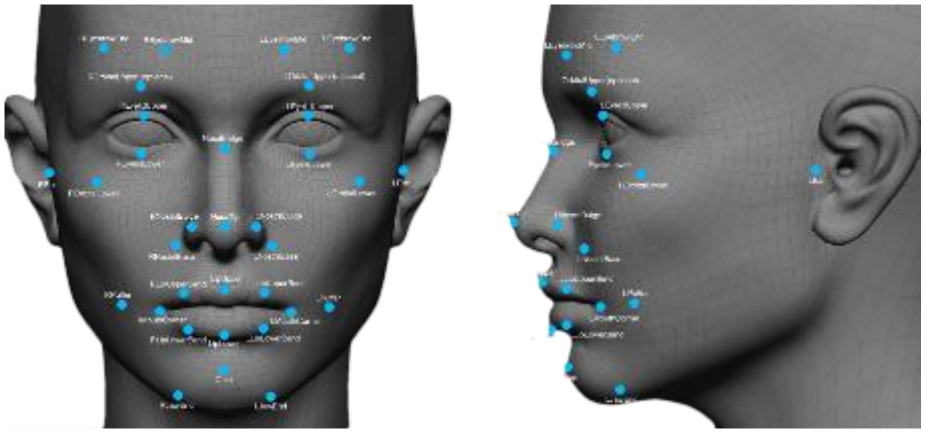
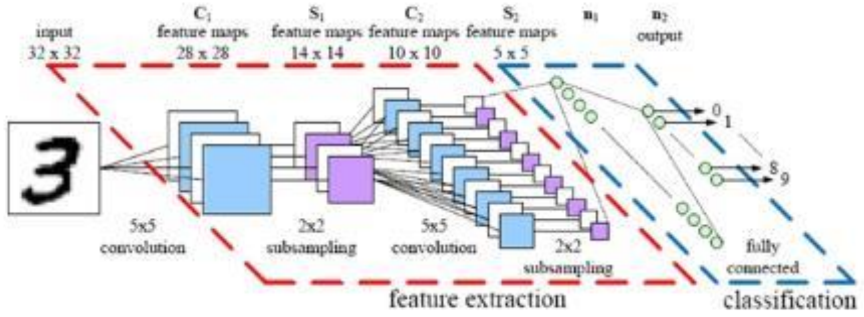


Convolutional neural networks. Part 1

Lecture # 8 of MIPT course: Introduction to machine learning



Speaker: Alexey O. Seleznev
PhD in Computational Chemistry

5VISION TEAM

Moscow
2017

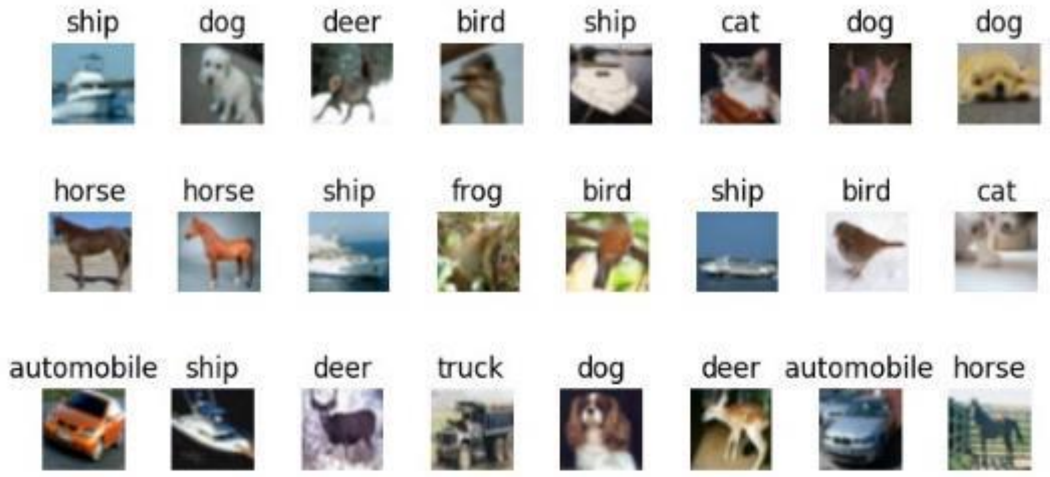
OUTLINE

Motivation for Convolutional Neural Networks

Challenges in Computer Vision

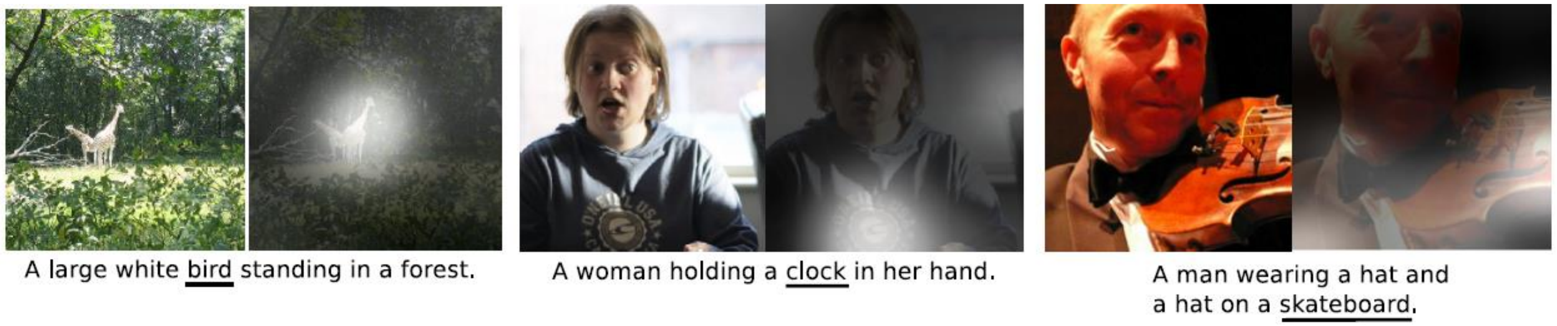
❖ **Typical Computer Vision Tasks**

- Image recognition:



