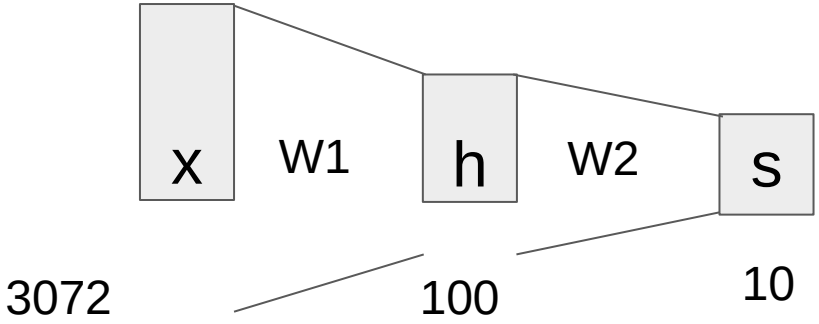
Лекция 5
Сверточные нейронные сети: базовые операции

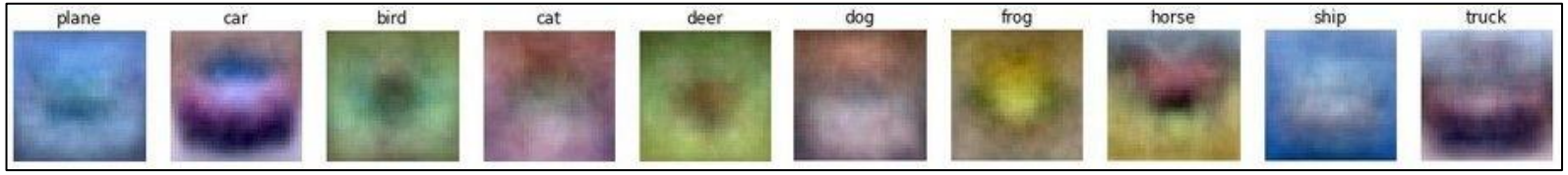
Нейронные сети

(**Было**) линейная оценка: $f = Wx$

(**Стало**) 2-слоя НС $f = W_2 \max(0, W_1 x)$

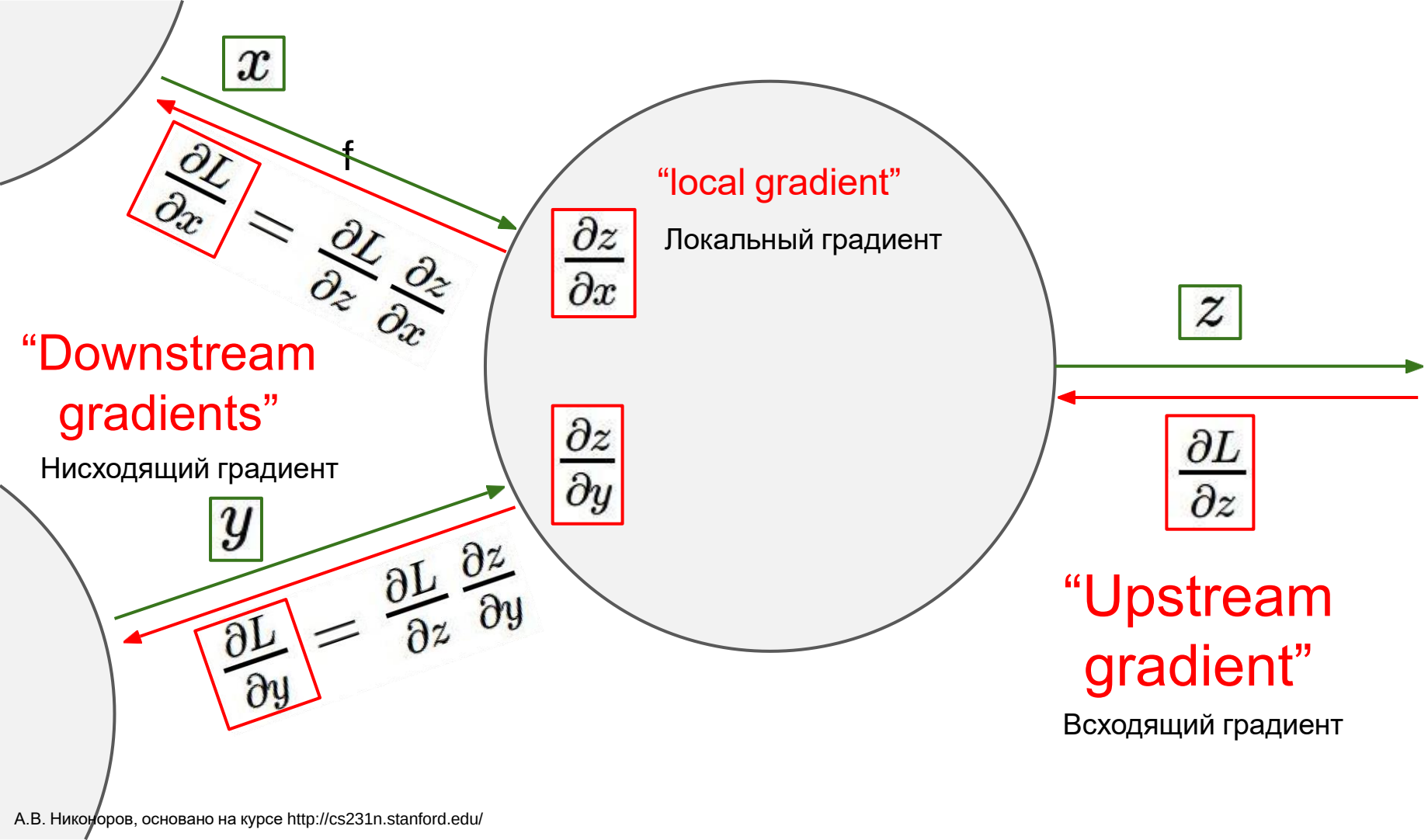


$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$



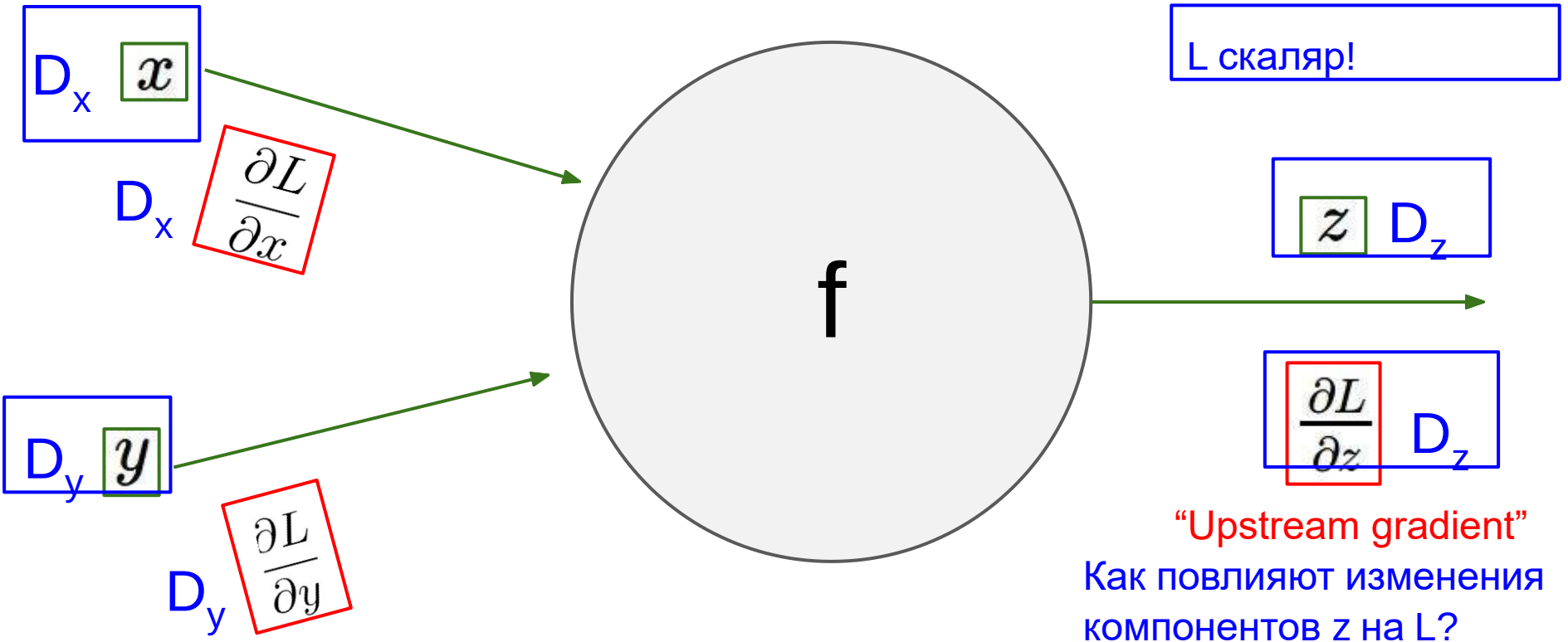
100 шаблонов вместо 10

Общие шаблоны для классов



А.В. Никоноров, основано на курсе http://cs231n.stanford.edu/

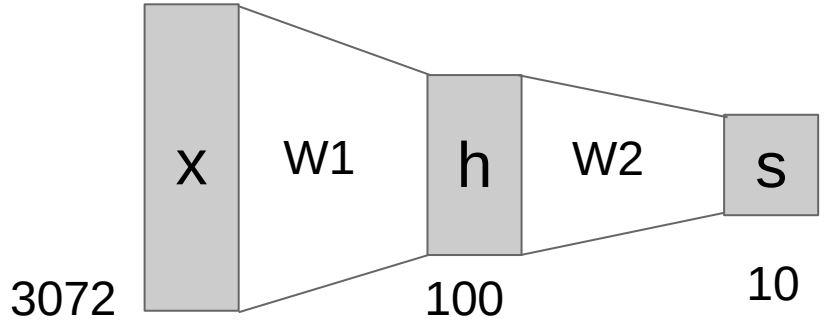
Градиенты переменных имеют ту же размерность что и сами переменные

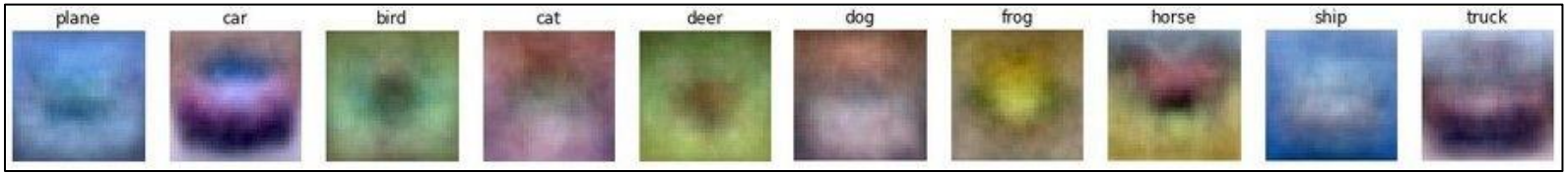


Соберем все вместе:

Линейный классификатор: $f = Wx$

Двухслойная нейросеть: $f = W_2 \max(0, W_1 x)$





Далее: сверточные НС

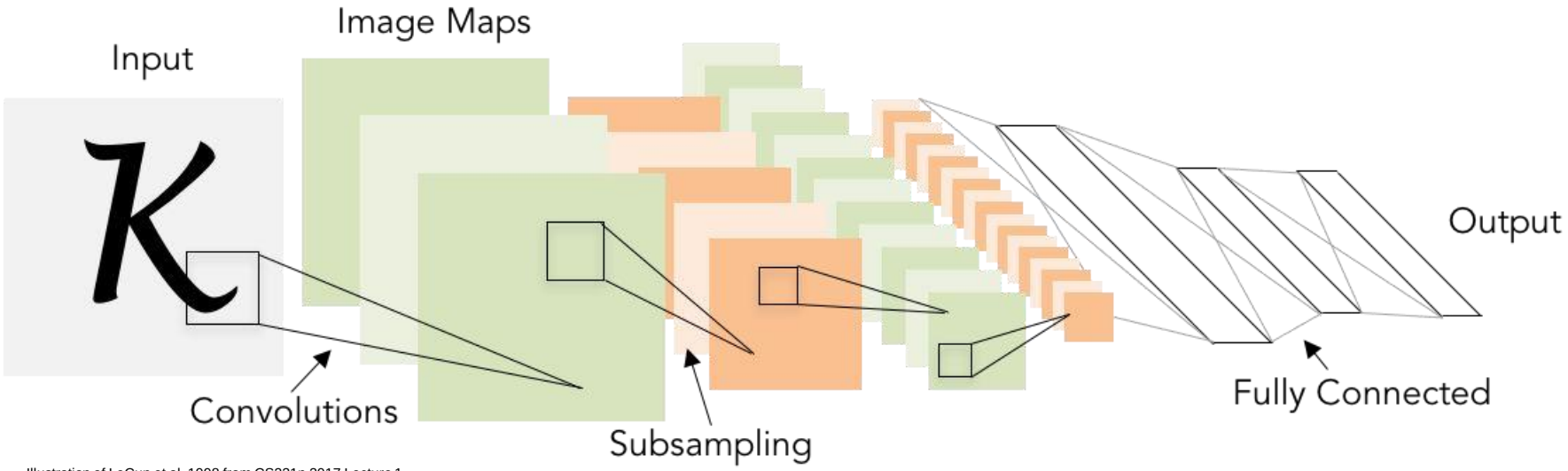


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

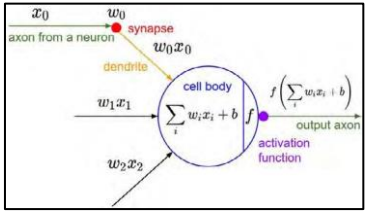
Немного истории...

Mark I Perceptron первая хардварная реализация.

Подключена к камере 20×20 дающей картинку в 400 пикселов.

распознает
буквы алфавита

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

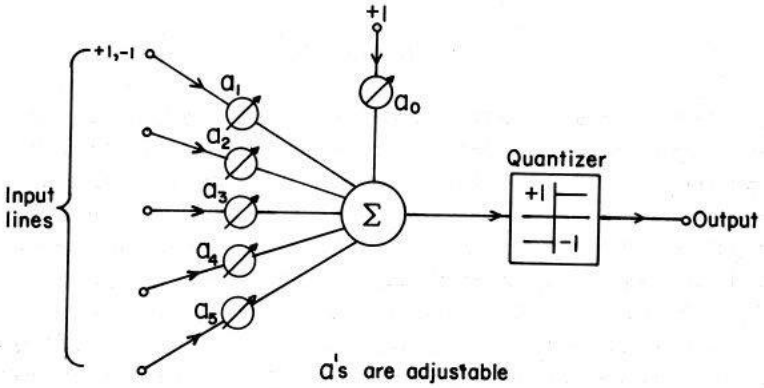


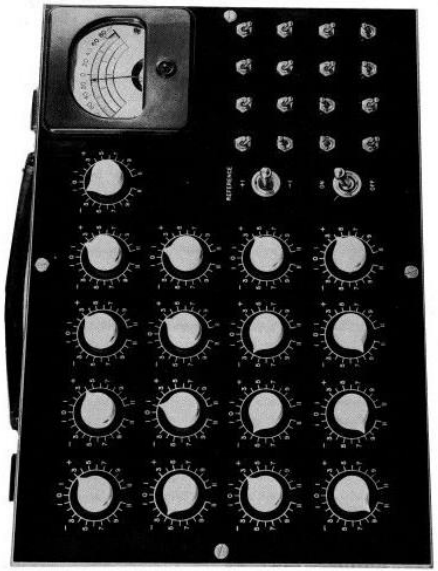
Обновление весов:

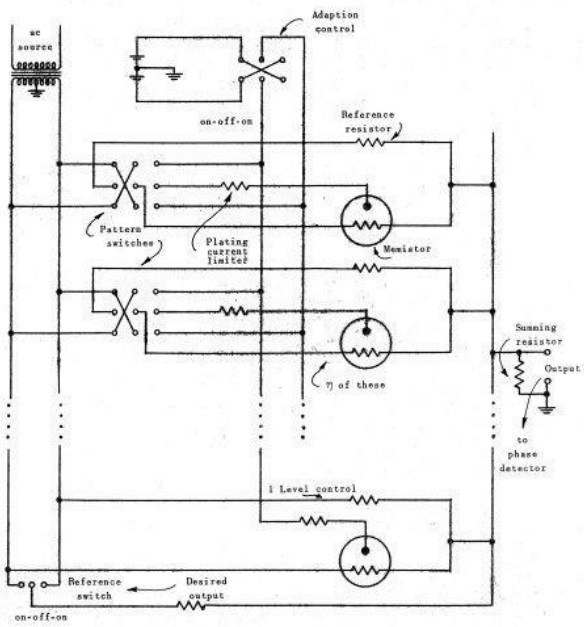
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$

Frank Rosenblatt, ~1957: Perceptron

A bit of history...







Widrow and Hoff, ~1960: Adaline/Madaline

These figures are reproduced from Widrow 1960, Stanford Electronics Laboratories Technical Report with permission from Stanford University Special Collections.

A bit of history...

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$

recognizable math

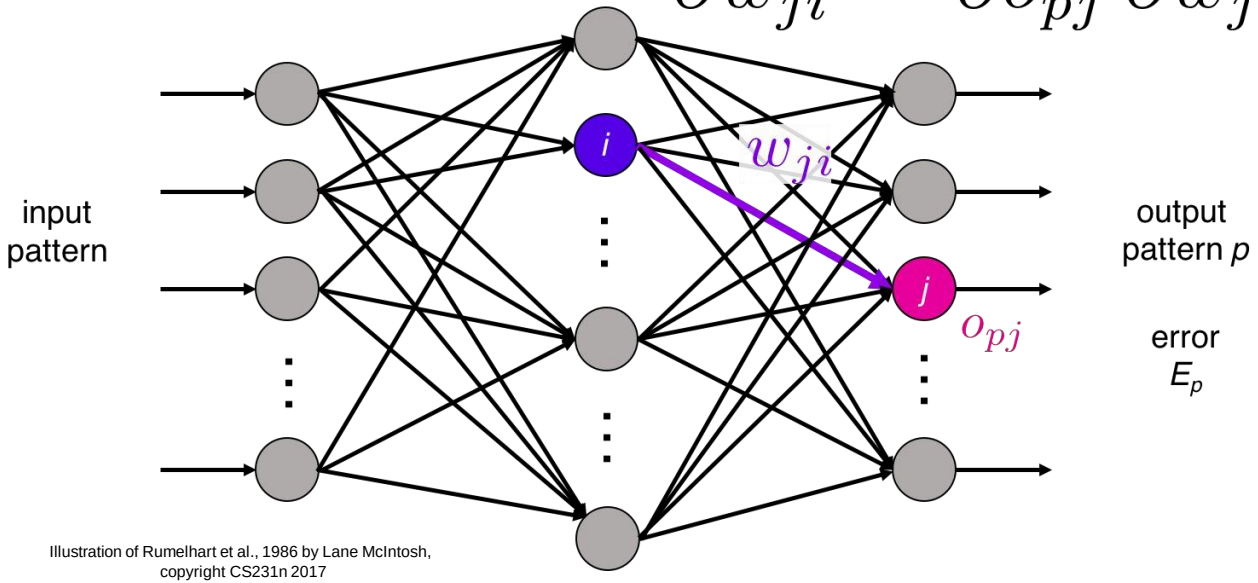


Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017

Rumelhart et al., 1986: First time back-propagation became popular

A bit of history...

