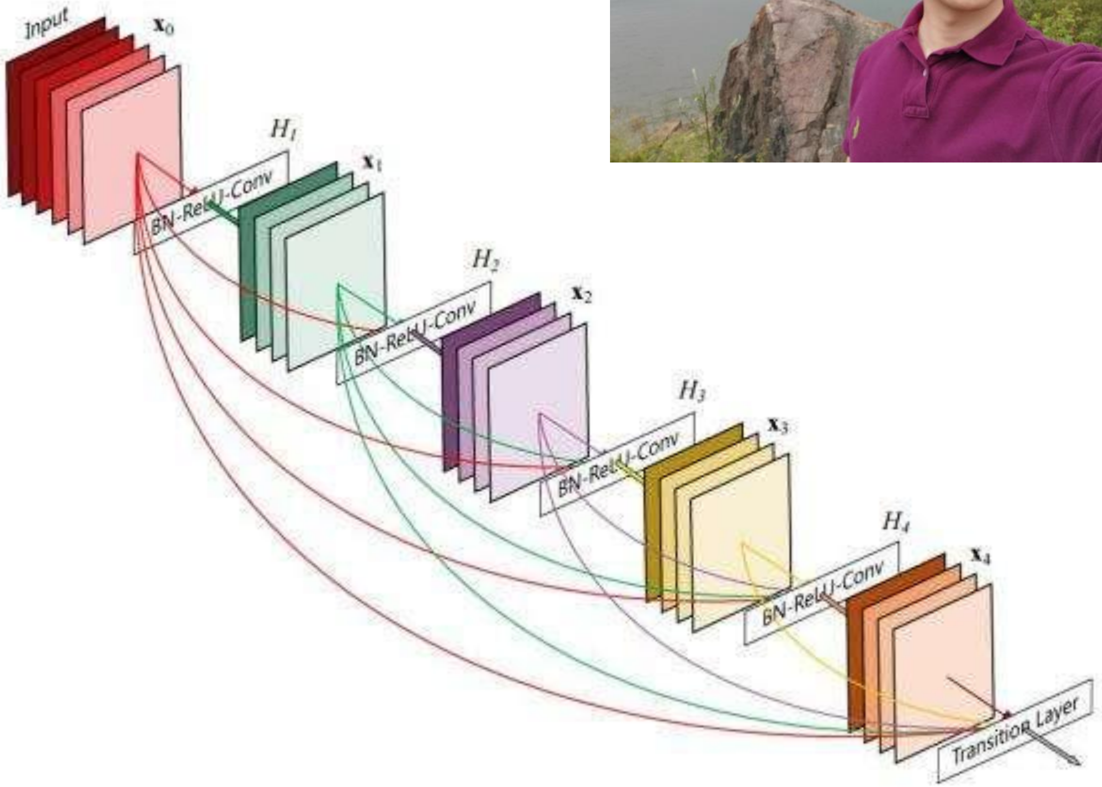
- ResNet: ILSVRC 2015 winner (3.6% top 5 error)

## MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

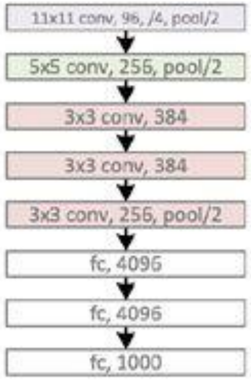
# DenseNet

- DenseNet
  - By Gao Huang et al.
  - Best paper award in CVPR 2017

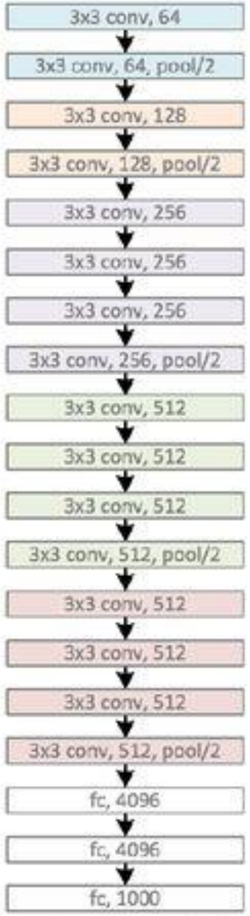


# Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

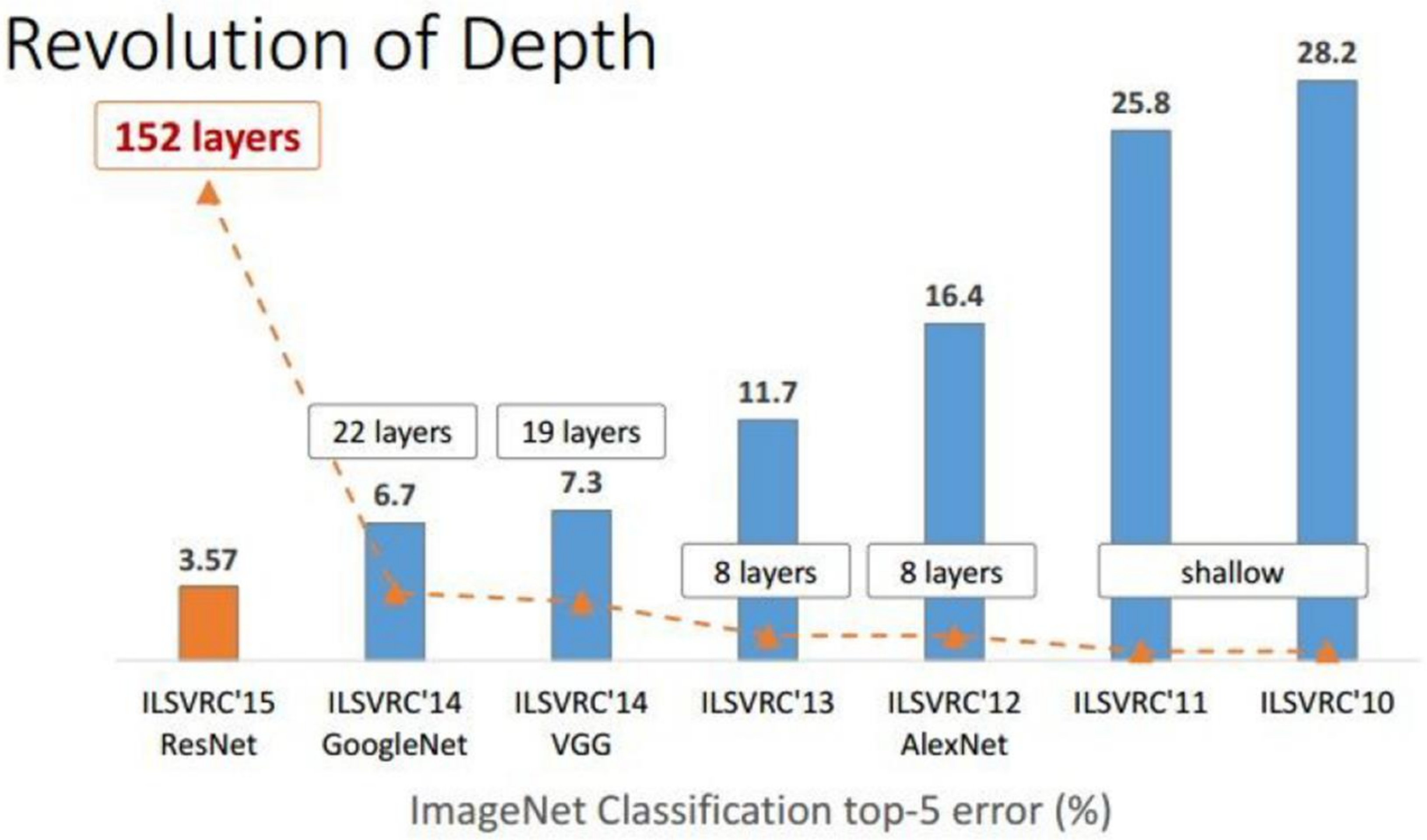



VGG, 19 layers
(ILSVRC 2014)




GoogleNet, 22 layers
(ILSVRC 2014)

# Revolution of depth



# Architecture comparison