Глубокое обучение в компьютерном зрении

Занятие 4
CNNs

Дмитрий Яшунин, к.ф.-м.н
IntelliVision

e-mail: yashuninda@yandex.ru

На прошлом занятии: **Классификация изображений**

Ключевая задача компьютерного зрения

К какому классу принадлежит изображение?
классы: человек, животное, автомобиль …

кот

На прошлом занятии: **Линейный классификатор**

Image

Array of **32x32x3** numbers (3072 numbers total)

CIFAR-10
50,000 training images
10,000 testing images
10 classes

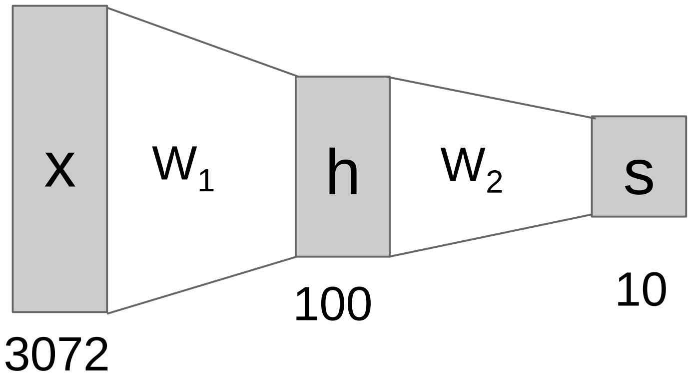
$$\underbrace{s = \boxed{f(x,W)}}_{10\times1} = \underbrace{\boxed{W}}_{10\times3072}\overbrace{\boxed{x}}^{3072\times1} + \boxed{b}\ 10\times1$$

s – scores
W – weights or parameters
x – image pixels
b – bias

На прошлом занятии: **Neural Networks**

Linear score function: $f = Wx$

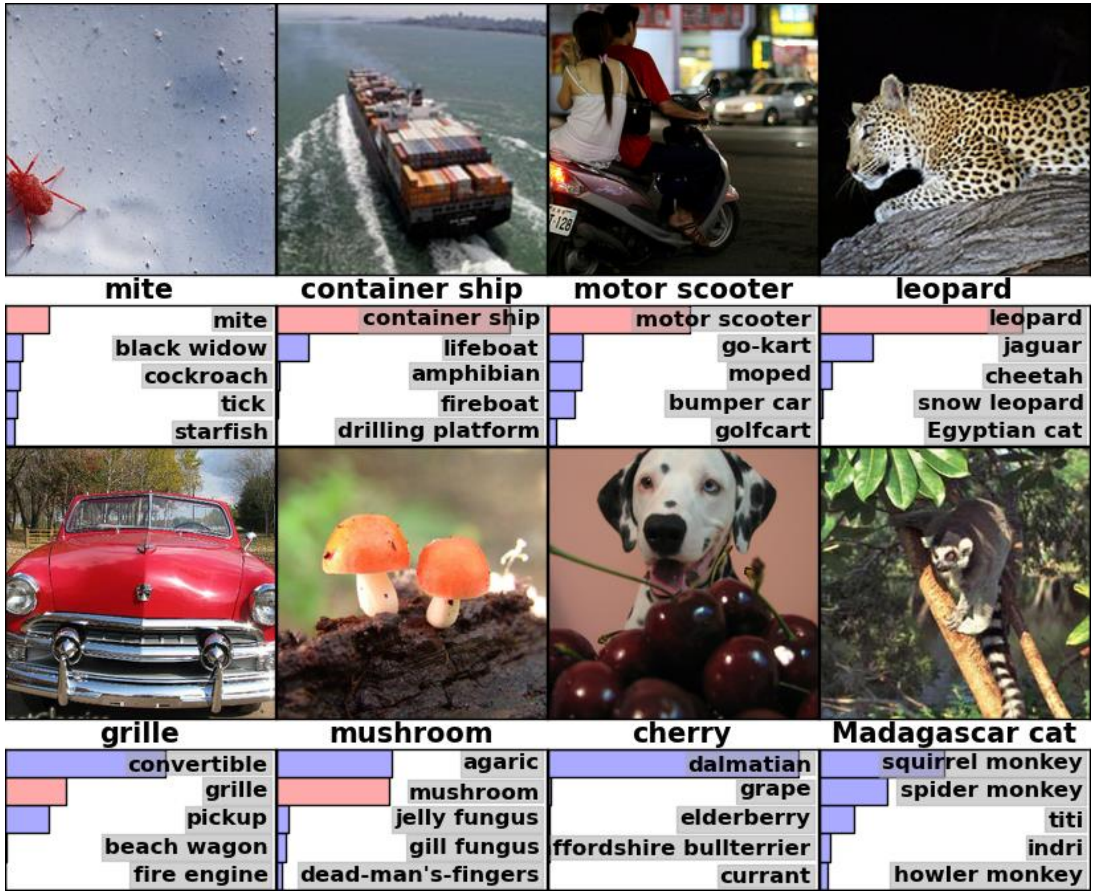
2-layer Neural Network: $f = W_2 \max(0, W_1 x)$



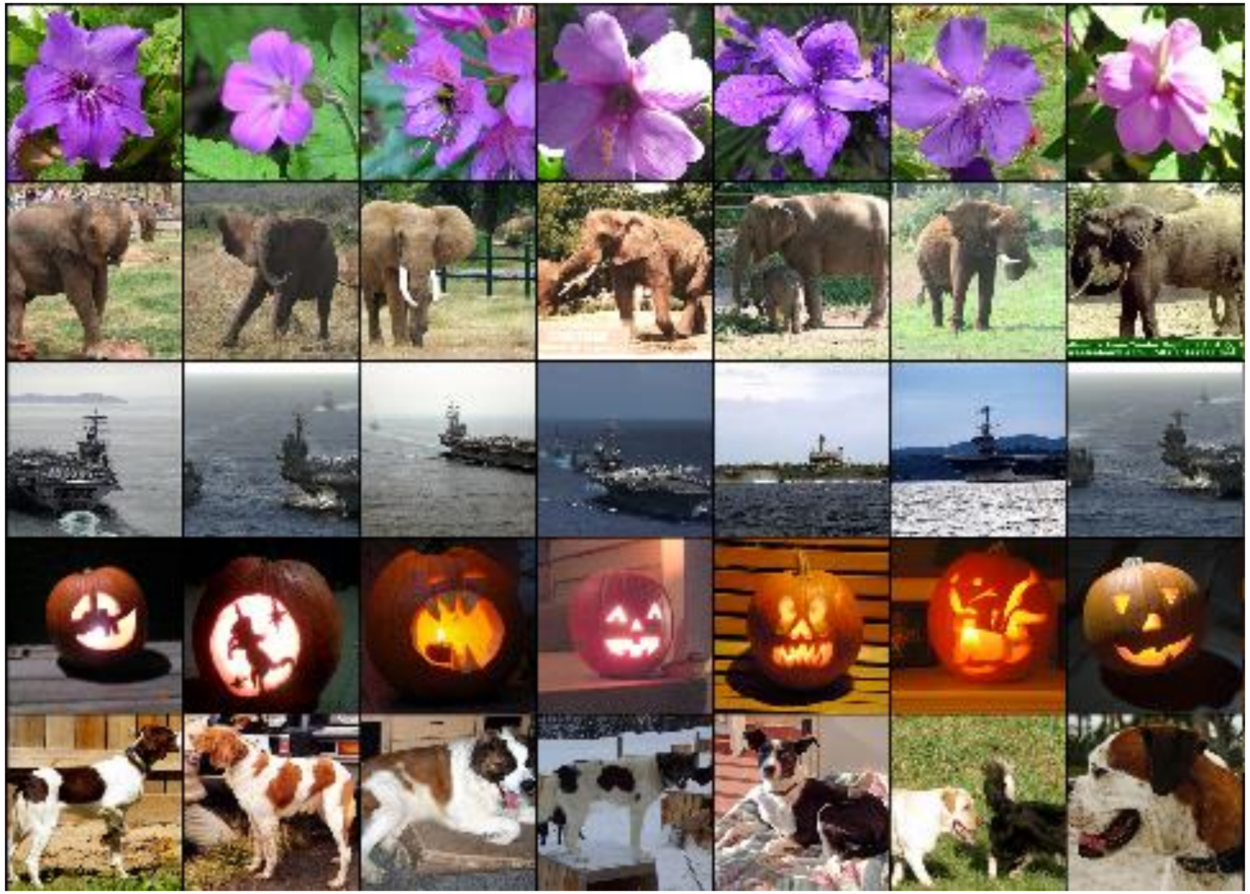
Сверточные сети

Применение сверточных сетей

Классификация изображений

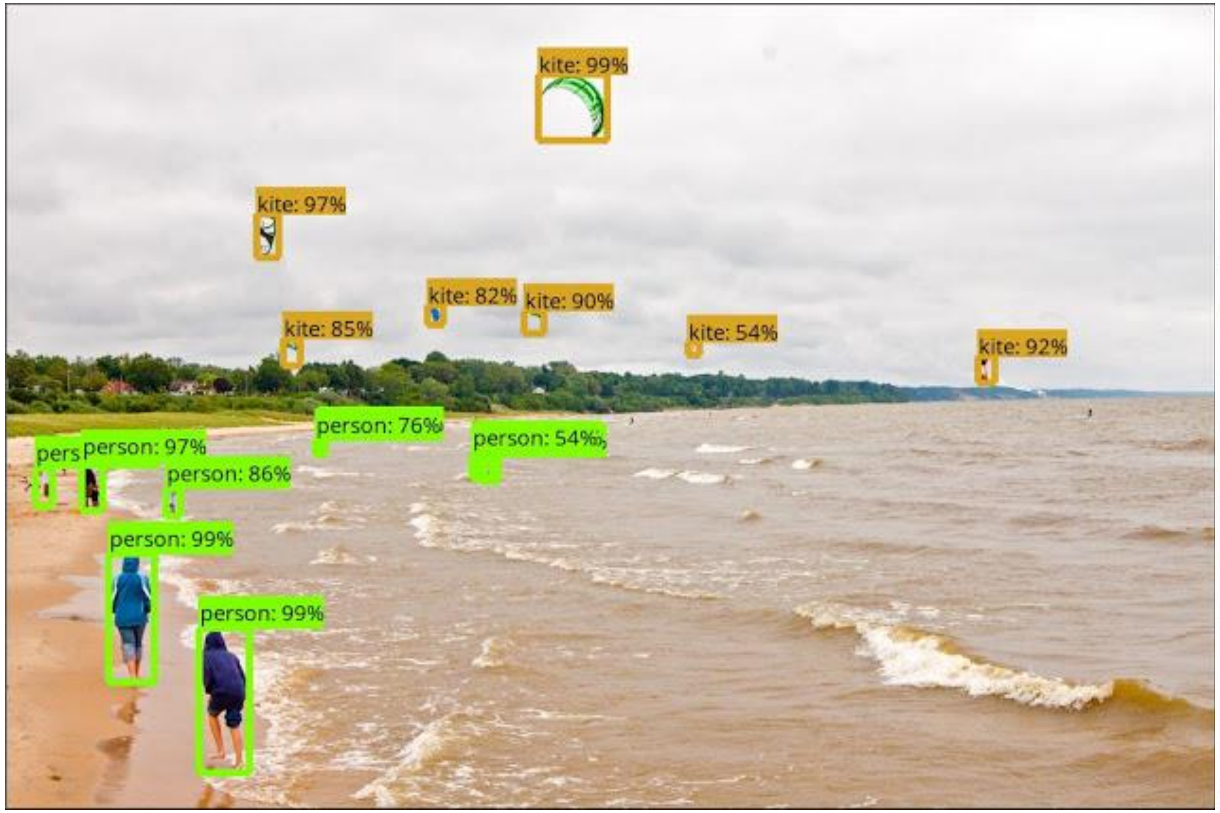


Поиск похожих изображений

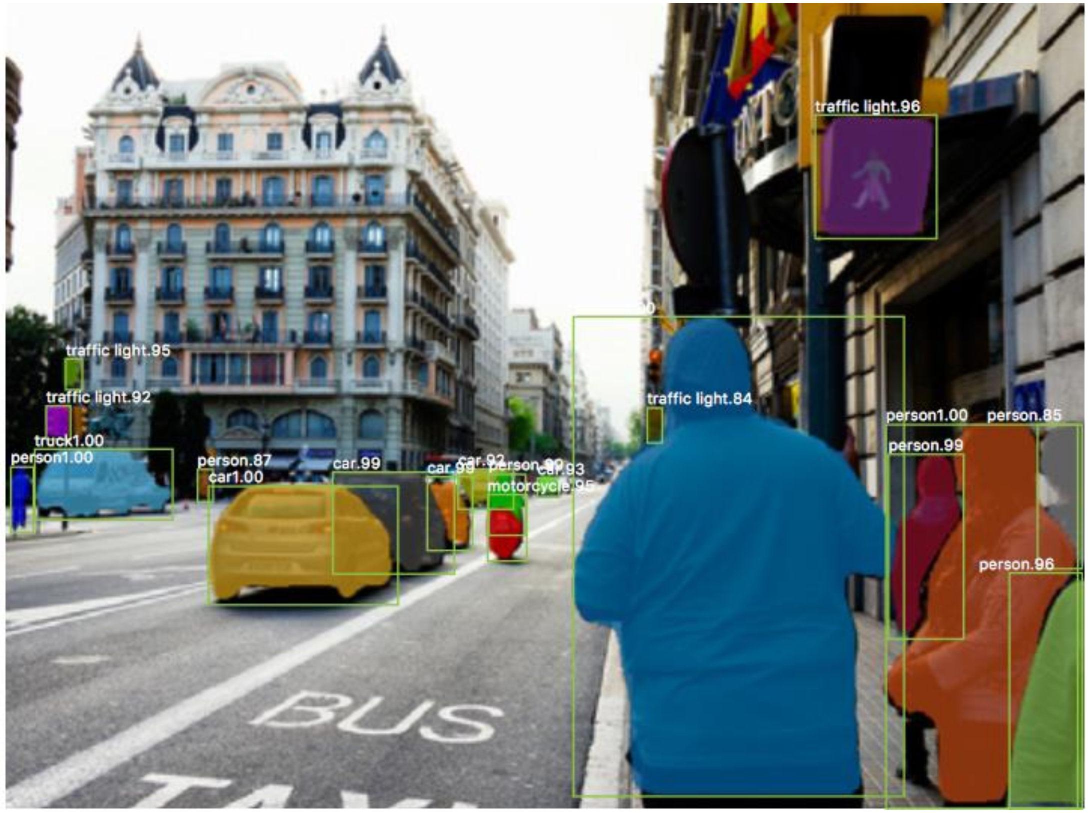


Применение сверточных сетей

Детектирование объектов

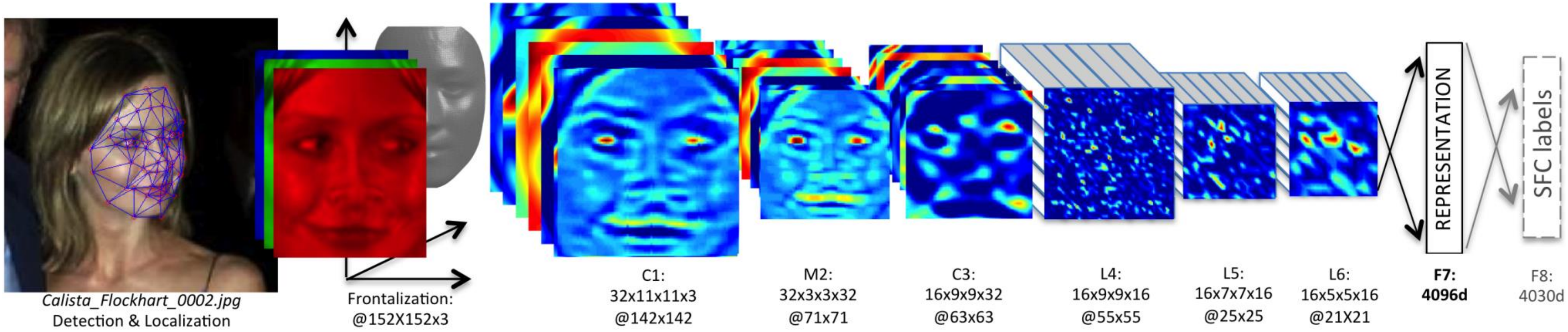


Сегментация (Instance segmentation)