[Hinton and Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, 2006]

Машина Больцмана

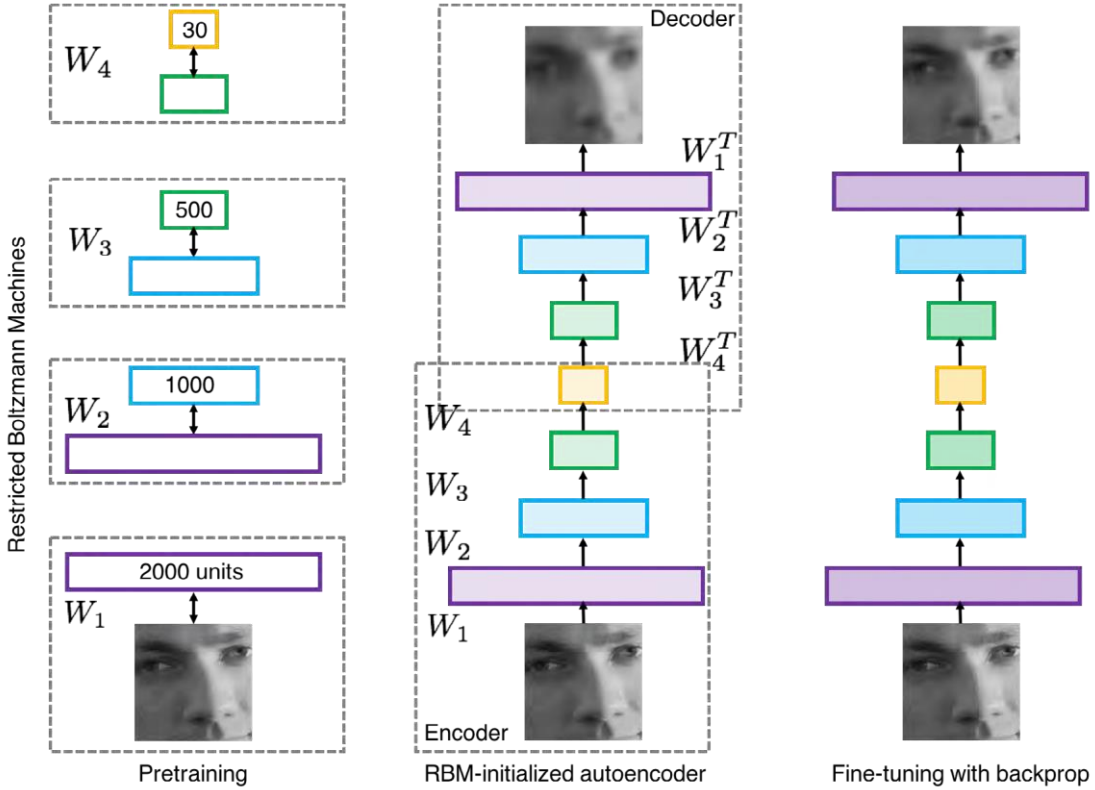


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

Первые крутые результаты

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

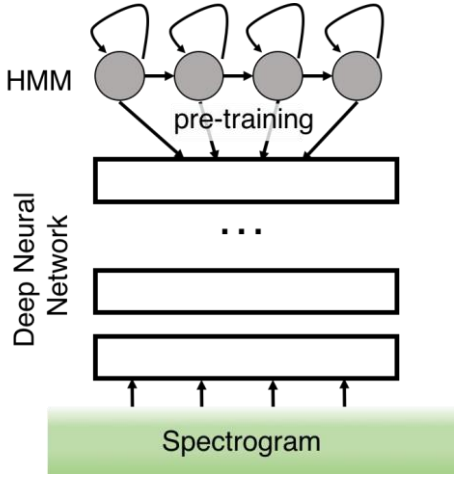
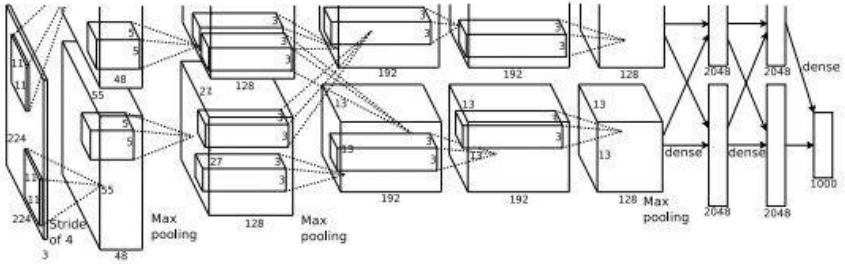
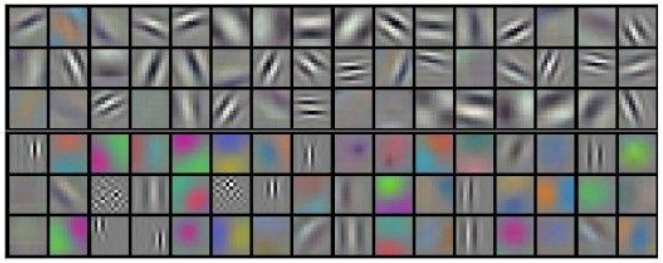


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

A bit of history:

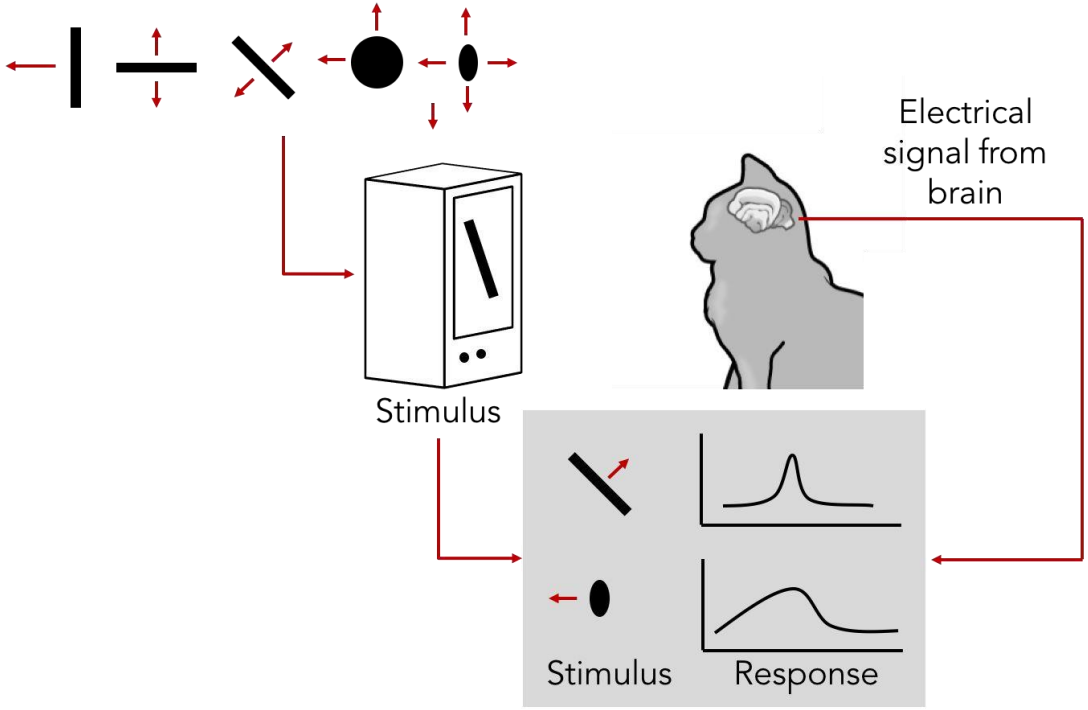
Hubel & Wiesel, 1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

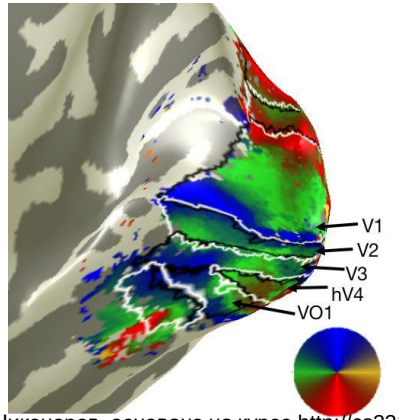
RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

1968...

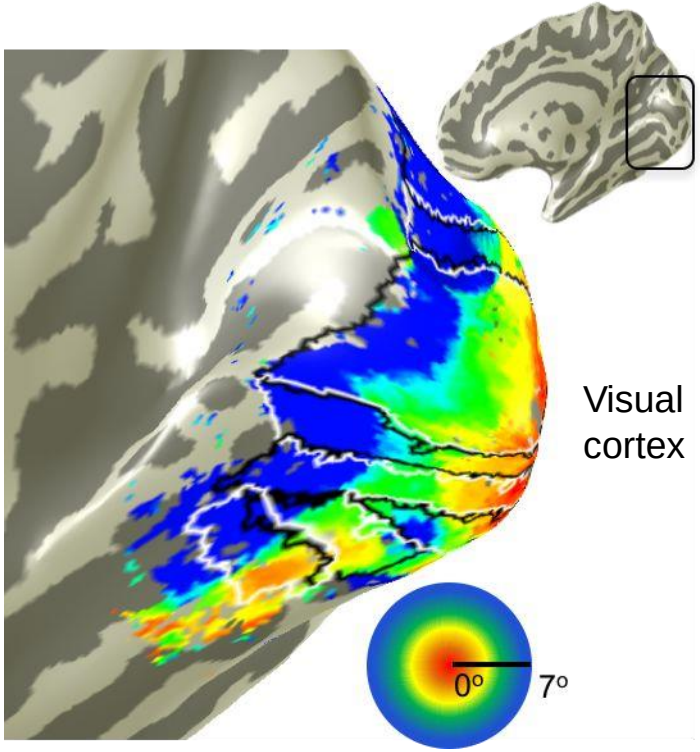


A bit of history

Сохранение топологии в визуальном кортексе:
соседние участки кортекса соответствуют соседним участкам поля зрения



Human brain



Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

Визуальный кортекс организован по слоям

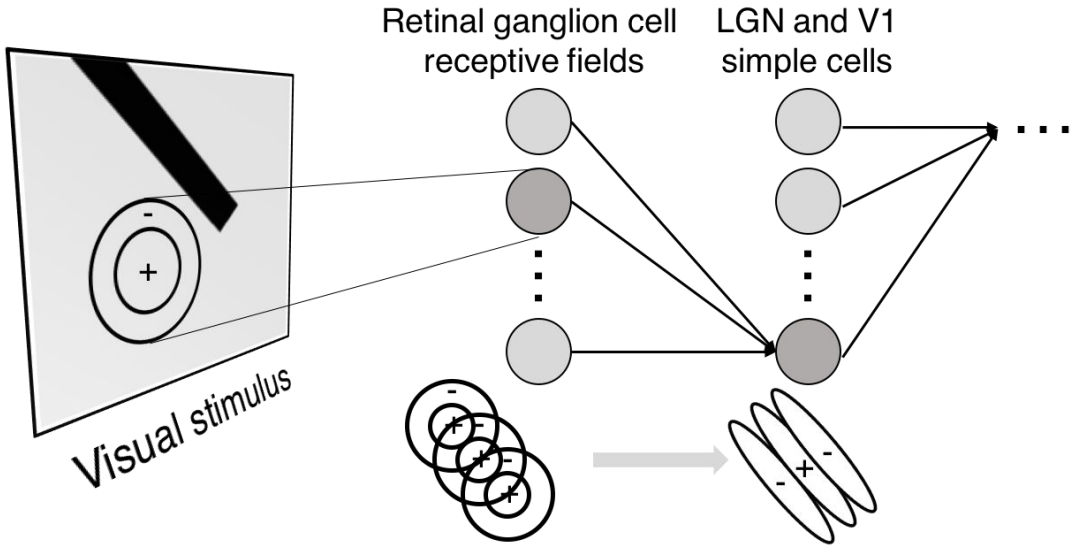
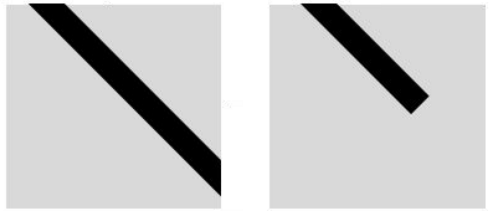


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point



No response

Response (end point)

А.В. Никоноров, основано на курсе http://cs231n.stanford.edu/

A bit of history

Rainer Goebel

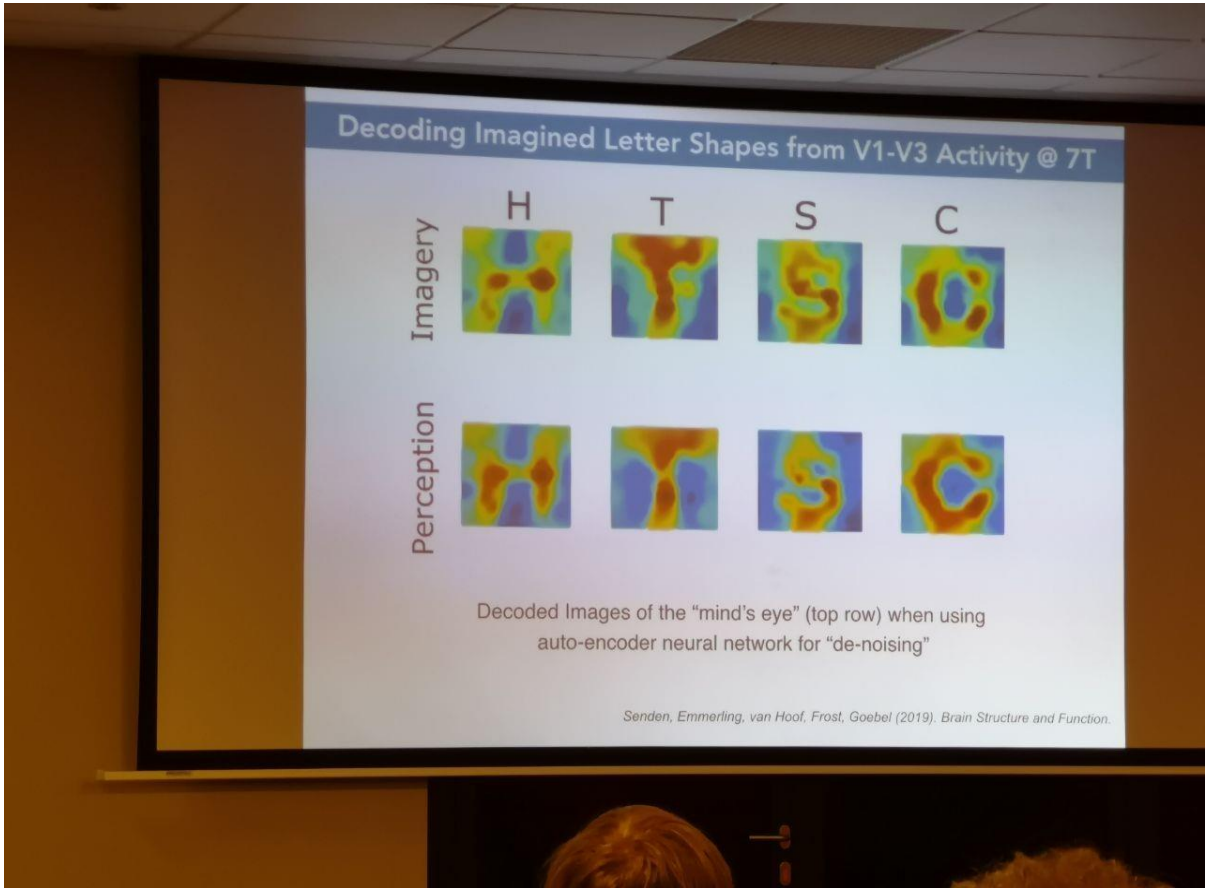
Professor of Cognitive Neuroscience, Maastricht University
Verified email at maastrichtuniversity.nl - Homepage

Ultra-High Field fMRI Human Brain Research Cognitive Neuroscience
Neural Network Modelling Brain-Computer Interfaces

ARTICLES CITED BY CO-AUTHORS

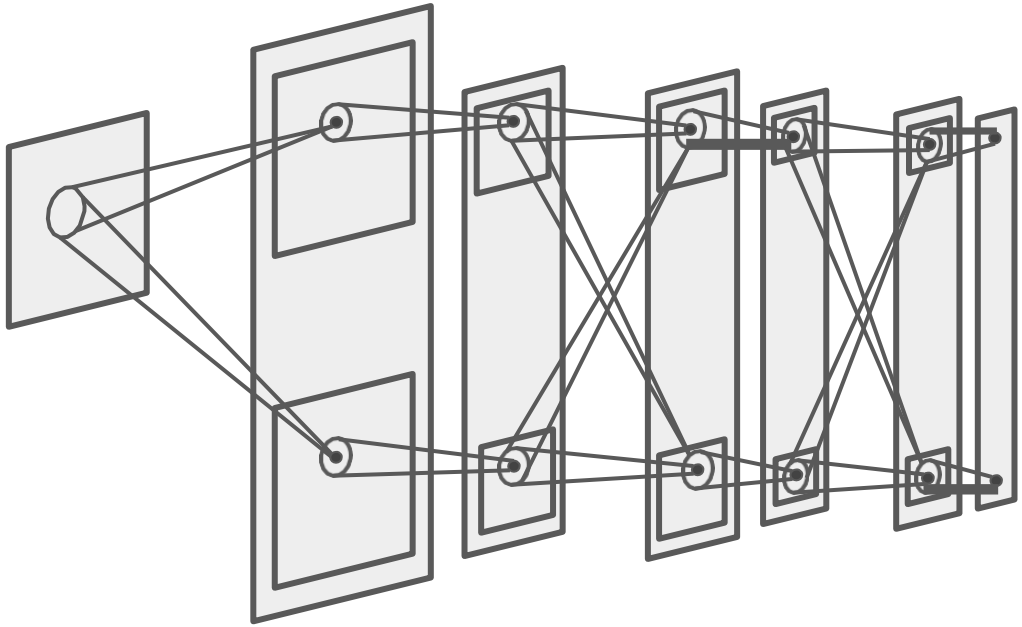
	All	Since 2015
Citations	34458	15380
h-index	101	67
i10-index	274	241

Создатель Turbo BrainVoyajer
Наша альтернатива для TBV:
http://opennft.org/



A bit of history:

Neocognitron
[Fukushima 1980]

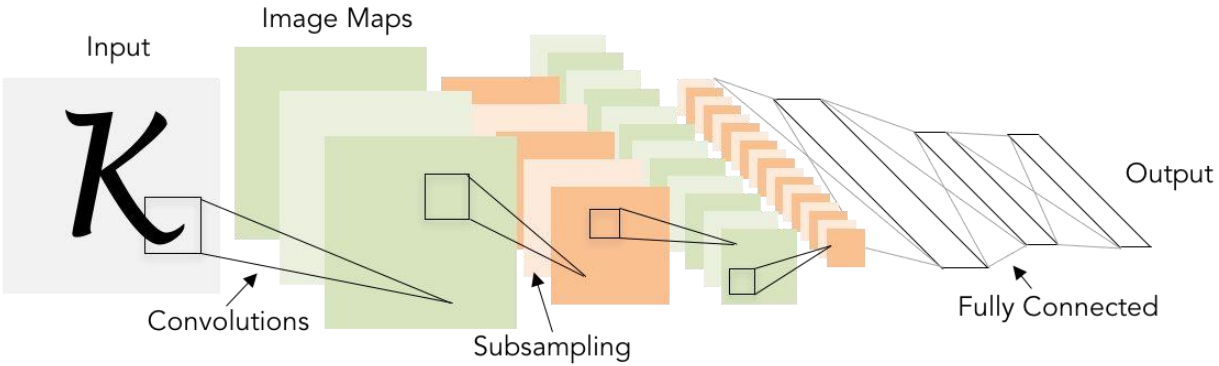


“sandwich” architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



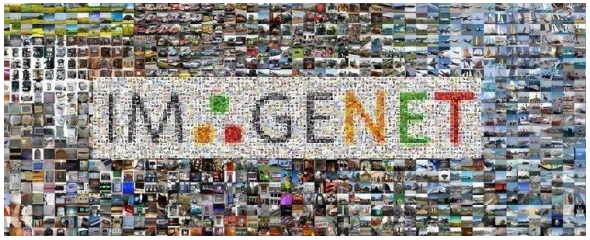
LeNet-5

Первая сверточная сеть!

