Lecture 7: Convolutional Neural Networks

Shuai Li

John Hopcroft Center, Shanghai Jiao Tong University

https://shuaili8.github.io

https://shuaili8.github.io/Teaching/VE445/index.html

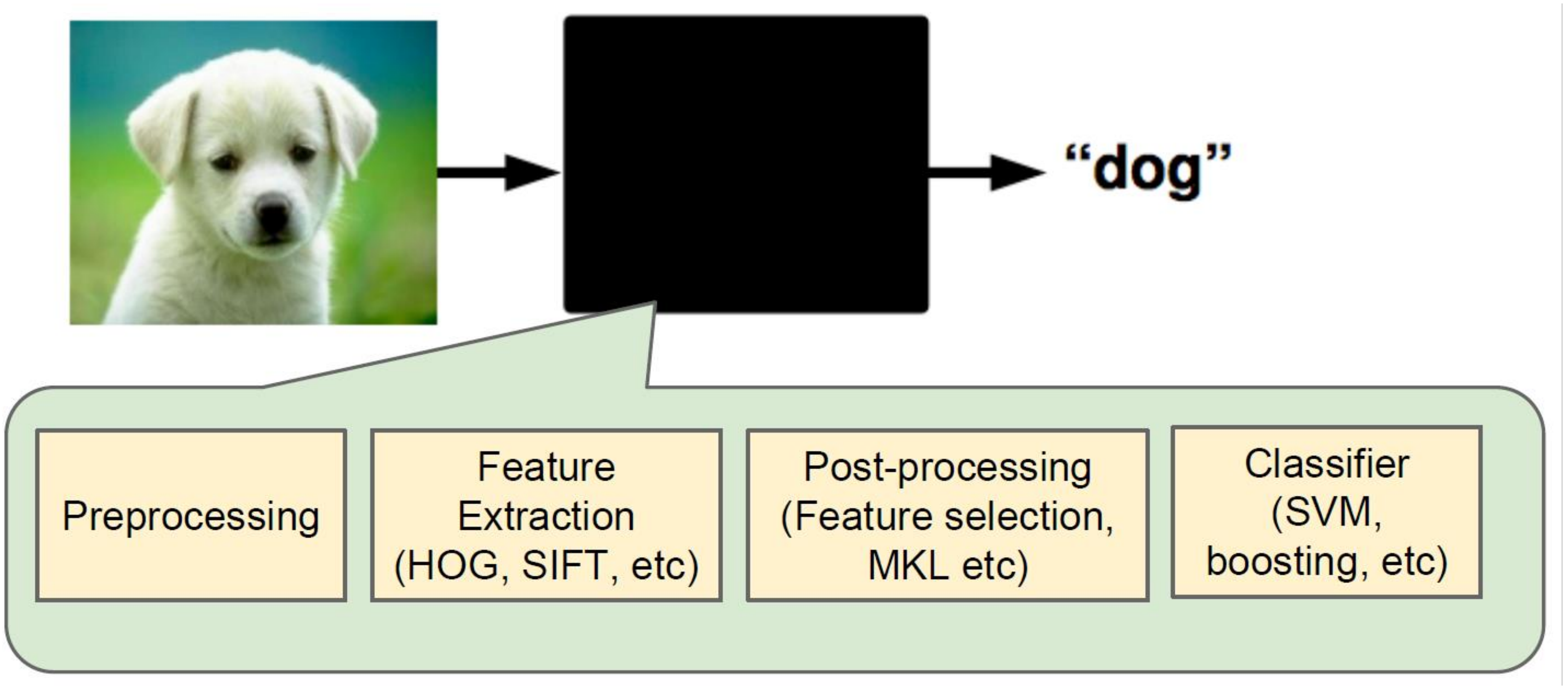
Outline

- Motivation
- What is convolution and benefits
- Parameters in CNN
- Pooling
- Training techniques
- Famous neural networks

Motivation

Previous pipeline of pattern recognition

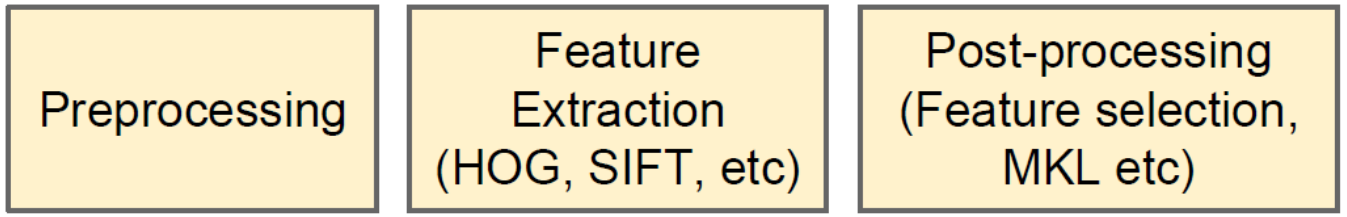
- The black box in a traditional pattern recognition problem


"dog"
Preprocessing
Feature Extraction (HOG, SIFT, etc)
Post-processing (Feature selection, MKL etc)
Classifier (SVM, boosting, etc)

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?


Preprocessing
Feature Extraction (HOG, SIFT, etc)
Post-processing (Feature selection, MKL etc)

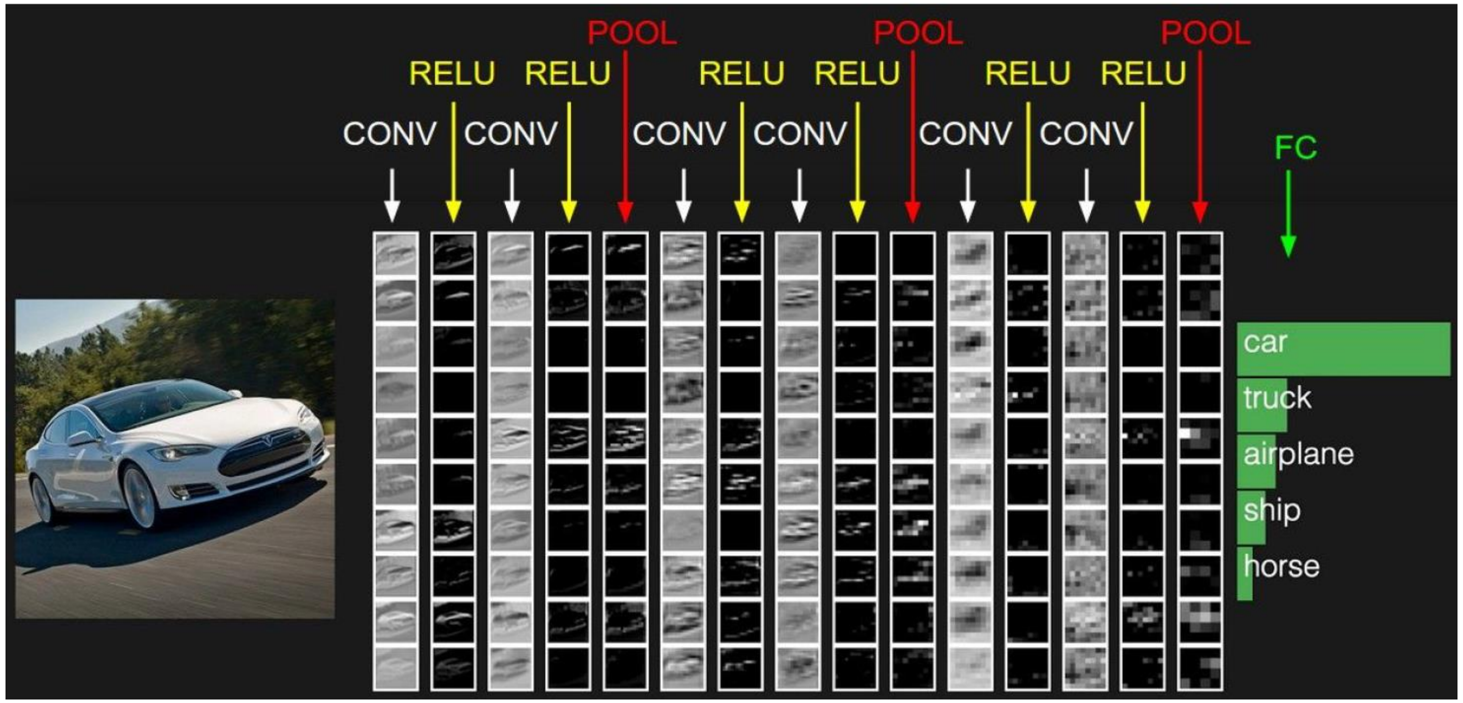
Objective

- 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

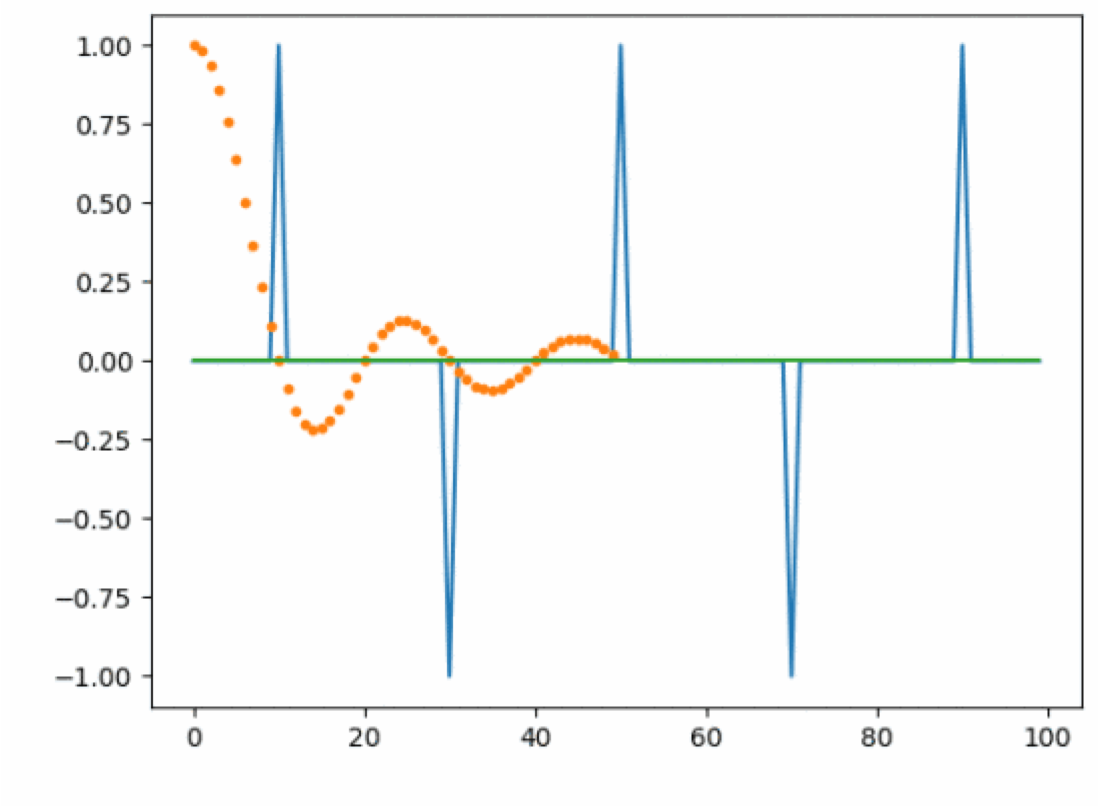
- The answer of an end-to-end recognition system is the convolution neural network
- It contains the following layers with flexible order and structure:
  - Convolution layer
  - Activation layer
  - Pooling layer

- Example of CNN:



What is Convolution and Benefits

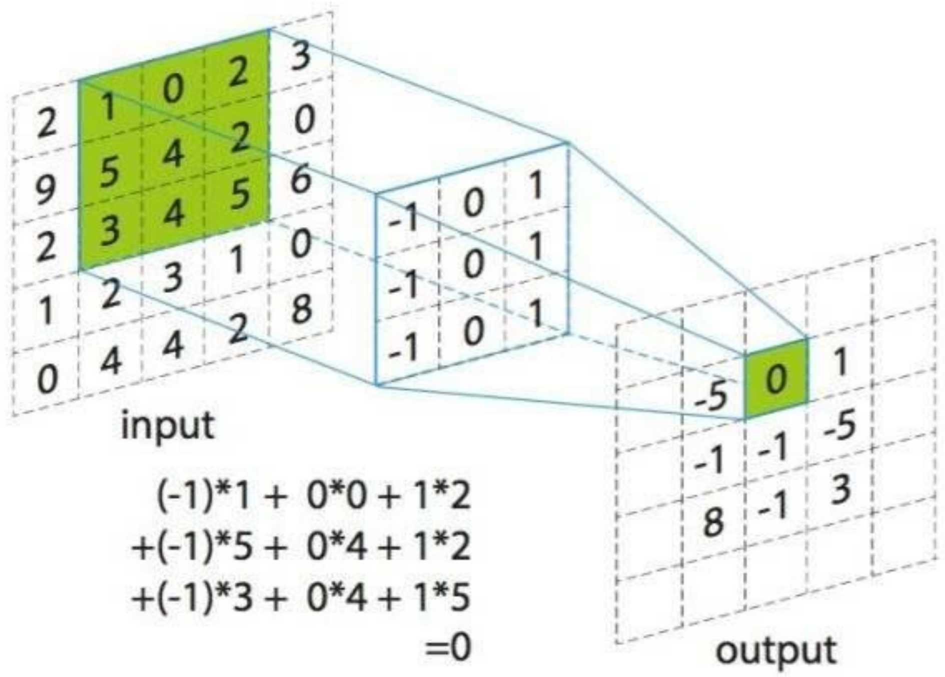
Convolution in math



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

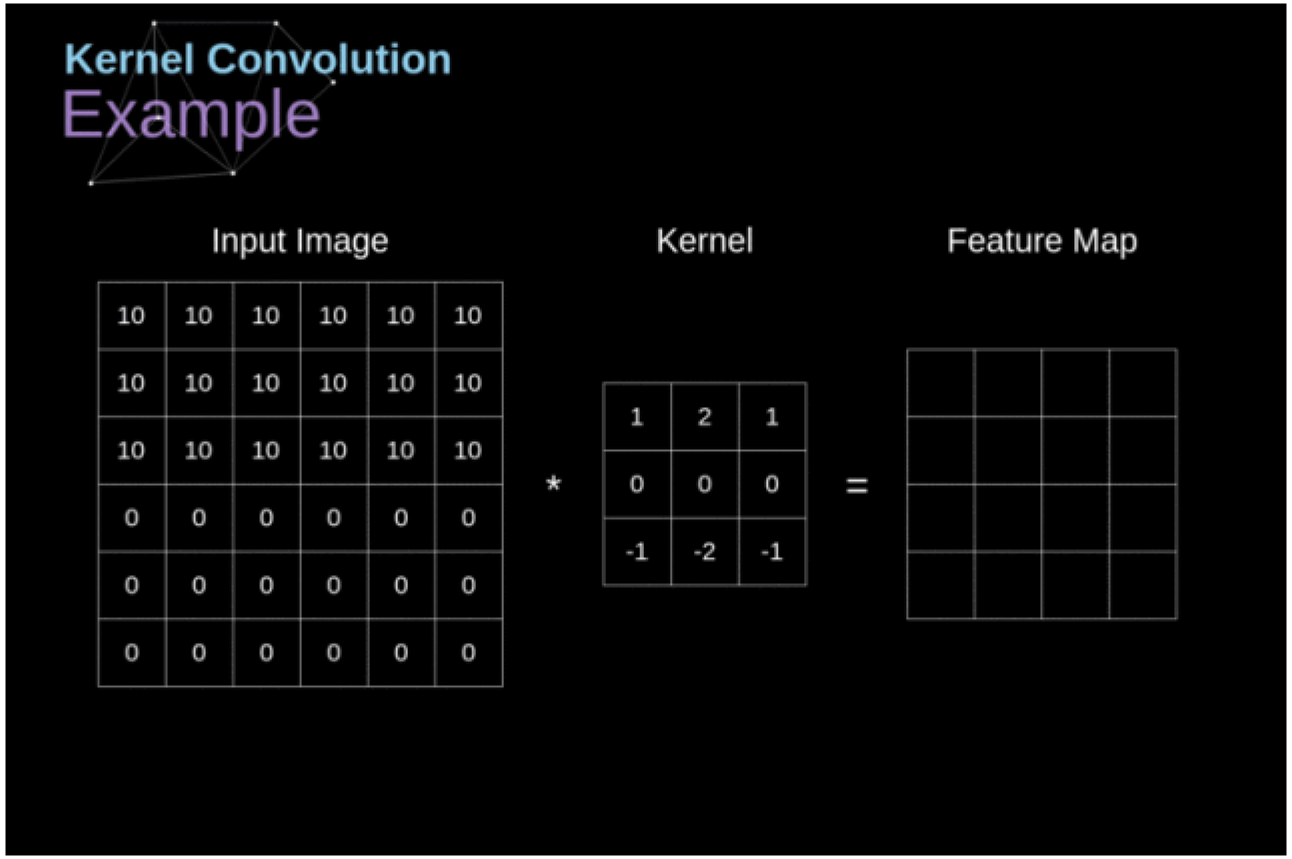
Convolution in neural network

- Given a matrix input, such as an image
- Using a smaller matrix to screening the input at every position at the input matrix
- Put the convolution results at corresponding positions