Применение сверточных сетей

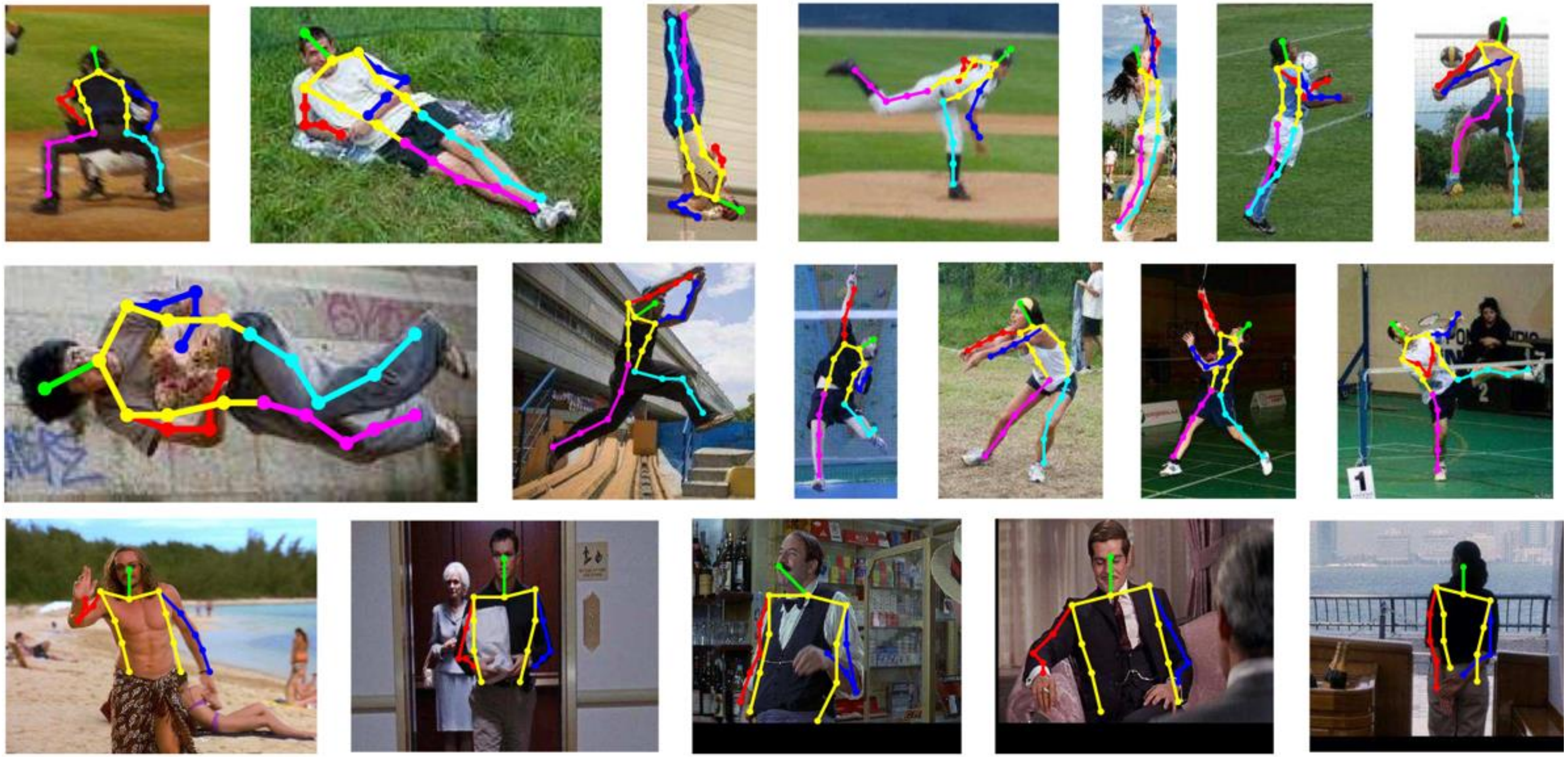
Распознавание лиц



Распознавание людей

Применение сверточных сетей

Определение позы

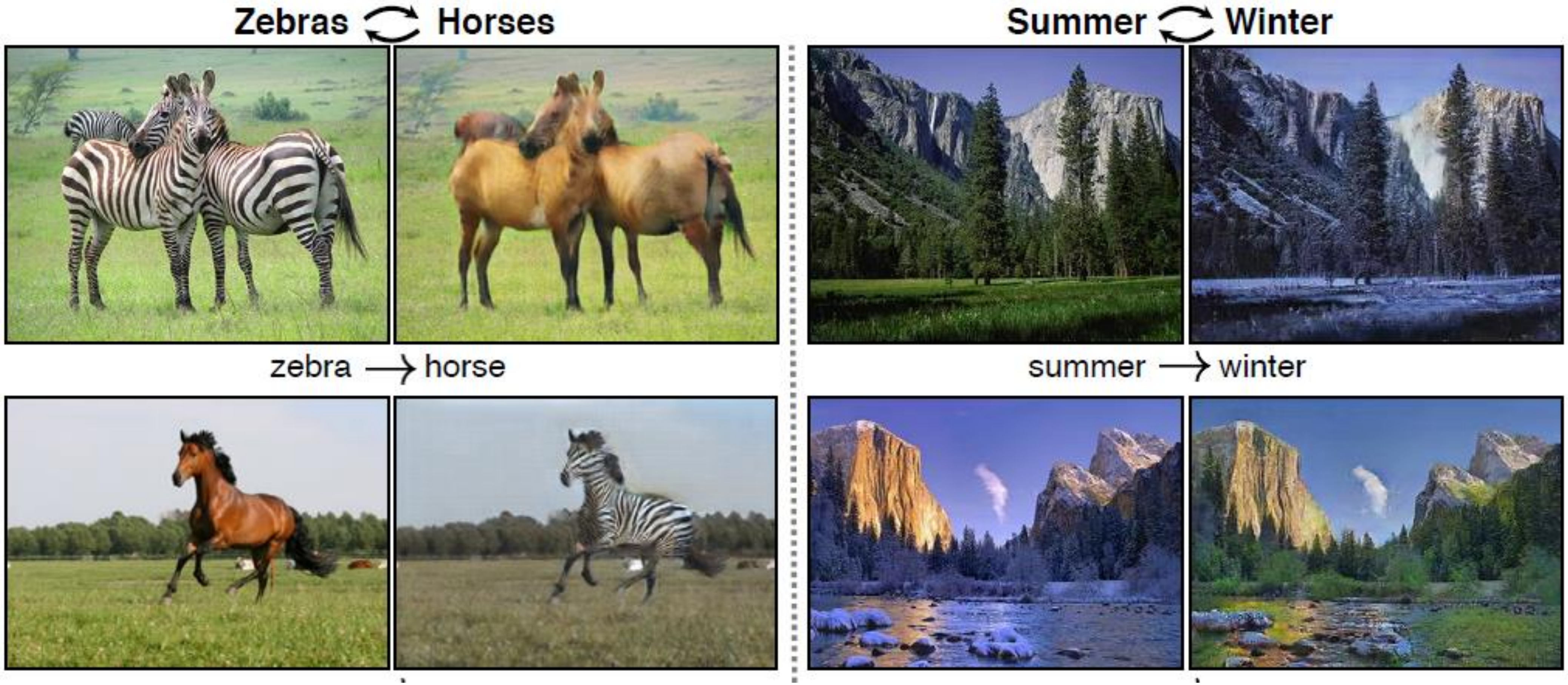


Применение сверточных сетей

Перемещение стиля
изображений
(Style Transfer)

Применение сверточных сетей

Преобразование изображений (Image-to-Image Translation)



Demo: https://www.youtube.com/watch?v=9reHvktowLY https://www.youtube.com/watch?v=Fea4kZq0oFQ

Применение сверточных сетей

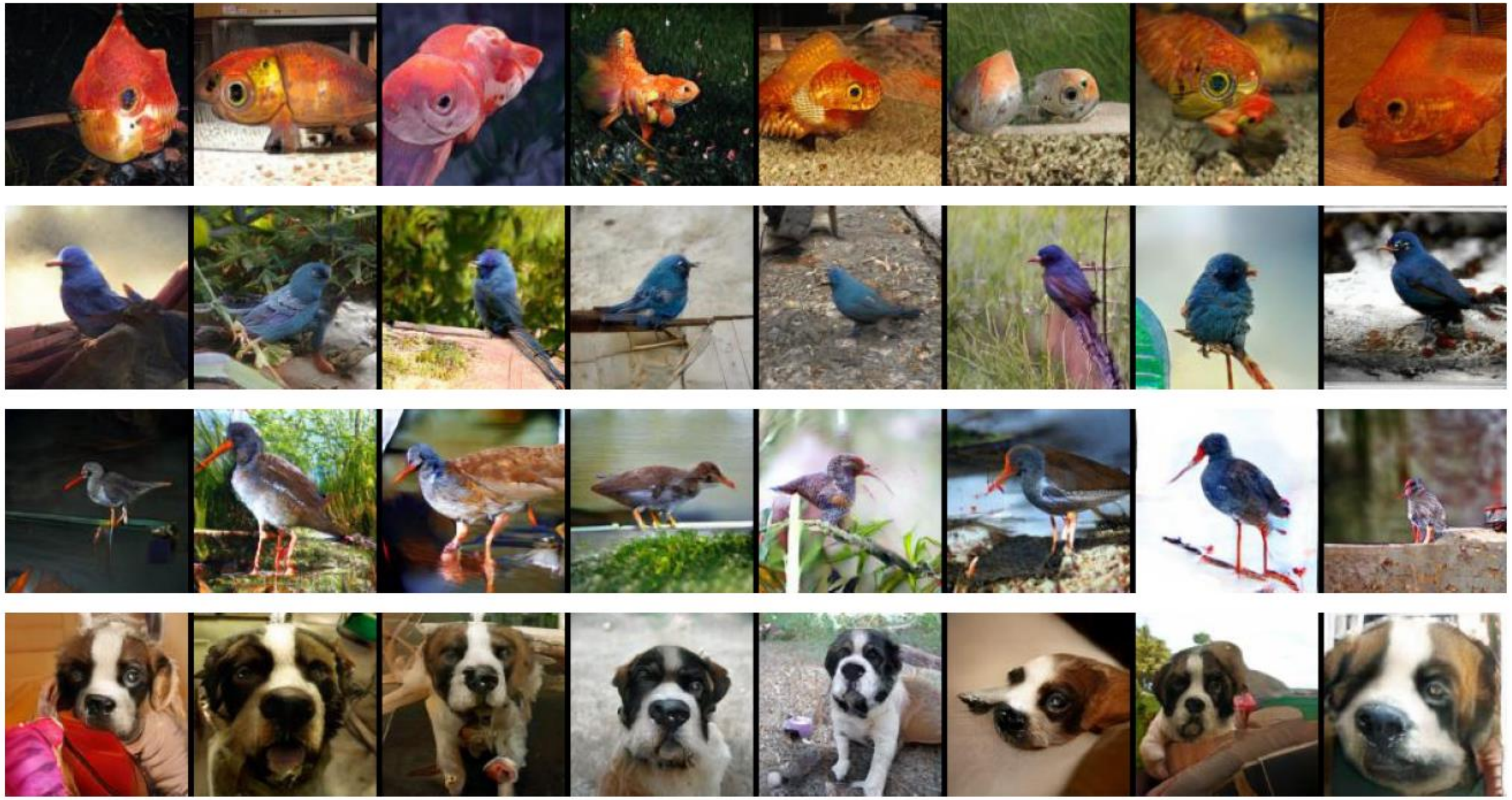
Генерация изображений с помощью генеративно-состязательных сетей (Generative Adversarial Network, GAN)

goldfish

indigo bunting

redshank

saint bernard

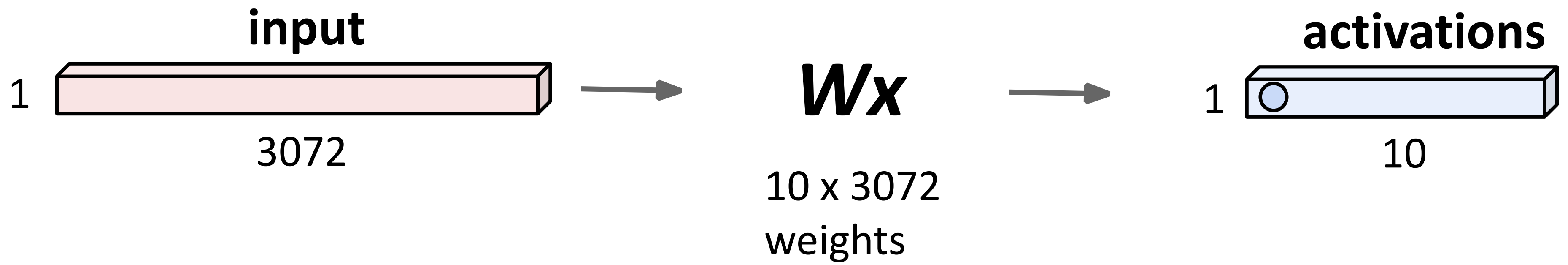


Demo: https://www.youtube.com/watch?v=XOxxPcy5Gr4

Han Zhang, et al, Self-Attention Generative Adversarial Networks, 2018

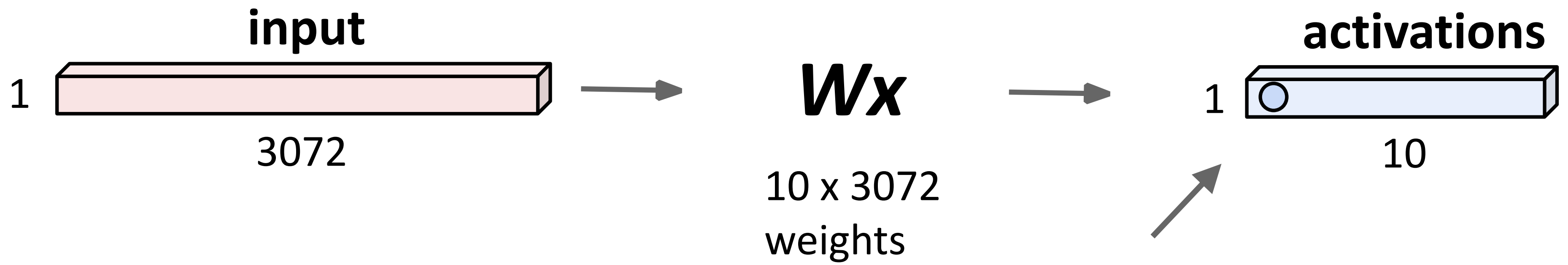
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