CIFAR database

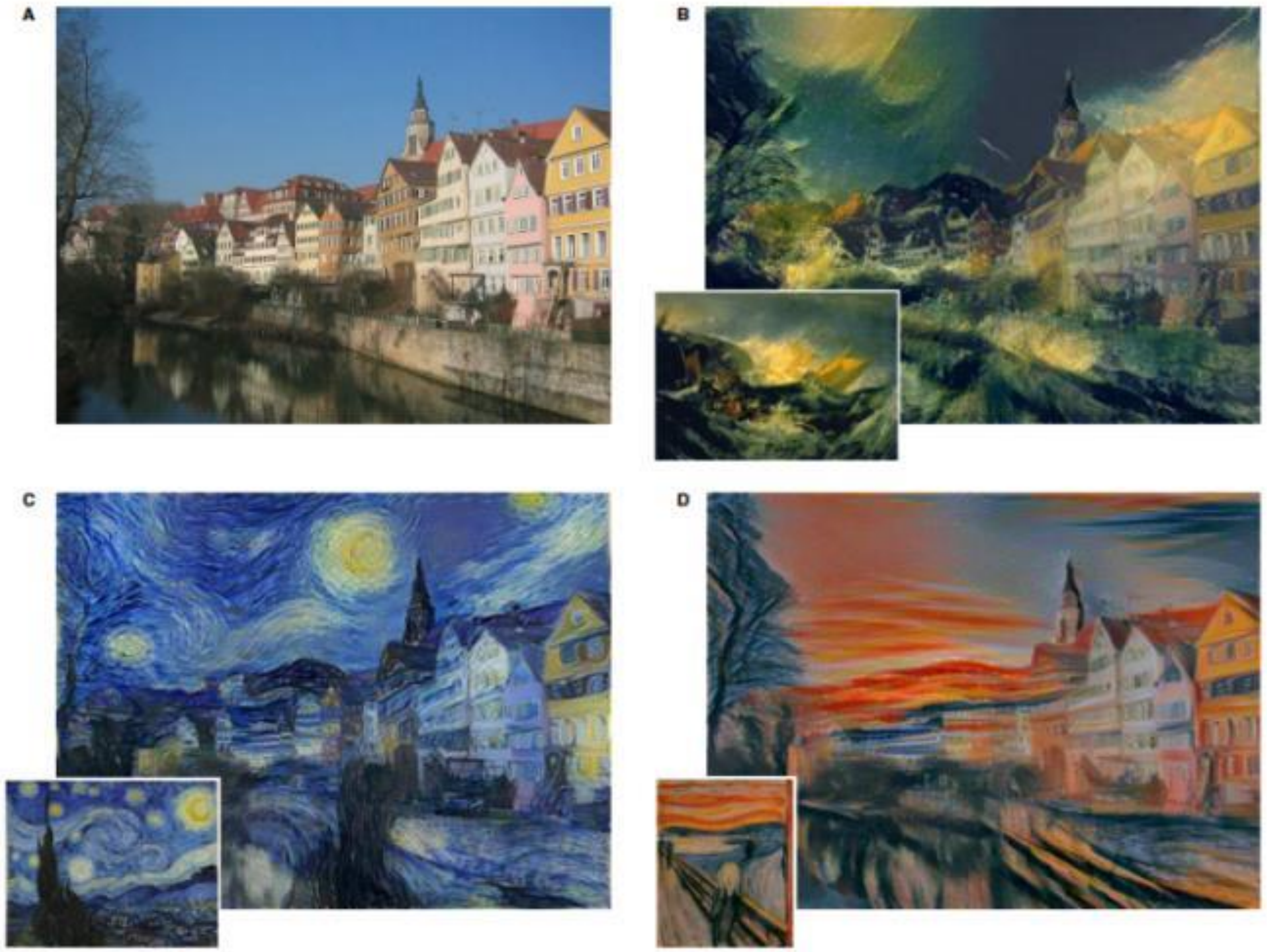
- Object detection/localization:


A large white bird standing in a forest.
A woman holding a clock in her hand.
A man wearing a hat and a hat on a skateboard.

Xu et al (2015)

Challenges in Computer Vision

- Transfer style:



Challenges in Computer Vision

- Generate new images related to those presented in some dataset:

Google deep dream

- Video analysis

Challenges in Computer Vision

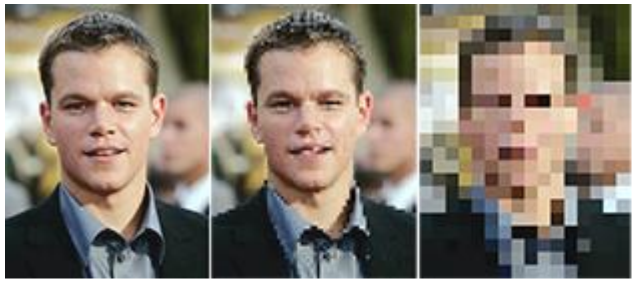
- **Major Challenges in Computer Vision**
  - Object recognition
  - Scene understanding
  - Concept learning (what is a book, a plane, a sheep)

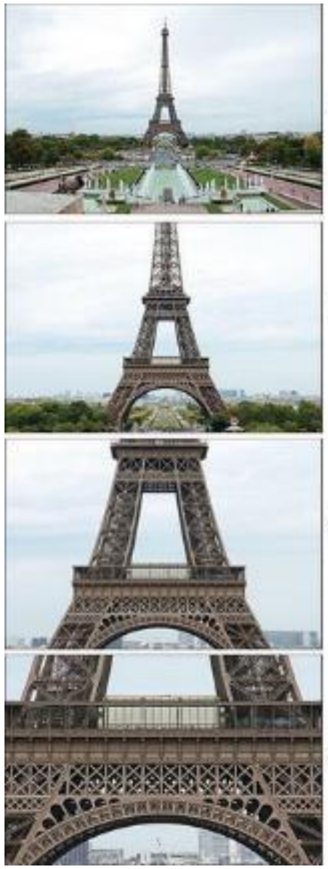
- **To solve them, we have, at least, to be able to identify same or similar objects in a set of images or frames**

Challenges in Computer Vision

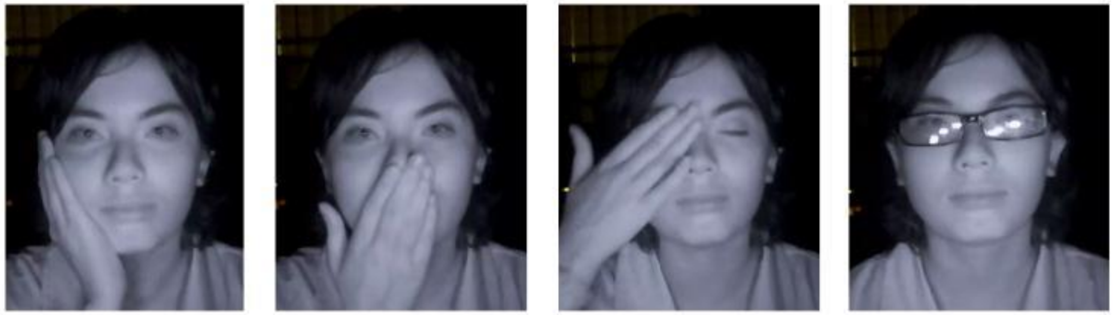
❖ **Problems with identifying similar or same objects**

- Different camera position and settings (angles, zoom, resolution):





- Occlusions and different illumination conditions:



Challenges in Computer Vision

- Different types of the same object

- Different conditions in which one may find the same object:

Challenges in Computer Vision

What allows us to correctly identify objects in all of the situations we just saw?