Convolution example

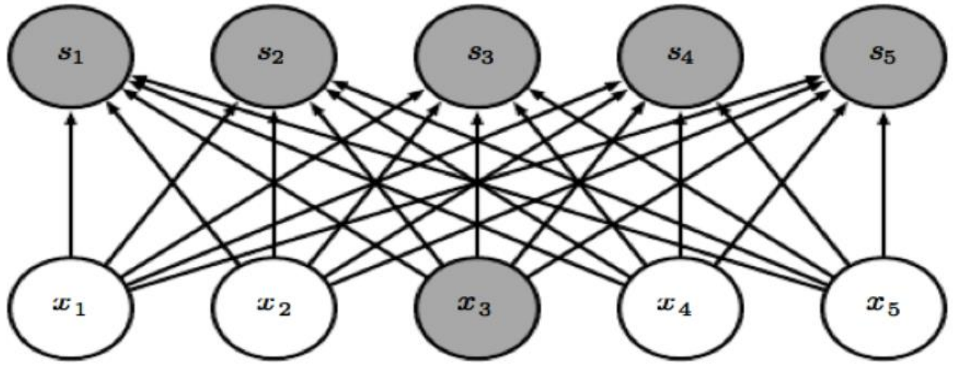
- Here is an animation of the convolution operation in CNN.

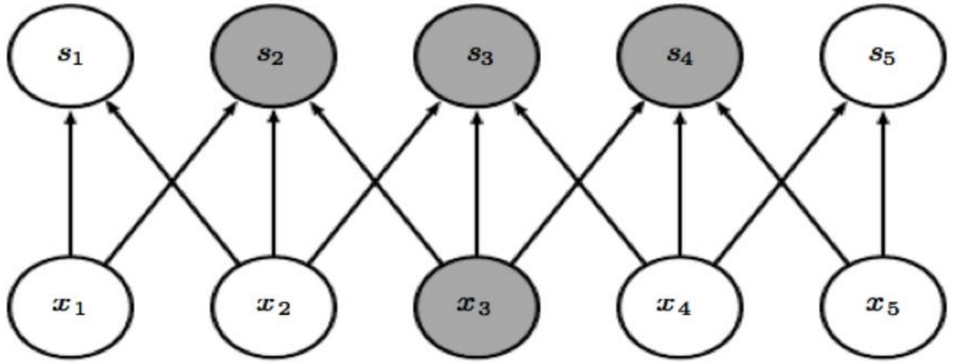


$$\begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} \, & \, & \, & \, \\ \, & \, & \, & \, \\ \, & \, & \, & \, \\ \, & \, & \, & \, \end{bmatrix}$$

Benefits of convolution

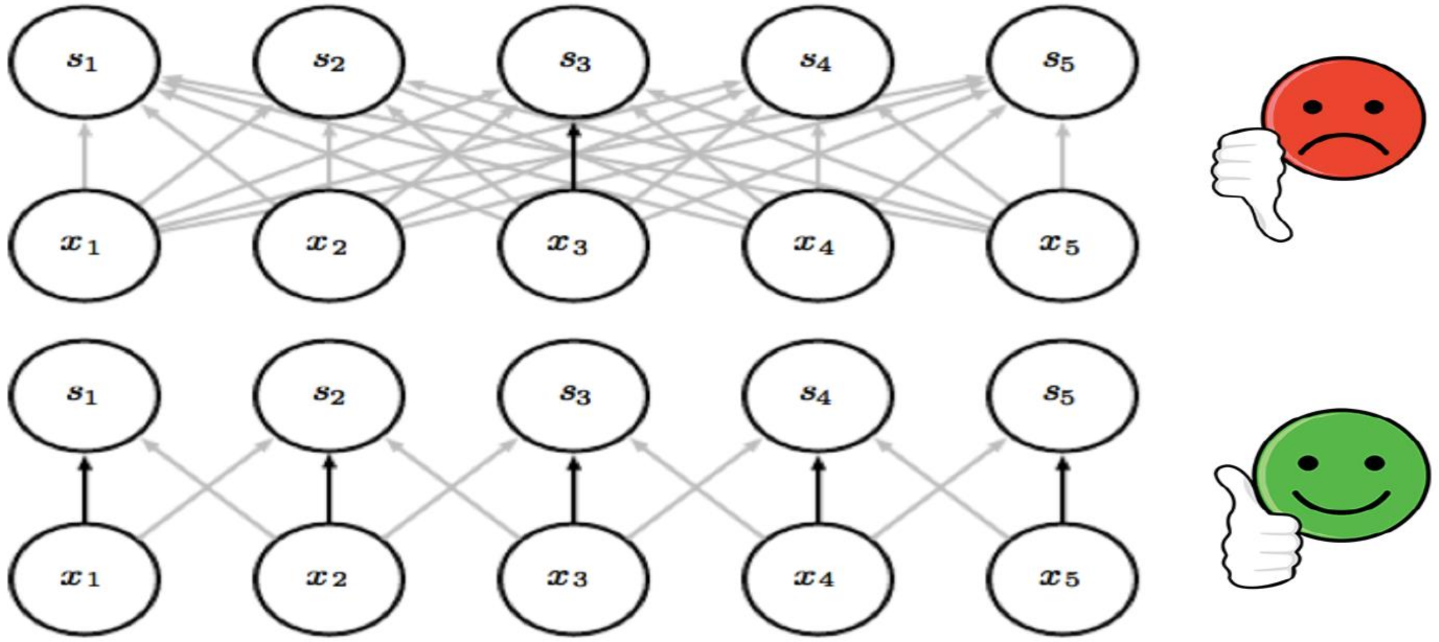
- Sparse connection means less computing burden
- Only need to compute the area with the filter at different position





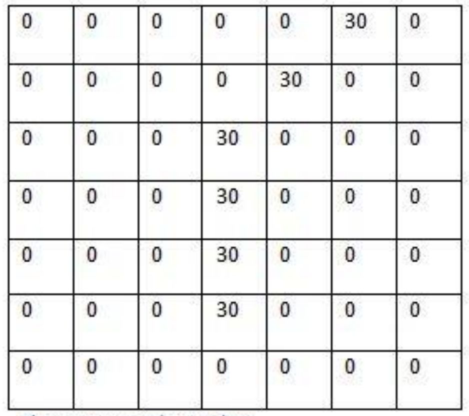
Benefits of convolution (cont.)

- Parameter sharing: the moving the filter, the value inside will not change, and be shared at different positions



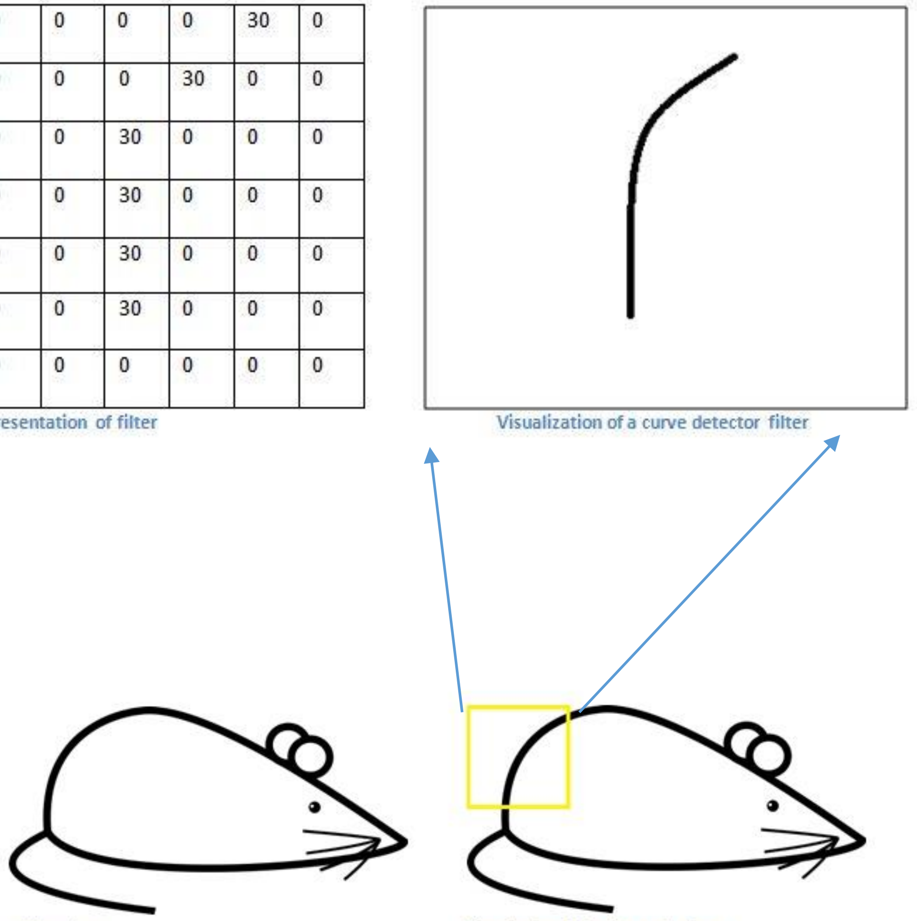
Interpretation of convolution

- Convolution can be used to find edge with particular shapes!
- Example: the kernel in the left represents the edge in the right, which is the back of a mouse



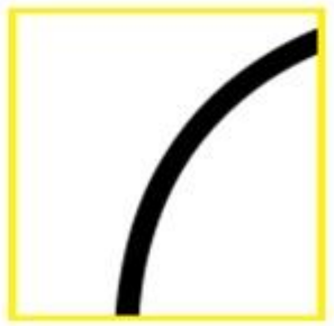
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

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

*

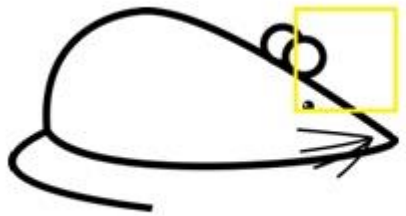
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

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 very 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

- To conclude, the filters in the CNN are used to find edges with particular pattern
- The filters in the deeper layers represent more abstract patterns

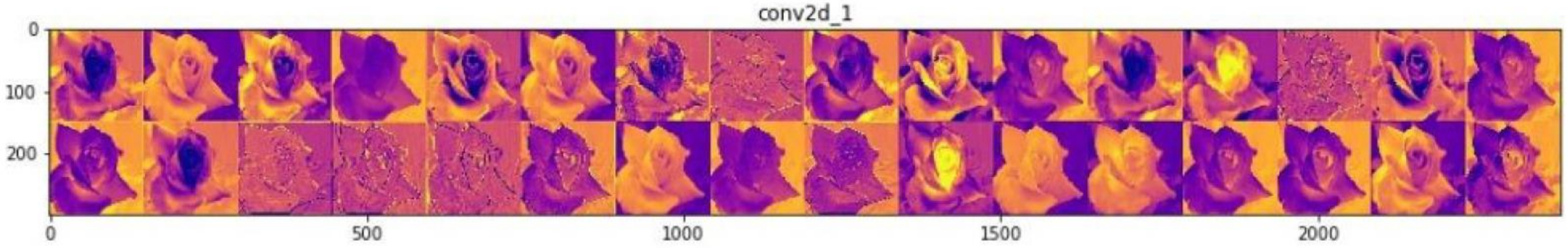
- Example input image:

- Example model:

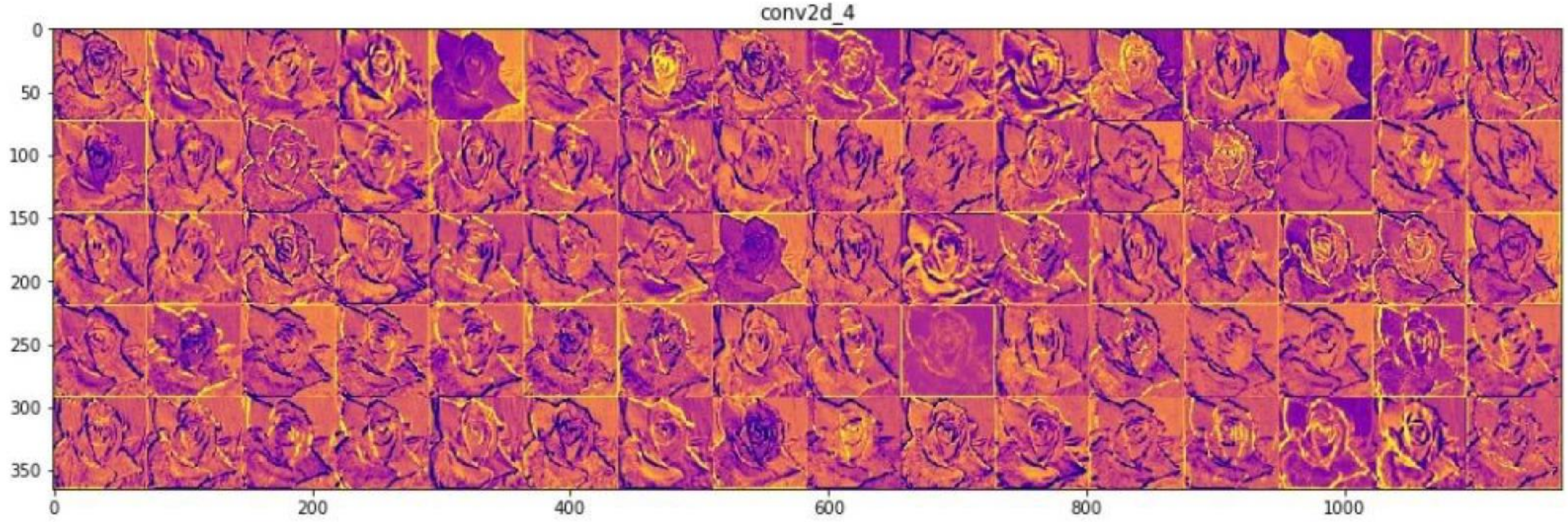
InceptionV3

Visualization (cont.)