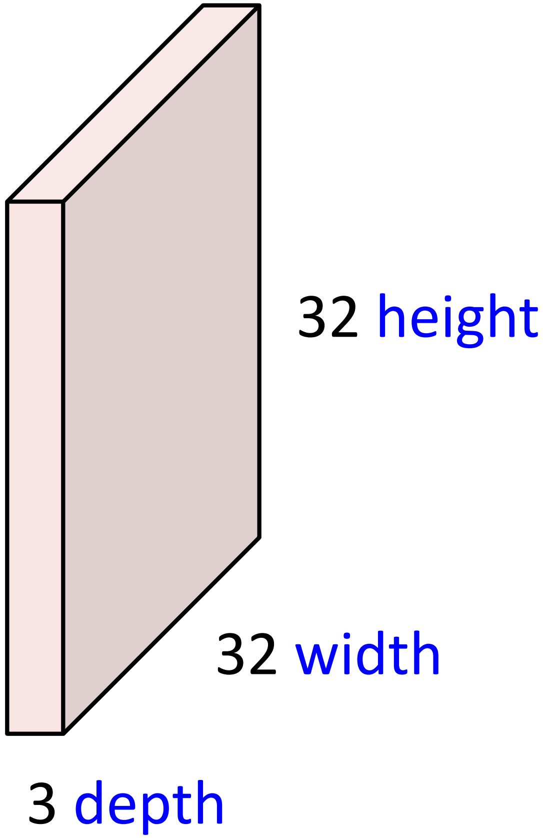
32x32x3 image -> stretch to 3072 x 1



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

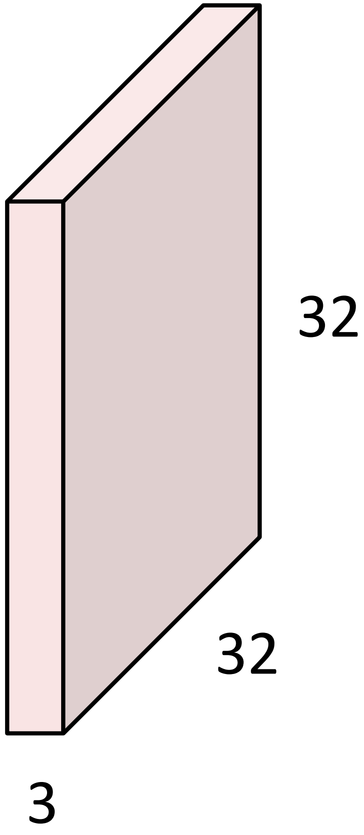
Convolutional Layer

32x32x3 image -> preserve spatial structure

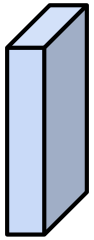


Convolutional Layer

32x32x3 image



5x5x3 filter

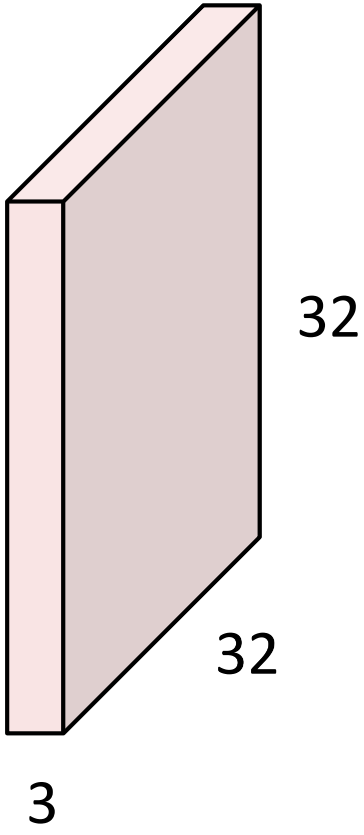


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

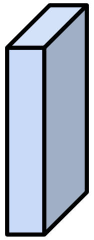
Convolutional Layer

Filters always extend the full depth of the input volume

32x32x3 image

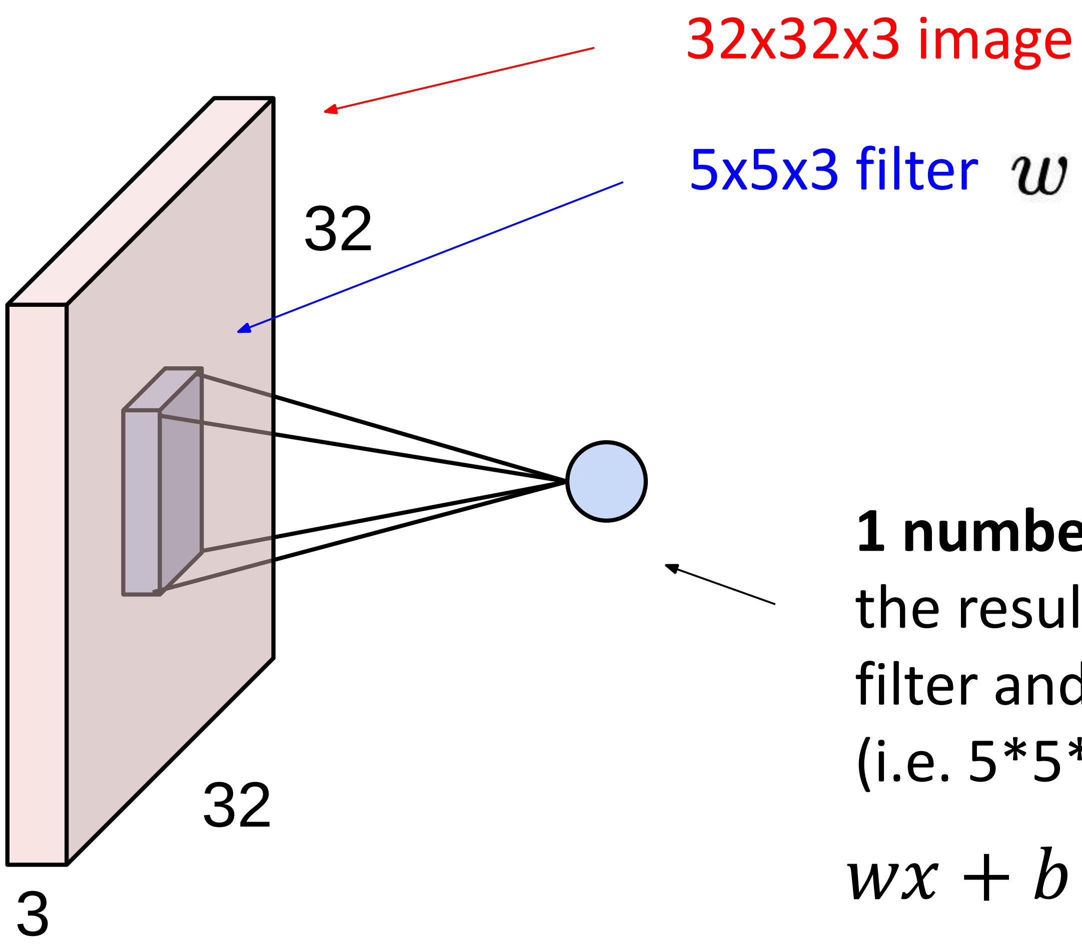


5x5x3 filter



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

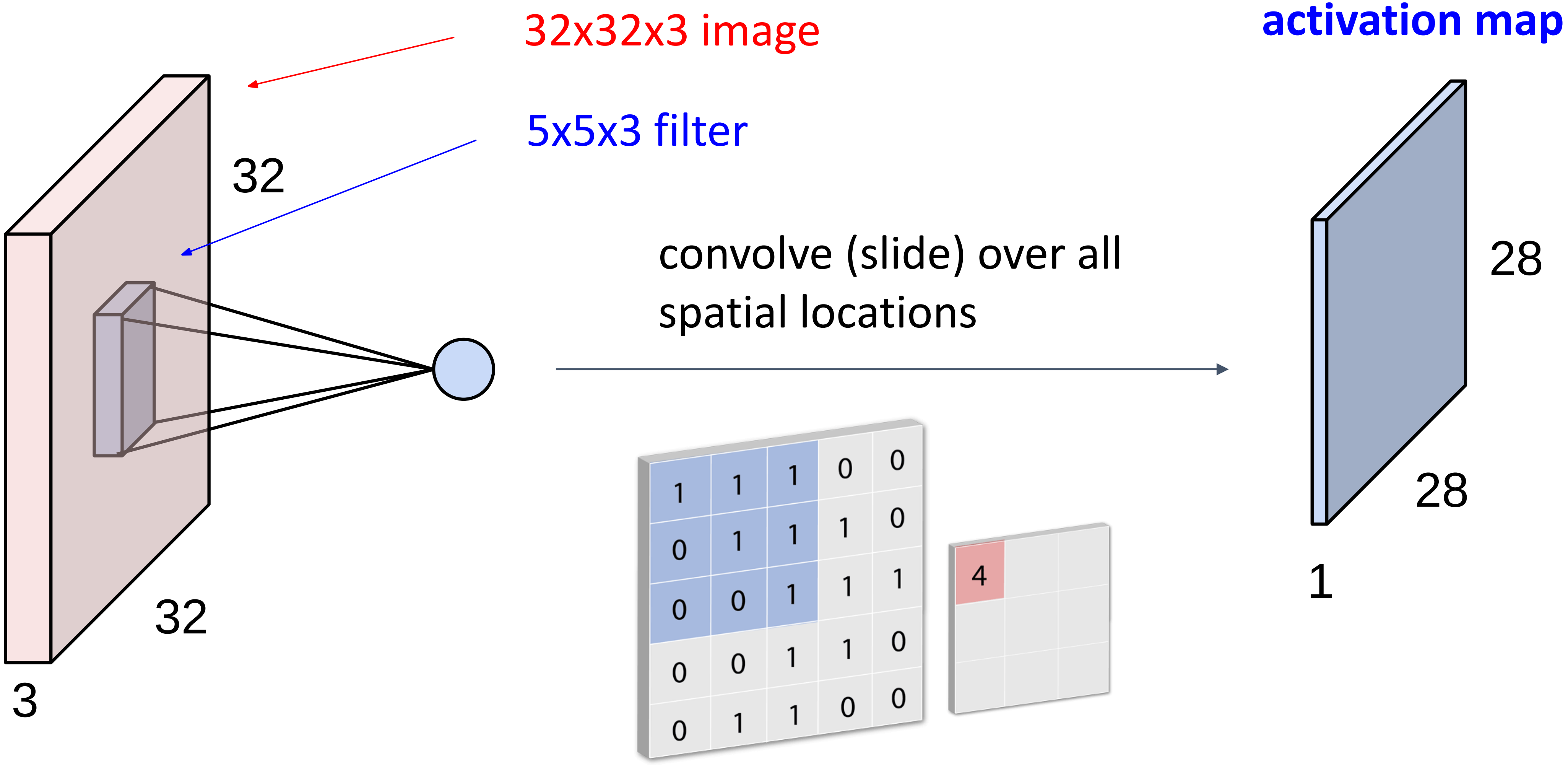
Convolutional Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$wx + b$$