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]



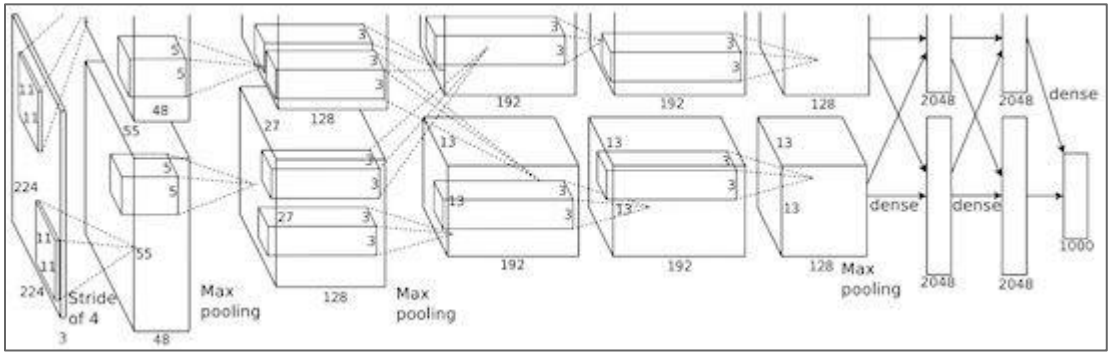


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

Сверточная сеть + GPU!

A bit of history

1989 G Cybenko
Теорема об универсальной аппроксимации

Approximation by superpositions of a sigmoidal function - Springer Link
https://link.springer.com/article/10.1007/BF02551274 - Перевести эту страницу
автор: G Cybenko - 1989 - Цитируется: 10688 - Похожие статьи

1998 Yann LeCun
сверточные сети

ieeexplore.ieee.org › document - Перевести эту страницу
Gradient-based learning applied to document recognition ...
Gradient-based learning applied to document recognition. ... A new **learning** paradigm, called graph transformer networks (GTN), allows such multimodule systems to be trained globally using **gradient-based** methods so as to minimize an overall performance measure. Two systems for online handwriting **recognition** are described.
автор: Y Lecun - 1998 - Цитируется: 28105 - Похожие статьи

2007 – Выход NVIDIA CUDA,

2009 – Google отказывается от нейронных сетей

2012 – AlexNet

[PDF] ImageNet Classification with Deep Convolutional Neural Networks
https://papers.nips.cc/.../4824-imagenet-classification-with-de... ▾ Перевести эту страницу
автор: A Krizhevsky - 2012 - Цитируется: 34232 - Похожие статьи

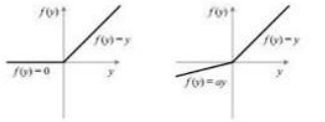


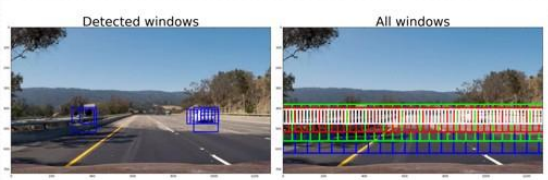
Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

Delving Deep into Rectifiers: Surpassing Human-Level Performance .
https://arxiv.org › cs ▾ Перевести эту страницу
автор: K He - 2015 - Цитируется: 3856 - Похожие статьи

IEEE CVPR Cite Score: 3.23 (2012), 6.19 (2015), 18.18 (2018)

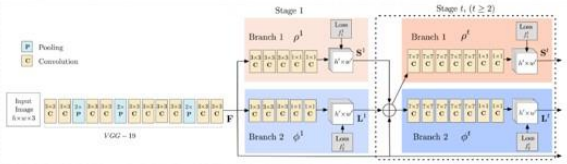
К нашим дням: СНС везде!

Fast Detection

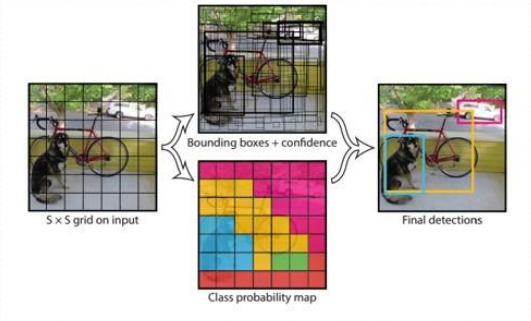


Sliding window detection -NO

Tracking



Real time pose tracking, CVPR 17



YOLO - You Only Look Once – Yes!

Openpose

Вычислительная фотография –

- Стекинг фото
- Сверхразрешение
- Съемка в темноте

Light.co

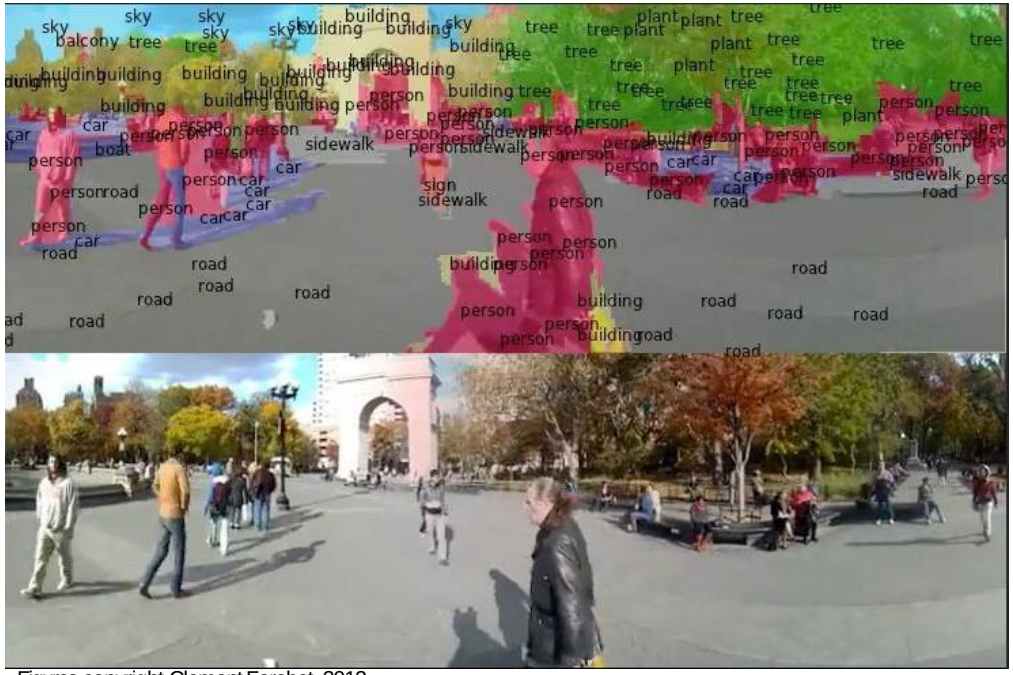
К нашим дням: CHC везде!

Detection

Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

К нашим дням: СНС везде!

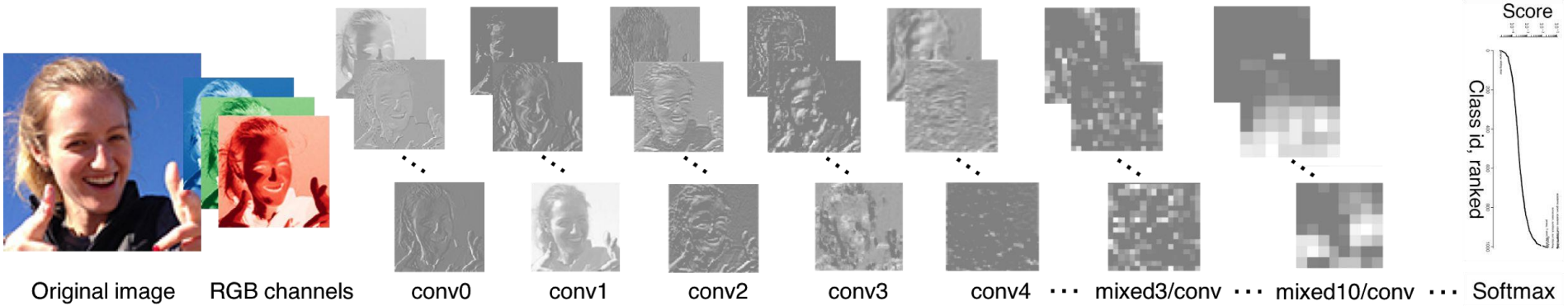
Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

NVIDIA Tesla line
(для обучения)

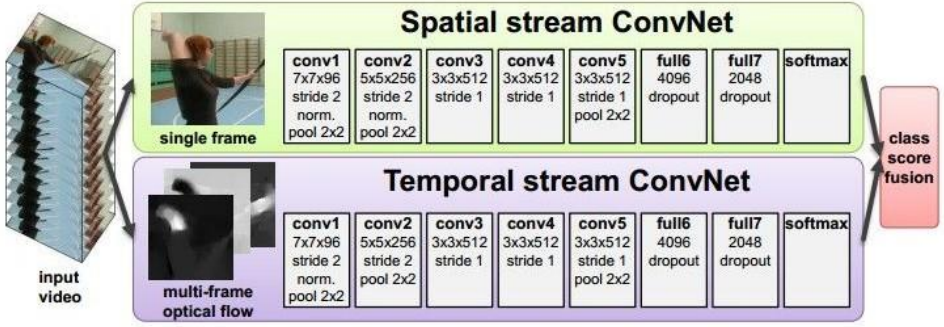
Для инференса повсеместно используют встраиваемые решения с ARM

К нашим дням: CHC везде!



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

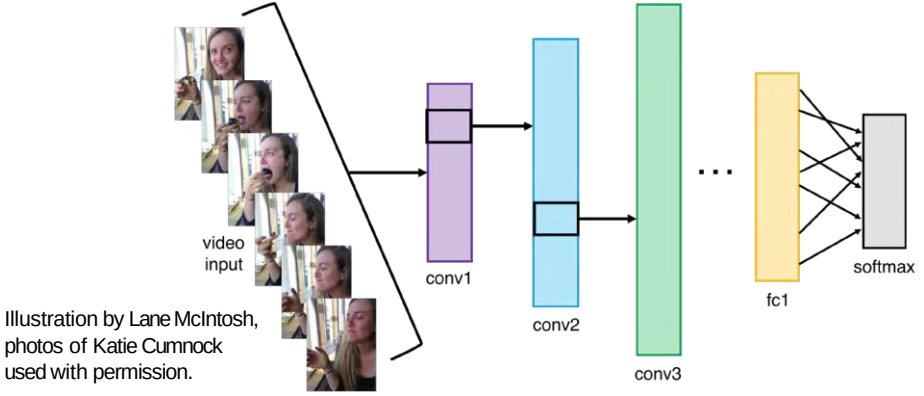


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

К нашим дням: CHC везде!

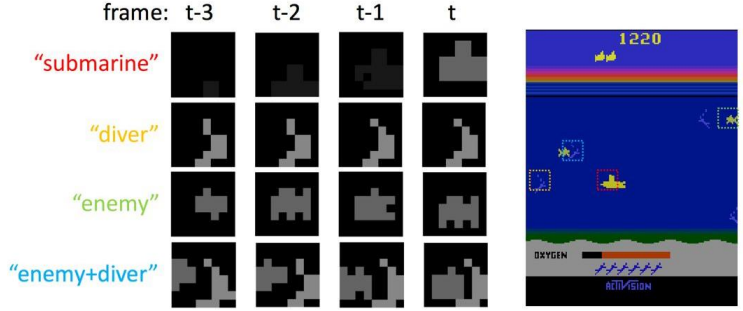
Classification

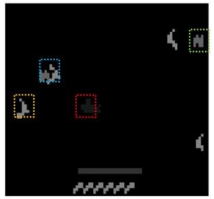
Retrieval

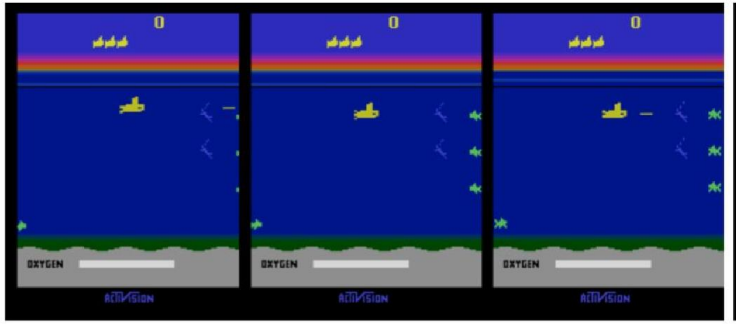
К нашим дням: CHC везде!

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]



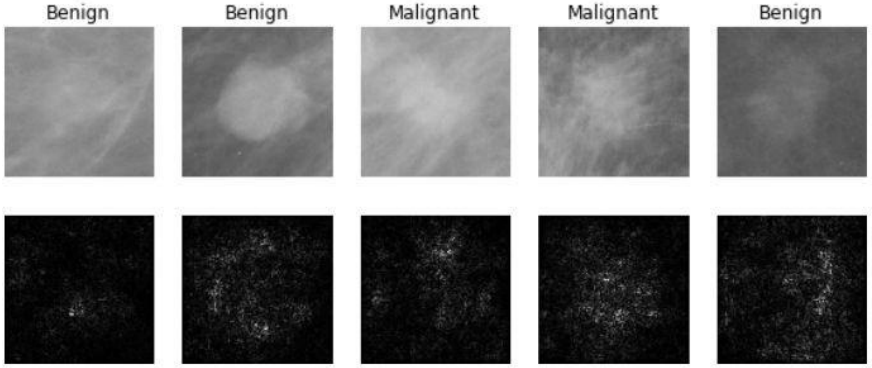




[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

К нашим дням: СНС везде!



[Levy et al. 2016]

[Dieleman et al. 2014]

[Sermanet et al. 2011]
[Ciresan et al.]

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

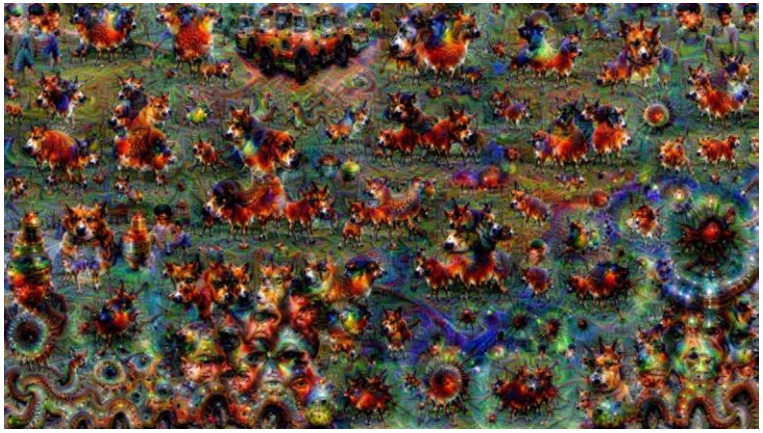
A cat sitting on a suitcase on the floor

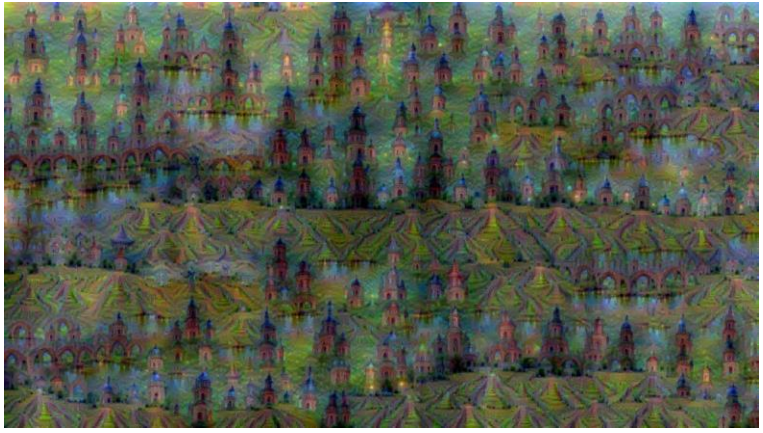
Somewhat related

A woman is holding a cat in her hand

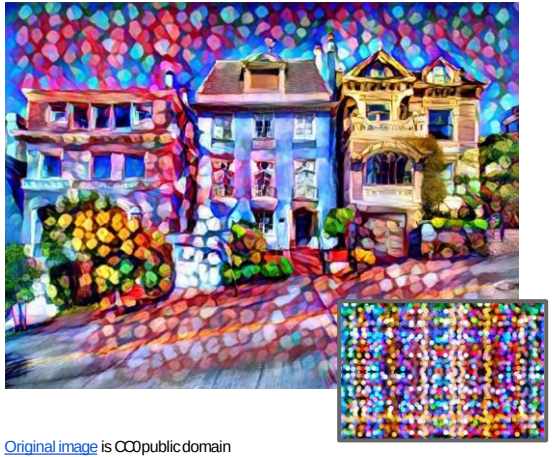
A woman standing on a beach holding a surfboard

Captions generated by Justin Johnson using Neuraltalk2

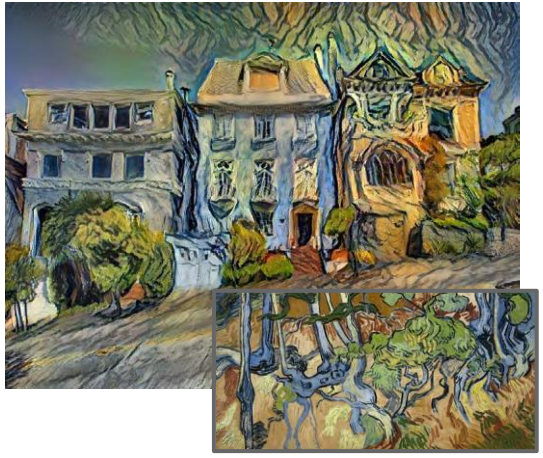




Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.



Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission

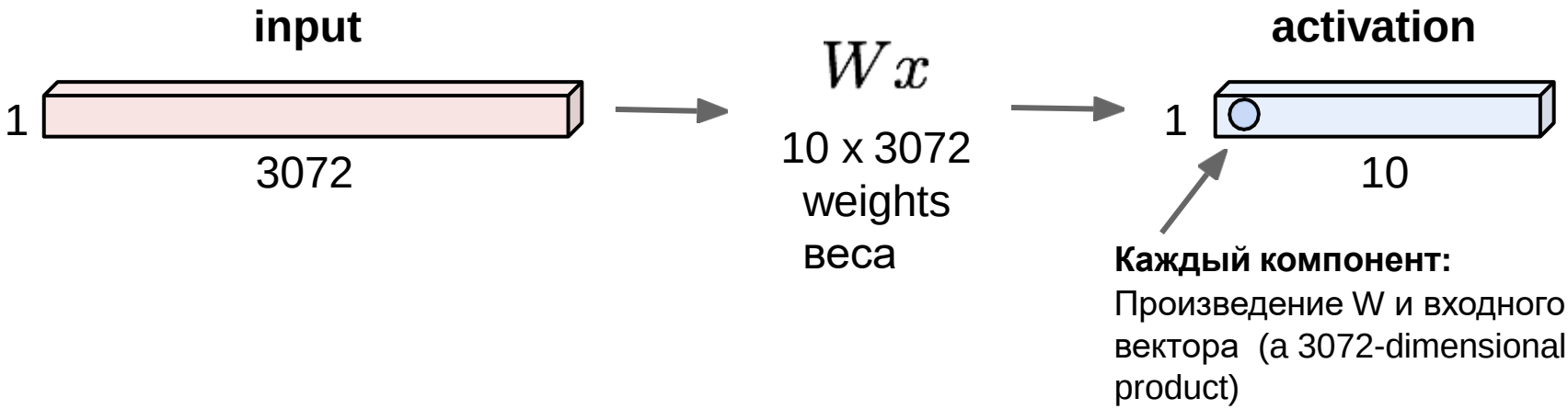


Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Convolutional Neural Networks
Сверточные нейронные сети
CNC

Fully Connected Layer Полносвязный слой

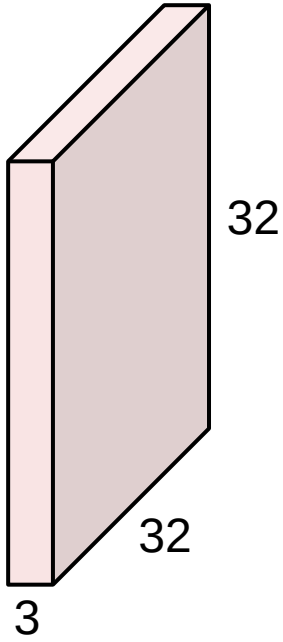
32x32x3 картинку -> к вектору 3072 x 1



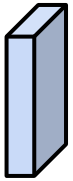
Convolution Layer - Сверточный слой

Ядро фильтра имеет ту же глубину, что и входное изображение

32x32x3 image

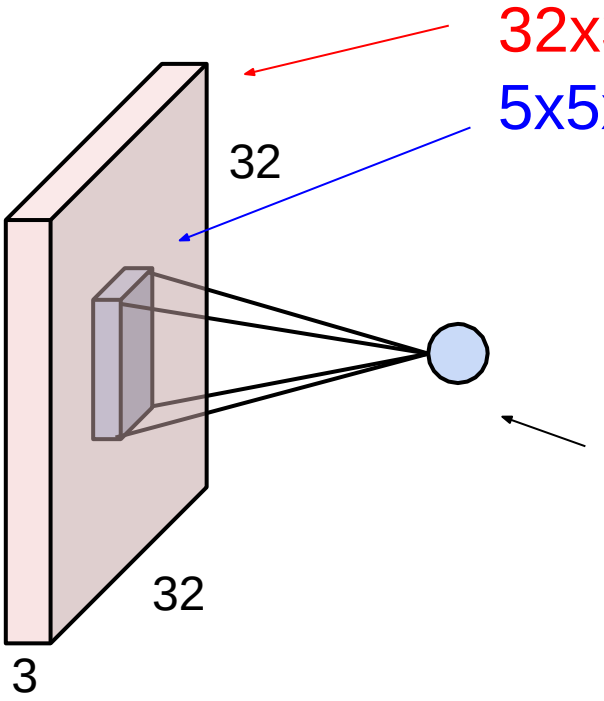


5x5x3 filter



Свертка с фильтром
“скалярные произведения со скользящим окном фильтра”

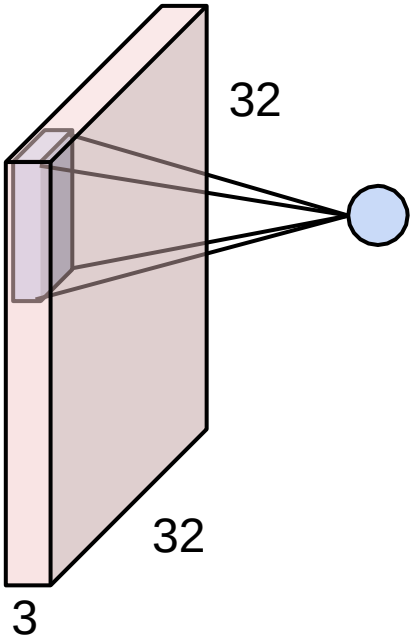
Сверточный слой



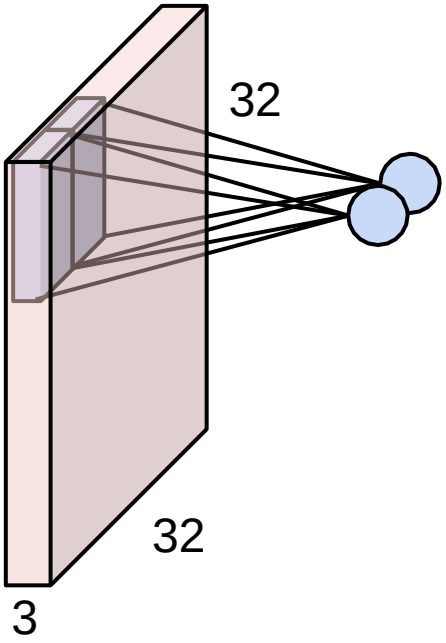
Один компонент:
Результат скалярного произведения 5x5x3 фрагмента изображения (5*5*3 = 75-мерное скалярное произведение + смещение)