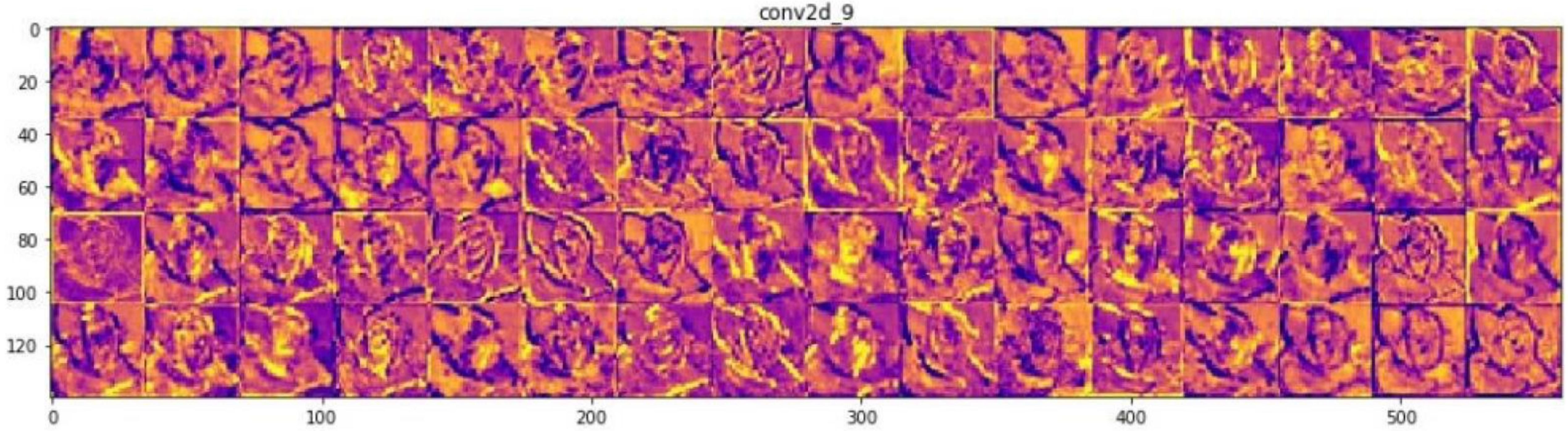
- The visualization of the filters in inceptionV3 in the first, fourth, and ninth layer



Visualization (cont.)

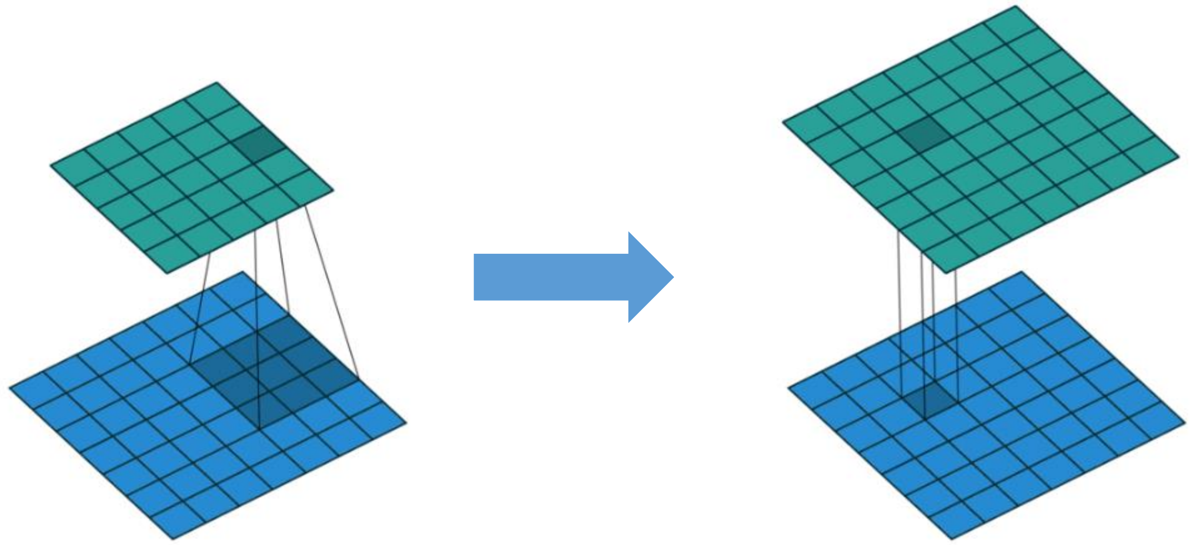


Visualization (cont.)



1*1 convolution

- Here we introduce a special filter, which as a size of 1*1

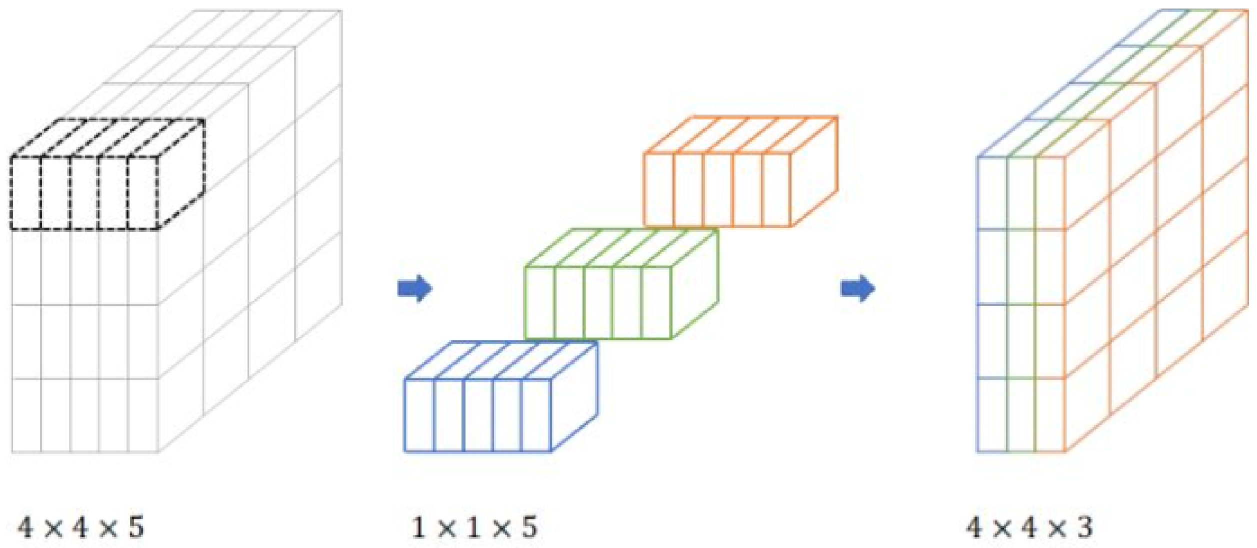


Convolution with large kernel Convolution with 1*1 kernel

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

1*1 convolution (cont.)

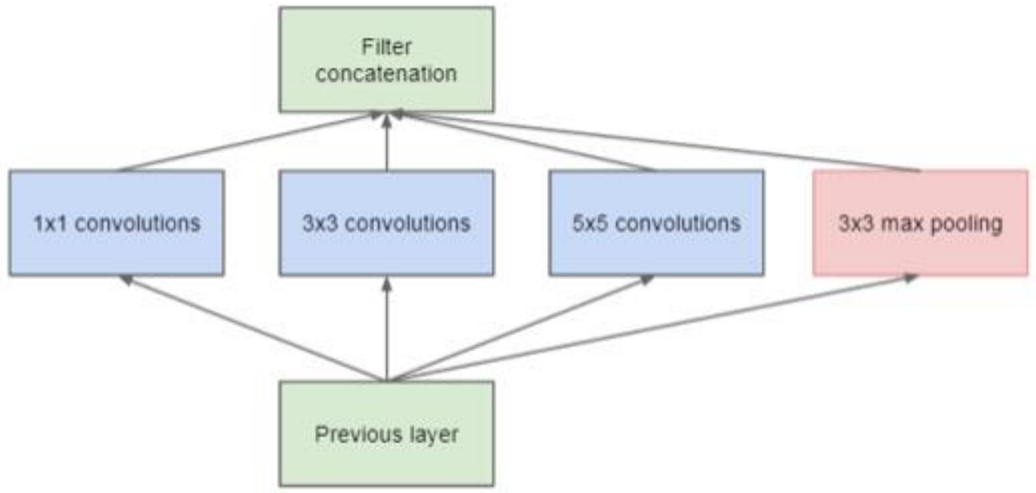
- The 1*1 convolution is very useful as it can reduce the computation complexity in CNN!



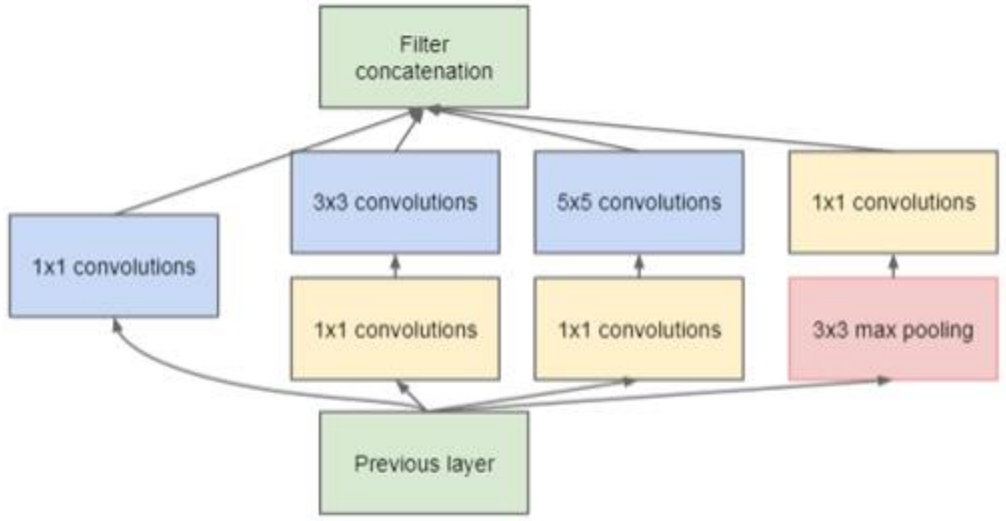
- Help reduce the number of channels
- In the example above, the size of the input data is reduced from 4*4*5 to 4*4*3.

1*1 convolution (cont.)

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



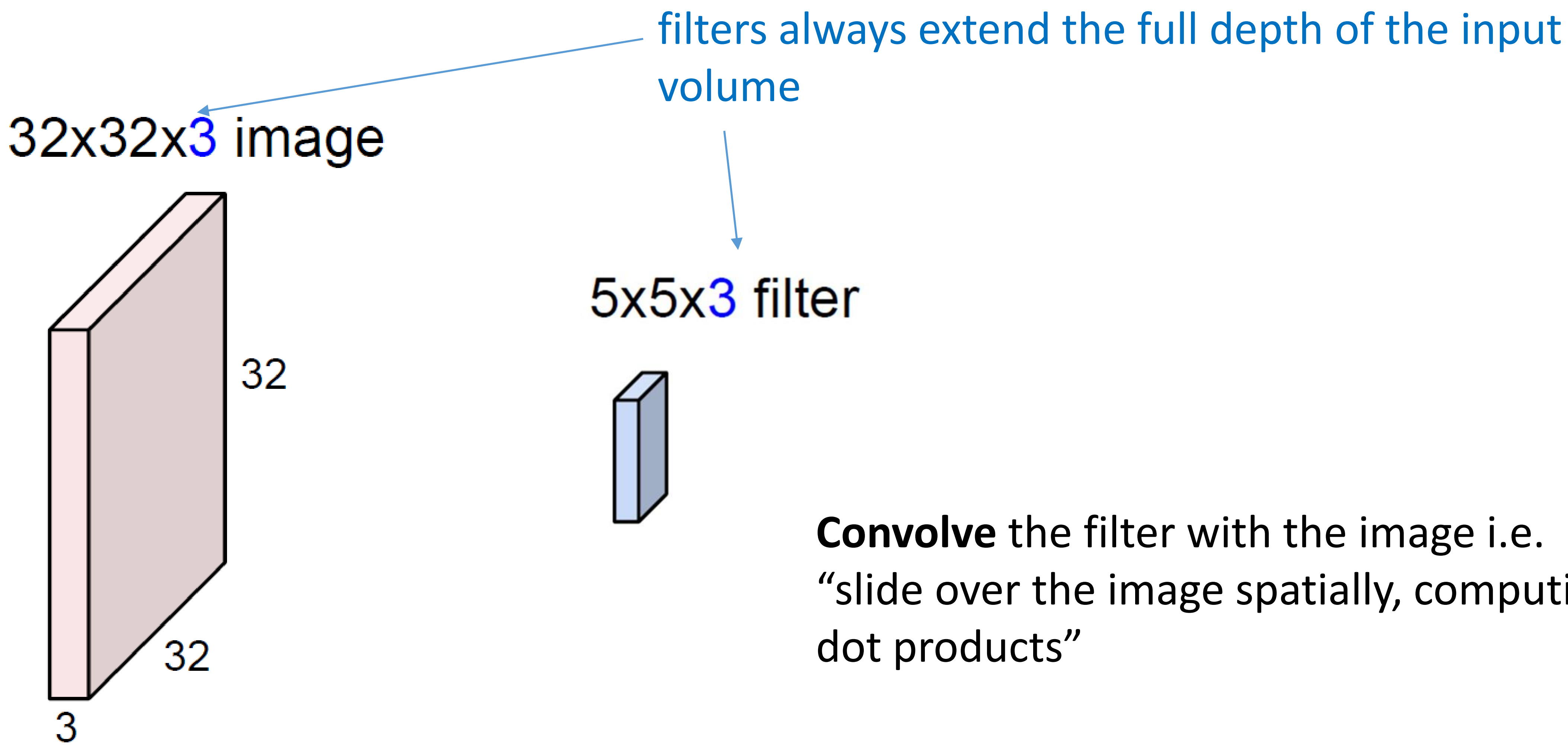
(a) Inception module, naïve version



(b) Inception module with dimension reductions

Parameters in CNN

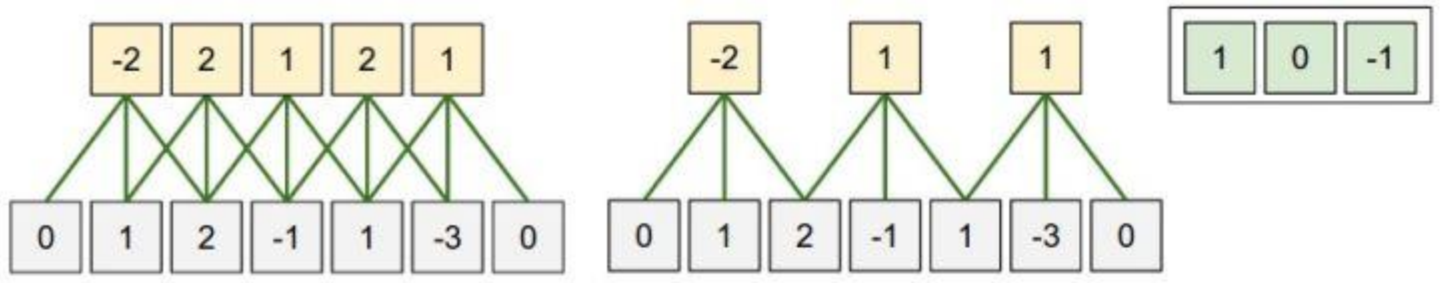
Filter depth



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Stride

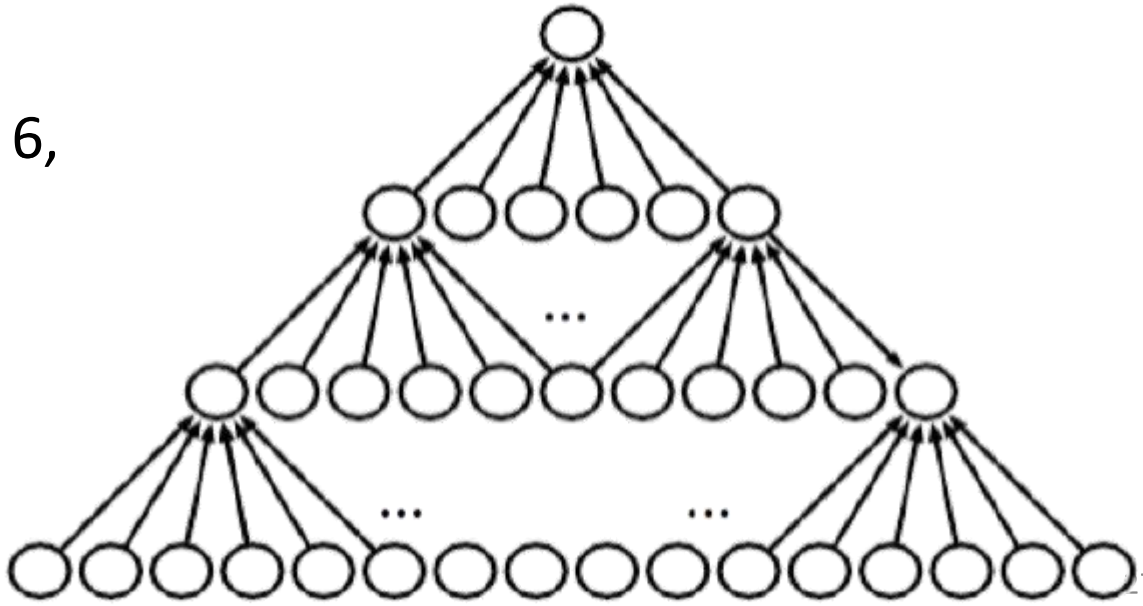
- The distance that the filter is moved in each step
- Examples of stride=1 and stride=2