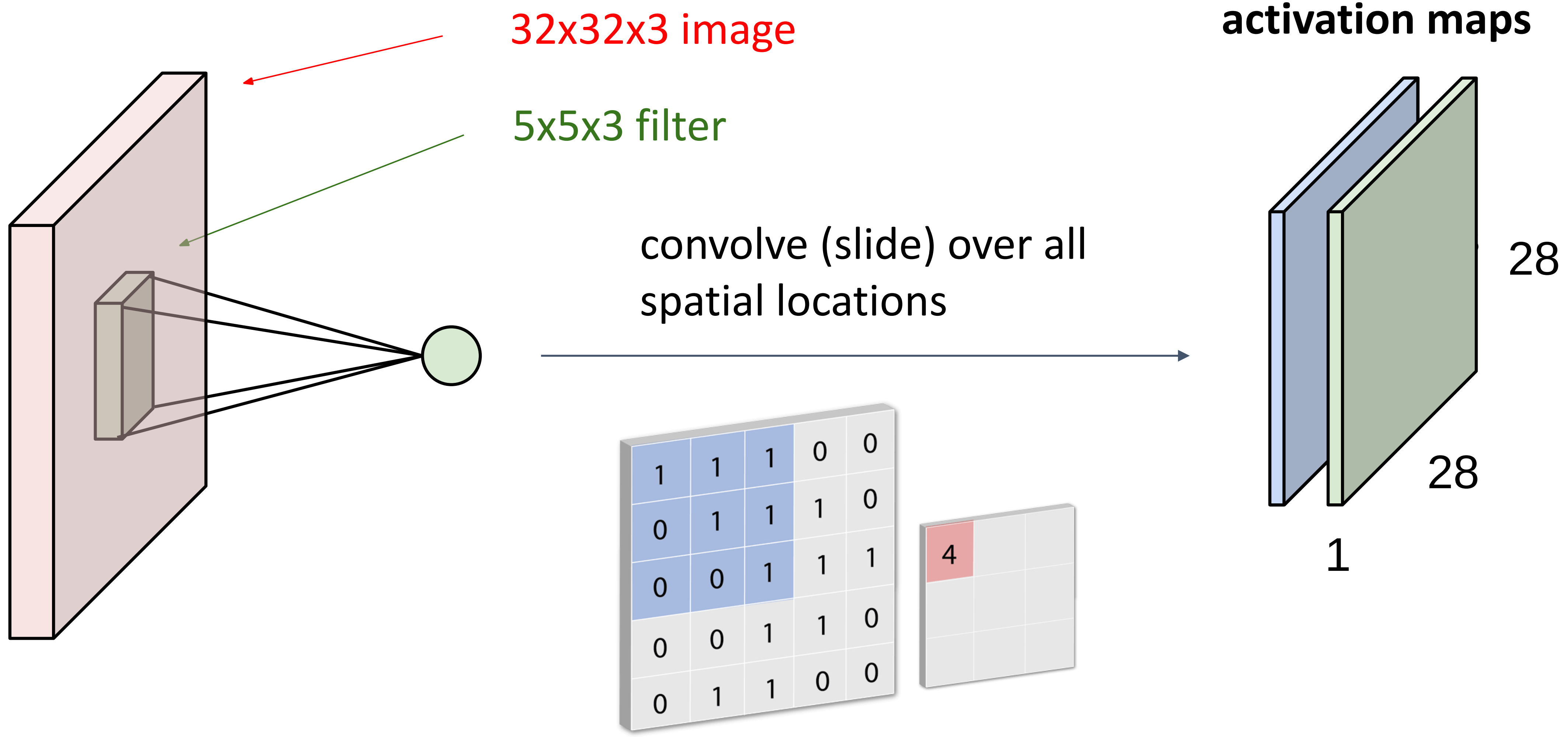
Convolutional Layer



example: image 5x5, filter 3x3

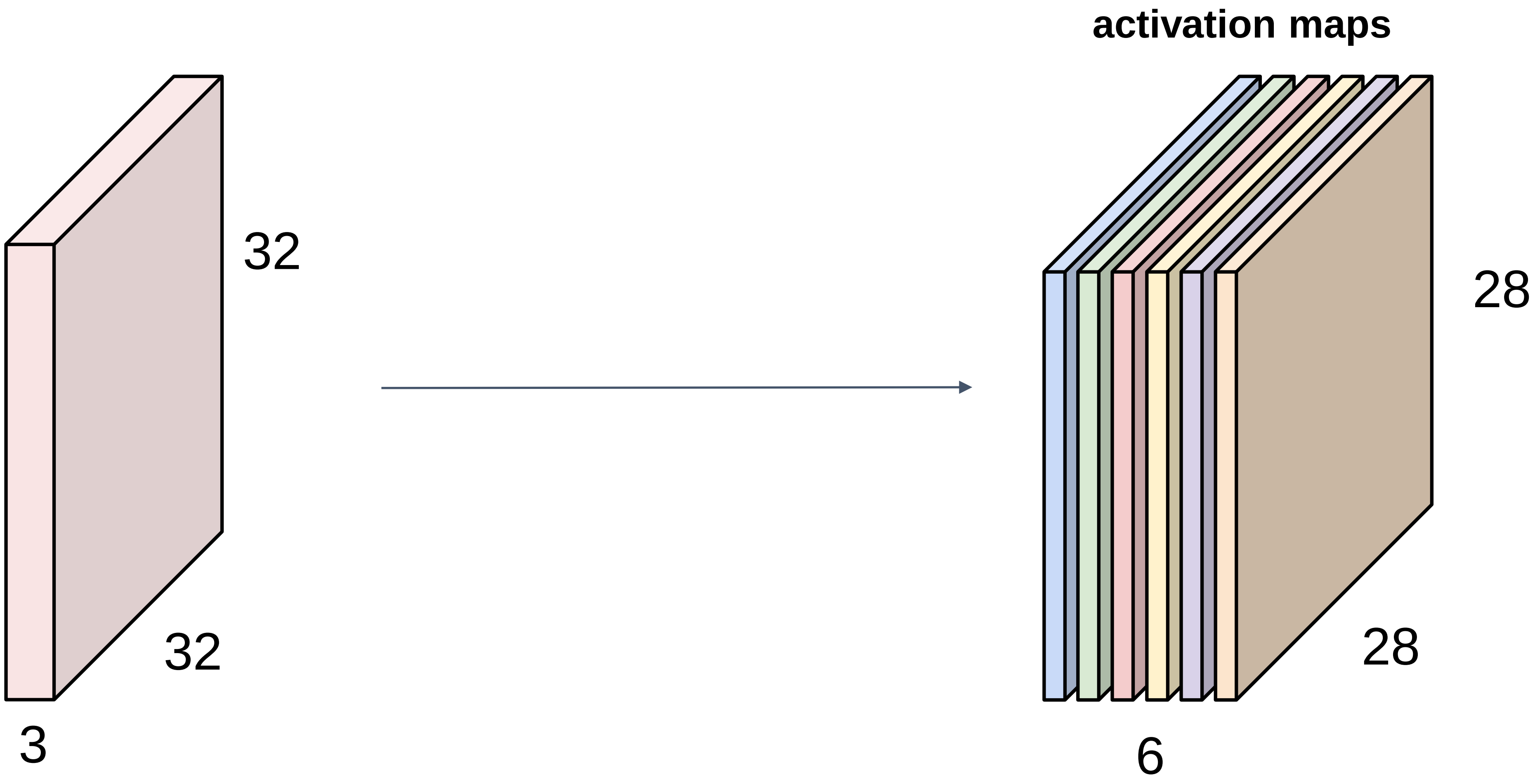
Convolutional Layer

consider a second, green filter



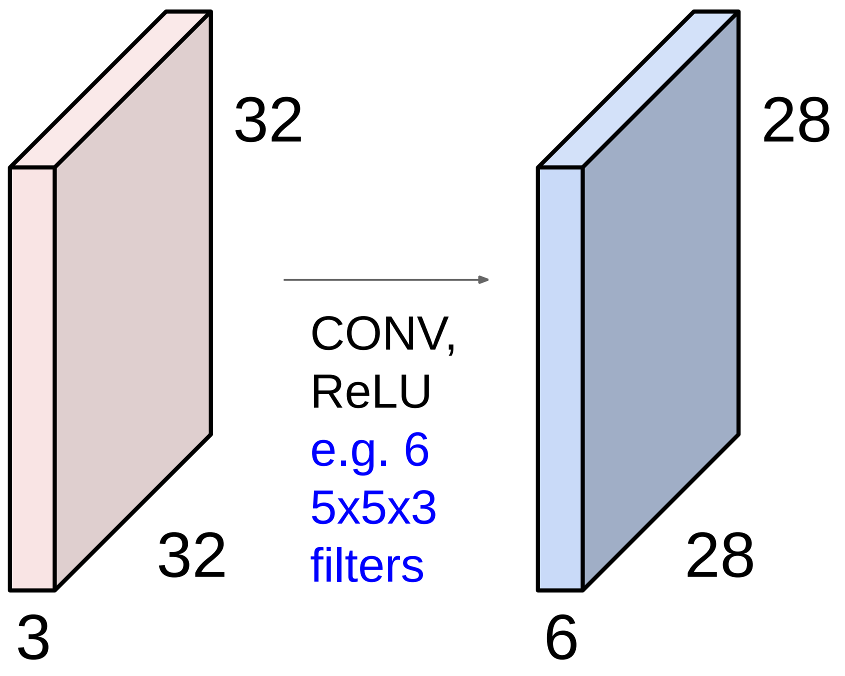
example: image 5x5, filter 3x3

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

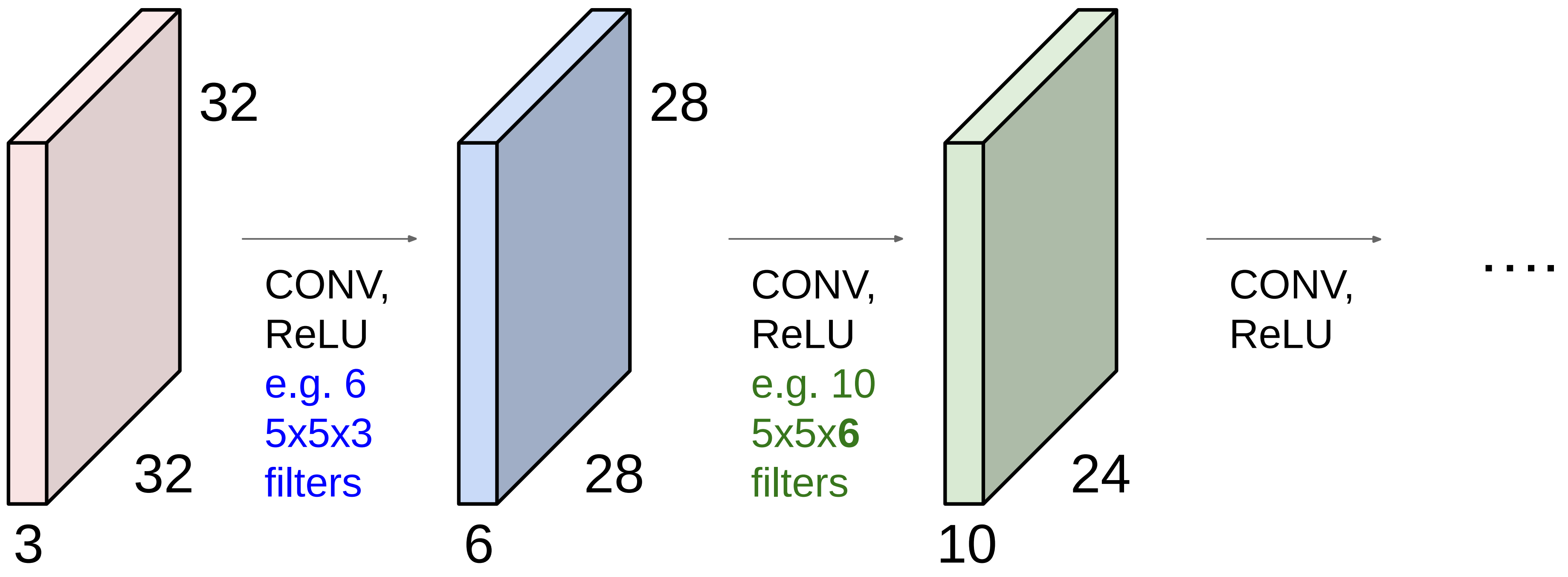


We stack these up to get a "new image" of size 28x28x6!

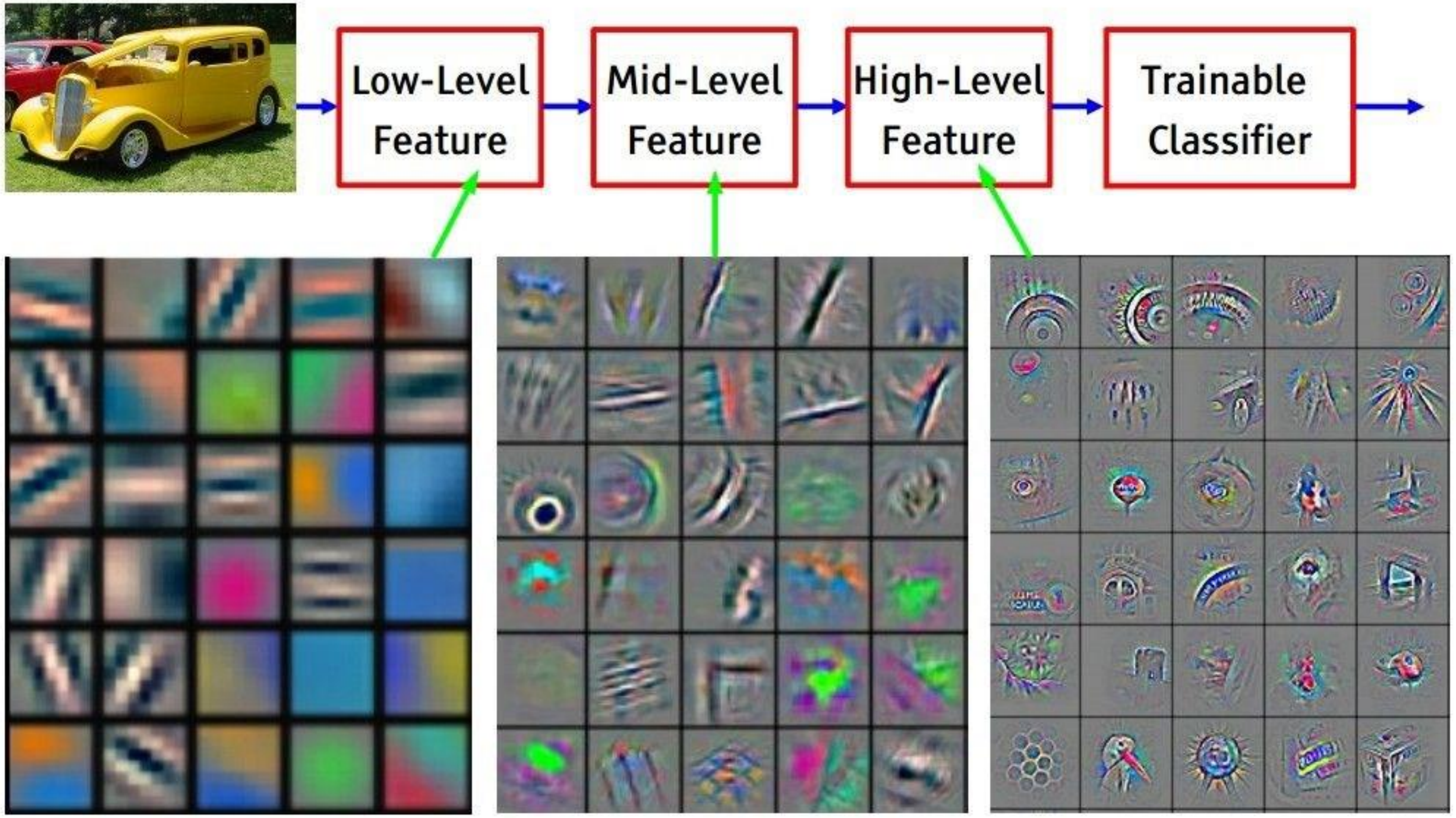
ConvNet is a sequence of Convolution Layers, interspersed with activation functions



ConvNet is a sequence of Convolution Layers, interspersed with activation functions