$$w^T x + b$$

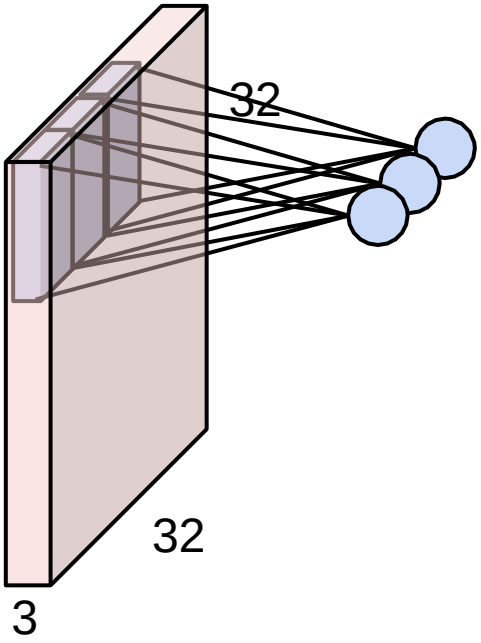
Сверточный слой



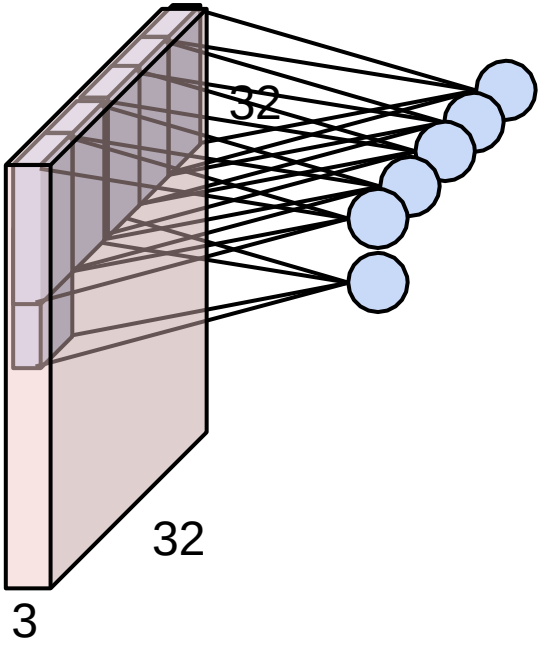
Сверточный слой



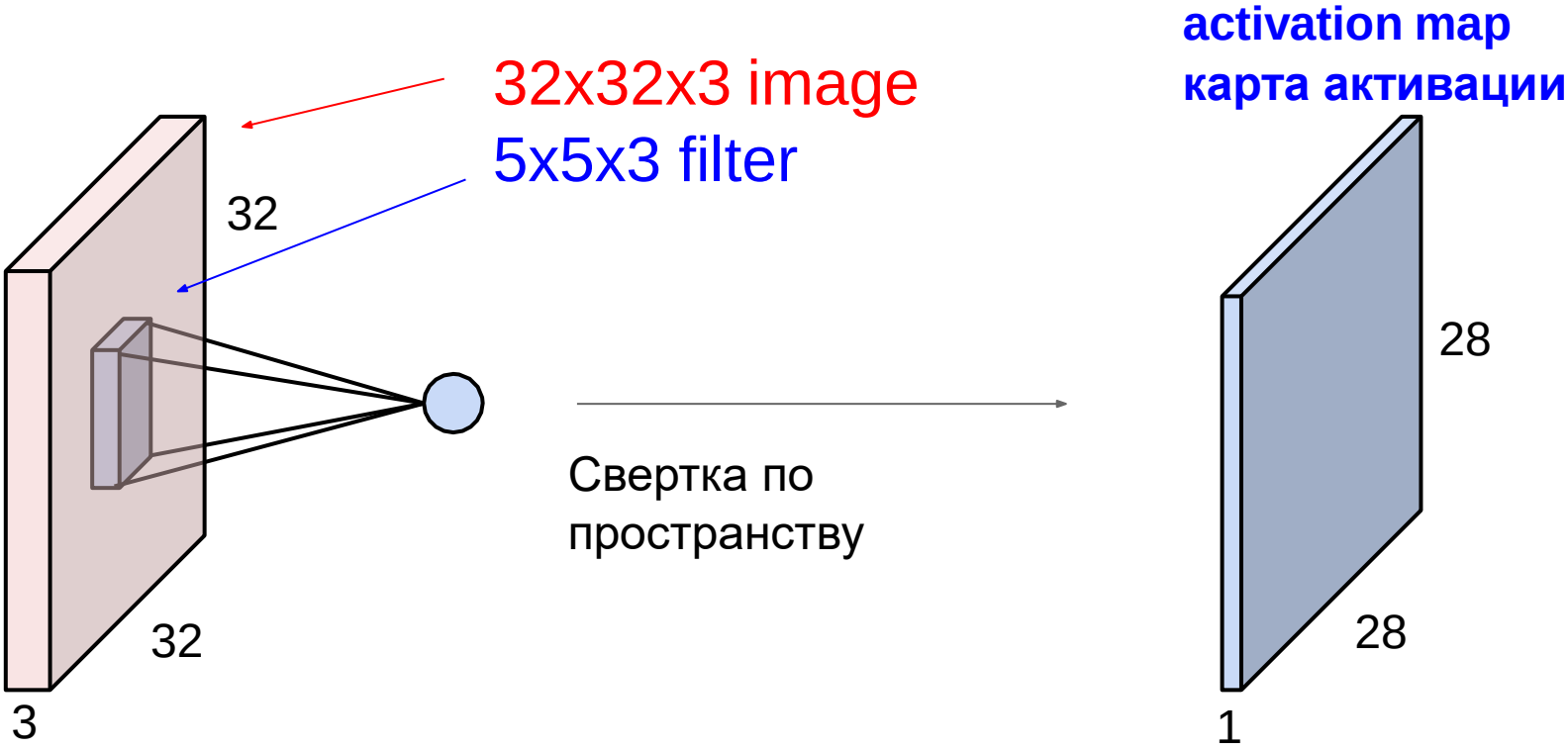
Сверточный слой



Сверточный слой

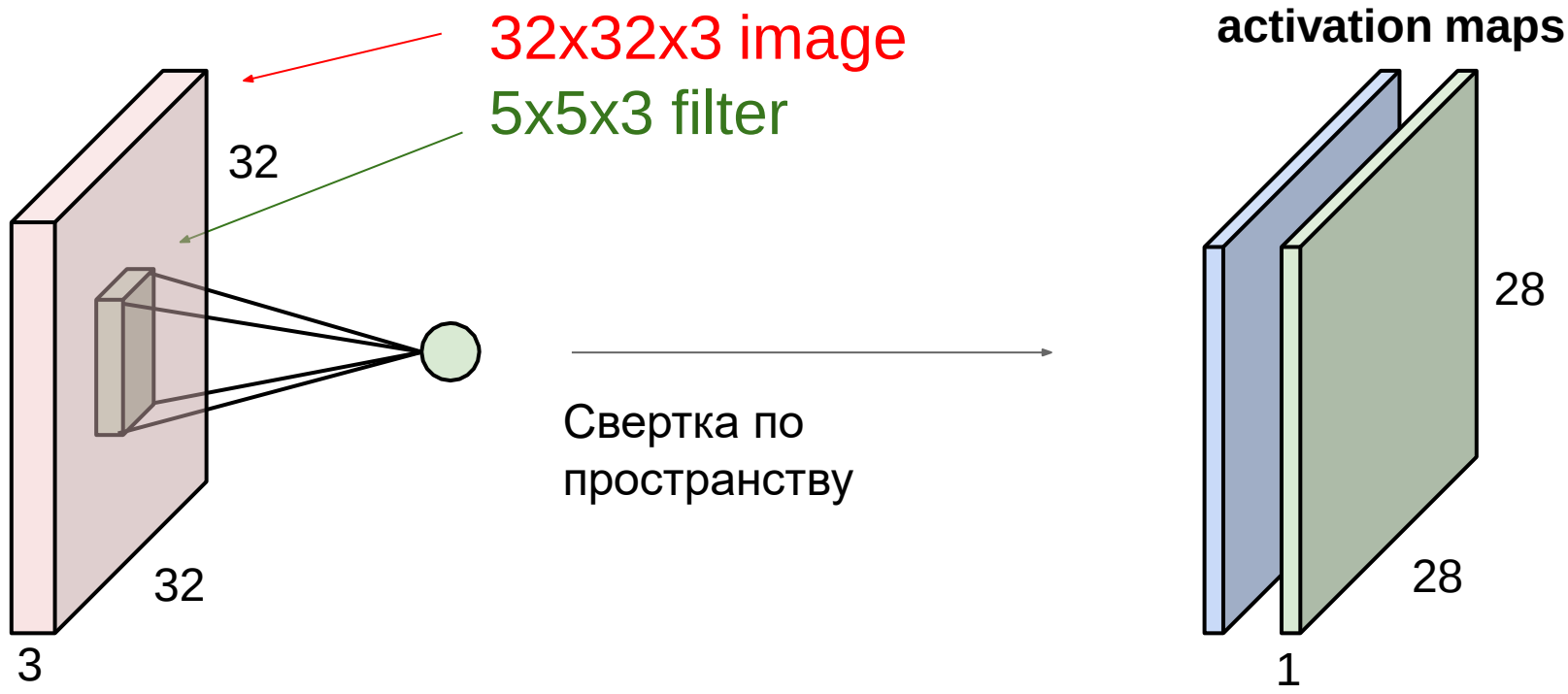


Сверточный слой

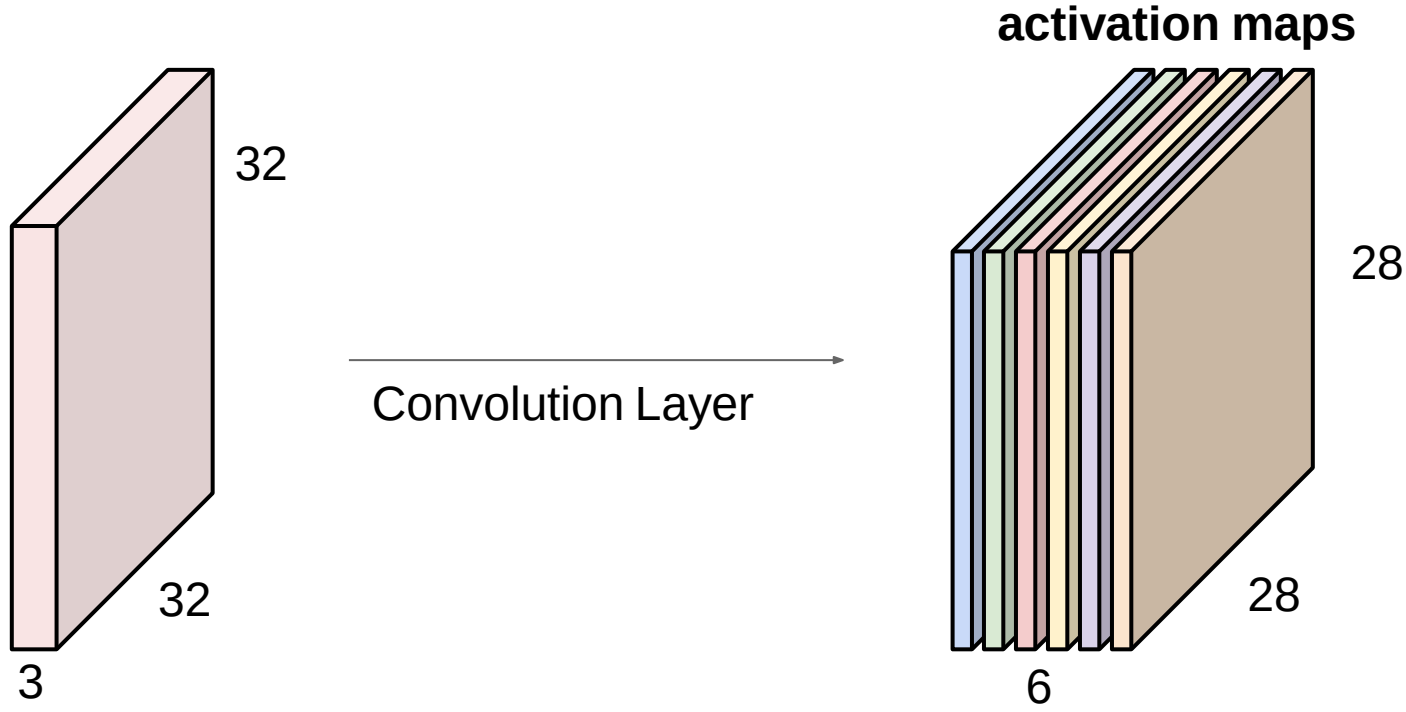


Сверточный слой

Добавим второй зеленый фильтр

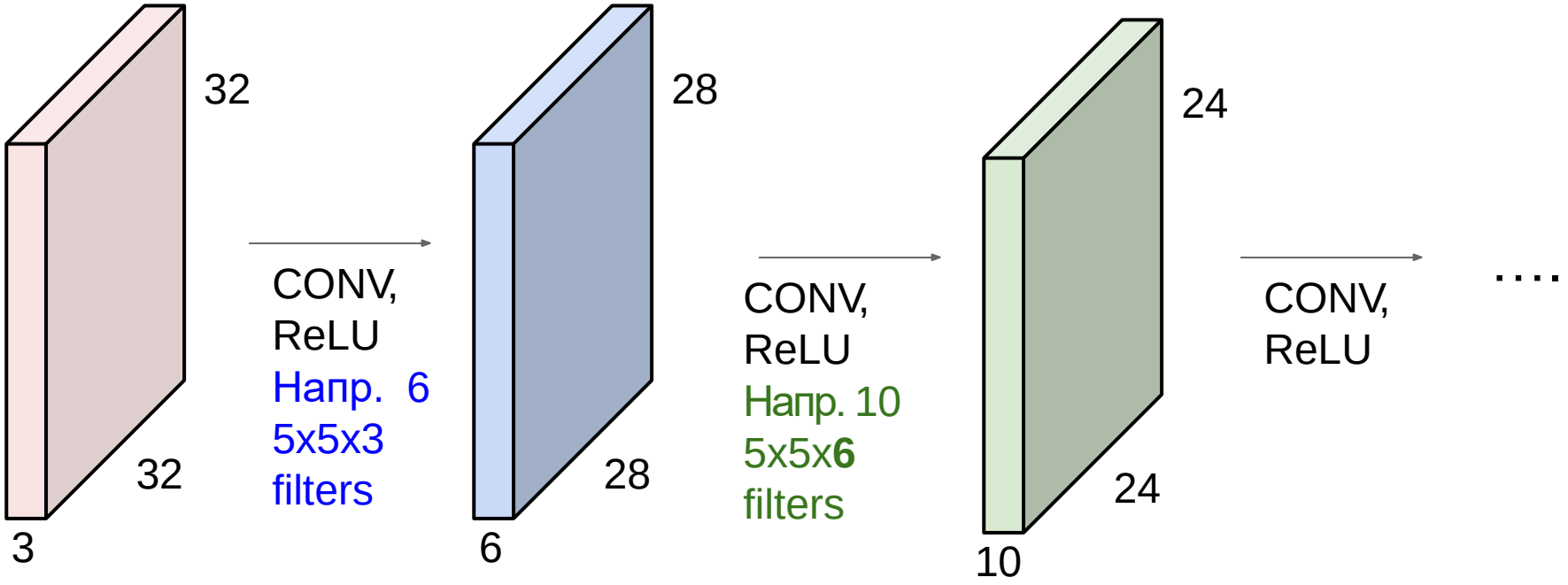


Пример: для 6 фильтров 5x5, получим 6 карт активации:



Эти карты формируют “новое” изображение 28x28x6!

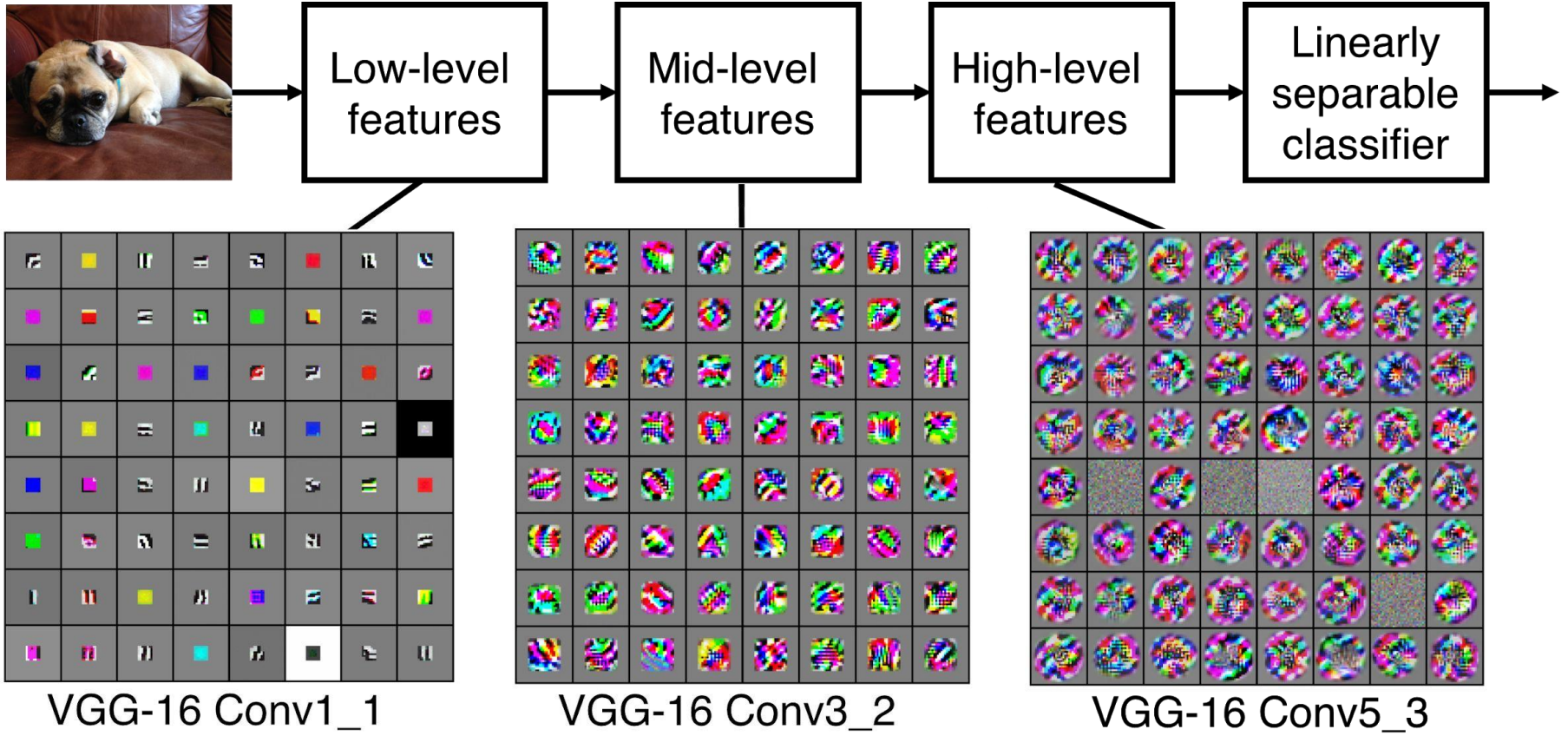
Концептуально: CHC это последовательность сверток и нелинейных активаций



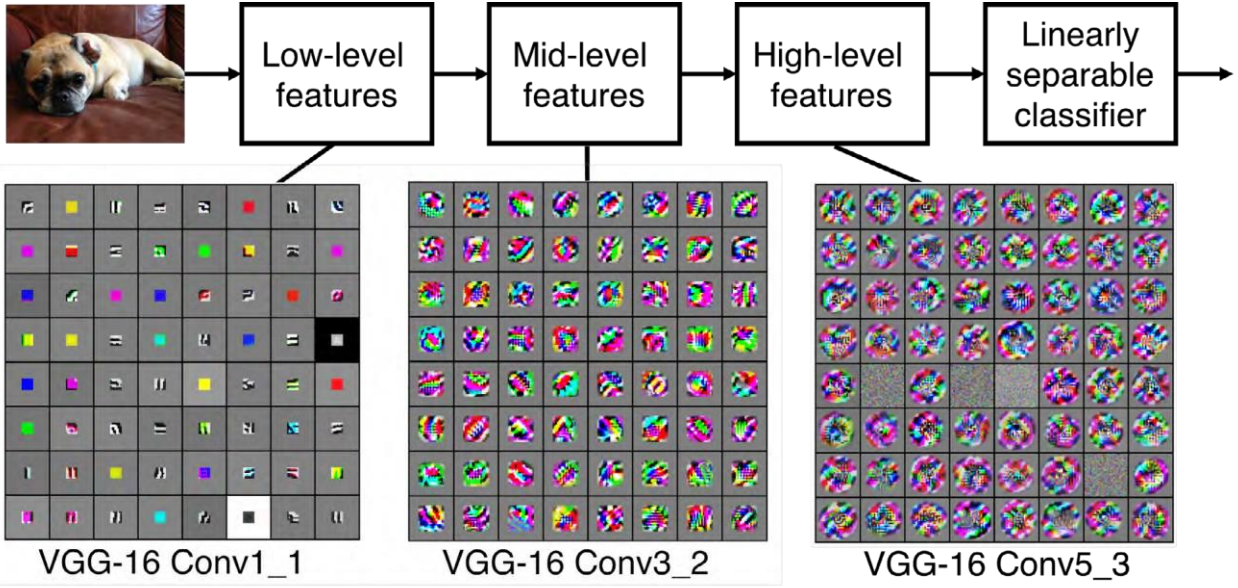
Концепция CNC:

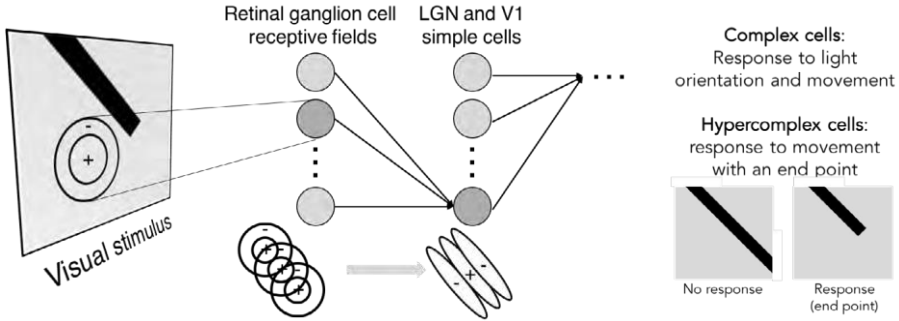
[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



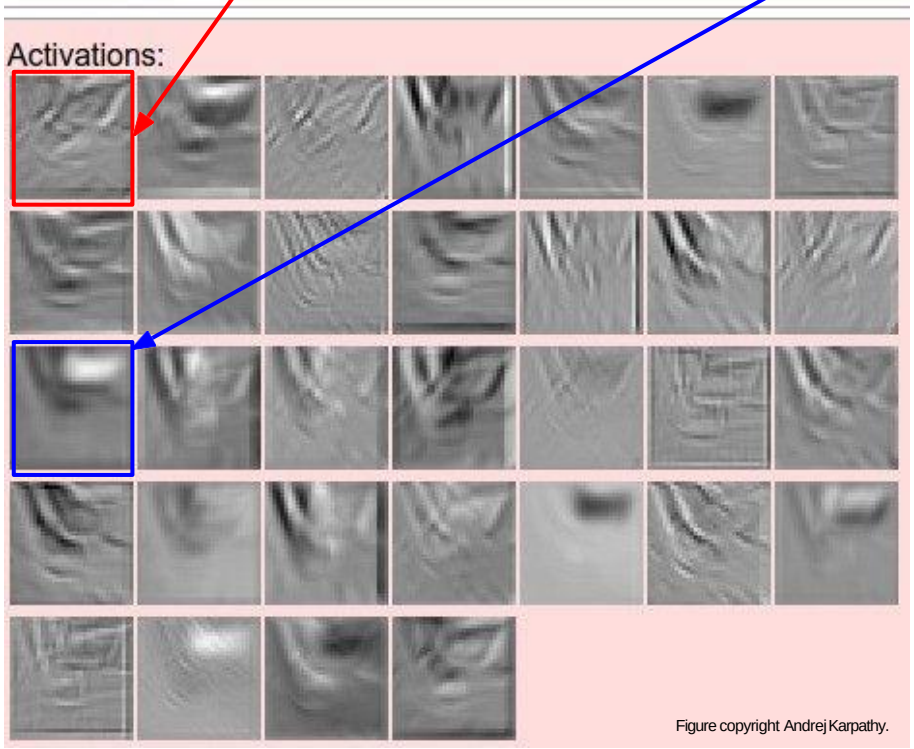
Сравним с визуальным кортексом





Один фильтр =>
Одна карта активации

фильтры 5x5
(Всего 32)

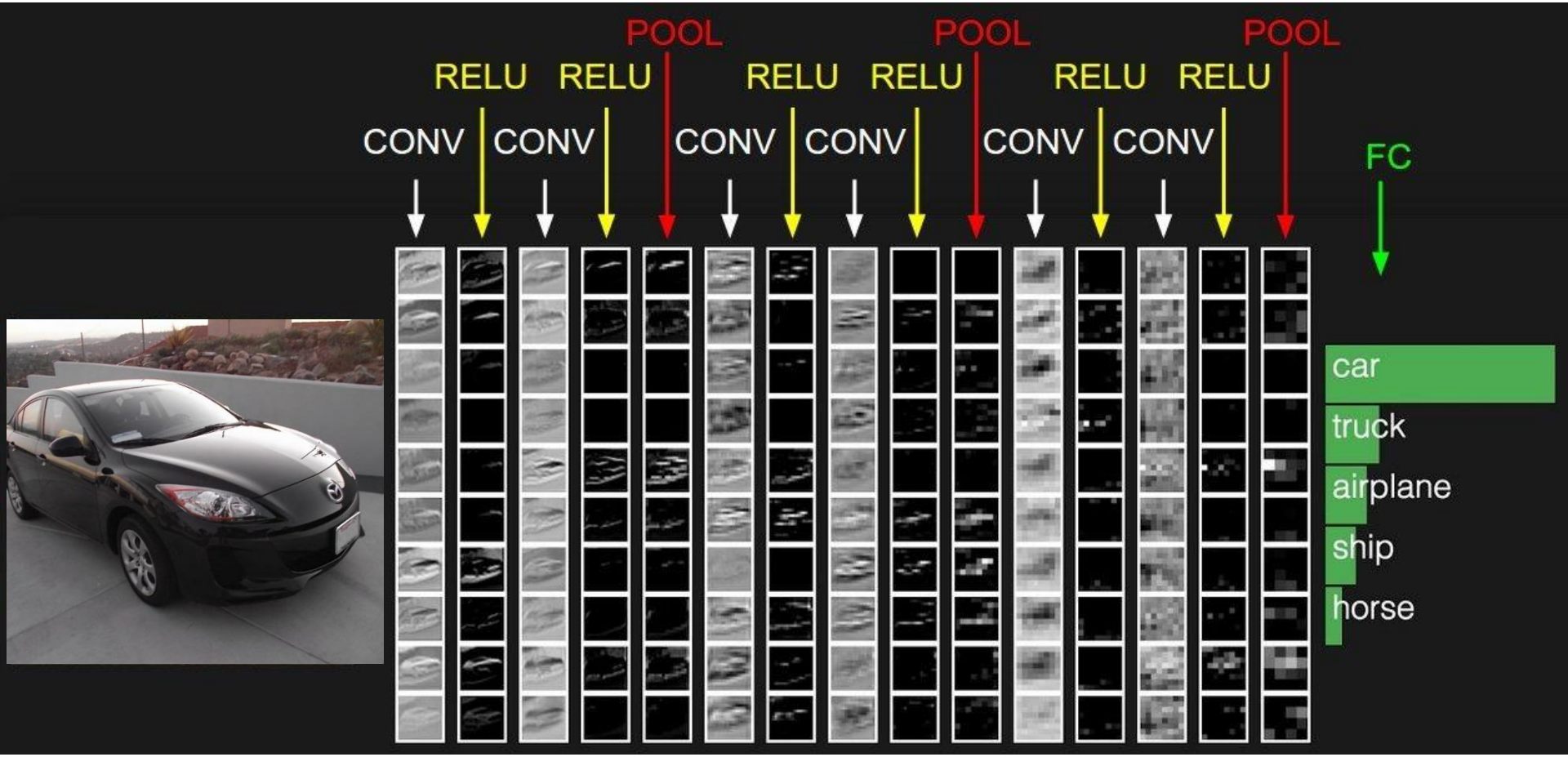


Свертка картинки и фильтра:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

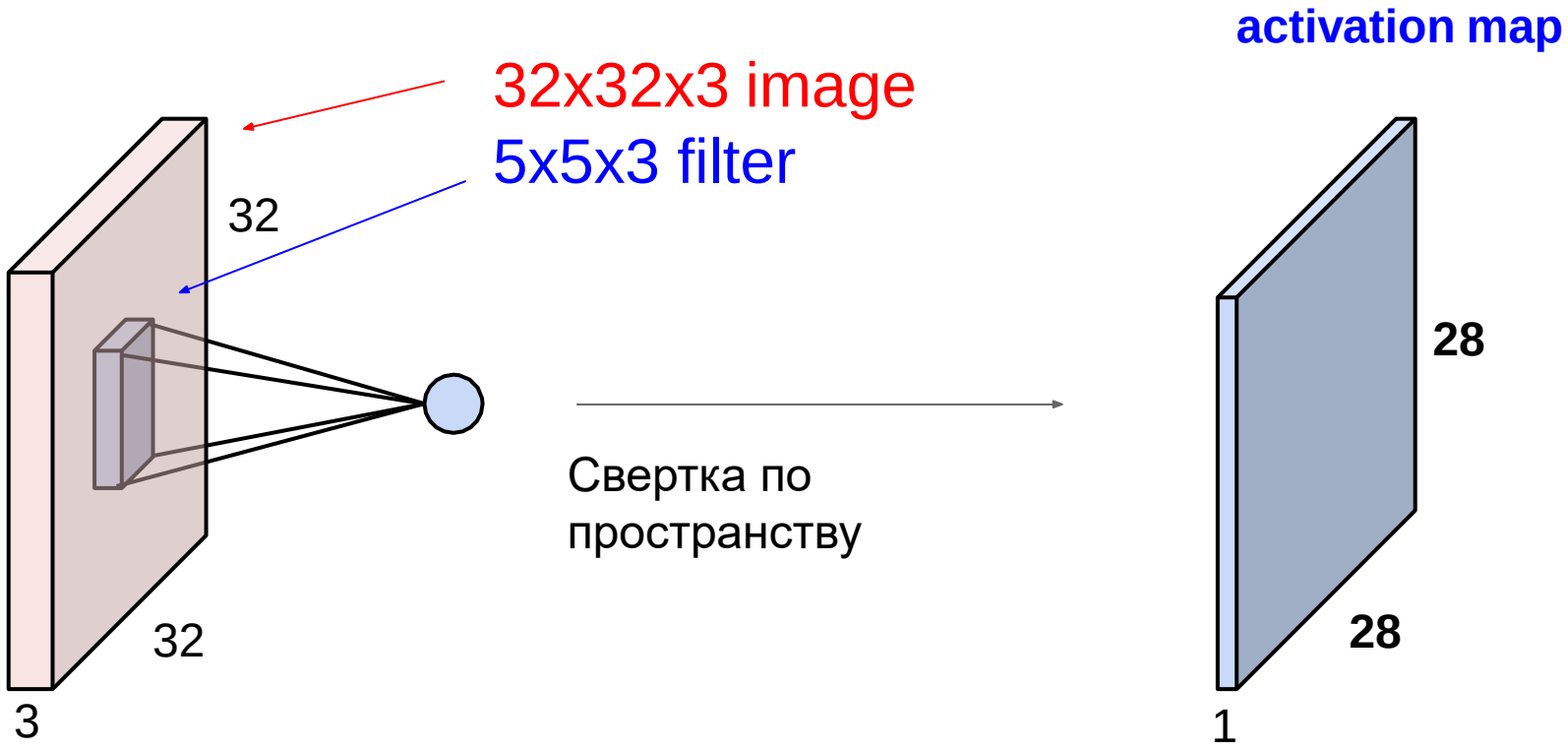
Поэлементное произведение скользящего окна фильтра и картинки