Padding

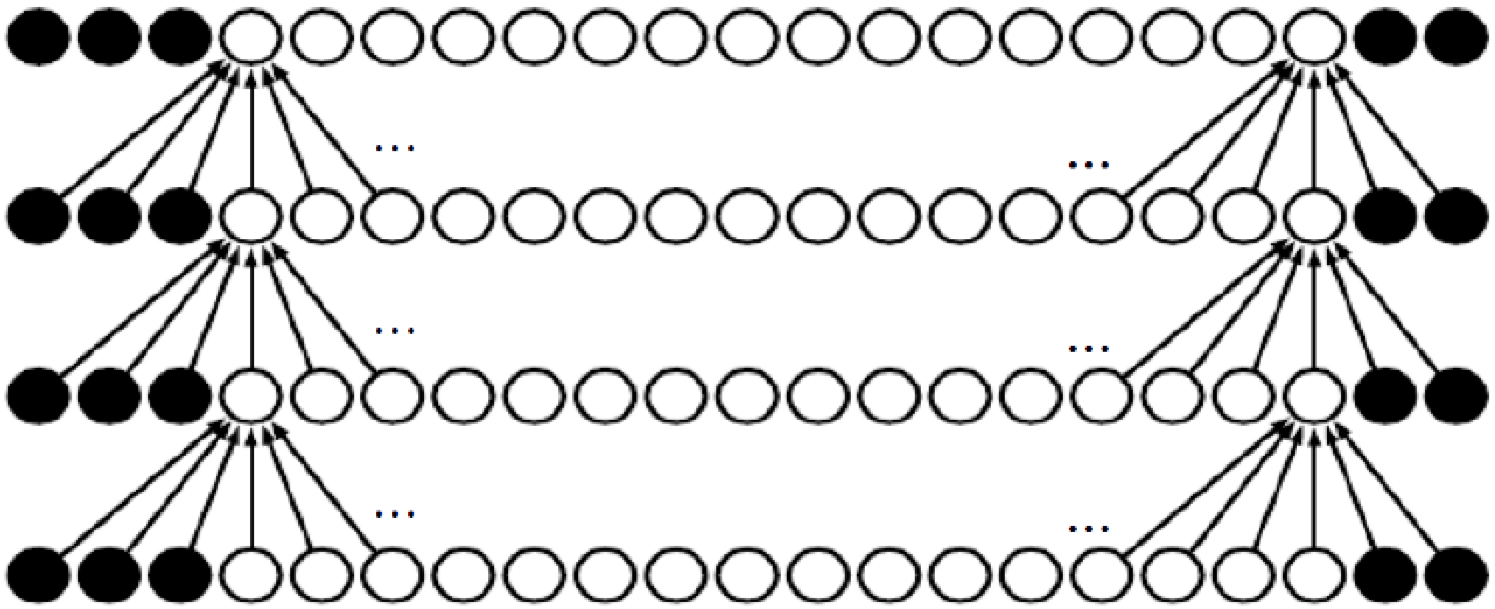
- A solution to the problem of data shrinking
- Data shrinking: as convolution can only happen within the border of the input data, the size of the data will become smaller as the network becomes deeper

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

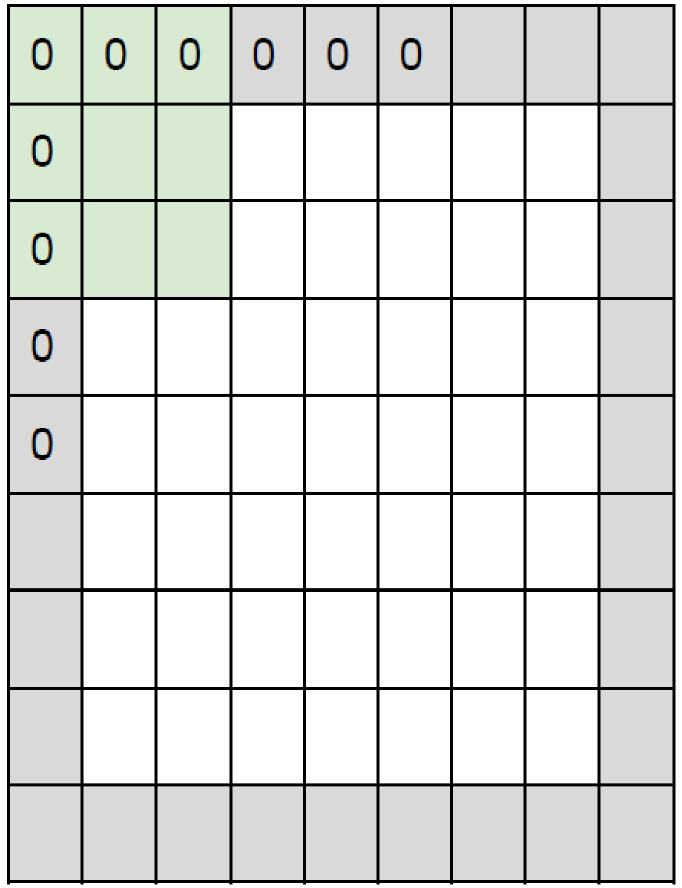


Padding (cont.)

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

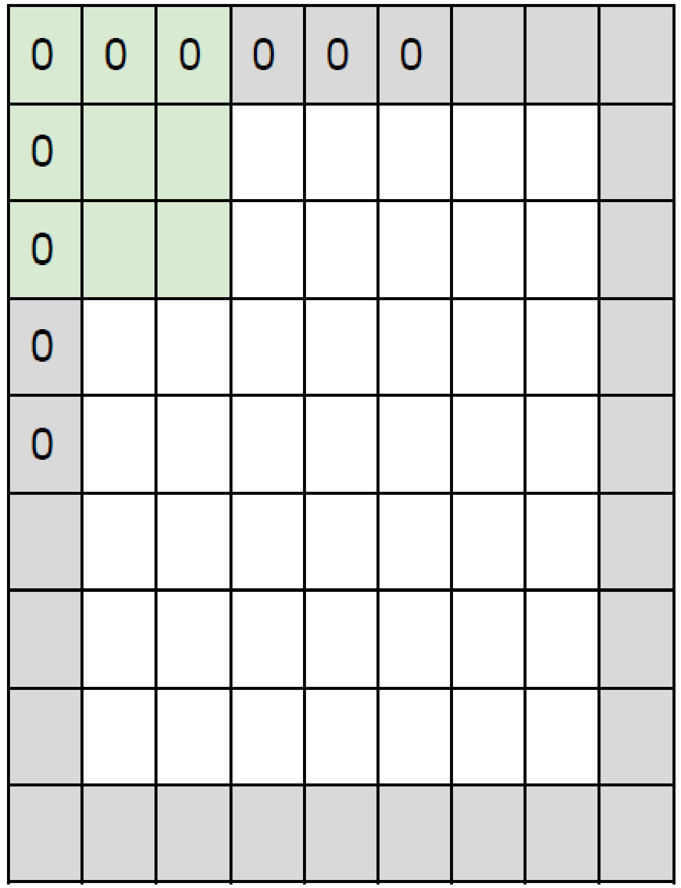


Example



- input 7x7
- **3x3** kernel, applied with **stride 1**
- **pad with 1 pixel** border => what is the size of the output?

Example (cont.)



- input 7x7
- **3x3** kernel, applied with **stride 1**
- **pad with 1 pixel** border => what is the size of the output?
- **7x7 output!**
- in general, common to see CONV layers with stride 1, kernels of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
- e.g. F = 3 => zero pad with 1

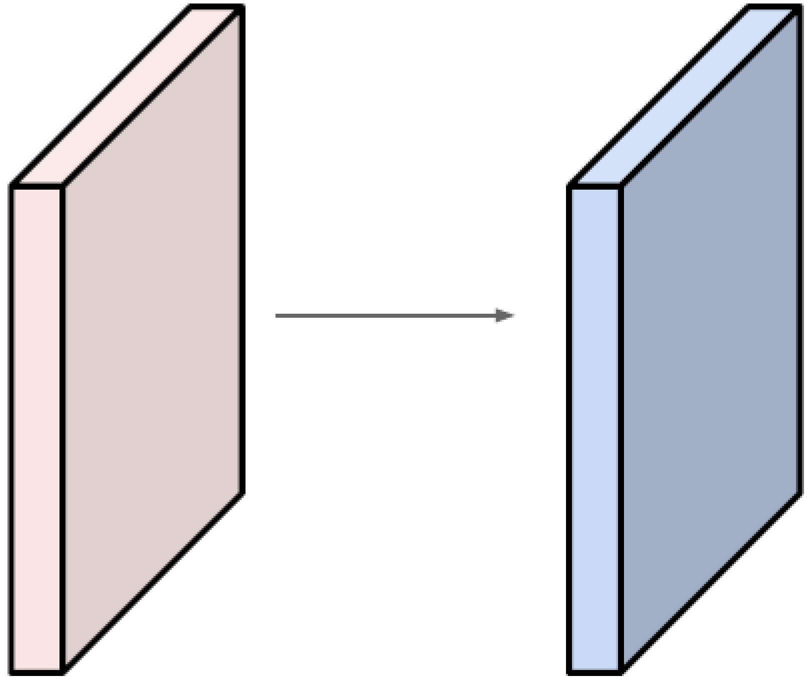
  F = 5 => zero pad with 2

  F = 7 => zero pad with 3

Example 2

- Input volume: 32x32x3
- 10 5x5 filters with stride 1, pad 2

- Output volume size: ?

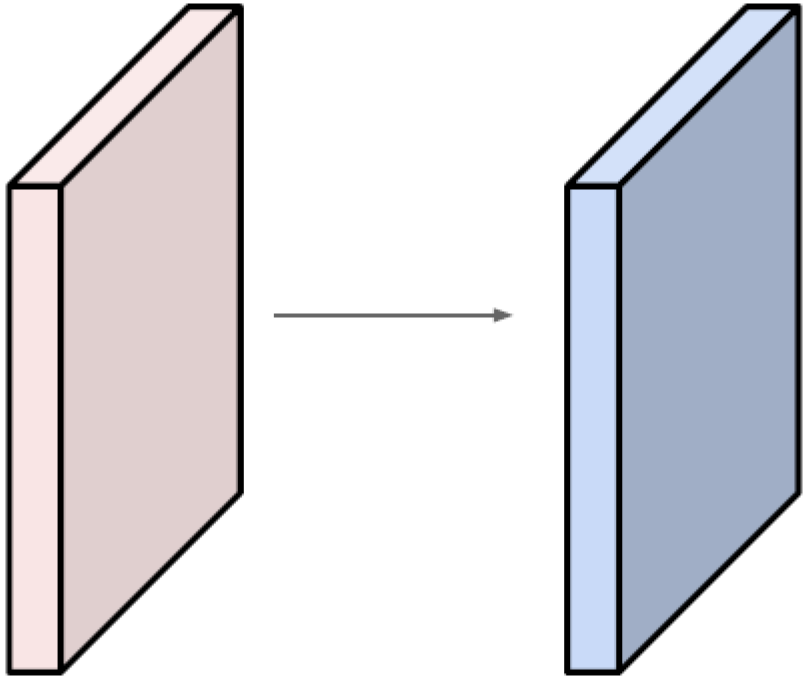


Example 2 (cont.)

- Input volume: 32x32x3
- 10 5x5 filters with stride 1, pad 2

- Output volume size: ?

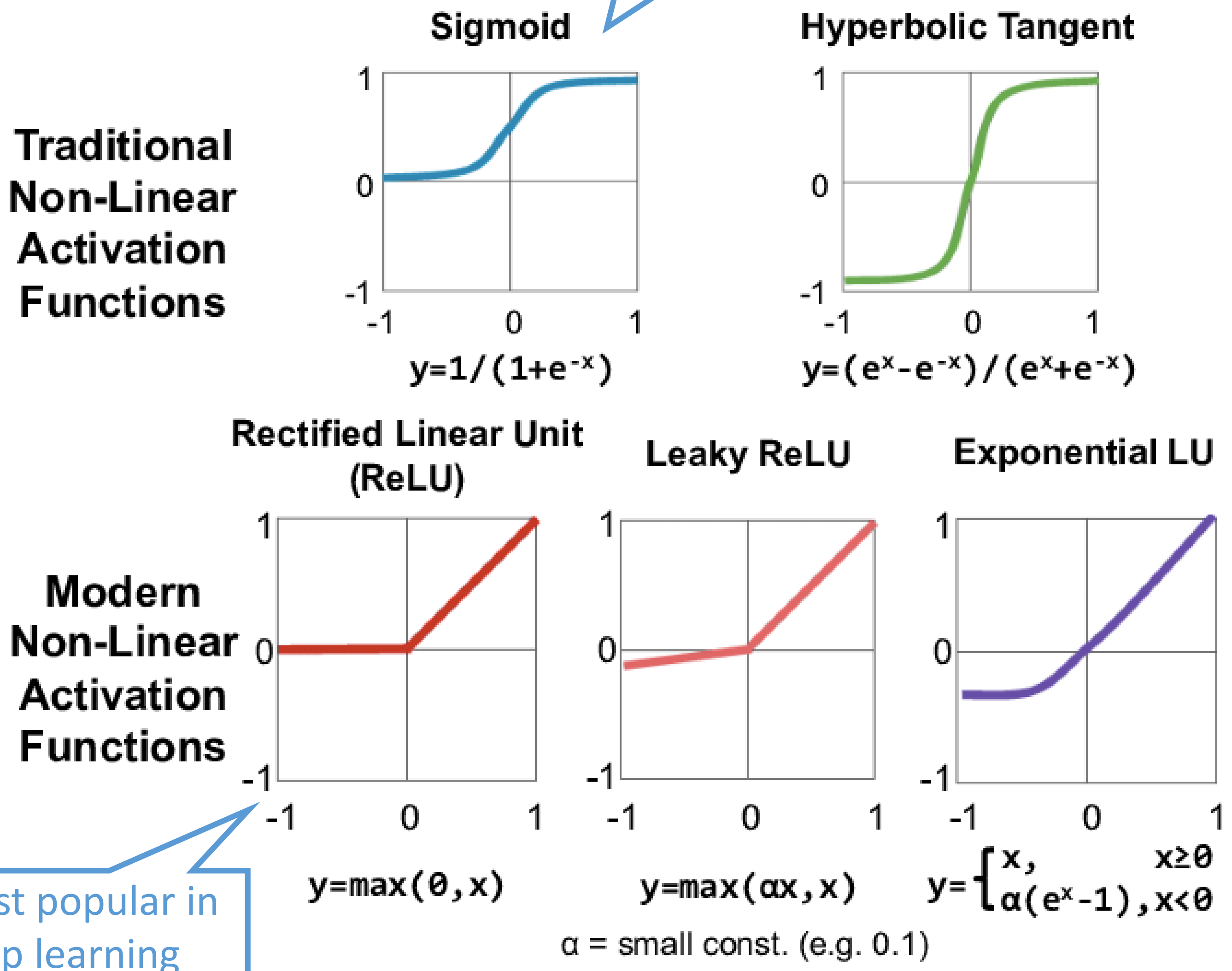
- (32+2*2-5)/1+1 = 32 spatially, so
- 32x32x10