- Some part of the target object is invariant with respect to different transformations (translation, rotation, resizing, changing color palitra, etc)
- We can find enough invariants to discover the real nature of the object with a high degree of certainty

Thus, to alleviate the aforementioned problems, we have to analyze small spatial zones of images in search for invariants characterizing our objects of interest

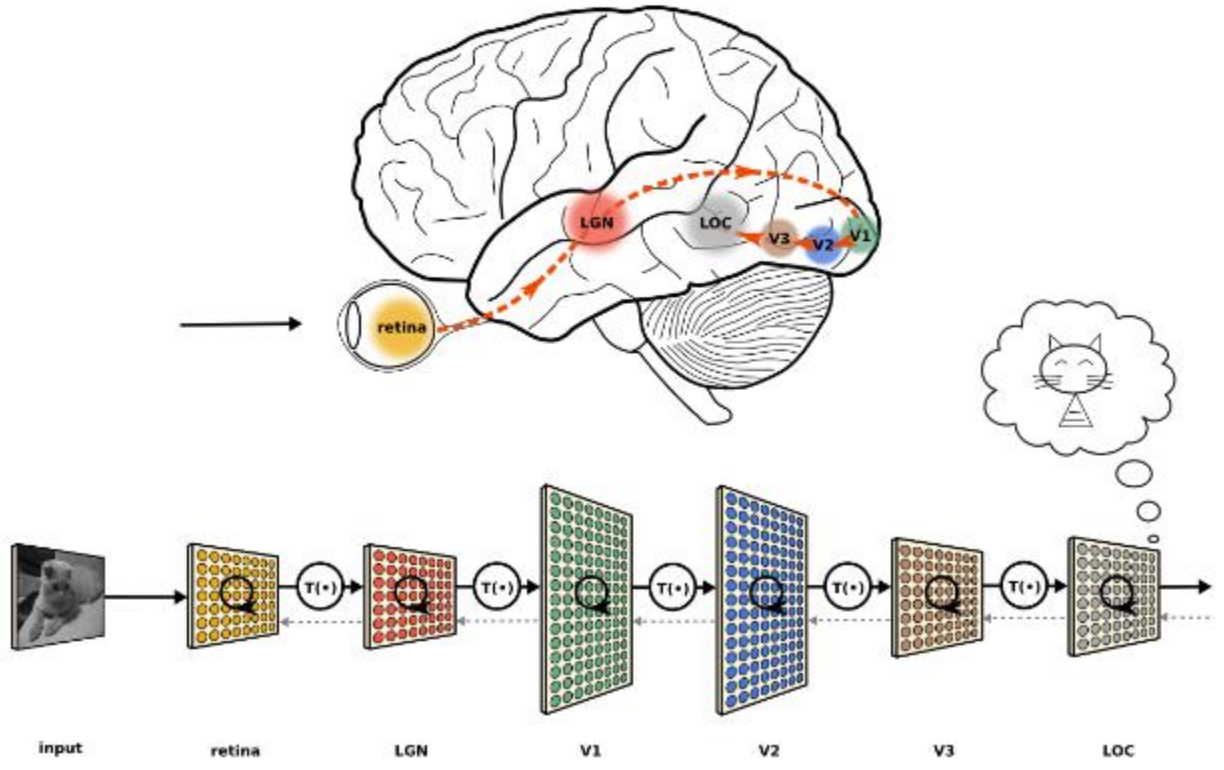
Pitfalls of Dense Layers

Why is dense layer inappropriate for the task? Why do we need a different kind of a layer?

- Huge number of parameters. E.g., to process a 512x512x3 RGB image, one needs some 8M parameters to connect him to 100 hidden units. Each pixel requires its own weight. Wasteful, considering our objective to find invariants
- Dense layer certainly can find invariants, but it binds them to the surrounding (e.g. background) so that network's generalization ability decreases
- Dense layers is not designed to process images of different sizes

How to address these restrictions? Let us see biological solution

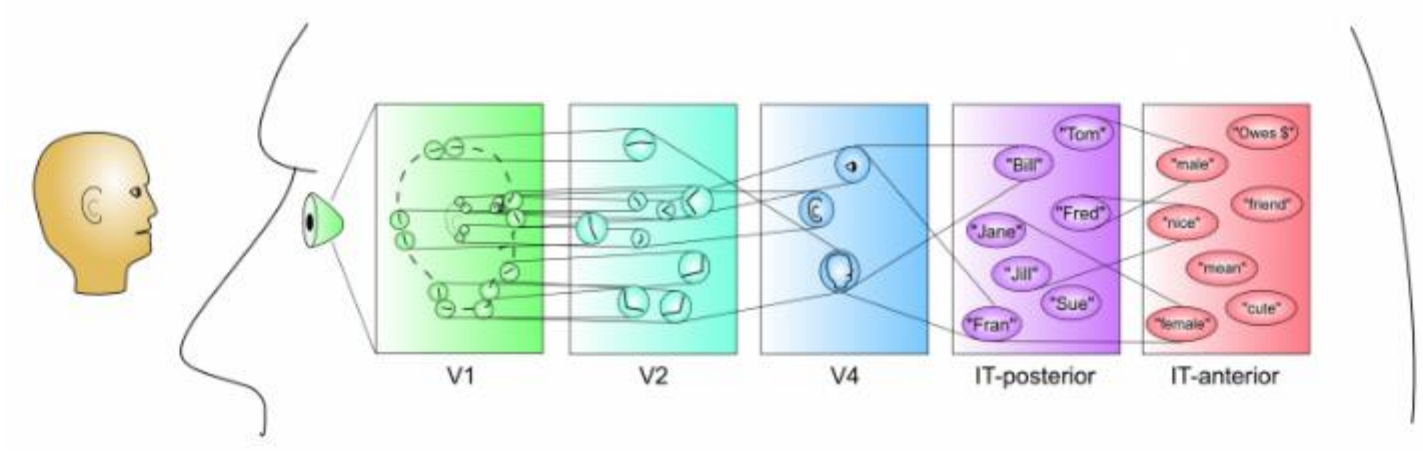
Biological Inspiration for CNN



We focus on a part of the brain called V1 – primary visual cortex.

It receives signals from retina through lateral geniculate nucleus.

Biological Inspiration for CNN



https://grey.colorado.edu/CompCogNeuro/index.php/CCNBook/Perception

Three properties of V1 are of particular importance for us: [simplified model]

- It is arranged in a spatial map having a property of locality: light arriving at the lower half of the retina affect only the corresponding half of V1
- V1 contains simple cells activated by light to some extent linearly in a small, spatially localized receptive field
- V1 also has complex cells non-linearly responding to features similar to those detected by simple cells, but invariant to small shifts in the position of the feature.