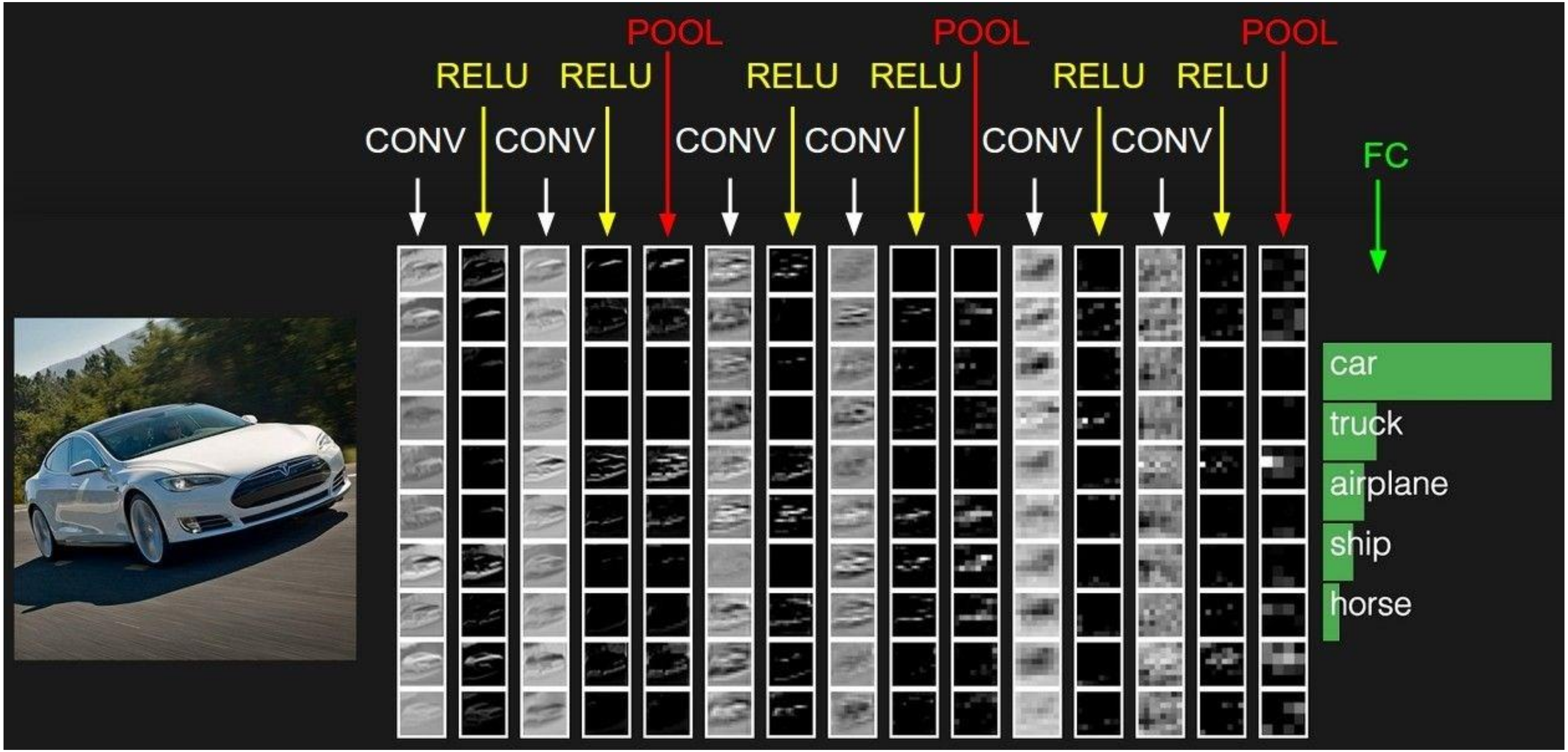


Визуализация признаков нейронной сети

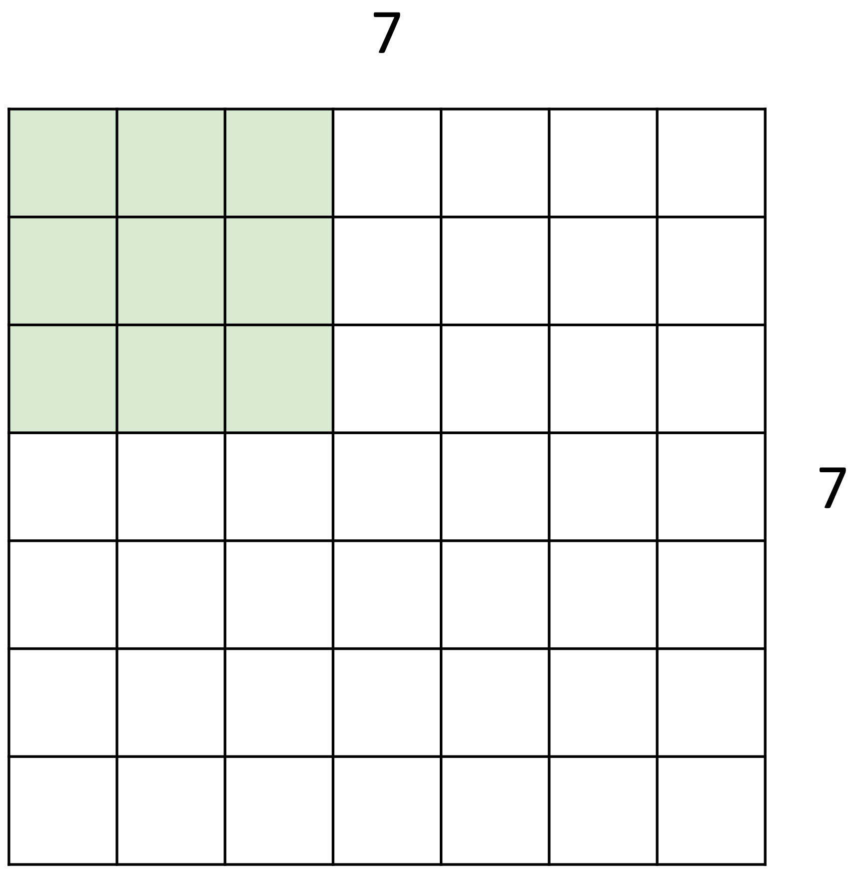


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Сверточная нейронная сеть

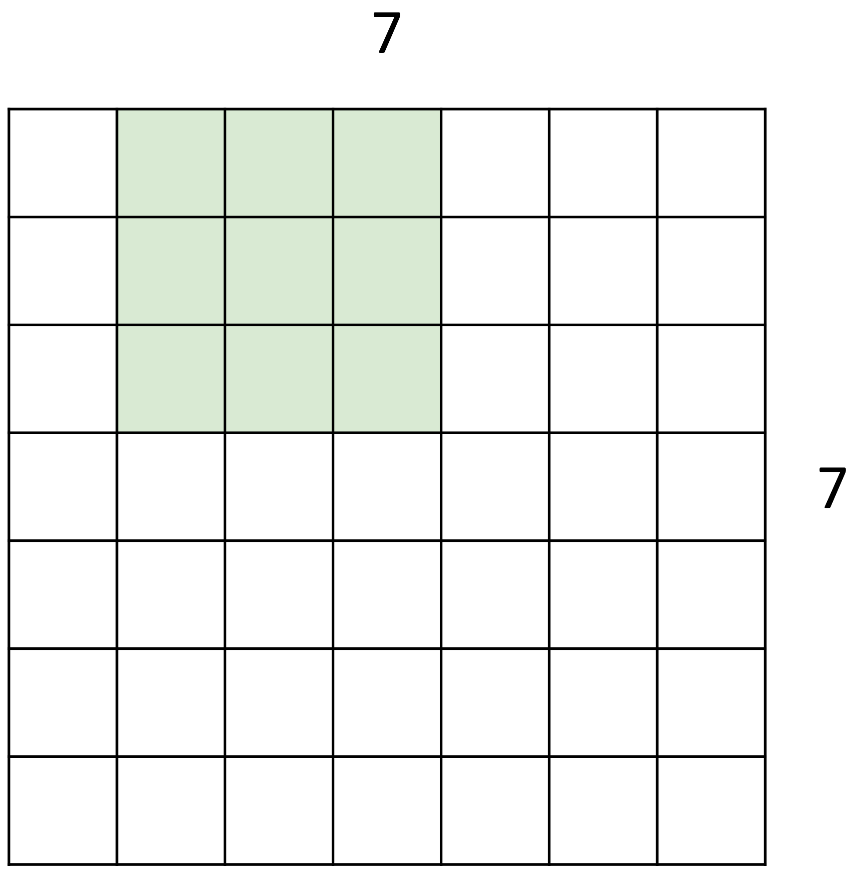


A closer look at spatial dimensions:



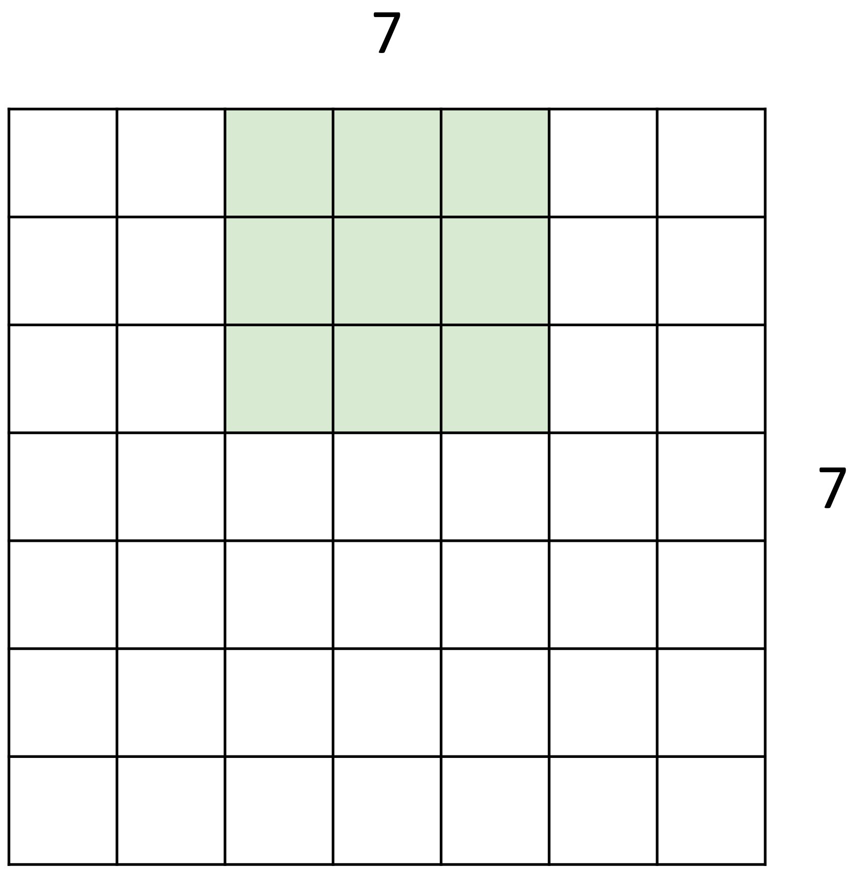
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



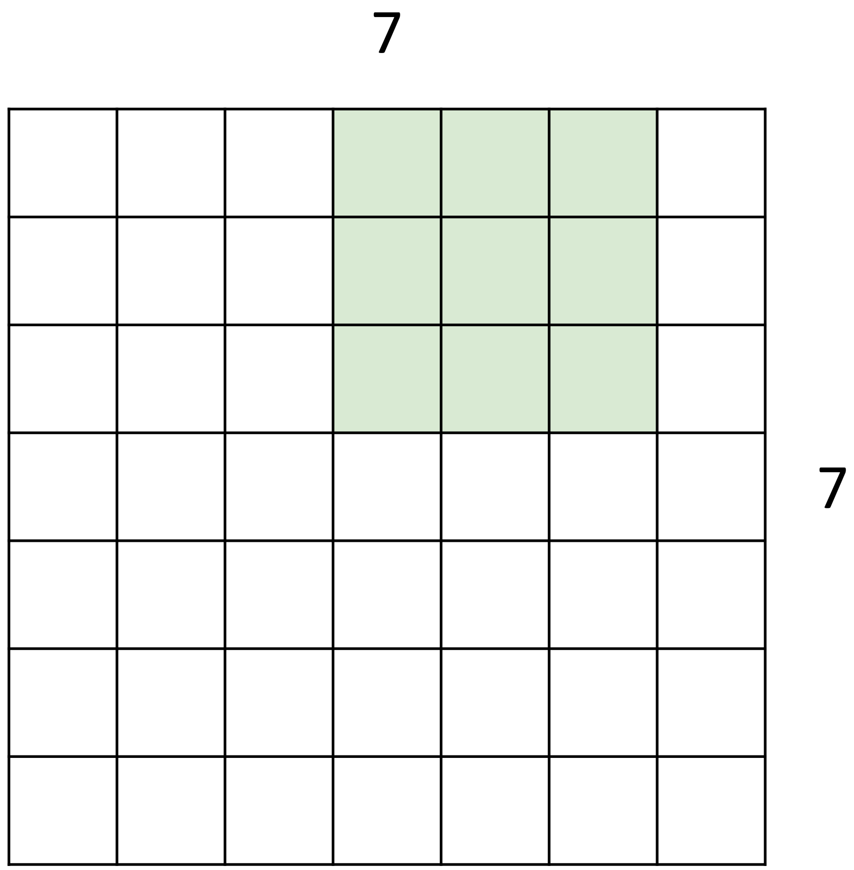
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



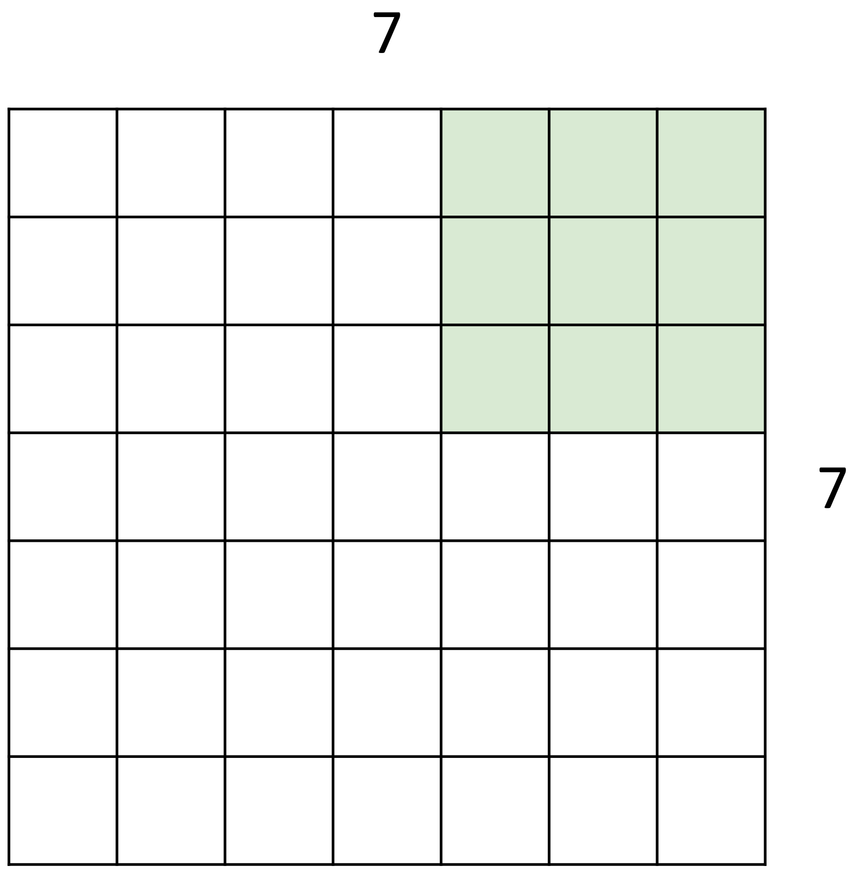
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