Pooling

Activation functions (Review)

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



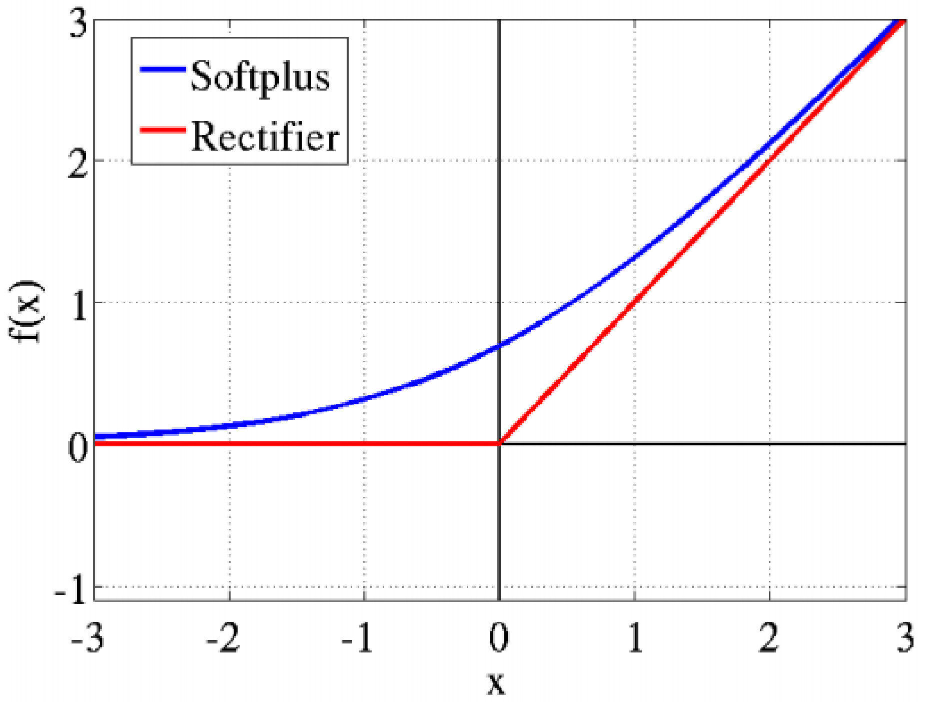
Most popular in fully connected neural network

Most popular in deep learning

ReLU activation function

- ReLU (Rectified linear unity) function

$$f_{\text{ReLU}}(x) = \max(0, x)$$



- Its derivative

$$f_{\text{ReLU}}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- ReLU can be approximated by softplus function

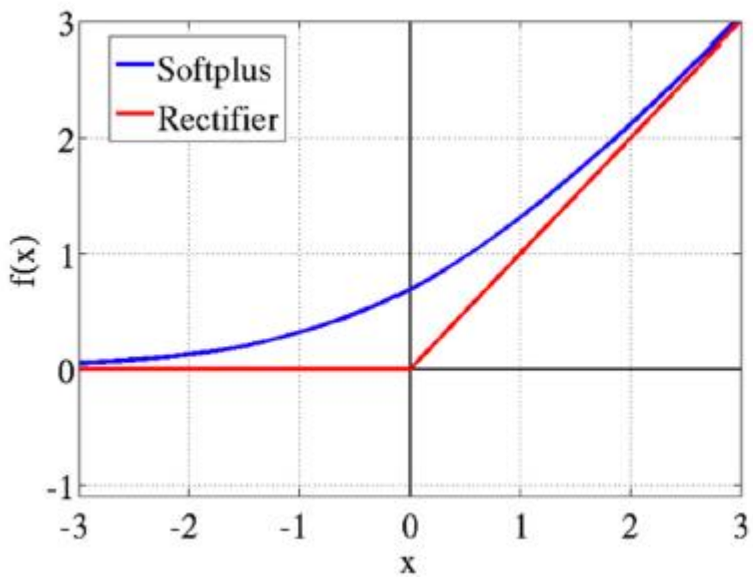
$$f_{\text{Softplus}}(x) = \log(1 + e^{x})$$

- 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

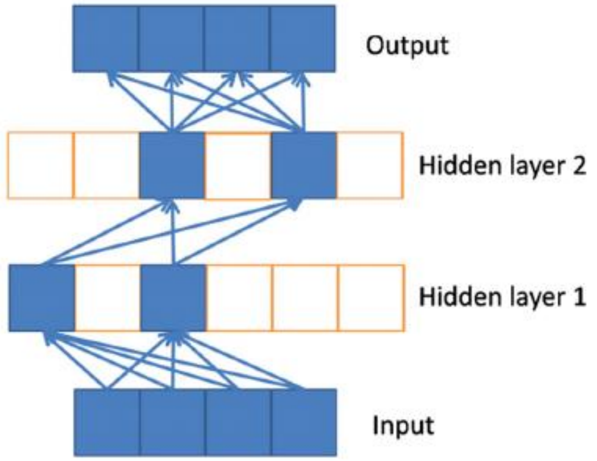
ReLU activation function (cont.)

- ReLU function

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



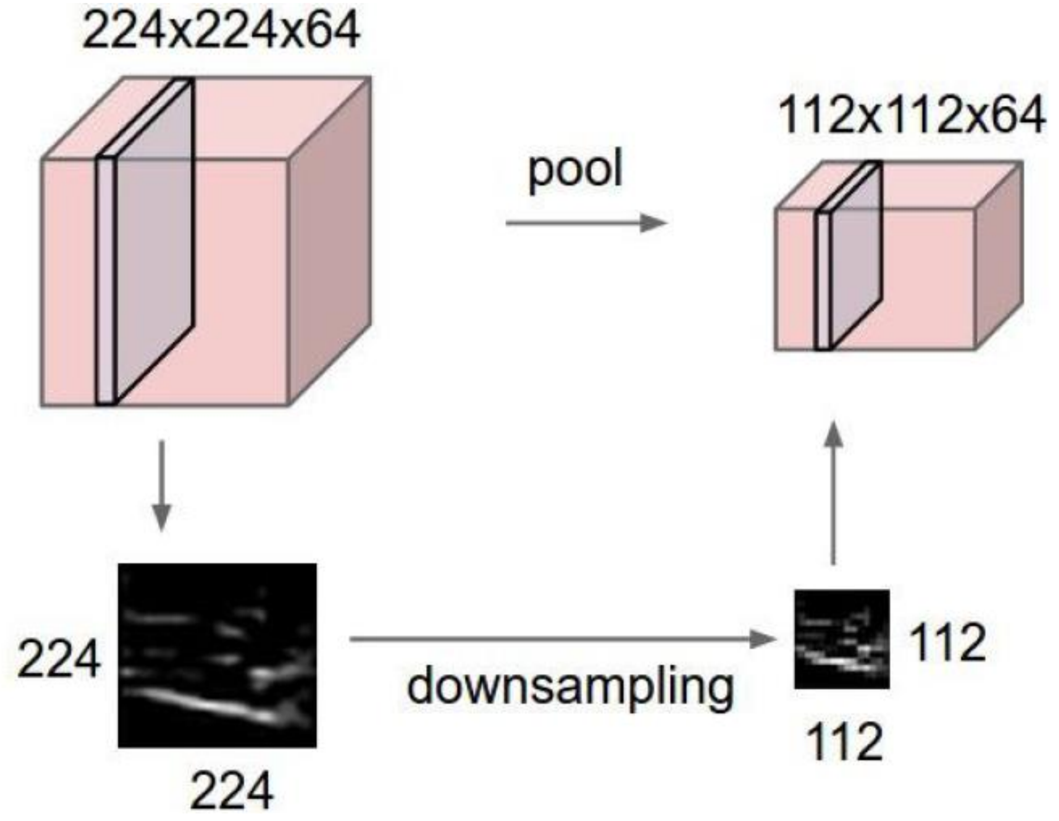
- The only non-linearity comes from the path selection with individual neurons being active or not
- It allows sparse representations:
    - for a given input only a subset of neurons are active



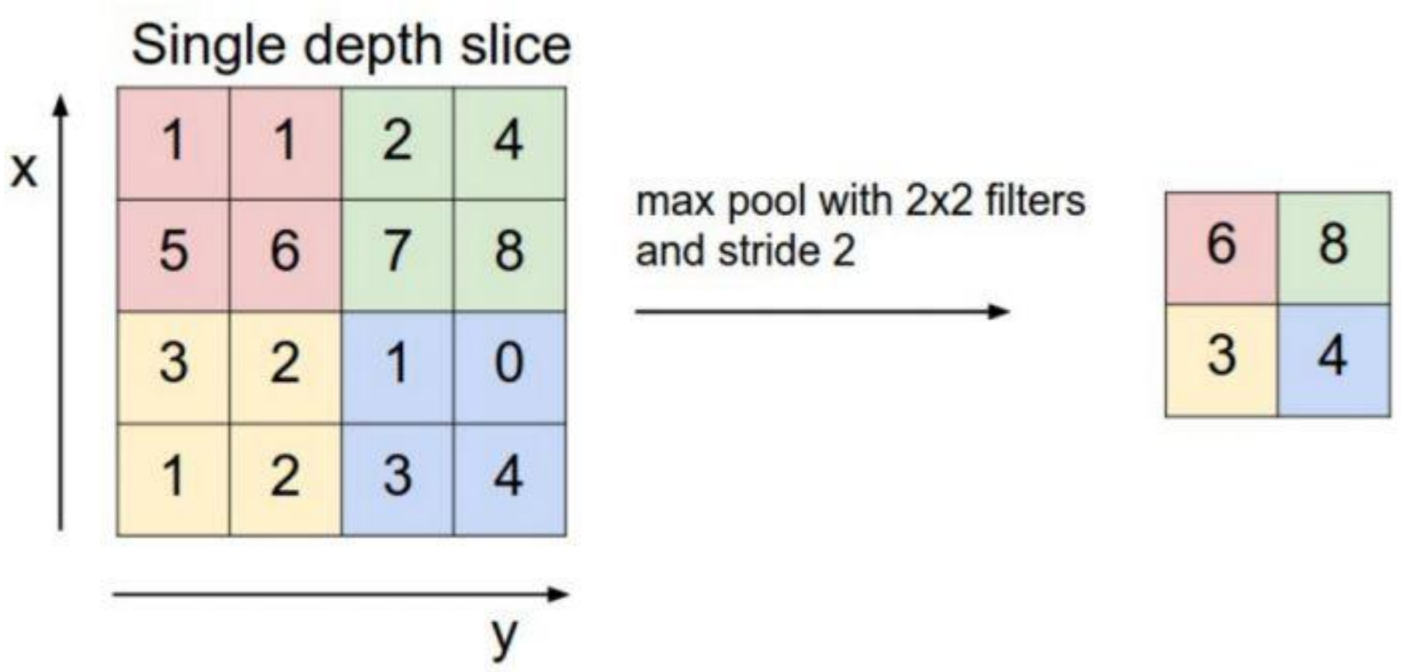
Sparse propagation of activations and gradients

Pooling layer

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



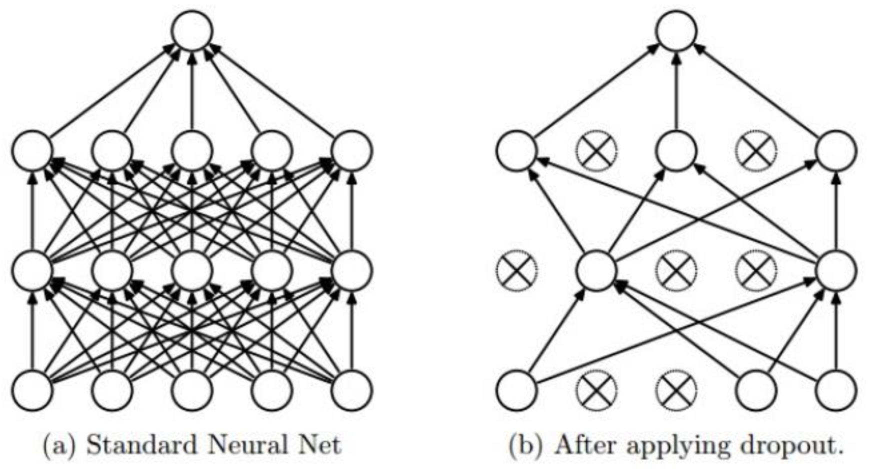
Pooling


Single depth slice
x
y
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

Training Techniques

Dropout

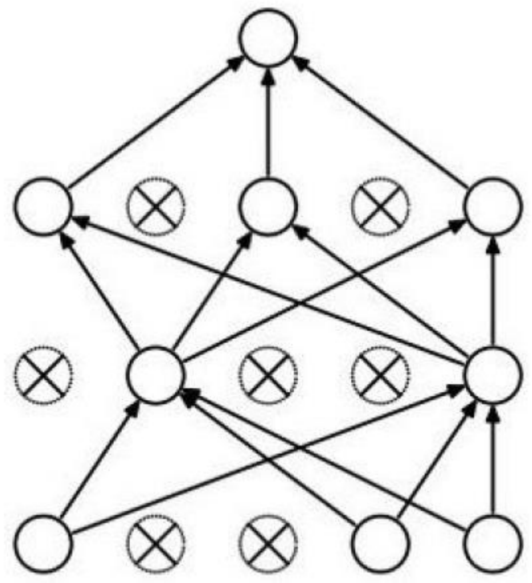
- Dropout randomly 'drops' units from a layer on each training step, creating 'sub-architectures' within the model.
- It can be viewed as a type of sampling of a smaller network within a larger network
- Prevent neural networks from overfitting

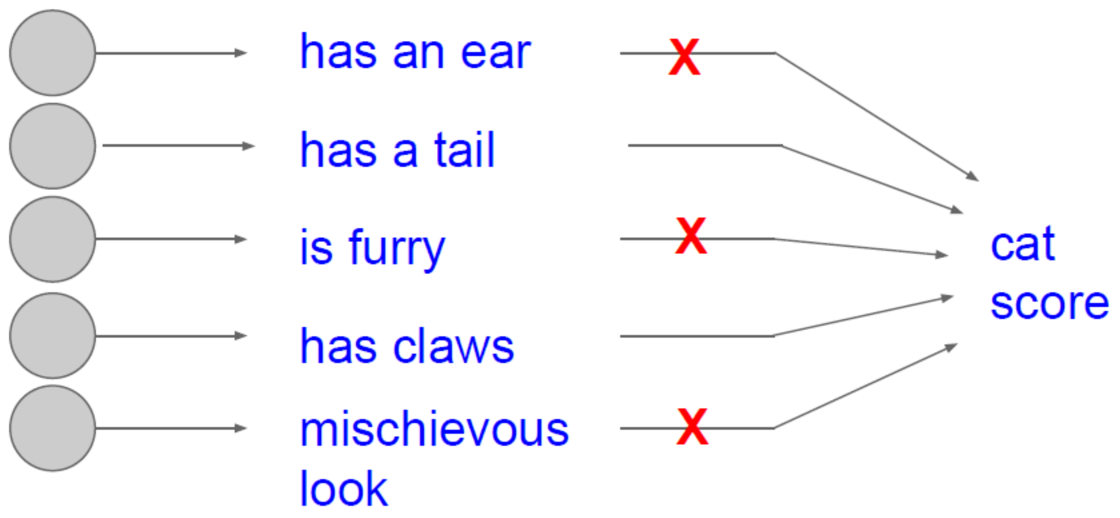


Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

Dropout (cont.)

Forces the network to have a redundant representation.





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 standard diavation of sqrt(2/n), where n is the number of outputs to the 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 unconvergence.

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 input for the ith neuran in jth layer.