It is believed that V2, V4 have something in common with V1. Information collected and processed by V layers is supposed to be processed in a more abstract way by inferotemporal cortex (IT)

Key Properties of CNN

Three properties that we expect from CNN:

- **Sparse interactions**. Not every input neuron is connected to every output neuron at the same time. Output neurons are organized in filters travelling across the grid of input neurons. So, by doing so, we decrease the number of necessary connections. Idea inspired by both pitfalls of dense layers and working principle of V1's simple cells.
- **Parameter sharing**. Each filter is used at every position of the input. Except for learning a separate set of parameters for every location, we learn only one set.
- **Equivariance to translations**. Idea inspired by both pitfalls of dense layers and working principle of V1's complex cells

We do not apply a grid of keys to every point

We test relatively small number of keys and apply each of them to every tile of the image

Key Elements of CNN

Convolution Layer

❖ **Convolution operation**

- Formal definition

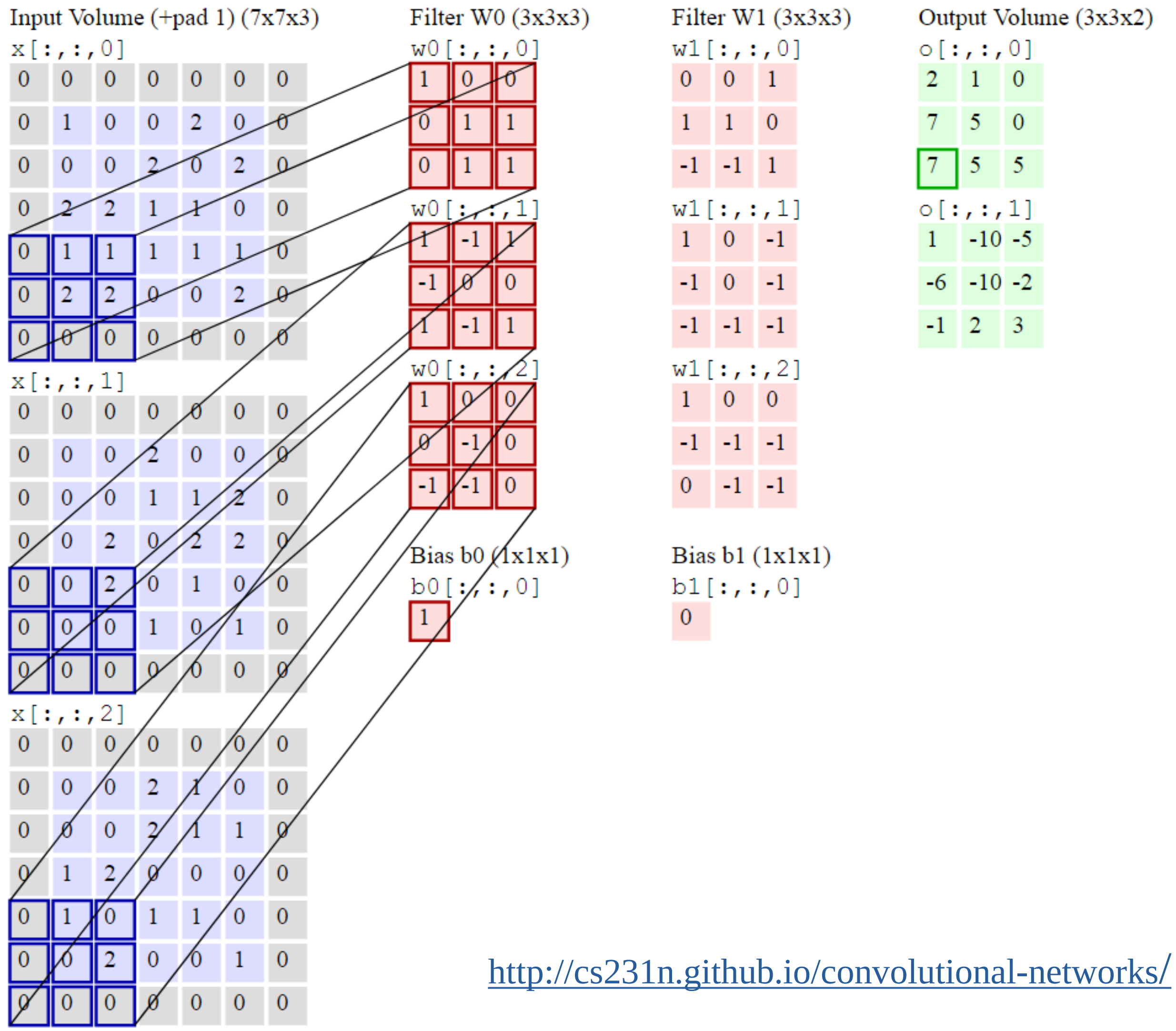
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

where I is an image, K is a kernel, or filter, i, j, m, n – pixel coordinates

- Idea behind convolution operation. The greater the sum is, the greater is the affinity between the kernel and the location under consideration.

Convolution Layer

- Convolution operation visualization:



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	0	0	2	0	0
0	0	0	2	0	2	0
0	2	2	1	1	0	0
0	1	1	1	1	1	0
0	2	2	0	0	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	0	2	0	0	0
0	0	0	1	1	2	0
0	0	2	0	2	2	0
0	0	2	0	1	0	0
0	0	0	1	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	0	2	1	0	0
0	0	0	2	1	1	0
0	1	2	0	0	0	0
0	1	0	1	1	0	0
0	0	2	0	0	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	0	0
0	1	1
0	1	1

w0[:,:,1]

1	-1	1
-1	0	0
1	-1	1

w0[:,:,2]

1	0	0
0	-1	0
-1	-1	0

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	1
1	1	0
-1	-1	1

w1[:,:,1]

1	0	-1
-1	0	-1
-1	-1	-1

w1[:,:,2]

1	0	0
-1	-1	-1
0	-1	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

2	1	0
7	5	0
7	5	5

o[:,:,1]

1	-10	-5
-6	-10	-2
-1	2	3

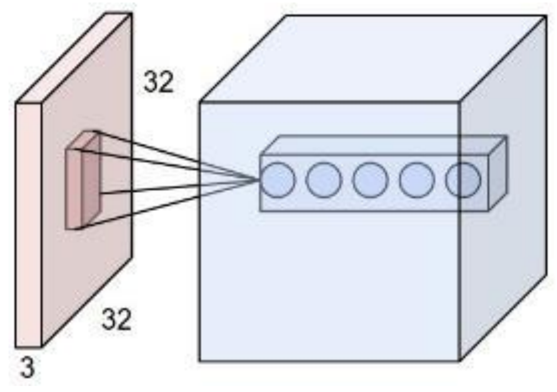
http://cs231n.github.io/convolutional-networks/