Концепция:

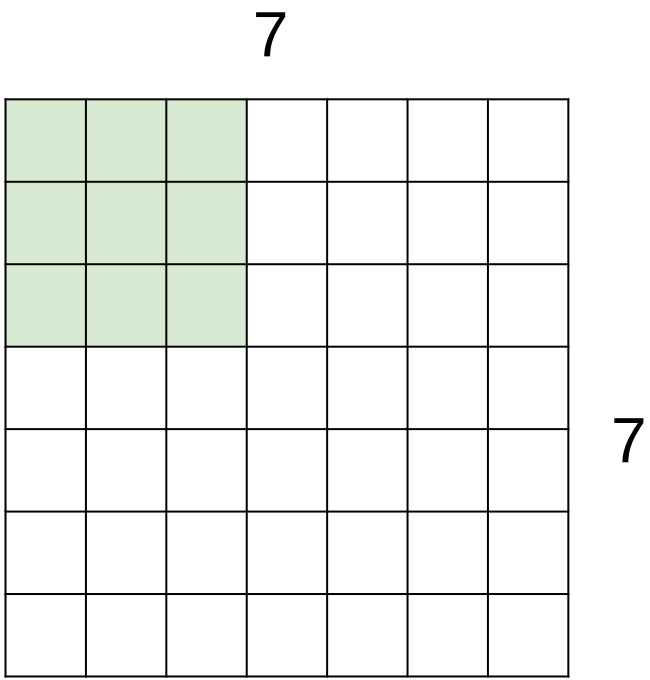


А.В. Никоноров, основано на курсе http://cs231n.stanford.edu/

Что с размерностями:



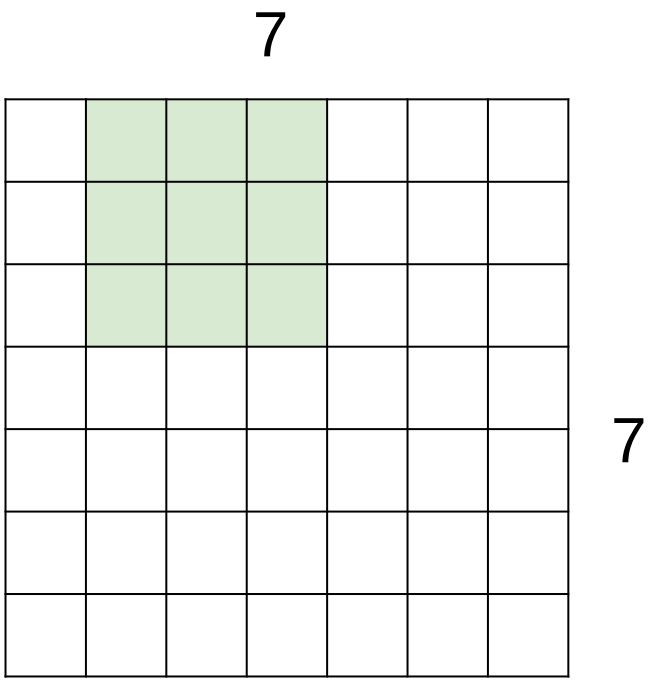
Размерности:



7x7 вход

3x3 фильтр

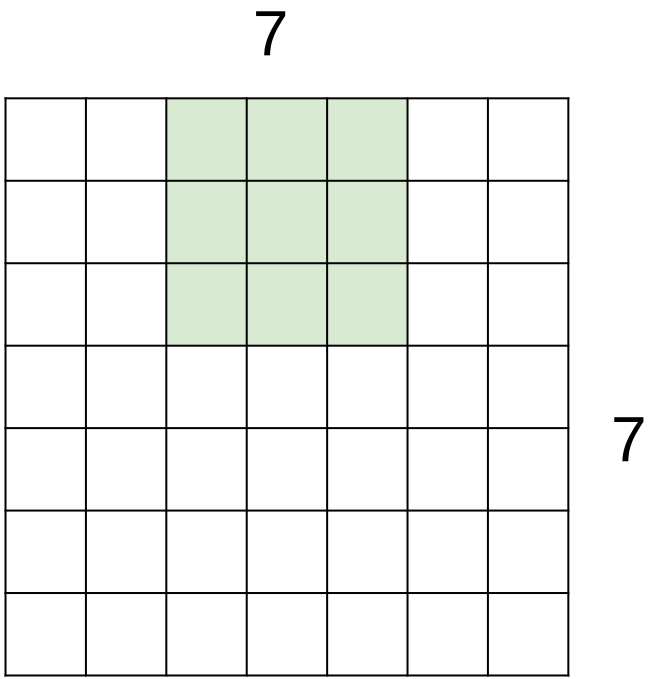
Размерности:



7x7 вход

3x3 фильтр

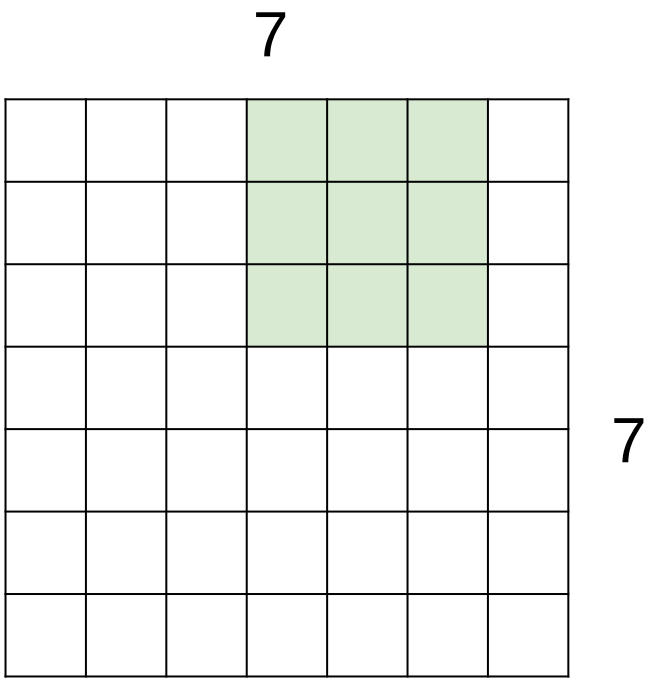
Размерности:



7x7 вход

3x3 фильтр

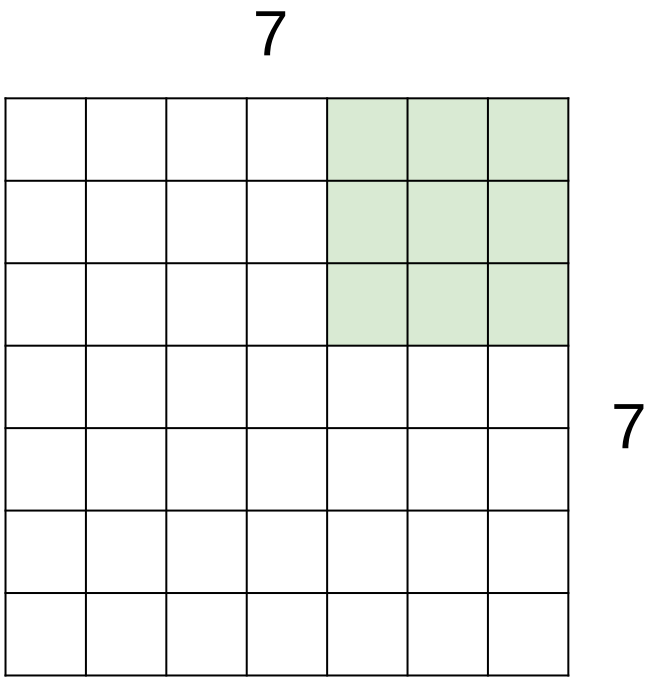
A closer look at spatial dimensions:



7x7 вход

3x3 фильтр

Размерности:

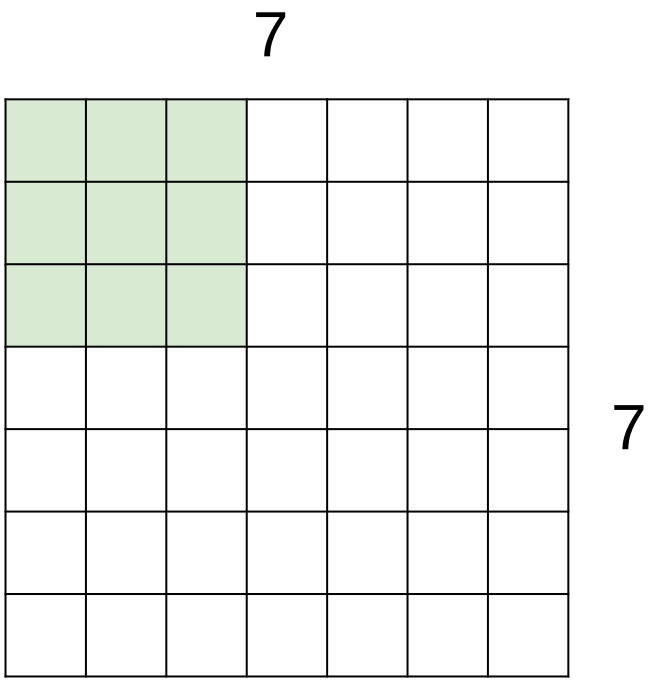


7x7 вход

3x3 фильтр

=> 5x5 выход!

Размерности:

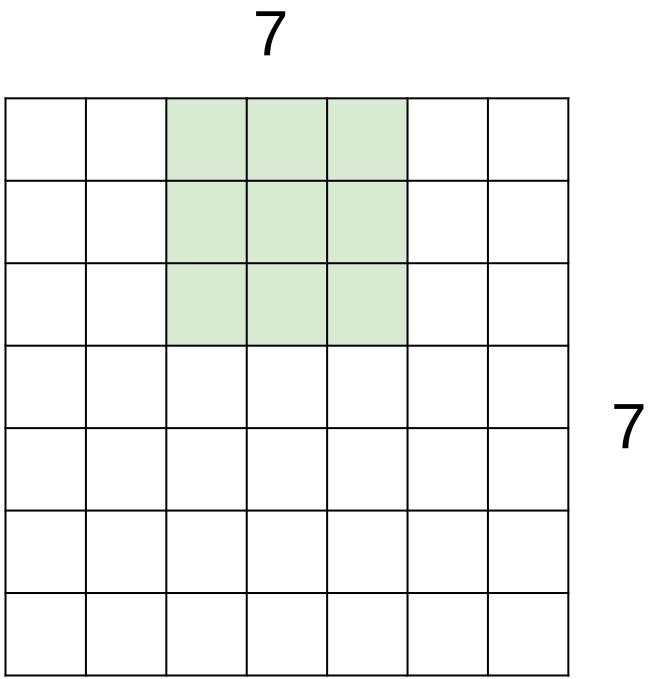


7x7 вход

3x3 фильтр

Шаг 2 (stride)

Размерности:

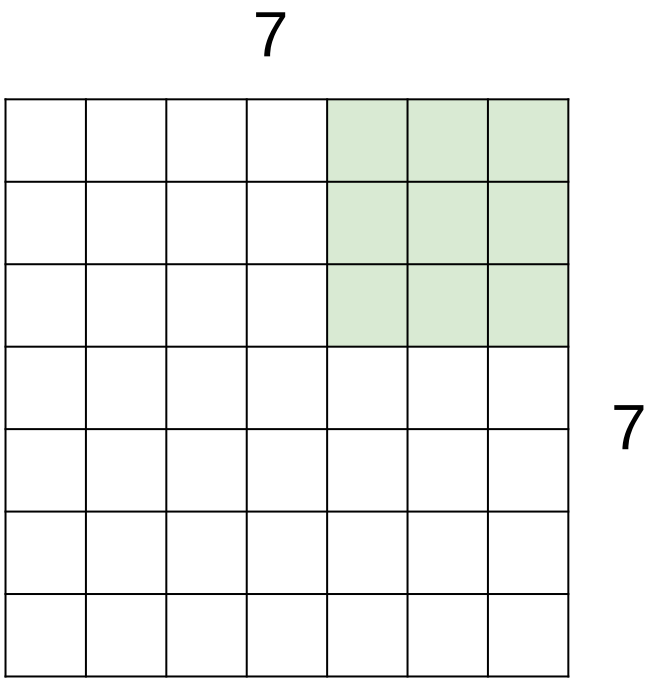


7x7 вход

3x3 фильтр

Шаг 2 (stride)

Размерности:



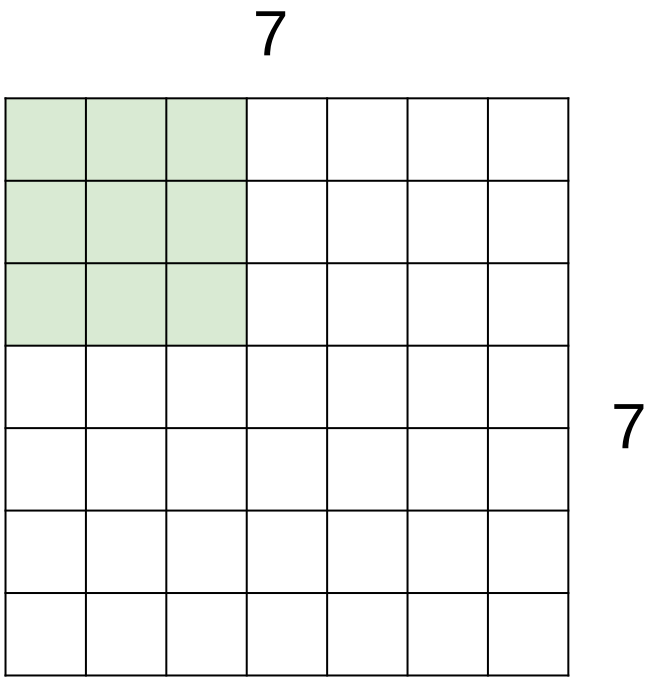
7x7 вход

3x3 фильтр

Шаг 2 (stride)

Выход 3x3!

Размерности:

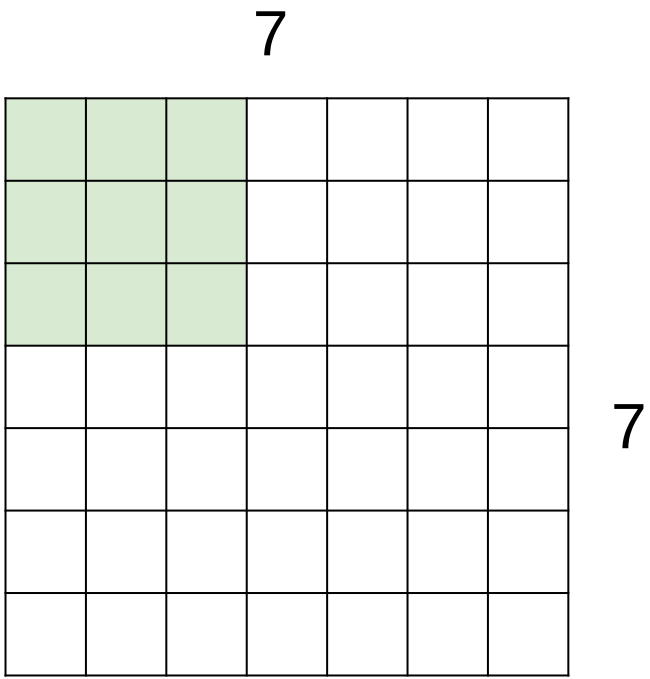


7x7 вход

3x3 фильтр

Шаг 3?

Размерности:



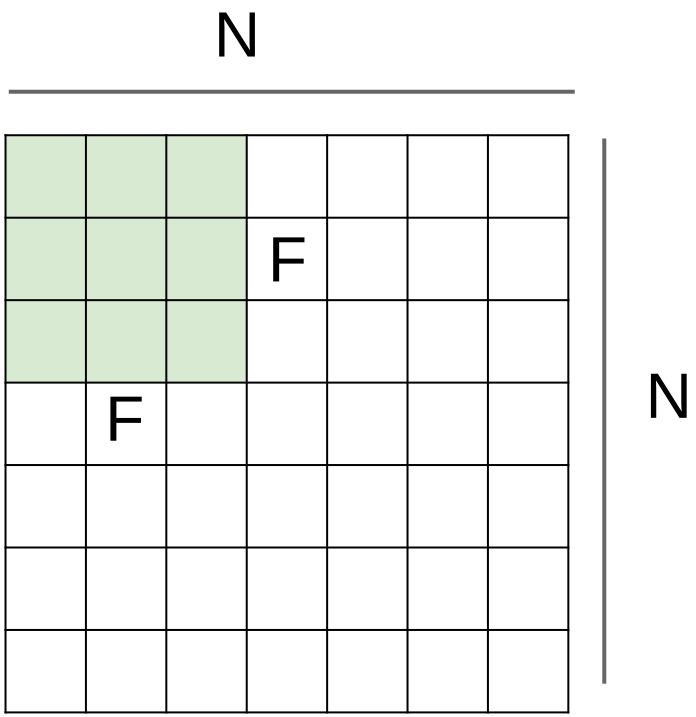
7x7 вход

3x3 фильтр

Шаг 3?

Не сработает!

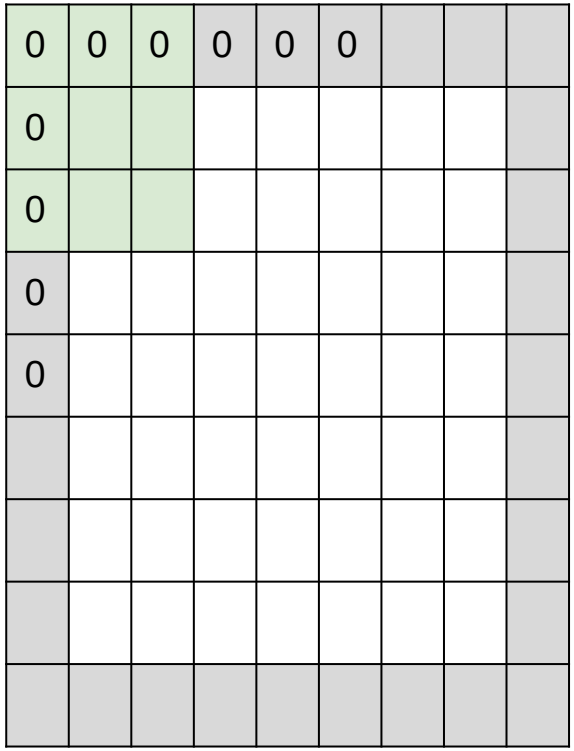
фильтр 3x3 к входу 7x7 с шагом 3 не применим.