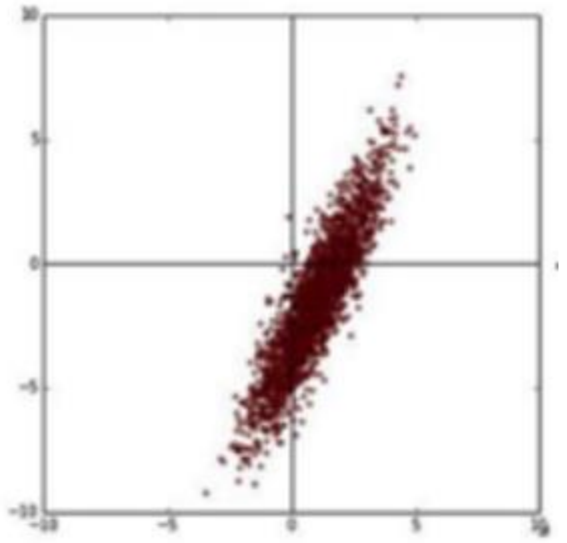
$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} Z_j^{(i)}$$

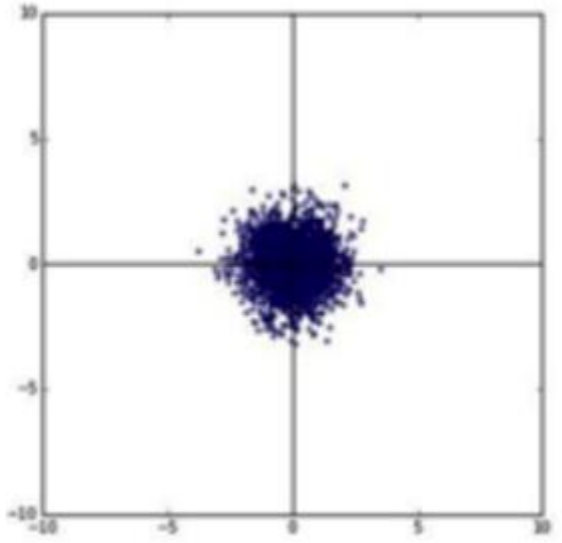
$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^{m} (Z_j^{(i)} - \mu_j)^2$$

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

Batch normalization

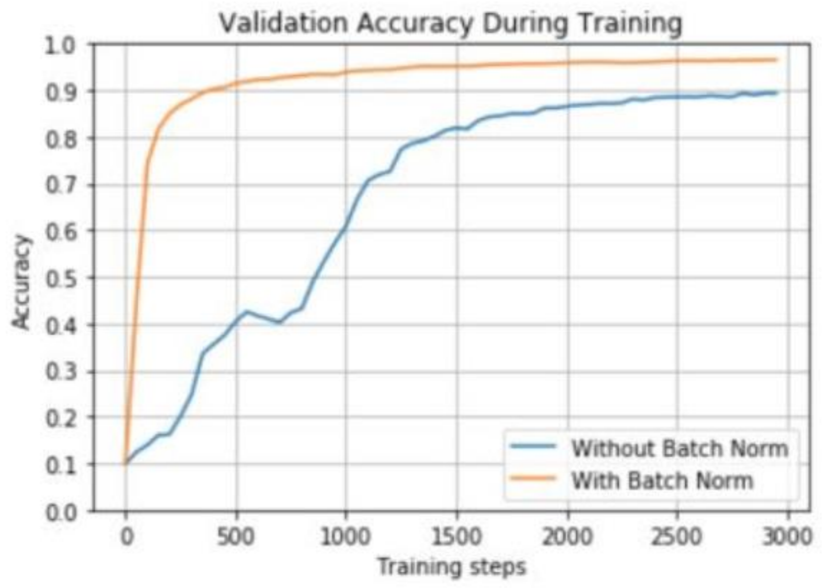
- 2D example of batch normalization

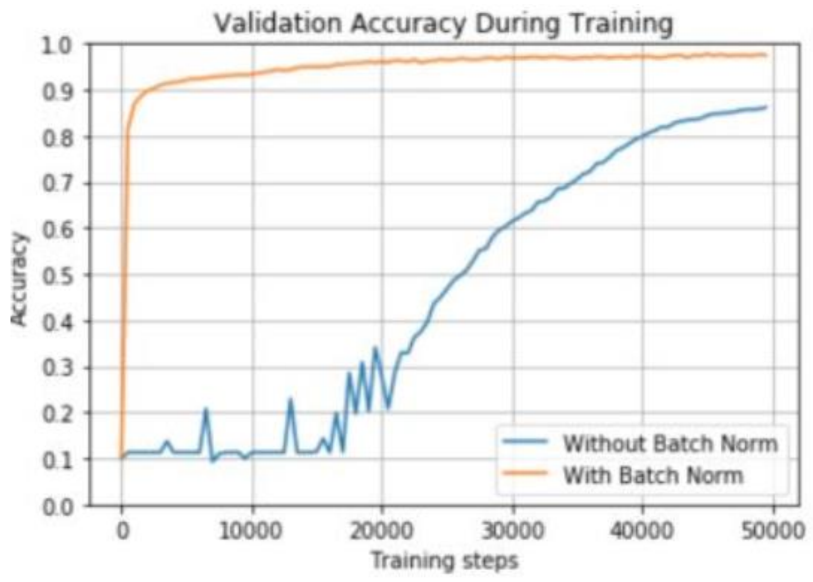




Batch Normalization

- Some experimental example

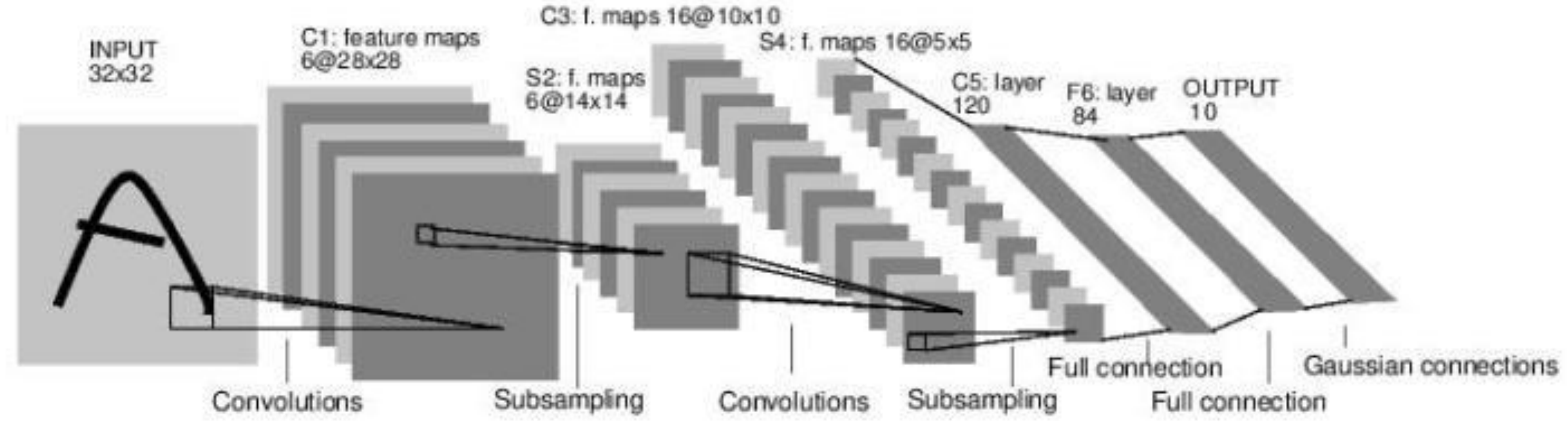




Famous Neural Networks

LeNet

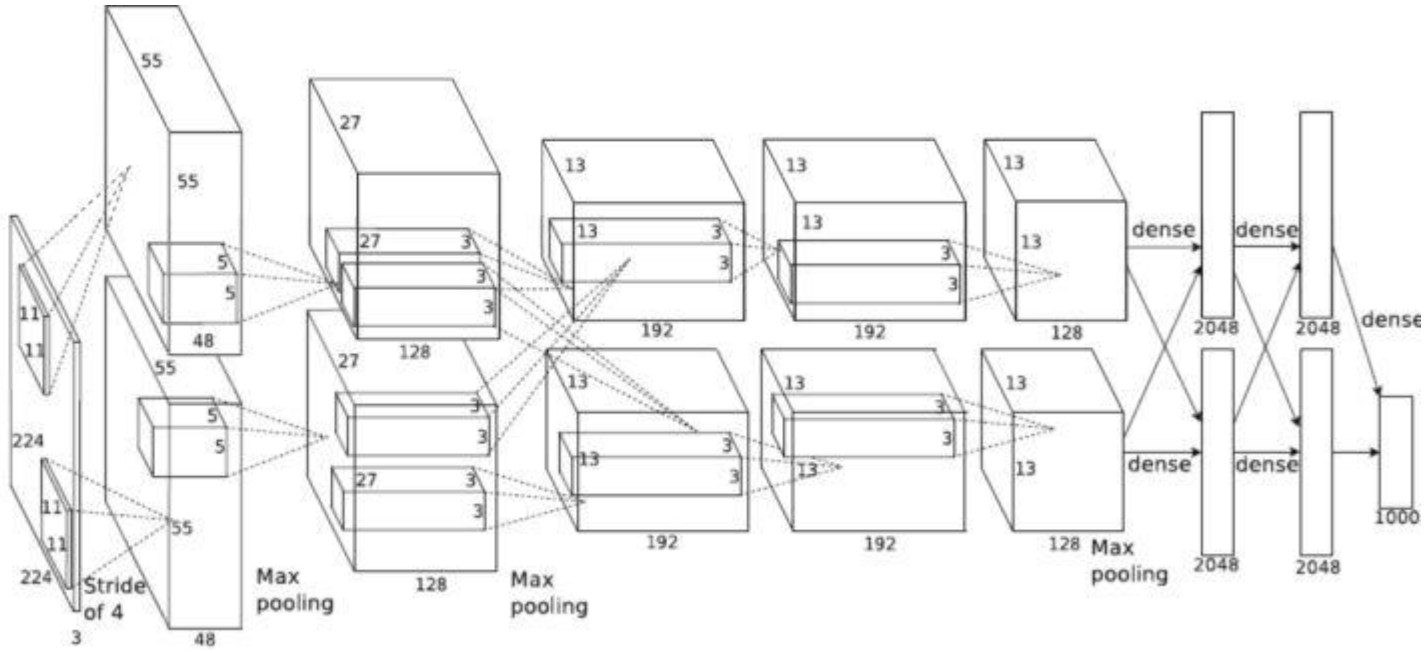
- LeNet: [LeCun et al., 1998]



Input: 28*28*1 image, Conv kernels were 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

- AlexNet: [Krizhevsky et al. 2012]



Input: 227x227x3 images

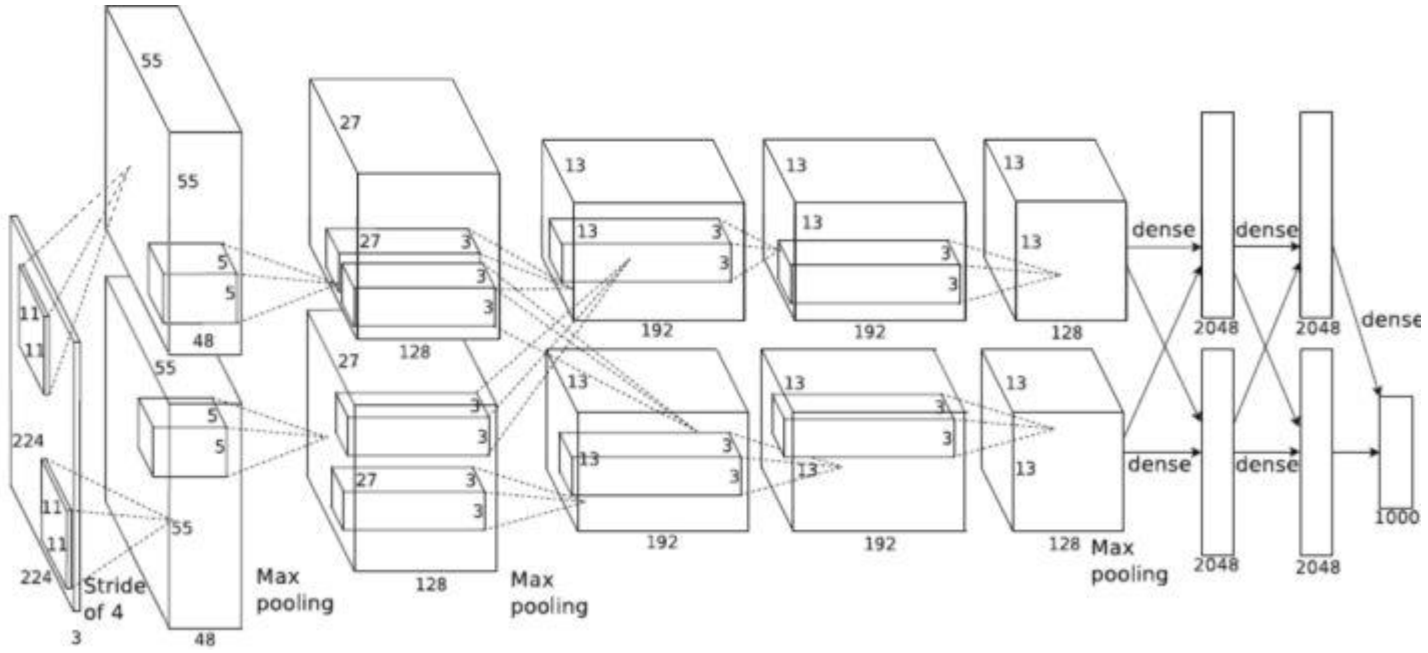
First layer (CONV1): 96 11x11 kernels applied at stride 4

=>

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

AlexNet (cont.)

- AlexNet: [Krizhevsky et al. 2012]



Input: 227x227x3 images

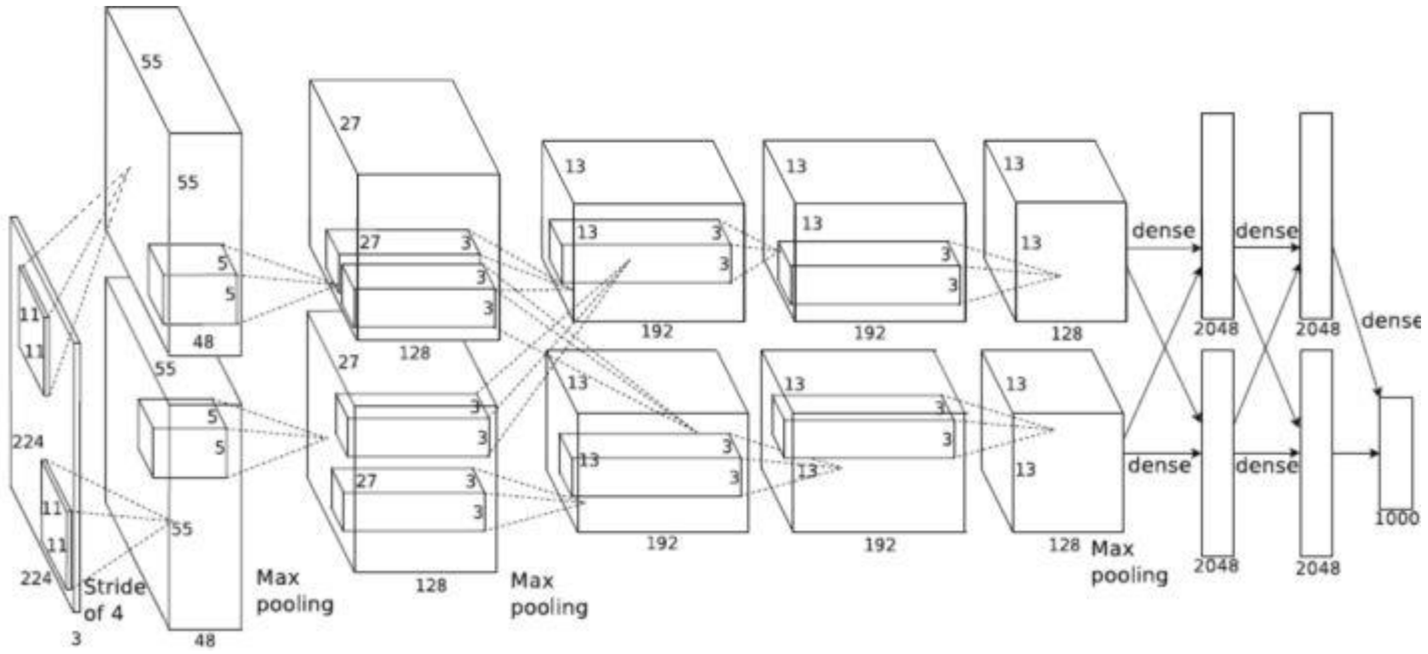
First layer (CONV1): 96 11x11 kernels applied at stride 4

=>

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

AlexNet (cont.)