5 min break

Convolution Layer

❖ **Convolution layer**

➢ Tensor operation for image with height = width = W



(width, height, nb_channels)

⇩

(new_width, new_height, nb_filters)

➢ Main parameters of the layer:

- Number of filters, N. Usually16, 32, or 64.
- Width of the filter, F. Usually (3, 3) or (4, 4).
- Padding, P (next slide)
- Stride, S (next slide)

Convolution Layer

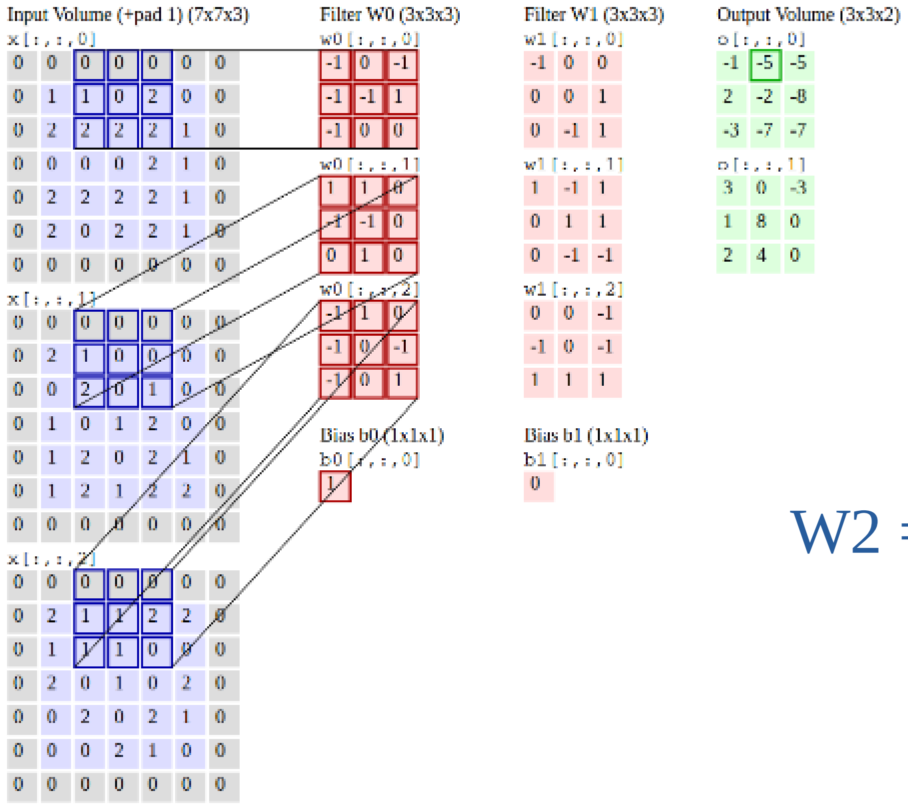
- What is padding? The width of the representation shrinks by one pixel less than the kernel width at each layer. Zero padding the input allows us to control the kernel width and the size of the output independently.
- In Keras, you have two padding options: ‘valid’ and ‘same’. Valid means no zero padding. In contrast, ‘same’ means automatic restoration of the size of the input
- What is stride? Shift with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.
- Usual choice of S and P: S = 1 or 2, padding = ‘valid’ or ‘same’
- The Pyramid Principle. Why should we stack convolution layers? Increase in representational power.

Convolution Layer

- How to find the size of output layer?

$$W2 = (W1-F+2P)/S+1$$

- Let us apply this equation to the following operation:



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	0	2	0	0
0	2	2	2	2	1	0
0	0	0	0	2	1	0
0	2	2	2	2	1	0
0	2	0	2	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	0	0	0	0
0	0	2	0	1	0	0
0	1	0	1	2	0	0
0	1	2	0	2	1	0
0	1	2	1	2	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	2	0
0	1	1	1	0	0	0
0	2	0	1	0	2	0
0	0	2	0	2	1	0
0	0	0	2	1	0	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	-1
-1	-1	1
-1	0	0

w0[:,:,1]

1	1	0
-1	-1	0
0	1	0

w0[:,:,2]

-1	1	0
-1	0	-1
-1	0	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	0	0
0	0	1
0	-1	1

w1[:,:,1]

1	-1	1
0	1	1
0	-1	-1

w1[:,:,2]

0	0	-1
-1	0	-1
1	1	1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

-1	-5	-5
2	-2	-8
-3	-7	-7

o[:,:,1]

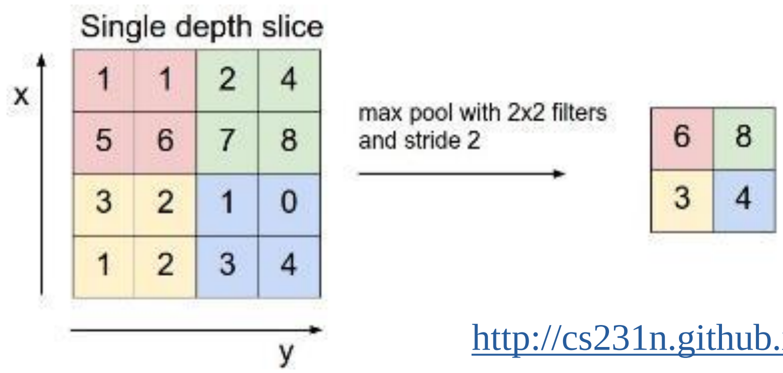
3	0	-3
1	8	0
2	4	0

$$W2 = (5-3+2*1)/2+1=3$$

Pooling Layer

❖ **Pooling layer**

- ➢ It replaces the output of the net at a certain location with a summary statistic of the nearby outputs (max, mean, L2)



http://cs231n.github.io/convolutional-networks/

- ➢ Why to use pooling:
  - Reduce number of parameters in subsequent layers
  - Help to make the representation approx. invariant to small translations of the input. BUT: information on position is lost
  - Make it easier to handle inputs with different sizes

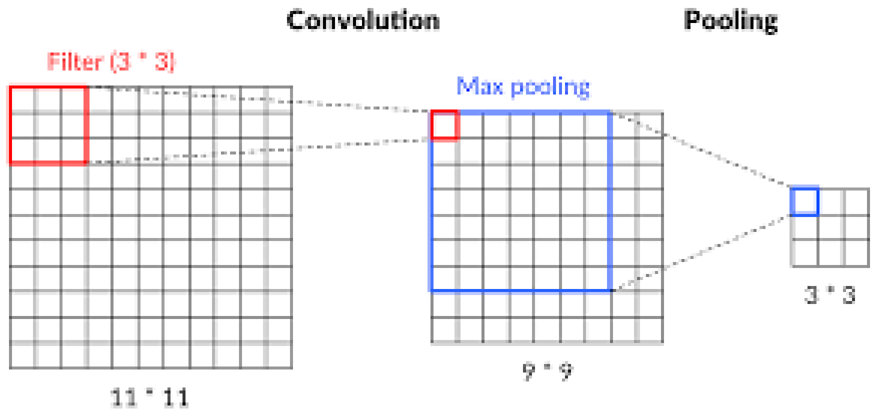
Pooling Layer

- Pooling layer parameters:
  - Size F
  - Stride S

- Usually, F is taken to be equal to 2-3, S to 2.