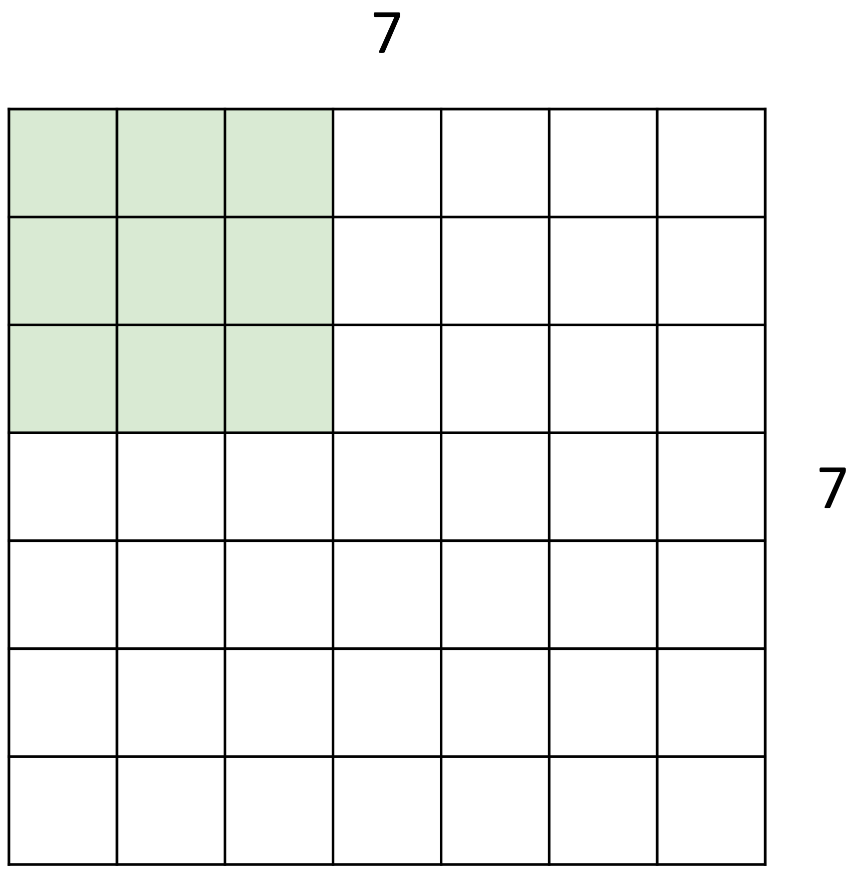
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

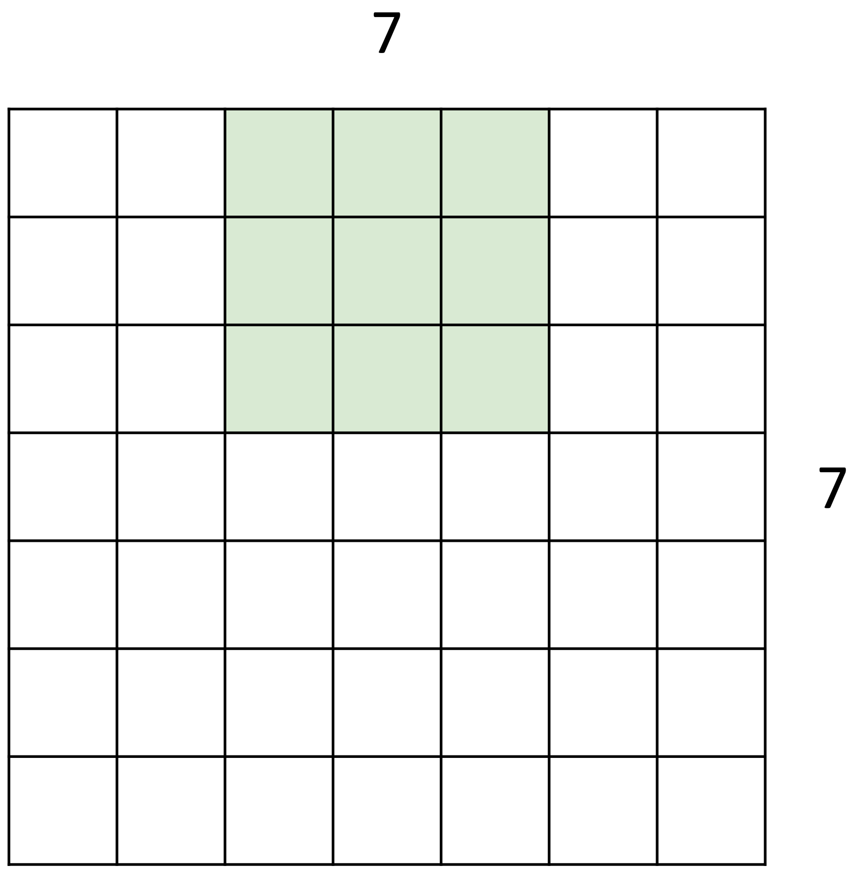
=> 5x5 output

A closer look at spatial dimensions:



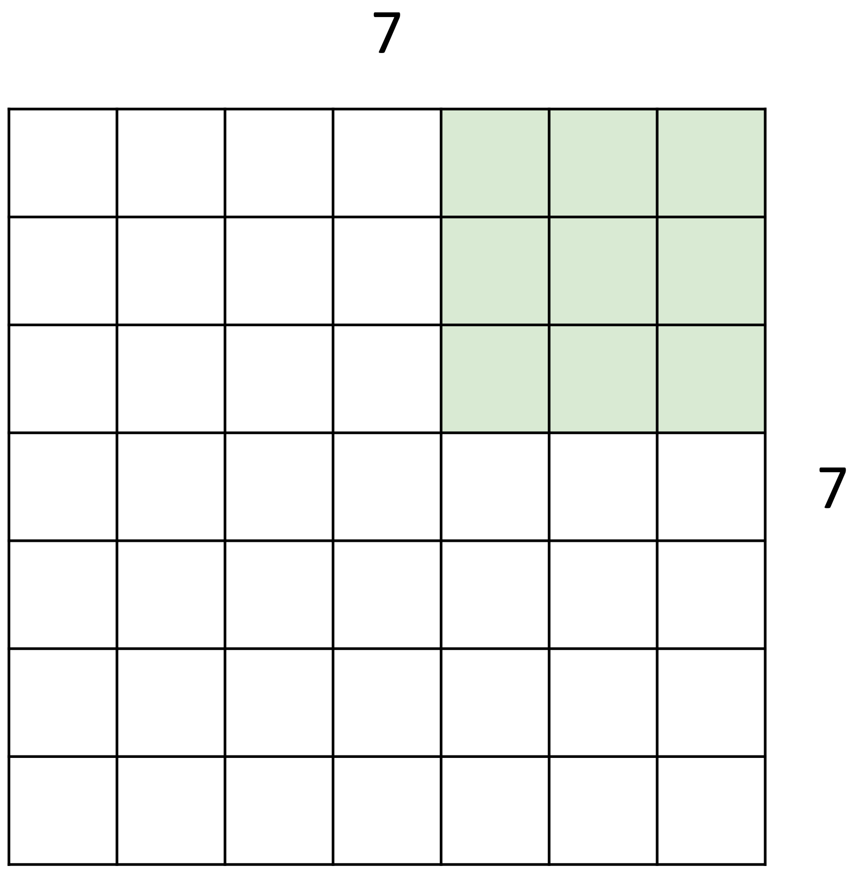
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



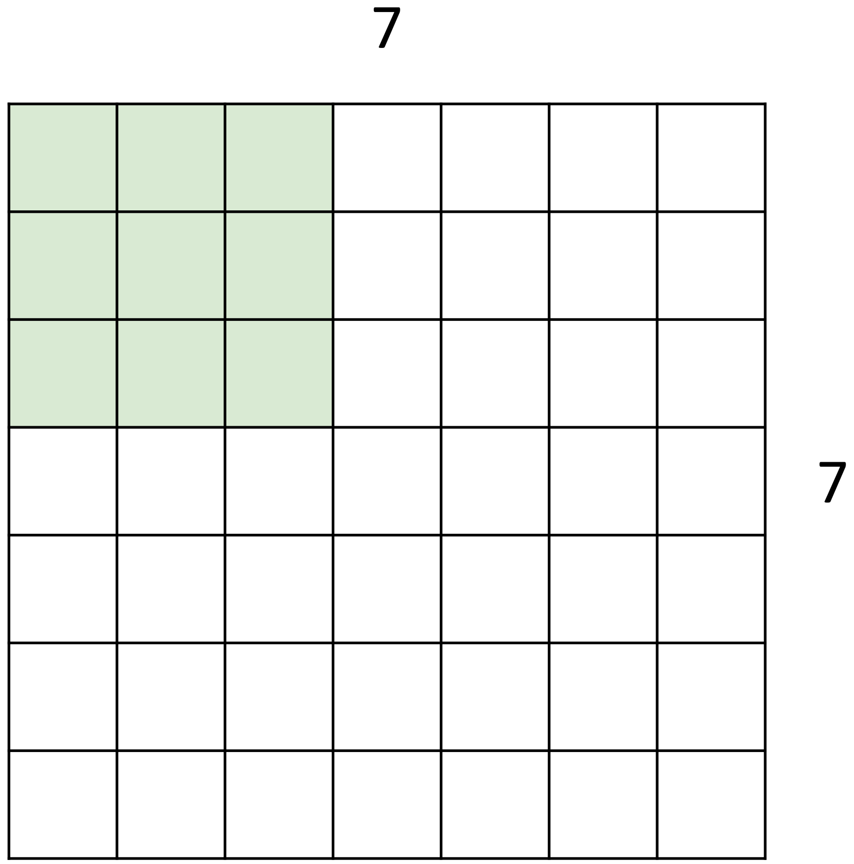
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



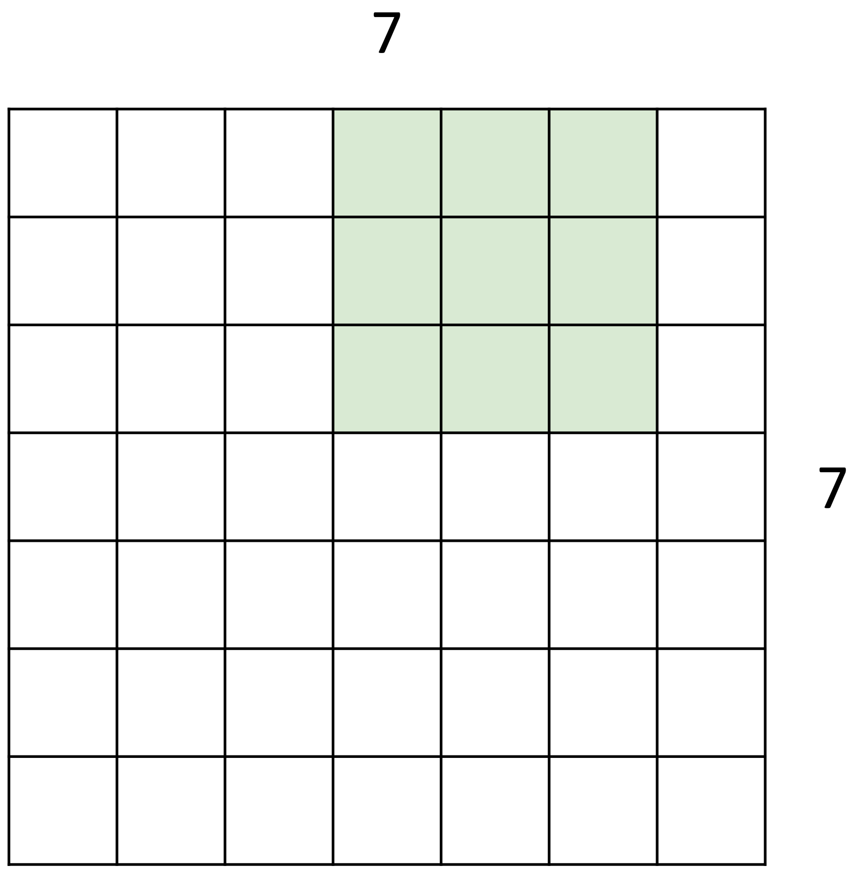
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



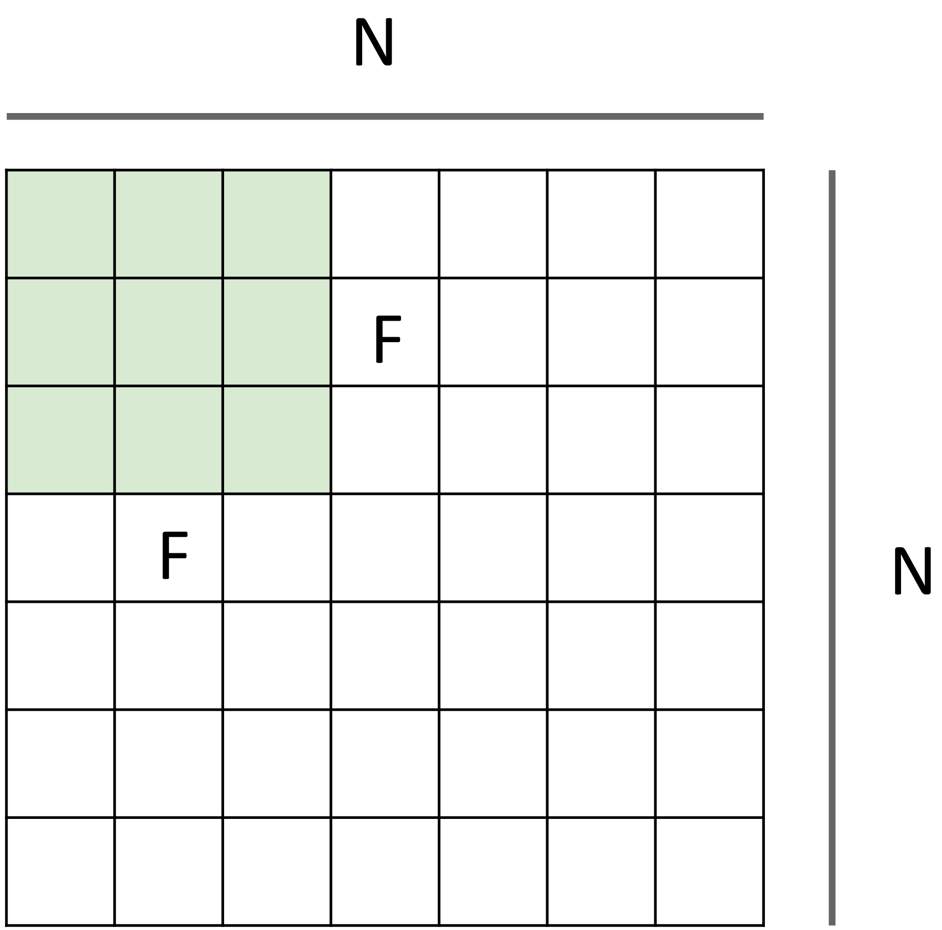
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