- AlexNet: [Krizhevsky et al. 2012]



Input: 227x227x3 images

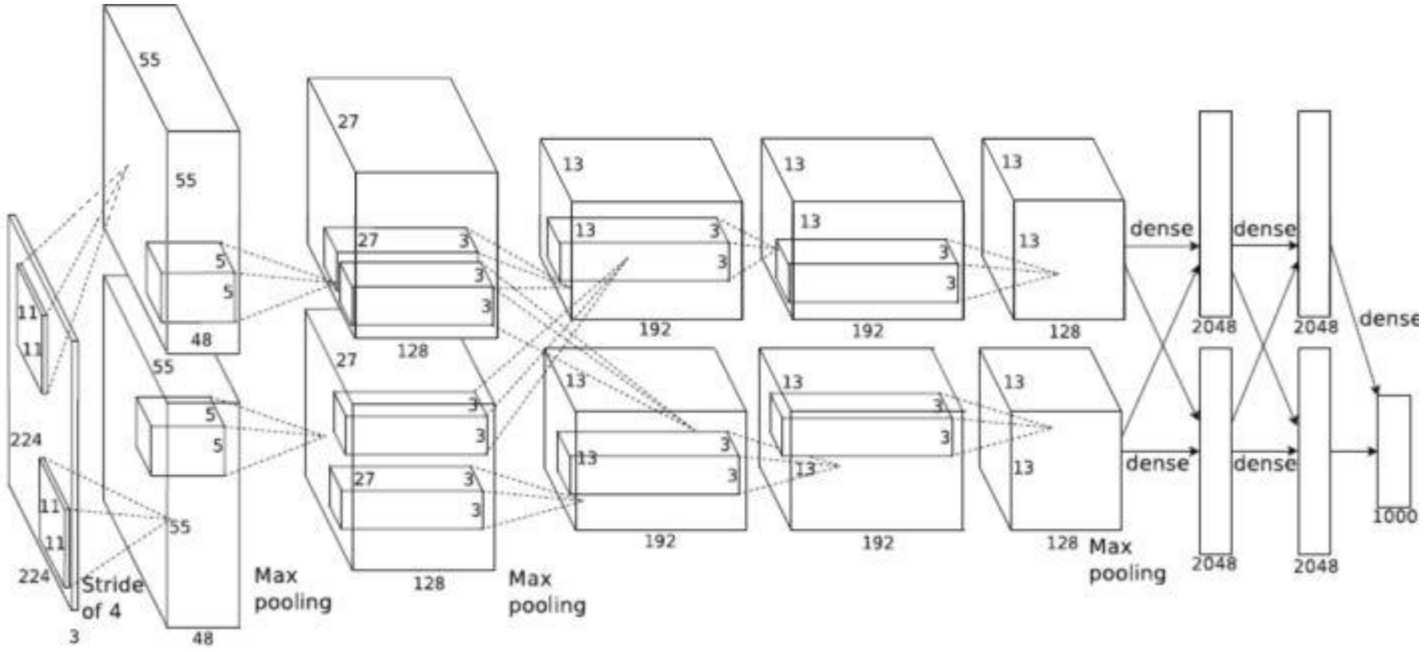
First layer (CONV1): 96 11x11 kernels applied at stride 4

=>

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

AlexNet (cont.)

- AlexNet: [Krizhevsky et al. 2012]



Input: 227x227x3 images
First layer (CONV1): 96 11x11 kernels applied at stride 4
=>
Q: What is the total number of parameters in this layer? (11*11*3)*96 = **35K**

VGGNet

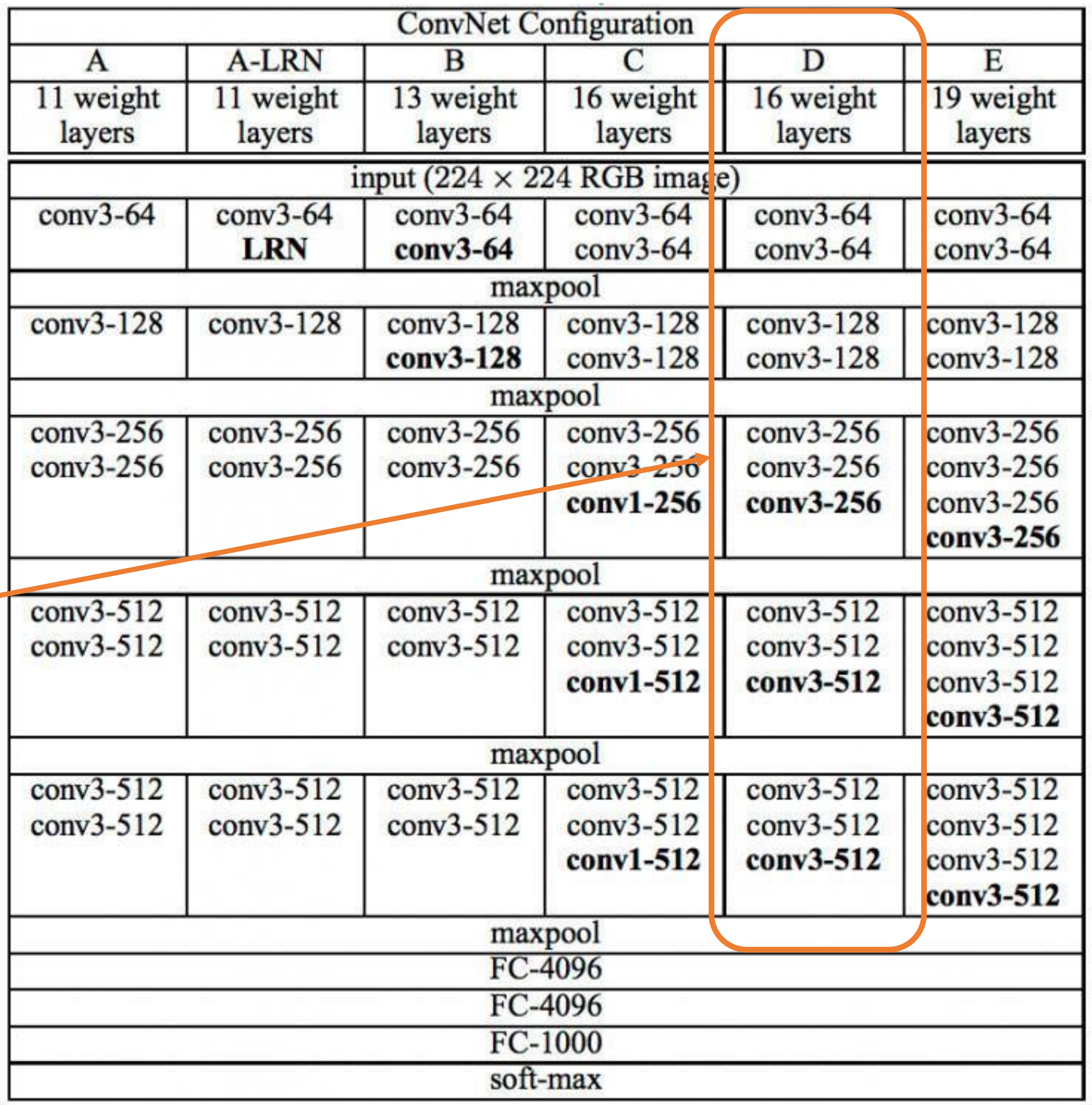
- VGGNet: [Simonyan and Zisserman, 2014]
- Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

Best model

- 11.2% top 5 error in ILSVRC 2013

  ->

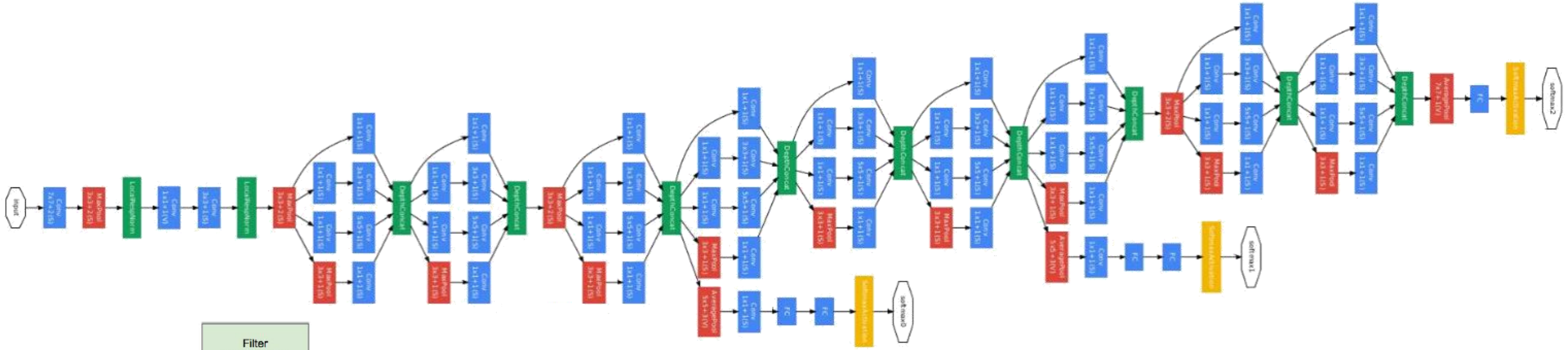
  7.3% top 5 error

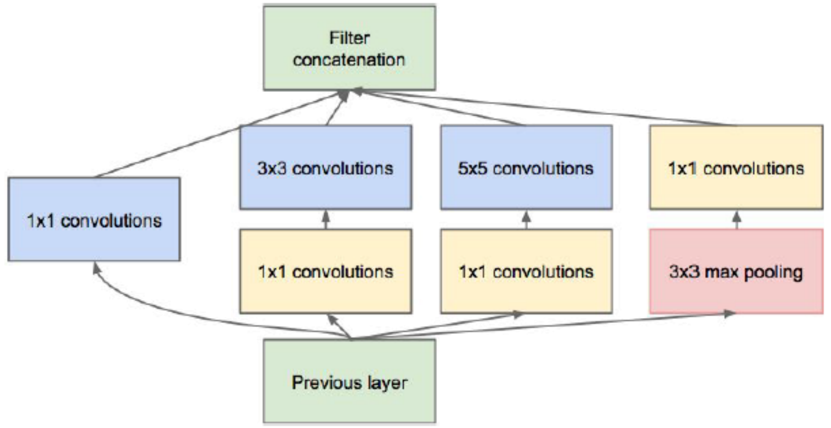


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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

GoogleNet

- GoogleNet: [Szegedy et al., 2014]

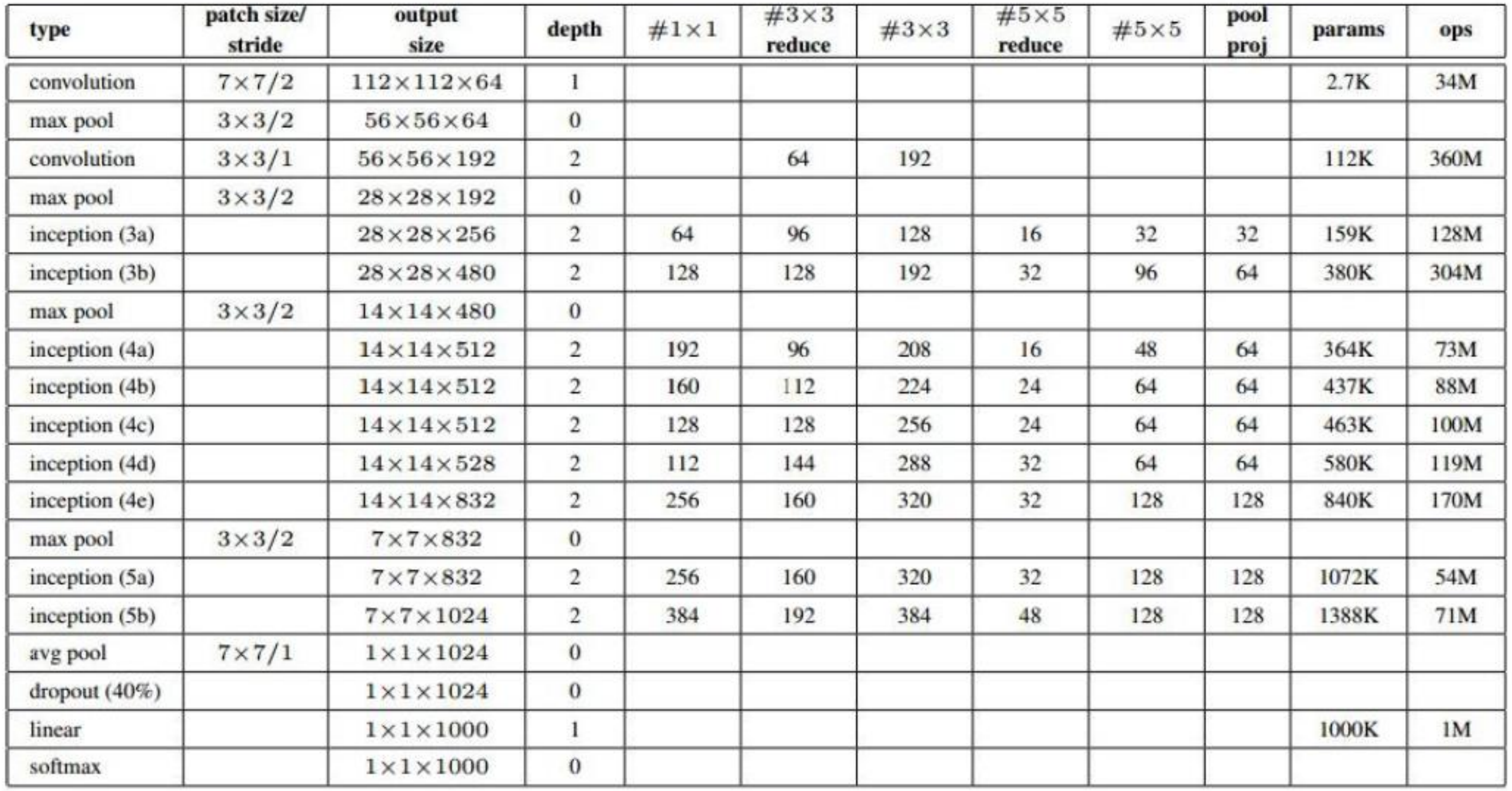




ILSVRC 2014 winner (6.7% top 5 error)

GoogleNet (cont.)