- How to find the output size?

$$W2=(W1-F)/S+1$$



Convolution and pooling as an infinitely strong priors:

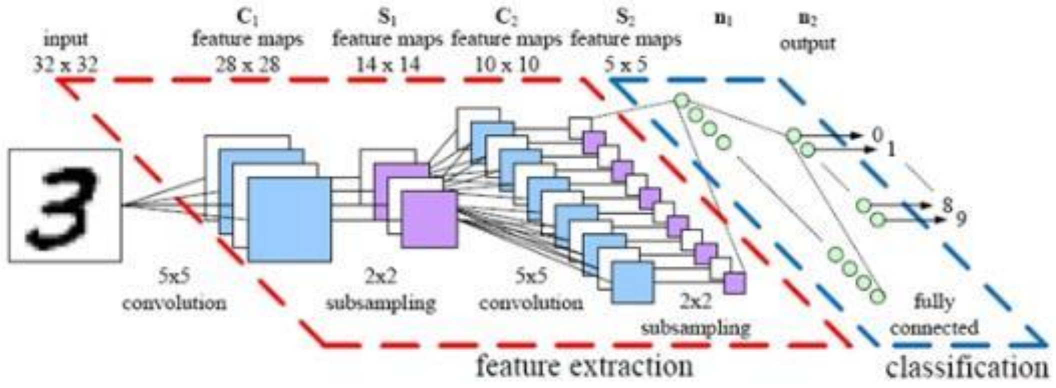
- Convolution layer is the result of imposing restrictions on weights of dense layer:
  - the weights for one hidden unit must be identical to the weights of its neighbor but shifted in space
  - the weights must be zero, except for in the small field
- Similar, pooling requires invariance to small translations

How to Build a Simple CNN

- Main recipe:

INPUT -> [[CONV -> ReLu]*N -> POOL?]*M -> [FC -> ReLu]*K -> FC

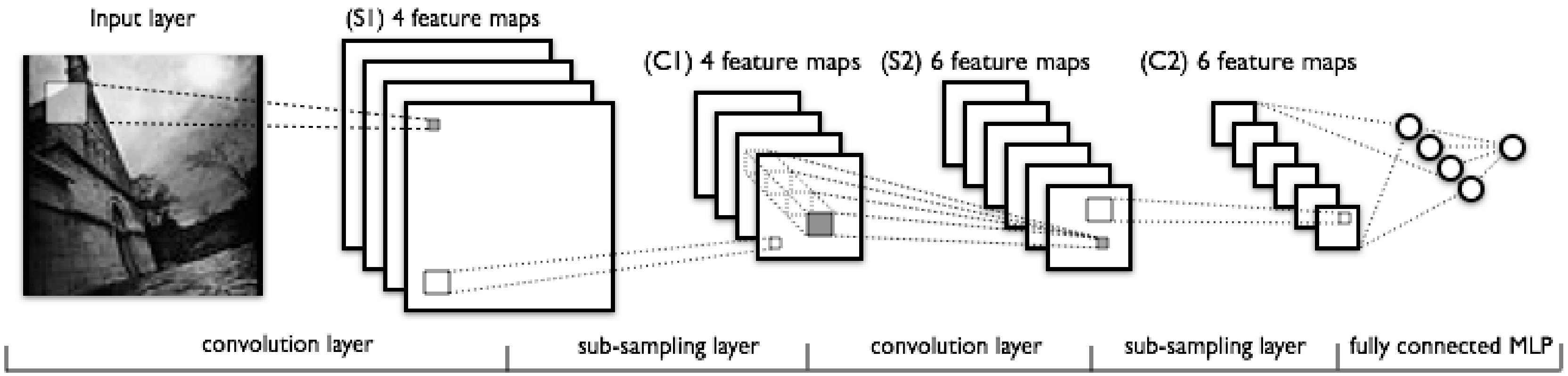
- Usual values of N, M and K: N=0,1,2,3 M>0, K=0,1,2
- ReLu is an analogy of simple cells in V1
- Recommendations:
  - The input layer should be divisible by 2 many times
  - The conv layers should use small filters (e.g. 3x3 or at most 5x5), a stride of S=1, and 'same' padding
  - Use max-pooling with F=2 and S=2