Размерность выхода:
(N - F) / stride + 1

Например N = 7, F = 3:
шаг 1 => (7 - 3)/1 + 1 = 5
шаг 2 => (7 - 3)/2 + 1 = 3
шаг 3 => (7 - 3)/3 + 1 = 2.33 упс

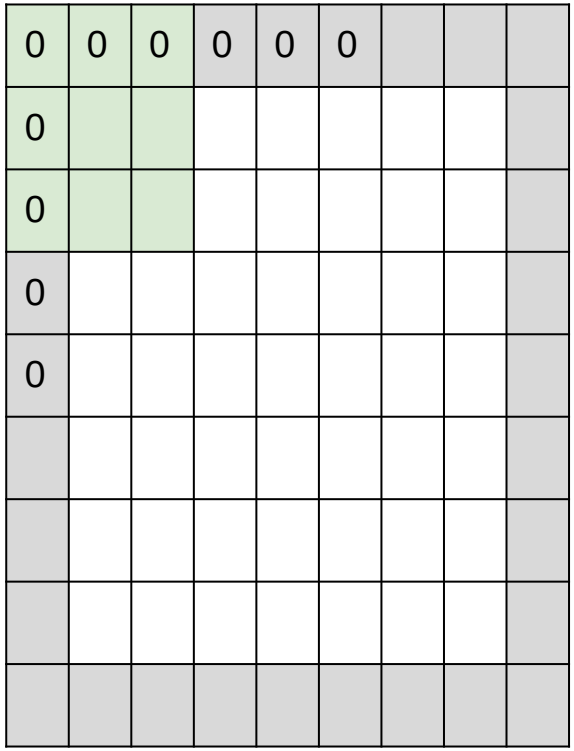
На практике используют zero padding – дополнение нулями



вход 7x7
3x3 фильтр шаг **1**
pad 1 => что на выходе?

(формула:)
(N - F) / stride + 1

На практике используют zero padding – дополнение нулями



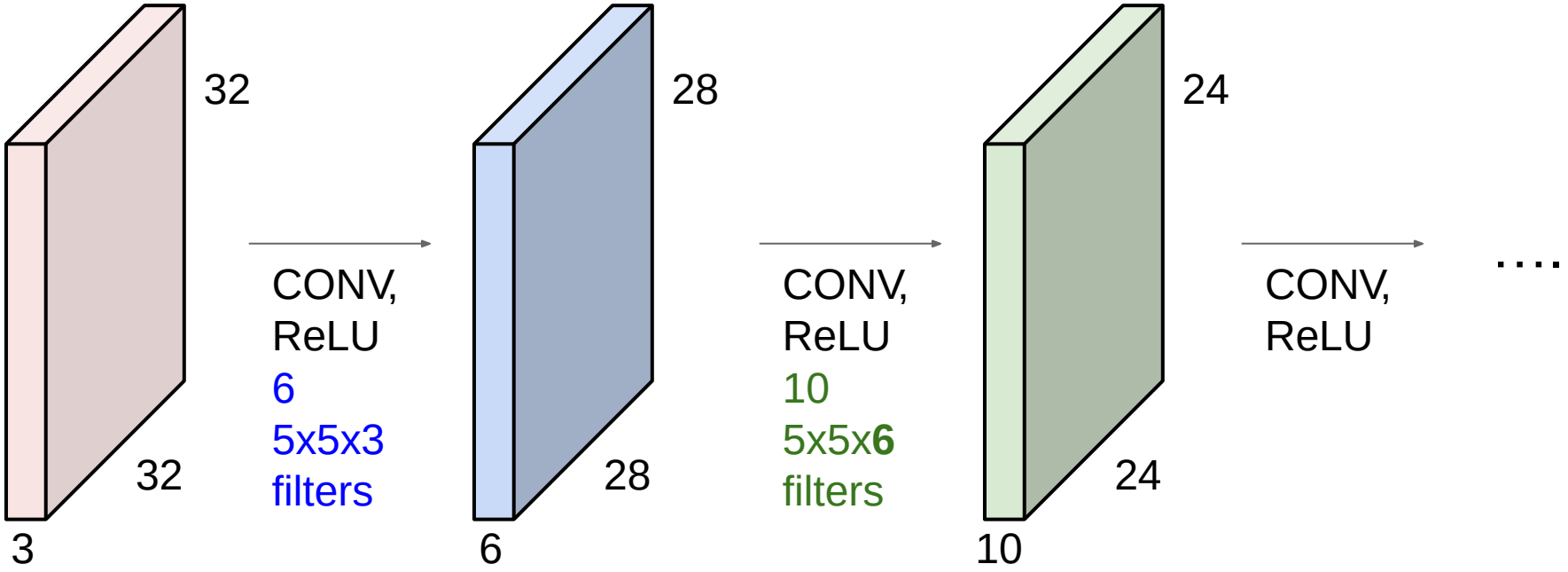
вход 7x7
3x3 фильтр шаг **1**
pad 1 => что на выходе?

Выход 7x7!
На практике CONV с шагом 1, фильтр FxF, делаем padding (F-1)/2. – сохраним размерность!

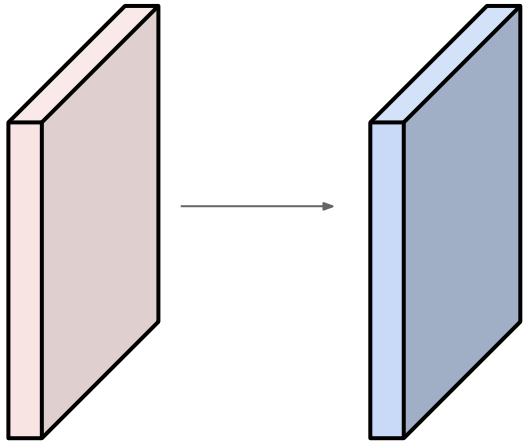
Напр. F = 3 => zero pad1
F = 5 => zero pad 2
F = 7 => zero pad 3

Вернемся…

Для входа 32x32 и сверток 5x5 результат быстро «усыхает» по пространству! (32 -> 28 -> 24 ...). Такое быстрое усыхание работает не очень.



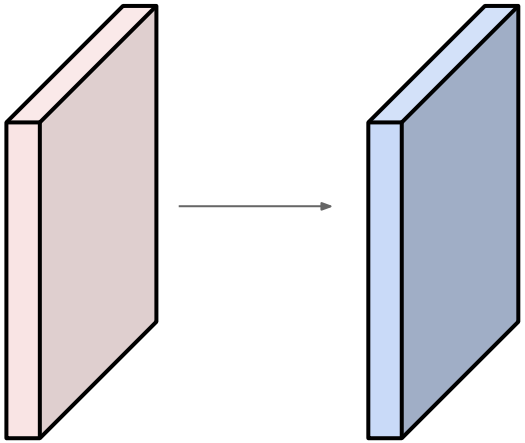
Примеры:



Вход: **32x32x3**
10 5x5 с шагом 1, паддинг 2

Размерность выхода: ?

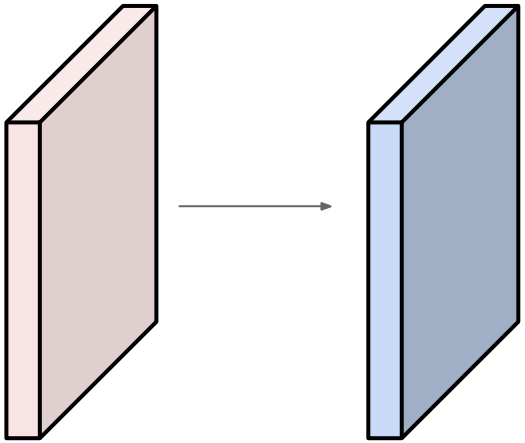
Пример:



Вход: **32x32x3**
10 5x5 фильтров шаг 1, pad 2

Размерность выхода:
(32+2*2-5)/1+1 = 32 по пространству,
Итого: **32x32x10**

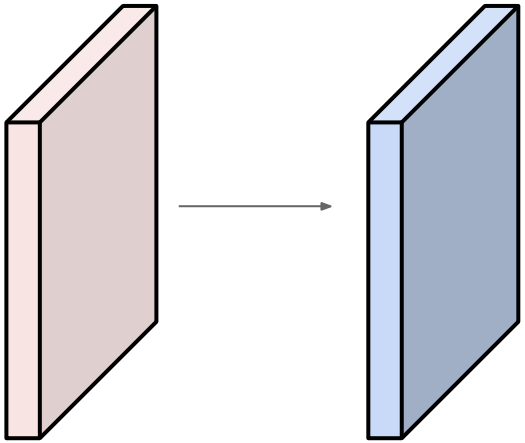
Примеры:



Вход: **32x32x3**
10 5x5 с шагом 1, паддинг 2

Количество весов в слое: ?

Пример:



Вход: **32x32x3**

10 5x5 фильтров шаг 1, pad 2

Количество весов в слое? Каждый фильтр 5*5*3 + 1 = 76 весов (+1 это смещение)

=> 76*10 = **760**

Сверточный слой: итоги

Для входа W_1 x H_1 x C

Conv слой имеет 4 гиперпараметра:

- Число сверток **K,** глубина
- Размер ядра (окна) **F**
- Шаг (stride) **S**
- zero padding **P**

Получим выход W_2 x H_2 x K

где:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

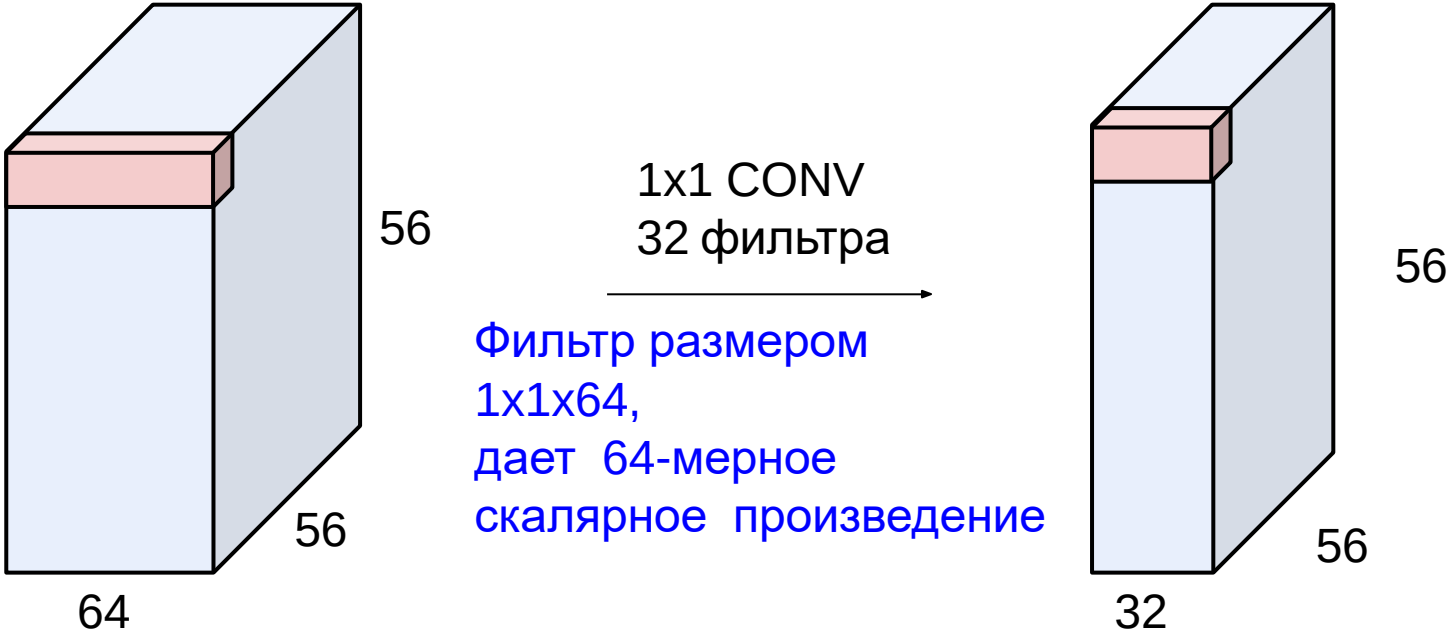
Число весов: F^2CK и K смещений

Обычно используют:

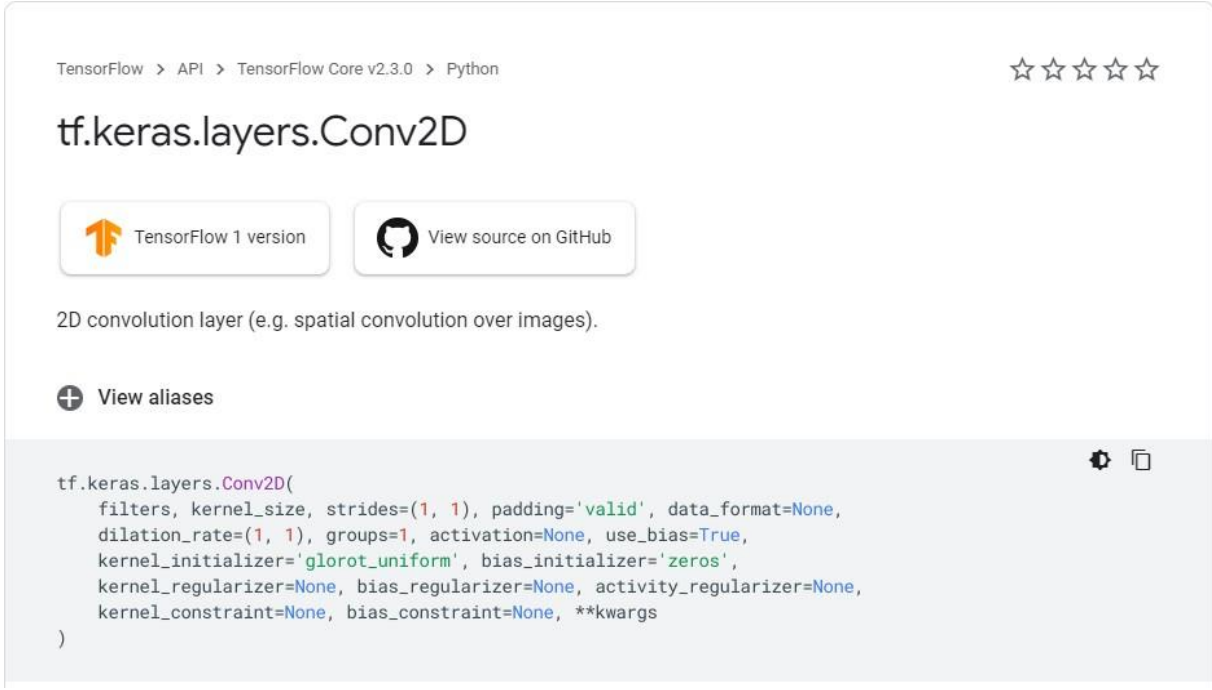
K = (степени 2 - 32, 64, 128, 512

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (подгоняем)
- F = 1, S = 1, P = 0

Кстати, свертки 1x1 отличный инструмент!



Пример: CONV слой в TF (tensorflow keras)



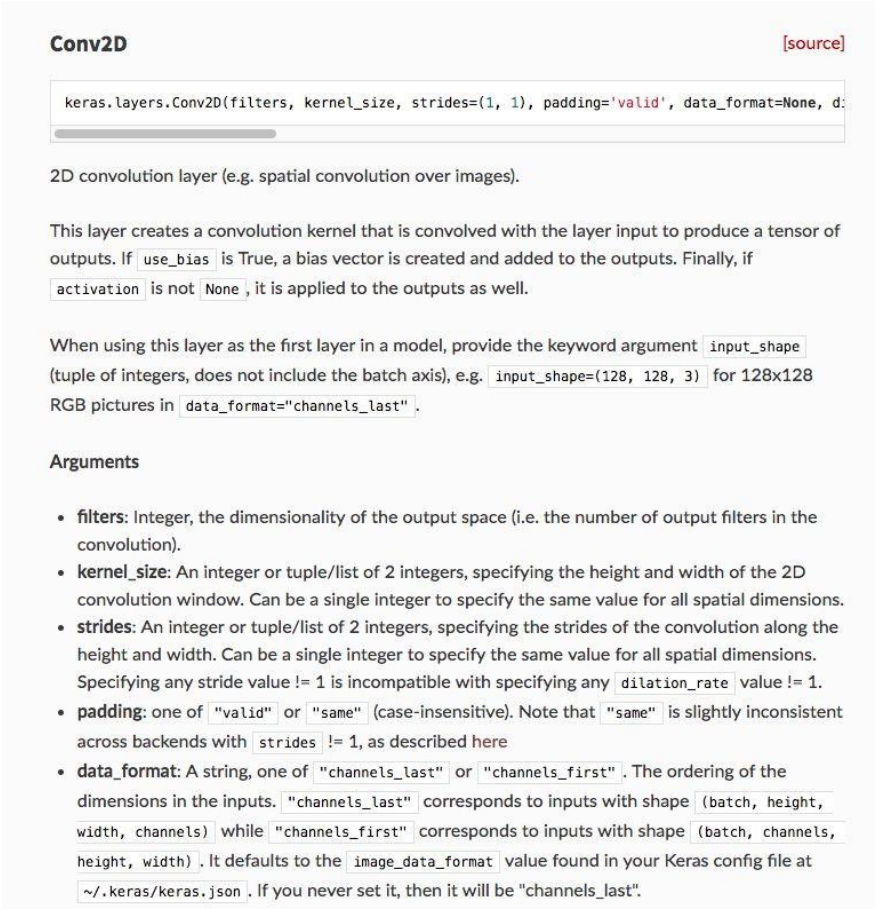
4 гиперпараметра Conv слоя:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

Пример: CONV слой в TF (tensorflow keras)

4 гиперпараметра Conv слоя:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**



Conv2D [source]