- GoogleNet

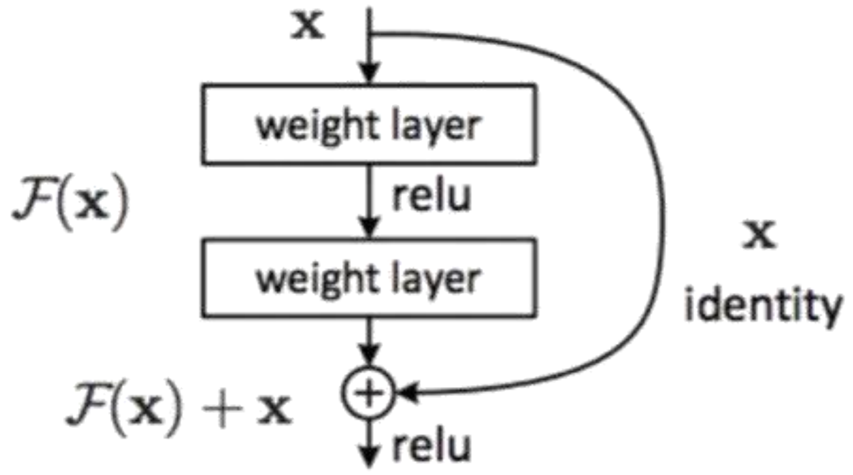


type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	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 params!
(Removes FC layers completely)
Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

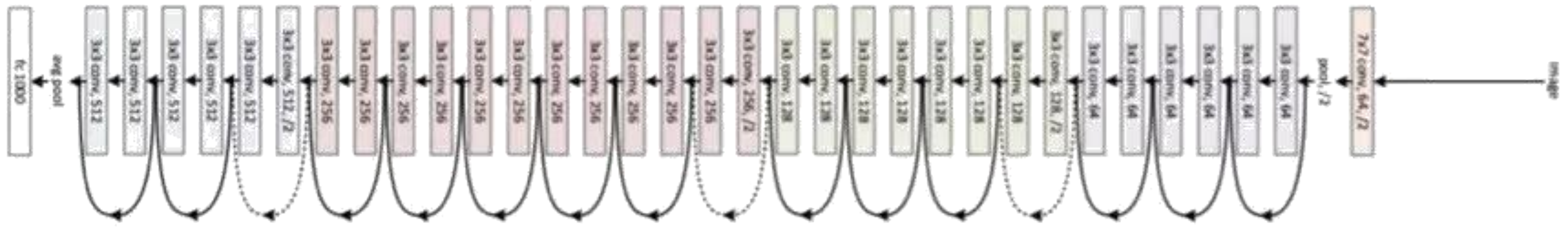
ResNet

- ResNet: [He et al., 2015] : solves the problem of drifting by adding the original input to later layers.



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

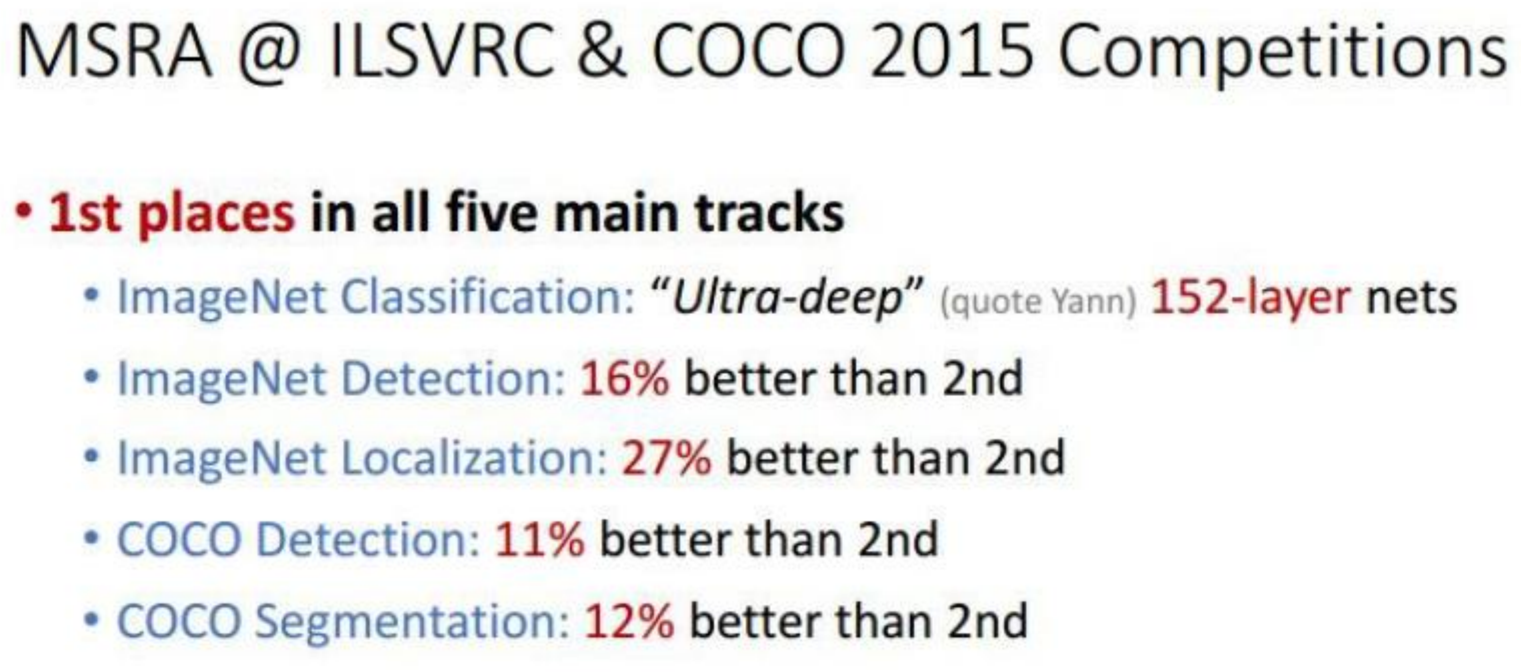
(a)



(b)

ResNet (cont.)

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

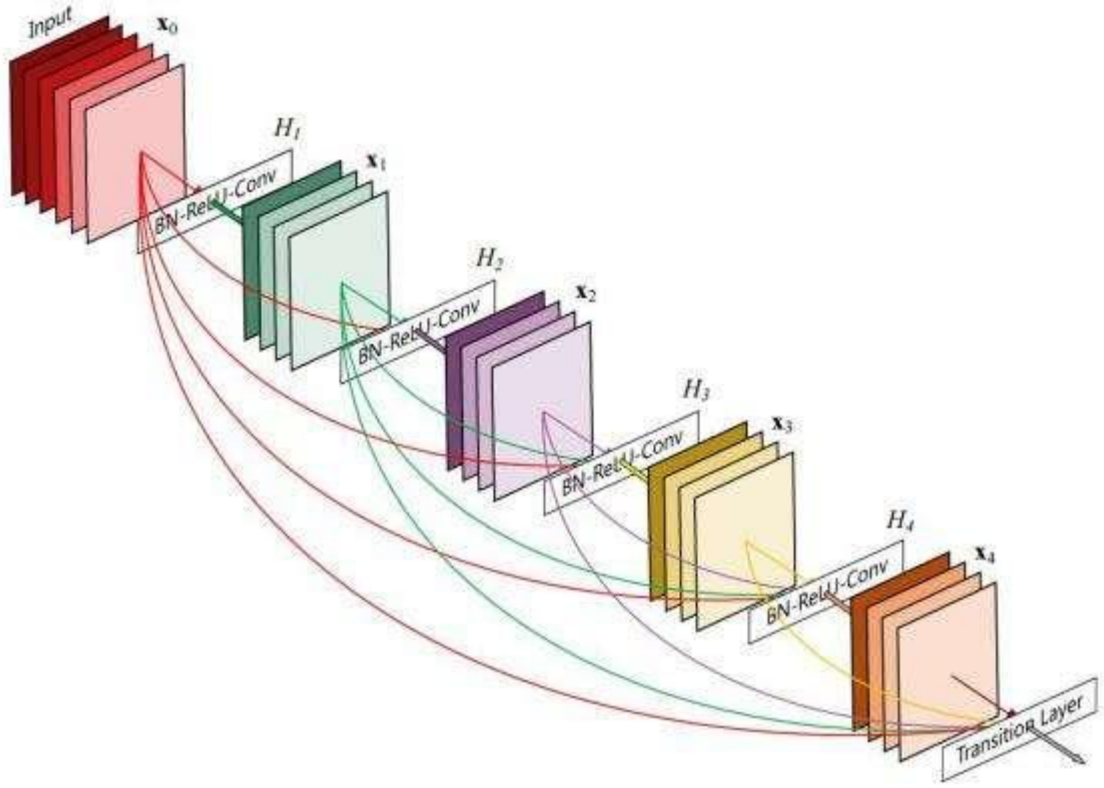


MSRA @ ILSVRC & COCO 2015 Competitions

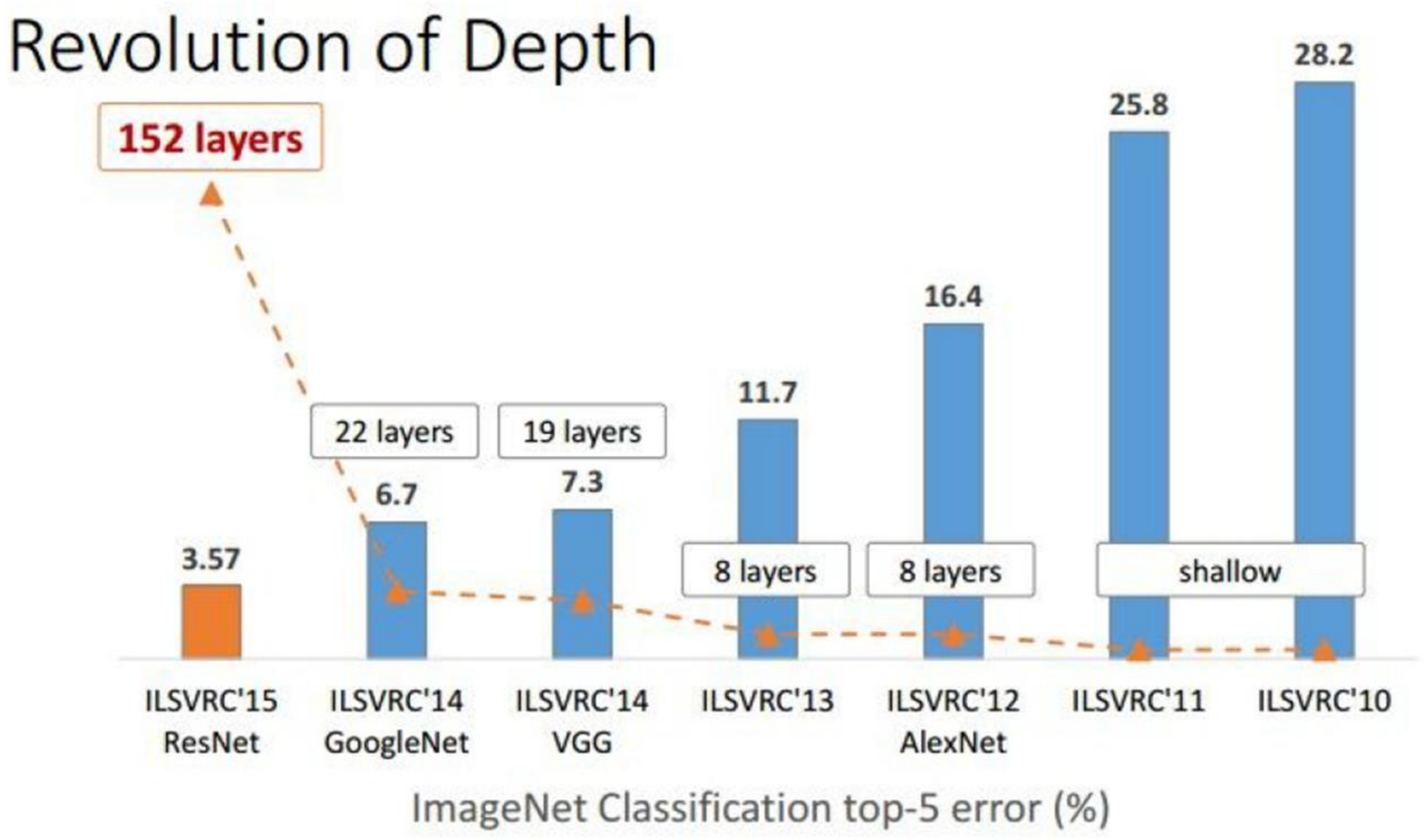
- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

DenseNet

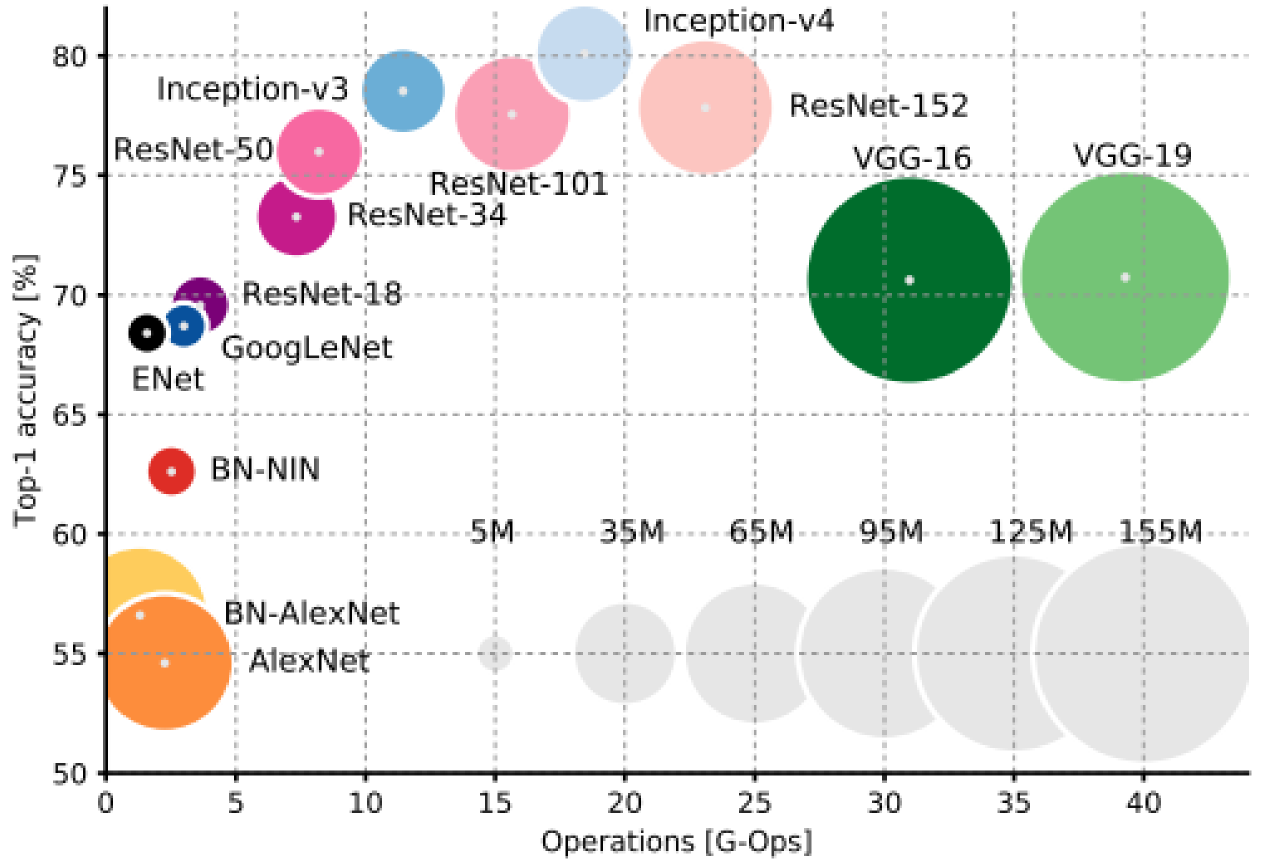
- DenseNet



Revolution of depth



Architecture comparison


Inception-v4
Inception-v3
ResNet-152
ResNet-50
VGG-16
VGG-19
ResNet-101
ResNet-34
ResNet-18
GoogLeNet
ENet
BN-NIN
BN-AlexNet
AlexNet
5M
35M
65M
95M
125M
155M
Top-1 accuracy [%]
Operations [G-Ops]