Famous CNNs

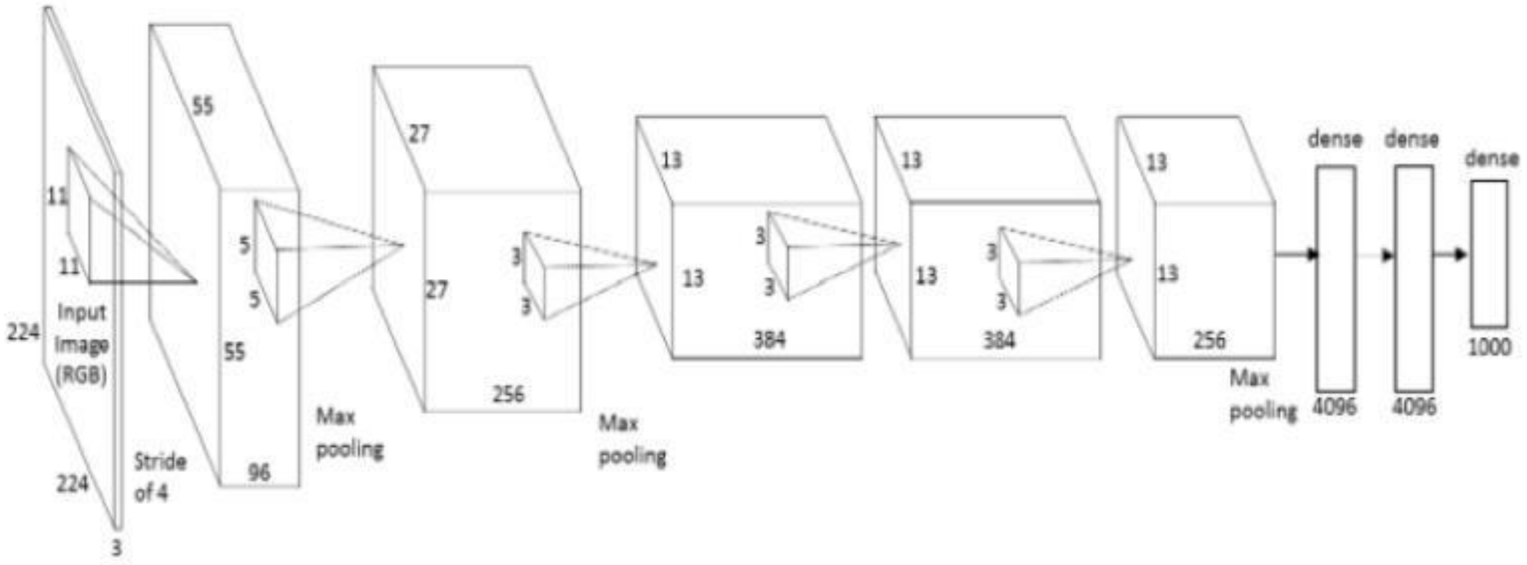
Famous CNNs

- **LeNet**. The first successful applications of Convolutional Networks developed by Yann LeCun in 1990's. Was used to read zip codes and digits. # of trainable parameters: < 1M



Famous CNNs

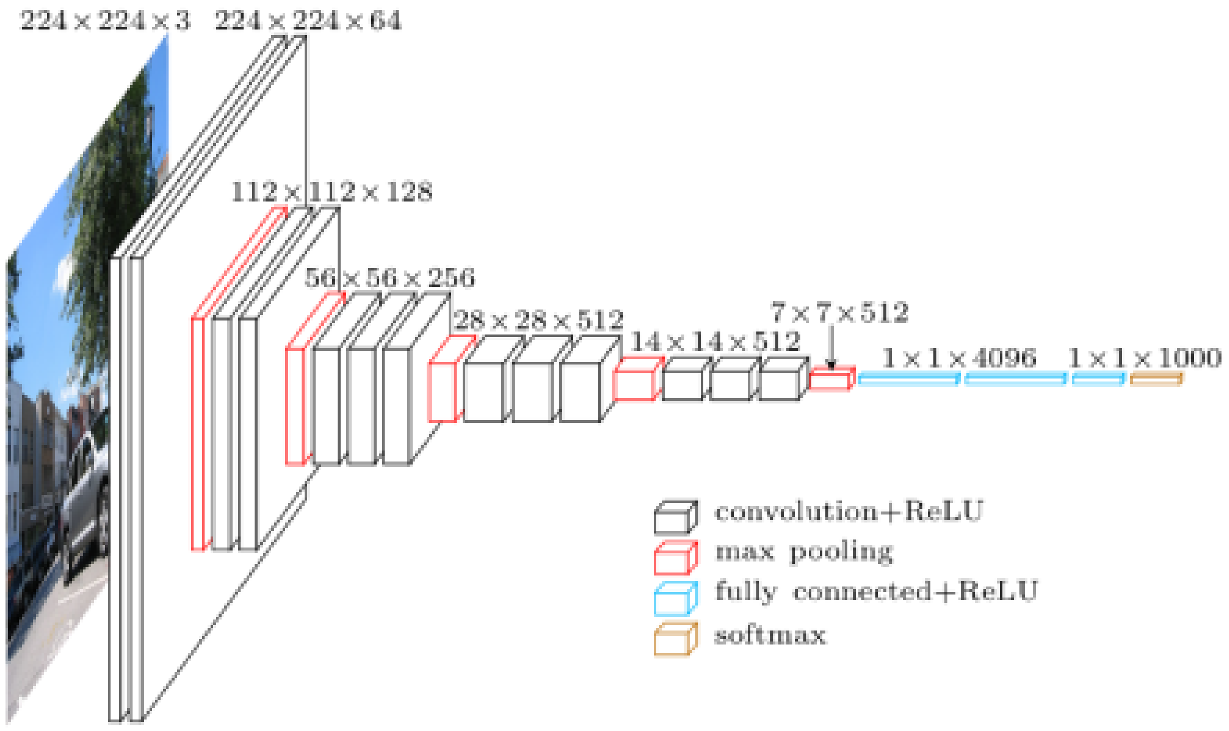
- **AlexNet** developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). Total # of parameters: 62M



Krizhevsky et al (2012)

Famous CNNs

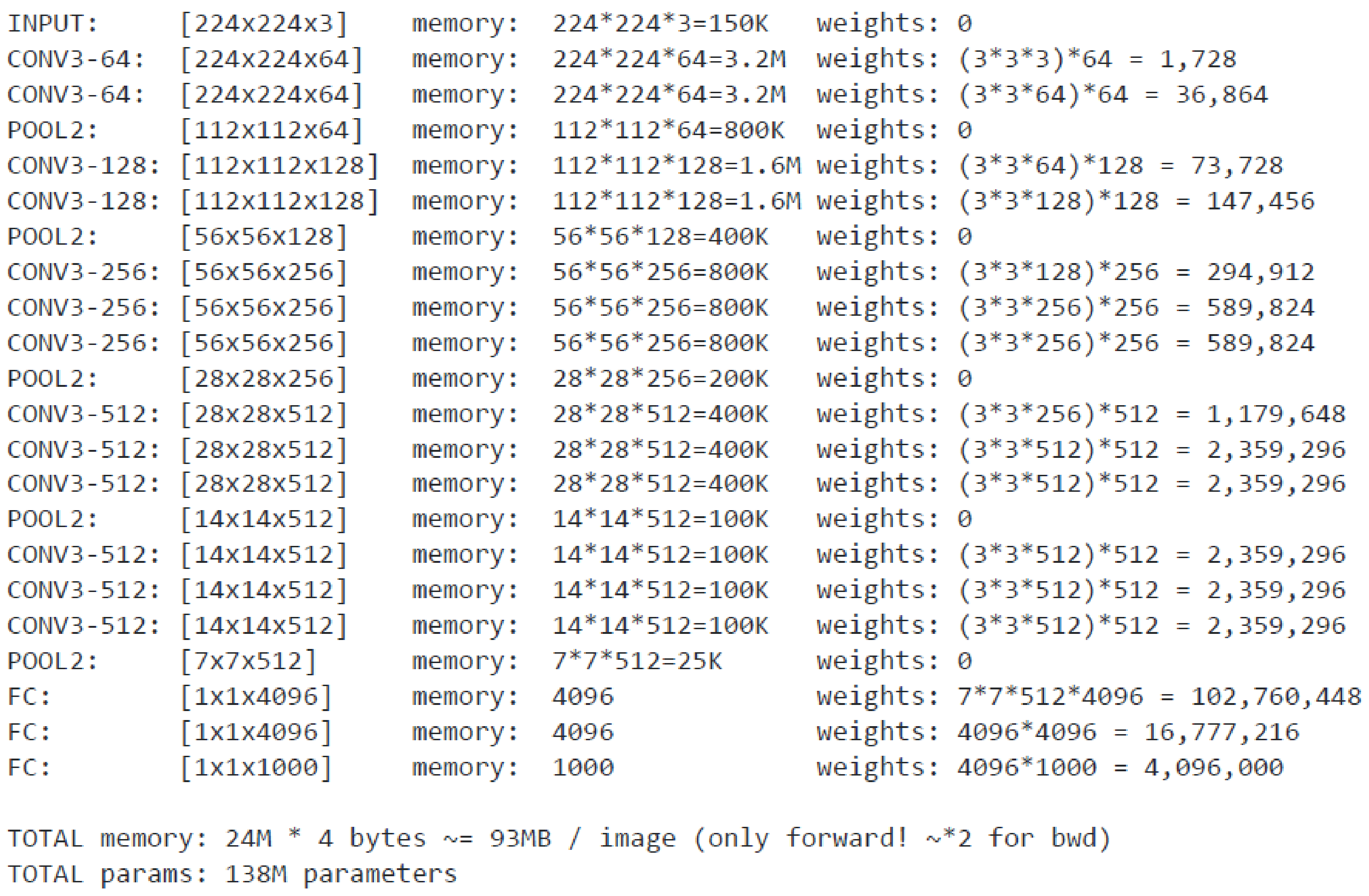
- **VGGNet**. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Total # of parameters: 138M. Main feature: depth-16