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, d
```

2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the batch axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

Arguments

- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: one of `"valid"` or `"same"` (case-insensitive). Note that `"same"` is slightly inconsistent across backends with `strides` != 1, as described here
- **data_format**: A string, one of `"channels_last"` or `"channels_first"`. The ordering of the dimensions in the inputs. `"channels_last"` corresponds to inputs with shape `(batch, height, width, channels)` while `"channels_first"` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

Пример: CONV слой в TF (tensorflow keras)

4 гиперпараметра Conv слоя:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

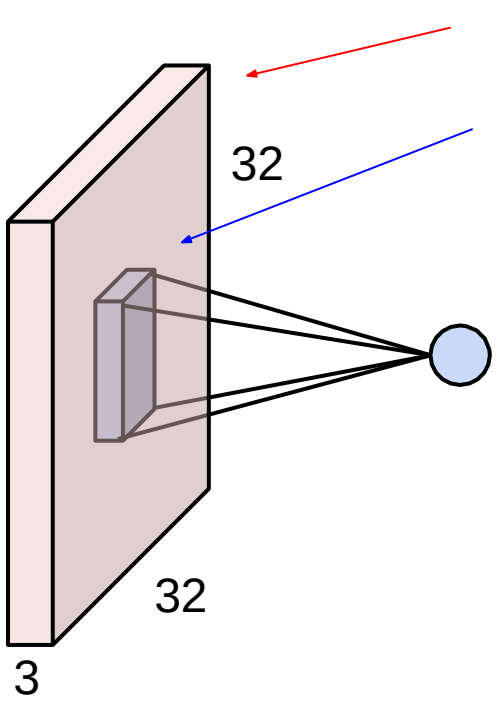
where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
    - At groups=1, all inputs are convolved to all outputs.
    - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
    - At groups= `in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$.

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

The brain/neuron view of CONV Layer

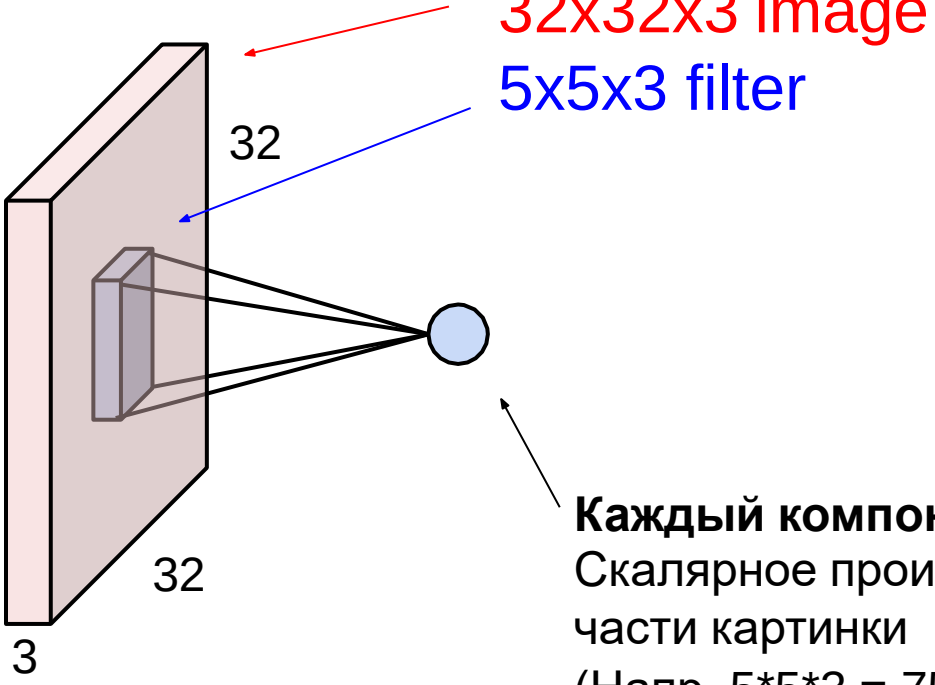


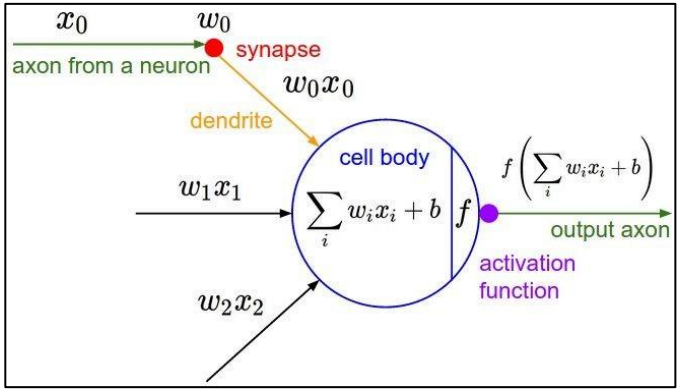
32x32x3 image

5x5x3 filter

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

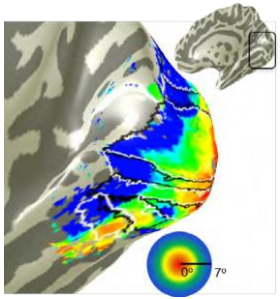
CONV слой с точки зрения нейробиологии





Это просто локальные связи нейрона...

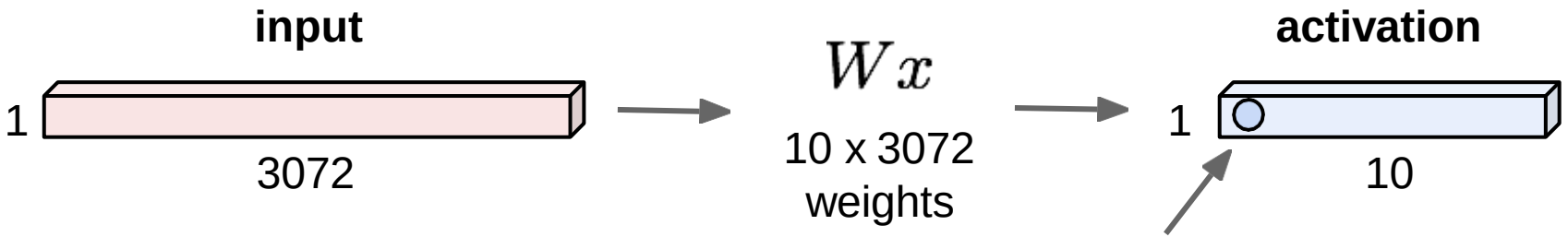
Каждый компонент:
Скалярное произведение фильтра и части картинки
(Напр. 5*5*3 = 75-мерное скалярное произведение)



Напомним: полносвязный слой

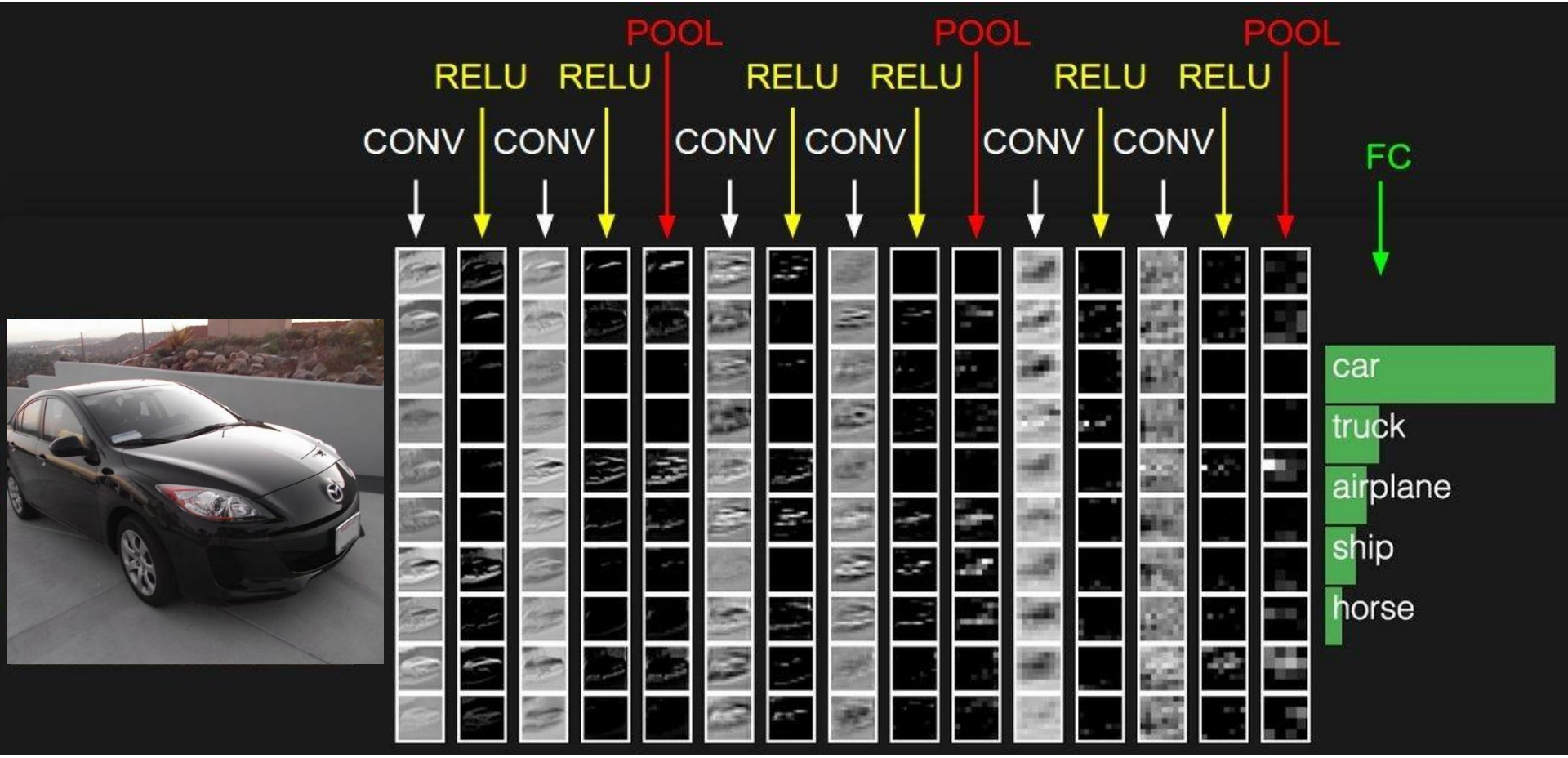
32x32x3 -> в вектор 3072 x 1

Каждый нейрон имеет на входе все данные



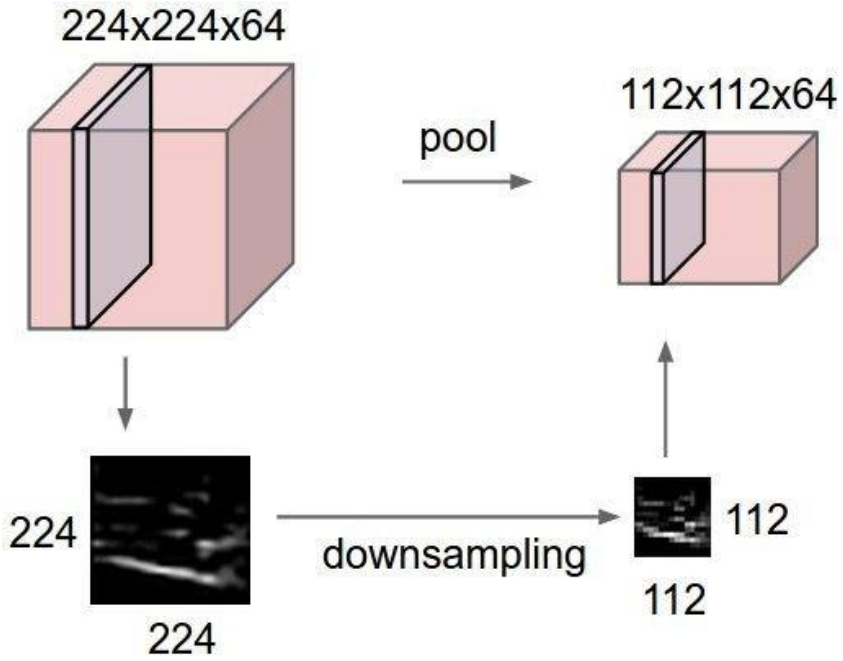
Каждый компонент выхода:
Скалярное произведение строки матрицы W и входа (3072-мерное скалярное произведение)

Добавим два новых слоя: POOL/FC

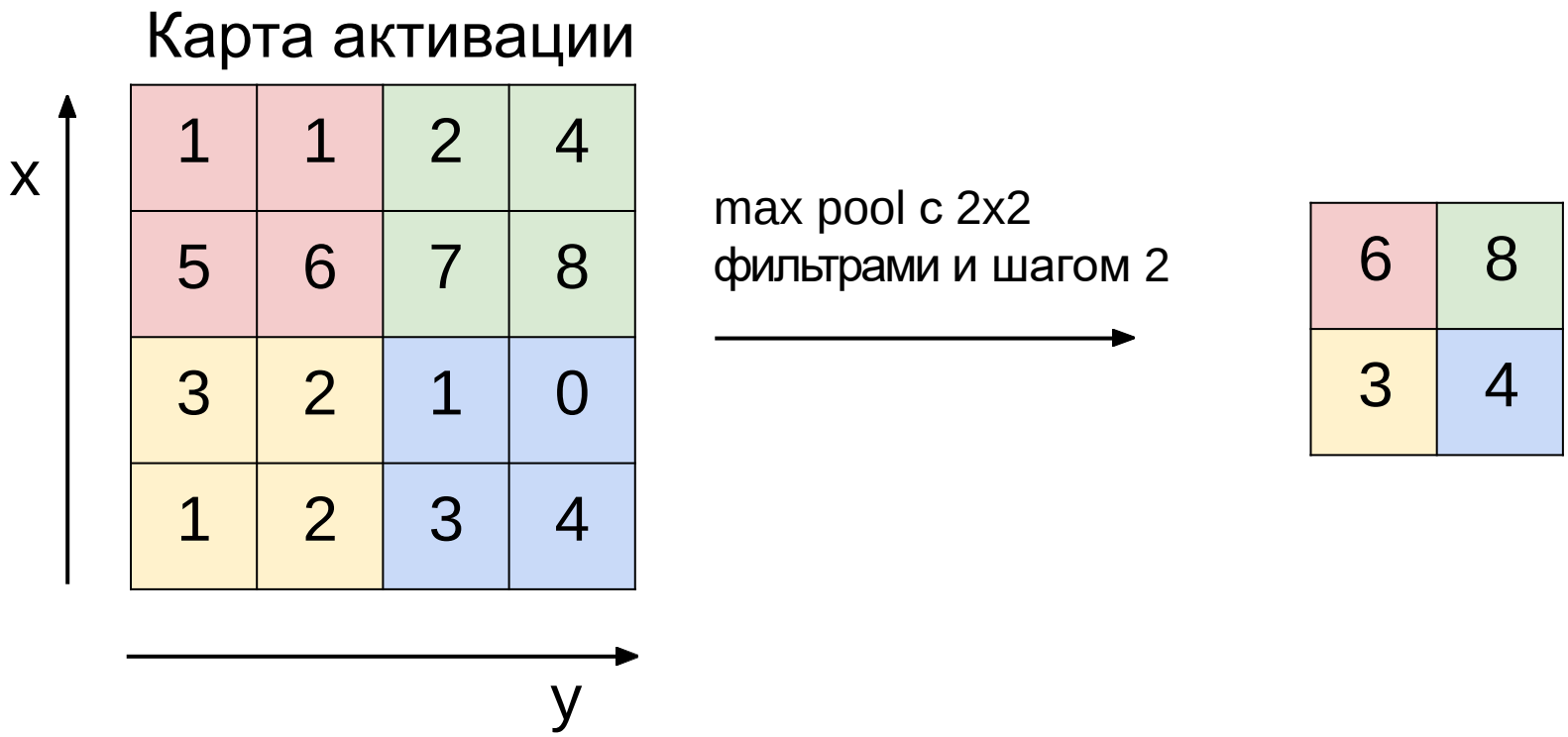


Pooling слой

- Снижает размерность
- Обрабатывает каждую карту активации независимо:



MAX POOLING


Карта активации
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool с 2x2
фильтрами и шагом 2
6 8
3 4

Pooling layer: итого

Для входа W_1 x H_1 x C
Pooling имеет 2 гиперпараметра:

- Размер **F**
- Шаг **S**

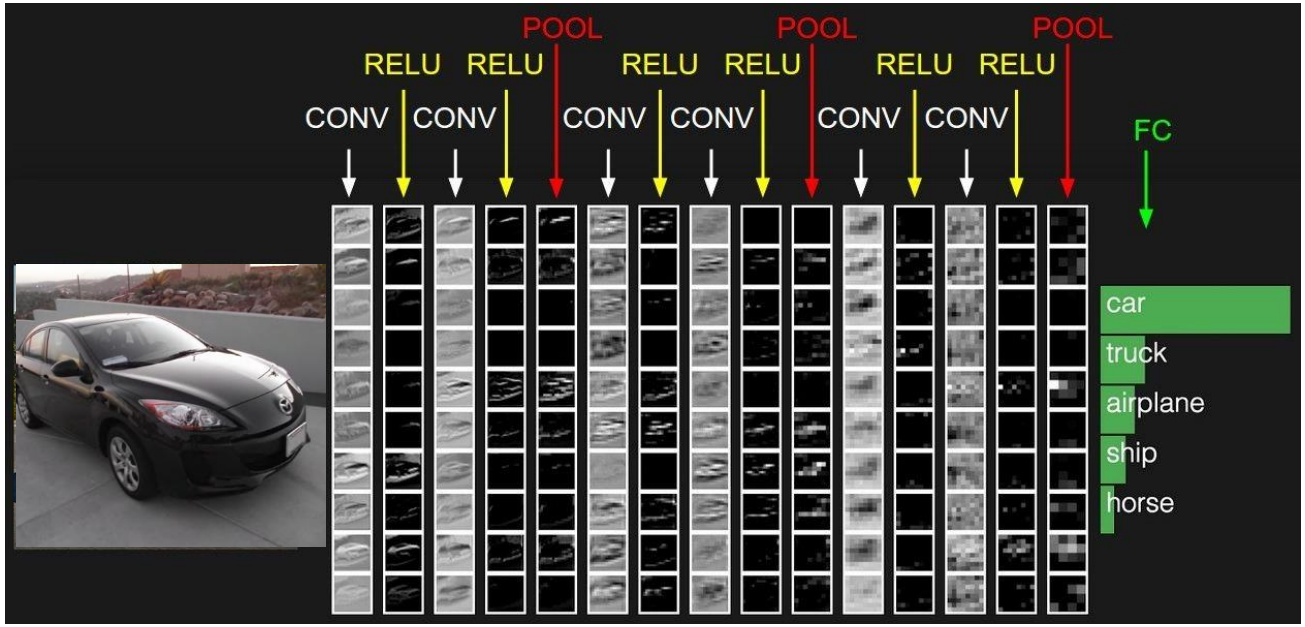
Выход имеет размер W_2 x H_2 x C:

- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Число параметров: 0 – необучаемый слой!

Fully Connected Layer (FC layer)

- Работает как однослойный перцептрон



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Итого

- ConvNets/CHC состоит из слоев CONV,POOL,FC
- Используют маленькие свертки (3*3) и глубокую архитектуру
- В тренде отказаться от POOL/FC слоев, сделать все свертками
- Исторически архитектуры имеют вид:

  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX

  где N ~5, M достаточно большое, 0 <= K <= 2.

  - Но позже возникают более хитрые архитектуры ResNet/GoogLeNet

Задача на дом:

Входное изображение:

[1 2 3 4 5]
[2 2 1 1 1]
[3 2 1 1 1]
[4 1 1 1 1]
[5 1 1 1 1]

CONV фильтр:

[0 -1 0]
[1 1 1]
[0 -1 0]

Посчитать выход сети: conv(depth=1, stride=2) -> ReLU -> MaxPool