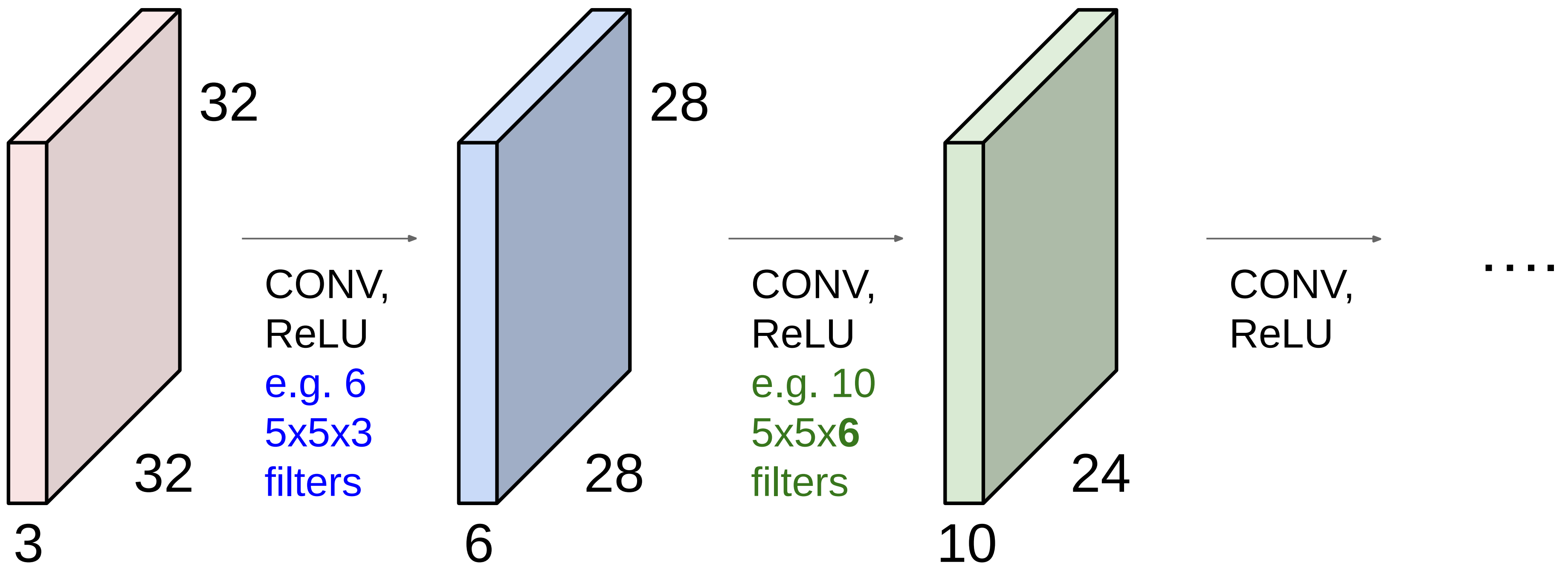


Output size:
(N - F) / stride + 1

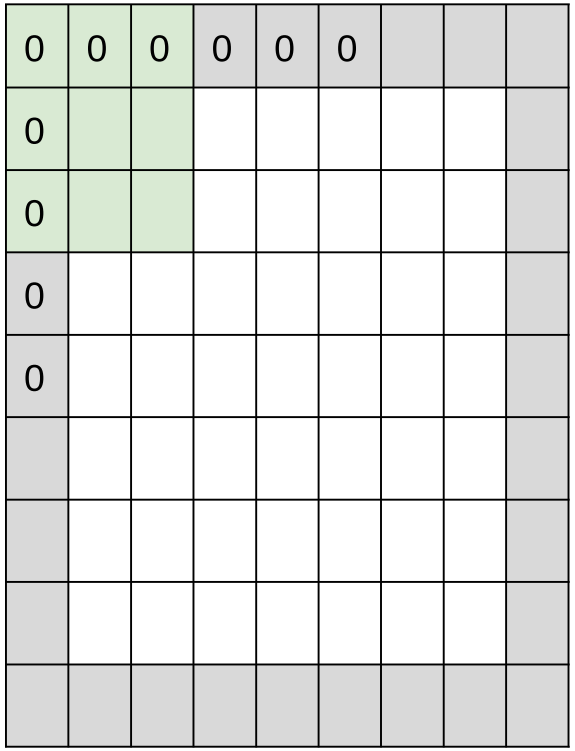
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



In practice: Common to zero pad the border



e.g. input 7x7

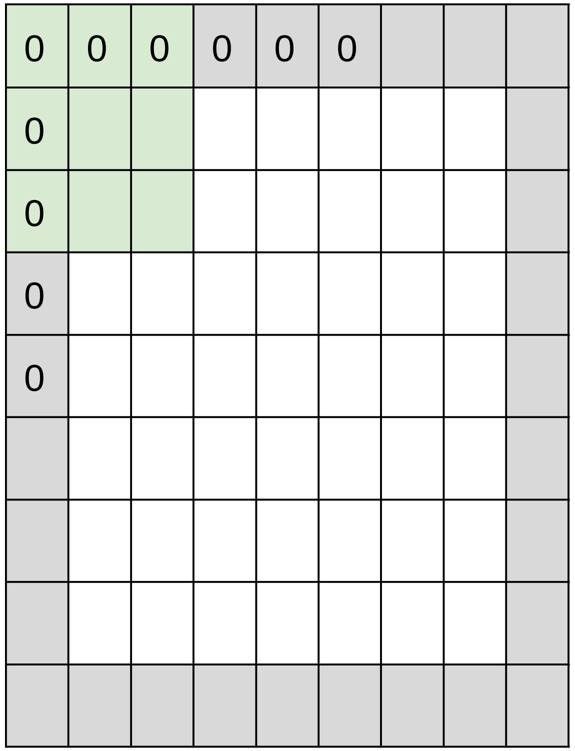
3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

(recall:)

(N - F) / stride + 1

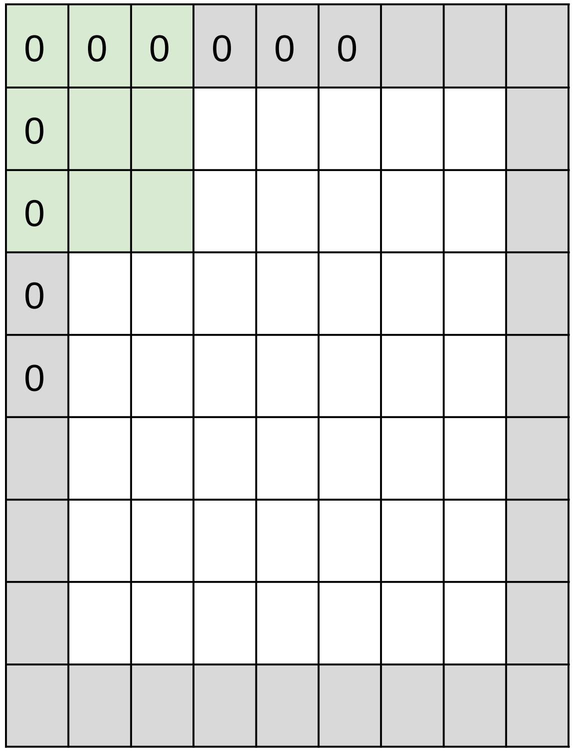
In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

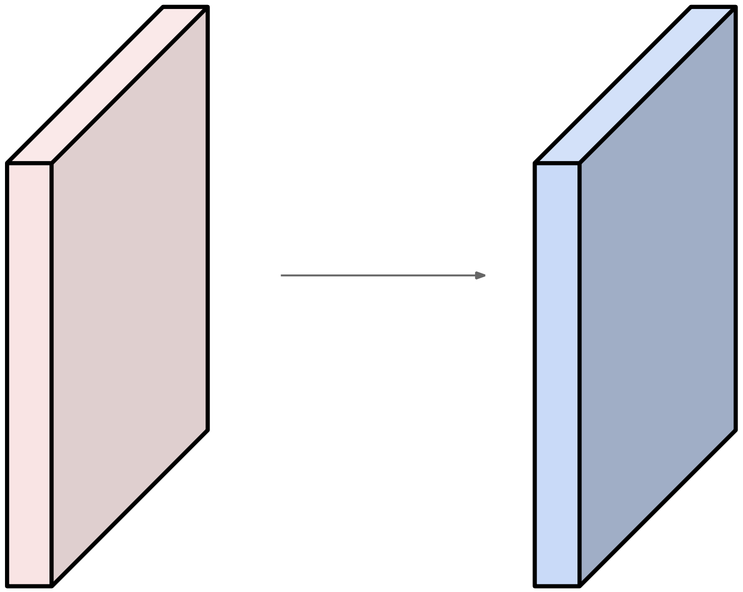
7x7 output!

in general, common to see CONV 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

Examples time:

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

Output volume size: ?



Examples time:

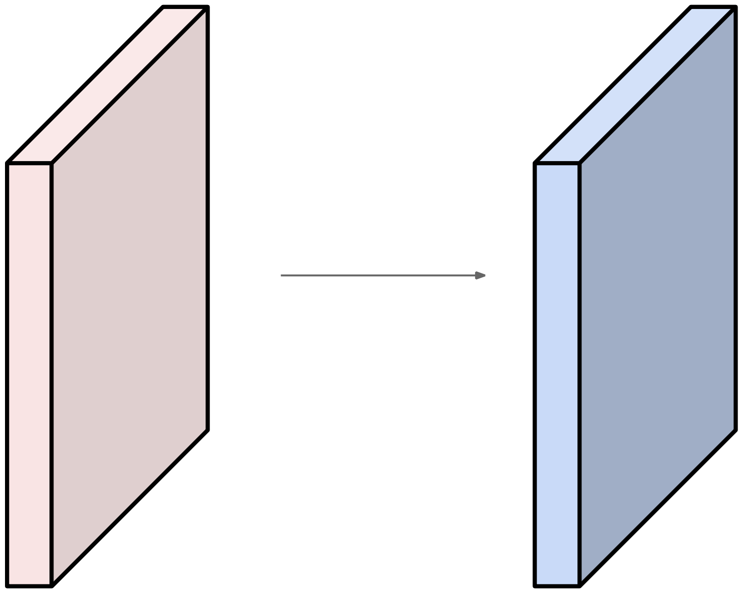
Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size:

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

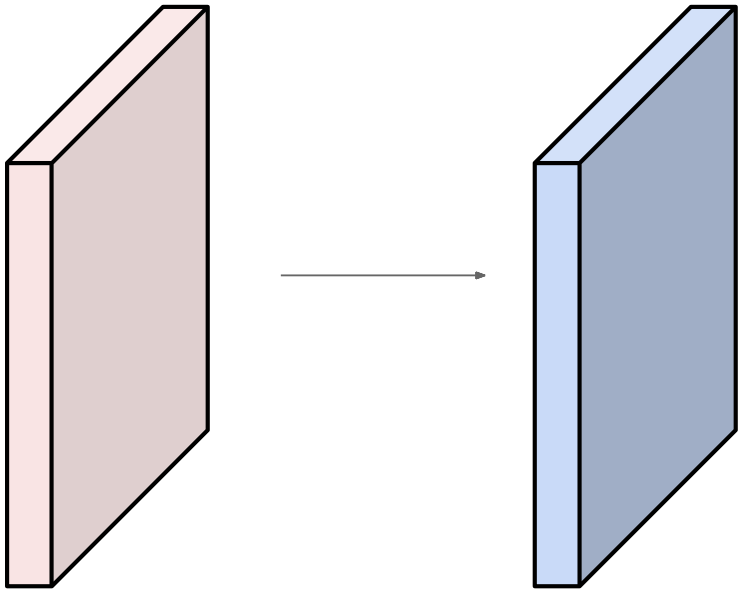
32x32x10



Examples time:

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

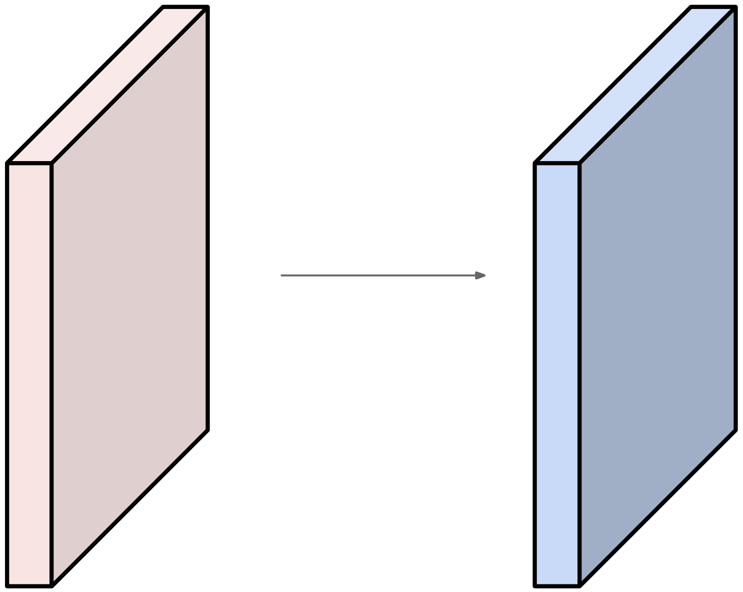
Number of parameters in this layer?



Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Summary. To summarize, the Conv Layer:

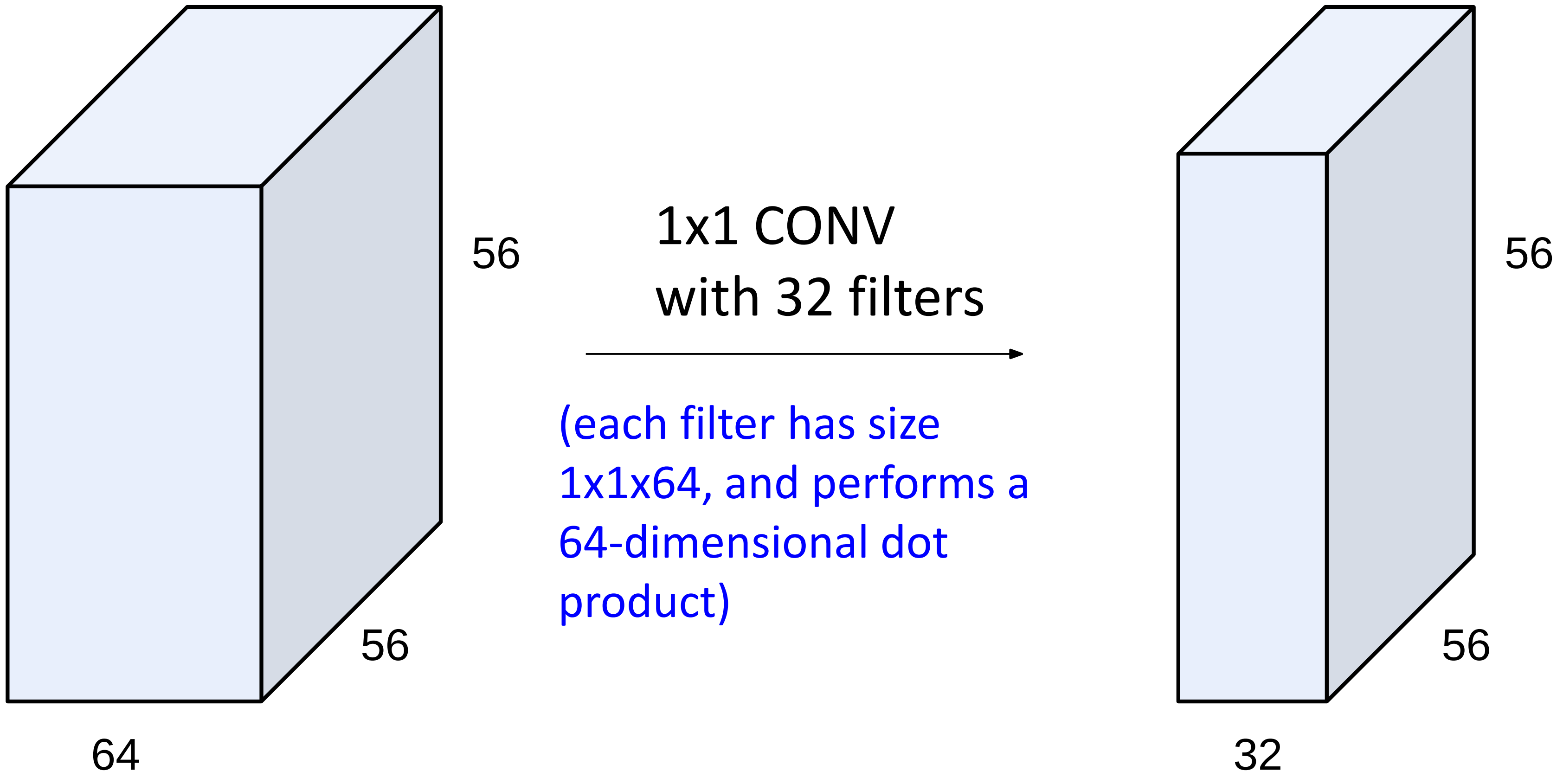
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