Simonyan et al (2015)

Famous CNNs

Parameter count for VGGNet:



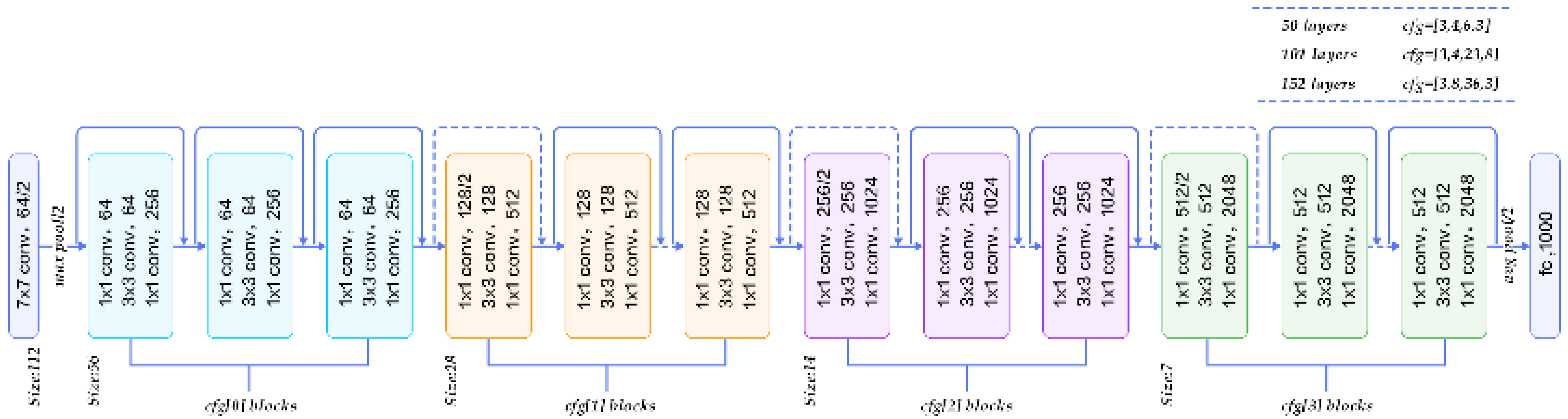
```
INPUT:     [224x224x3]    memory:  224*224*3=150K   weights: 0
CONV3-64:  [224x224x64]   memory:  224*224*64=3.2M  weights: (3*3*3)*64 = 1,728
CONV3-64:  [224x224x64]   memory:  224*224*64=3.2M  weights: (3*3*64)*64 = 36,864
POOL2:     [112x112x64]   memory:  112*112*64=800K  weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M weights: (3*3*128)*128 = 147,456
POOL2:     [56x56x128]    memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]    memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]    memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]    memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2:     [28x28x256]    memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]    memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]    memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]    memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2:     [14x14x512]    memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]    memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]    memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]    memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2:     [7x7x512]      memory:  7*7*512=25K      weights: 0
FC:        [1x1x4096]     memory:  4096             weights: 7*7*512*4096 = 102,760,448
FC:        [1x1x4096]     memory:  4096             weights: 4096*4096 = 16,777,216
FC:        [1x1x1000]     memory:  1000             weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

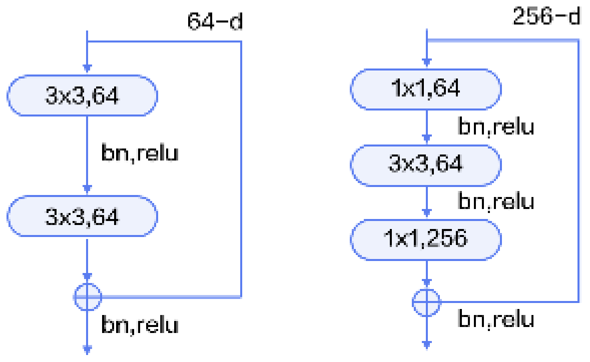
http://cs231n.github.io/convolutional-networks/#pool

Famous CNNs

❖ **ResNet** developed by Kaiming He et al. was the winner of ILSVRC 2015. It features special skip connections and a heavy use of batch normalization. State-of-the-art CNN.



He et al (2015)



Residual blocks

References

- K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, Y. Bengio. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. ArXiv 1502.03044
- A. Krizhevsky, I. Sutskever, G. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. In: NIPS Proceedings (2012)
- K. Simonyan, A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. ArXiv 1409.1556 (2015)
- K. He, X. Zhang, S. Ren, J. Sun. *Deep Residual Learning for Image Recognition*. ArXiv 1512.03385 (2015)

General Information

- Next Lecture:
  - Title: ‘Convolutional Neural Networks: Part 2’
  - Main topics:
    - CNN pretraining
    - Transfer learning
    - Visualization of CNN filters
    - Deconvolutional layers
    - CNNs for analyzing time series
    - Discussion on assignment #2
  - Schedule: next Wednesday 18:30
- Assignments:
  - Assignment #3 is now available on https://github.com/5vision/miptmlcourse
  - Assignment #3 due is 24 May 2017
  - Submissions for assignment #2 should be made through Kaggle:

    https://inclass.kaggle.com/c/classroom-diabetic-retinopathy-detection-competition