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

1x1 convolution layers make perfect sense



Example: CONV layer in Caffe

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Example: CONV layer in TensorFlow

```
conv2d(
    inputs,
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format='channels_last',
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer=None,
    bias_initializer=tf.zeros_initializer(),
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    trainable=True,
    name=None,
    reuse=None
)
```

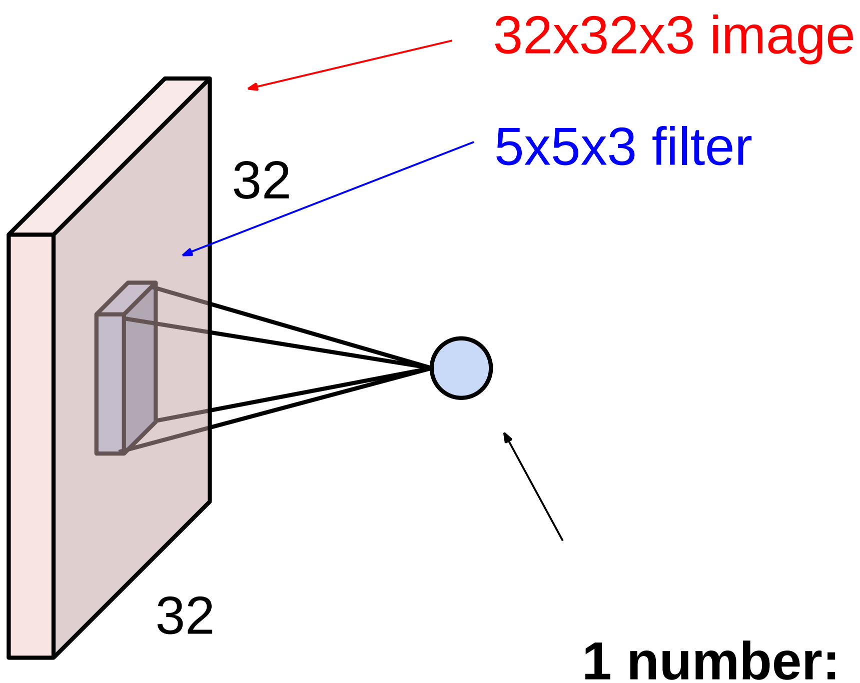
```
# Input Layer
input_layer = tf.reshape(features, [-1, 28, 28, 1])

# Convolutional Layer #1
conv1 = tf.layers.conv2d(
    inputs=input_layer,
    filters=32,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
```

Summary. To summarize, the Conv Layer:

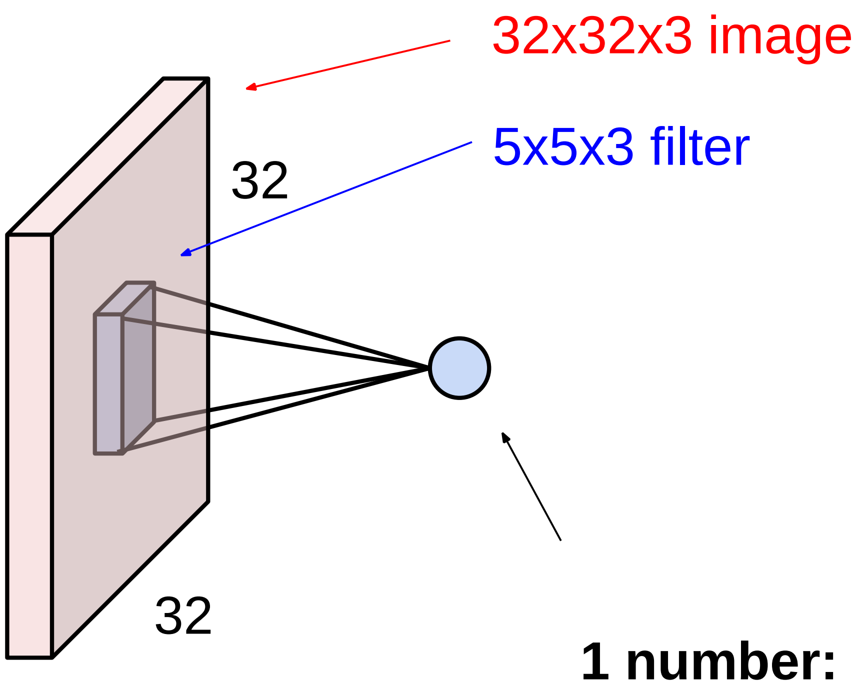
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

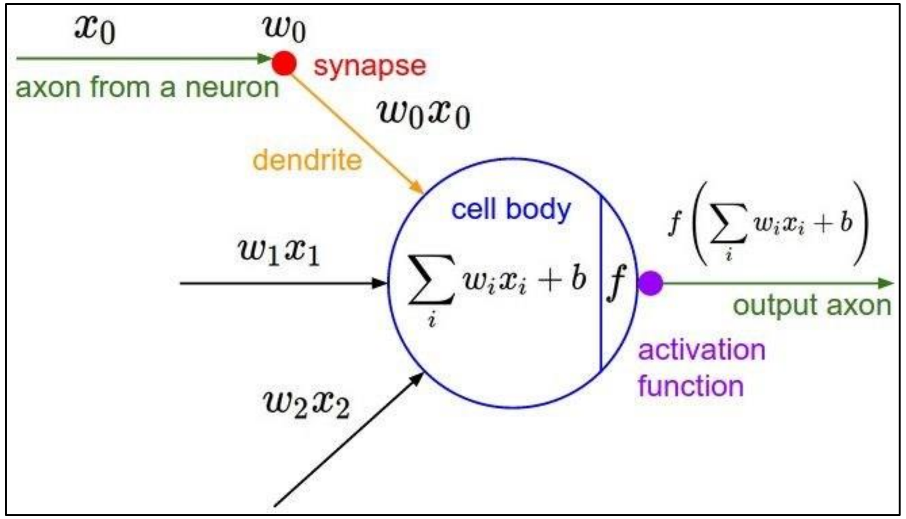
The brain/neuron view of CONV Layer



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)

The brain/neuron view of CONV Layer

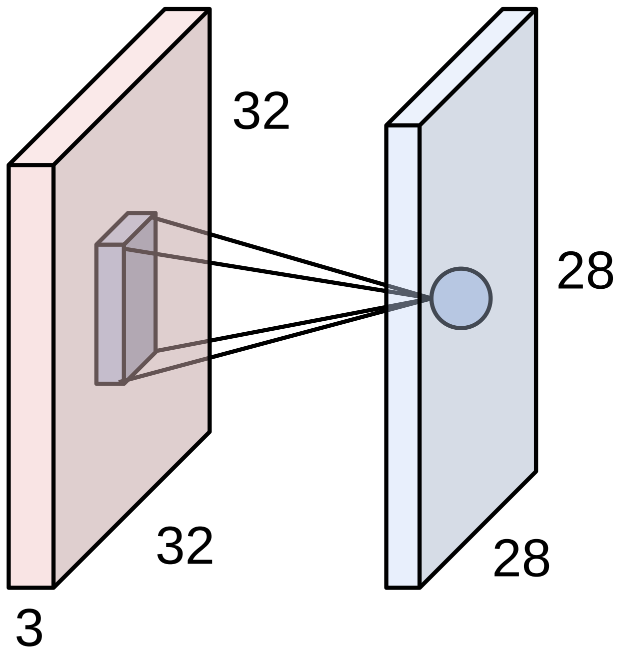


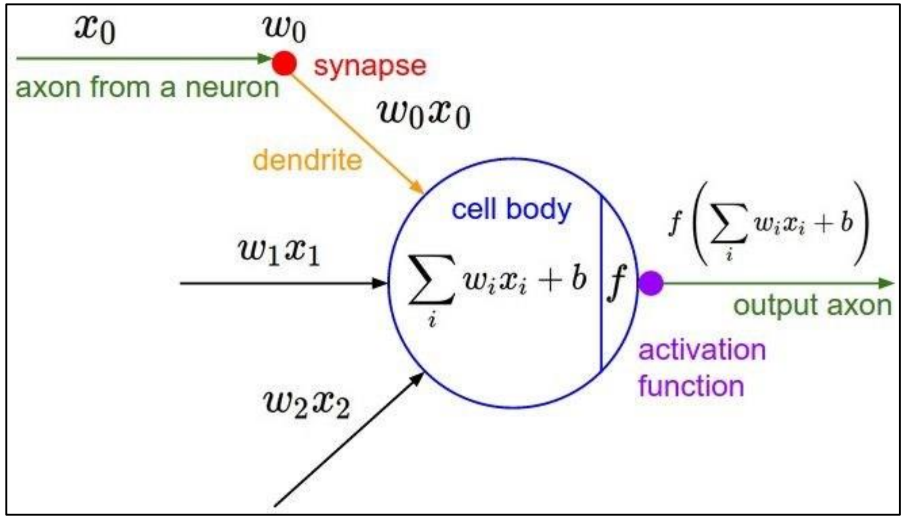


It’s just a neuron with local connectivity...

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)

The brain/neuron view of CONV Layer

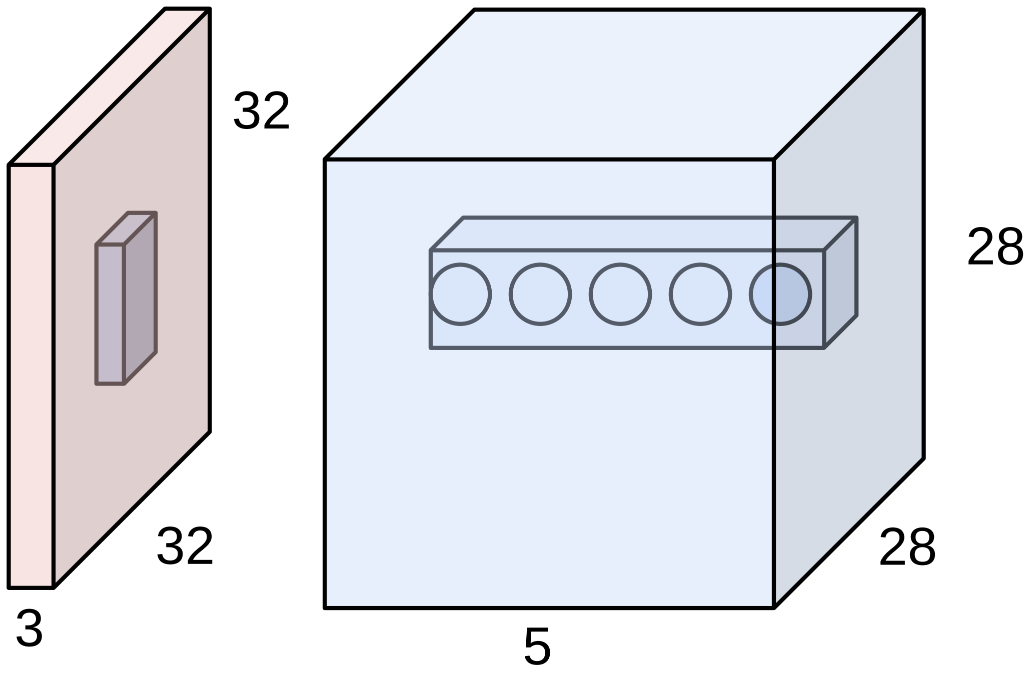


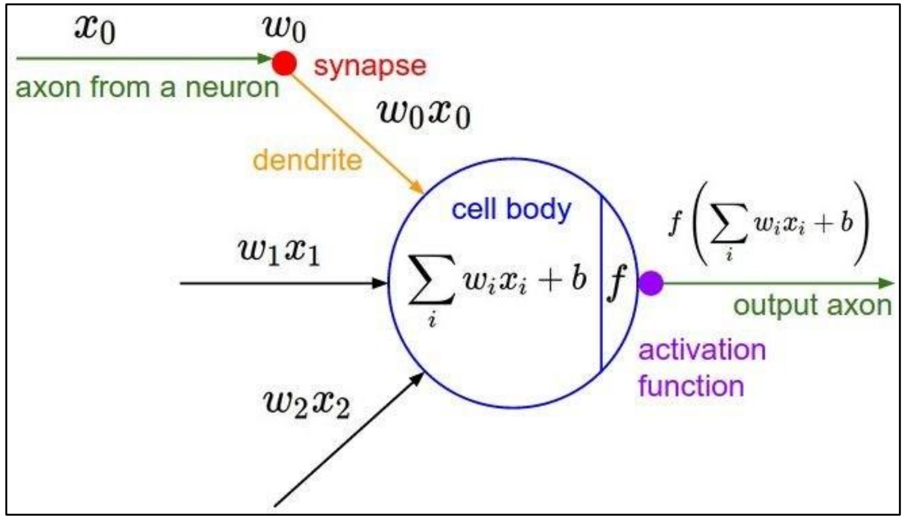


An activation map is a 28x28 sheet of neuron outputs:
1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron"

The brain/neuron view of CONV Layer





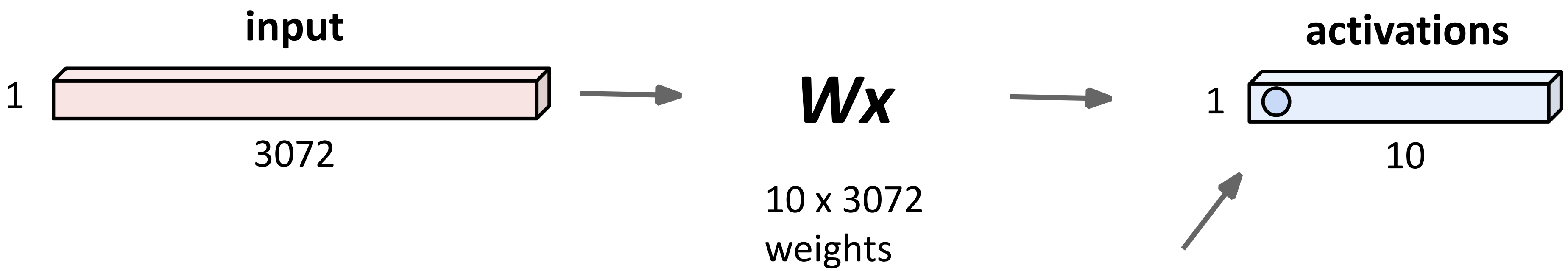
E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same 5 region in the input volume

Reminder: Fully Connected Layer

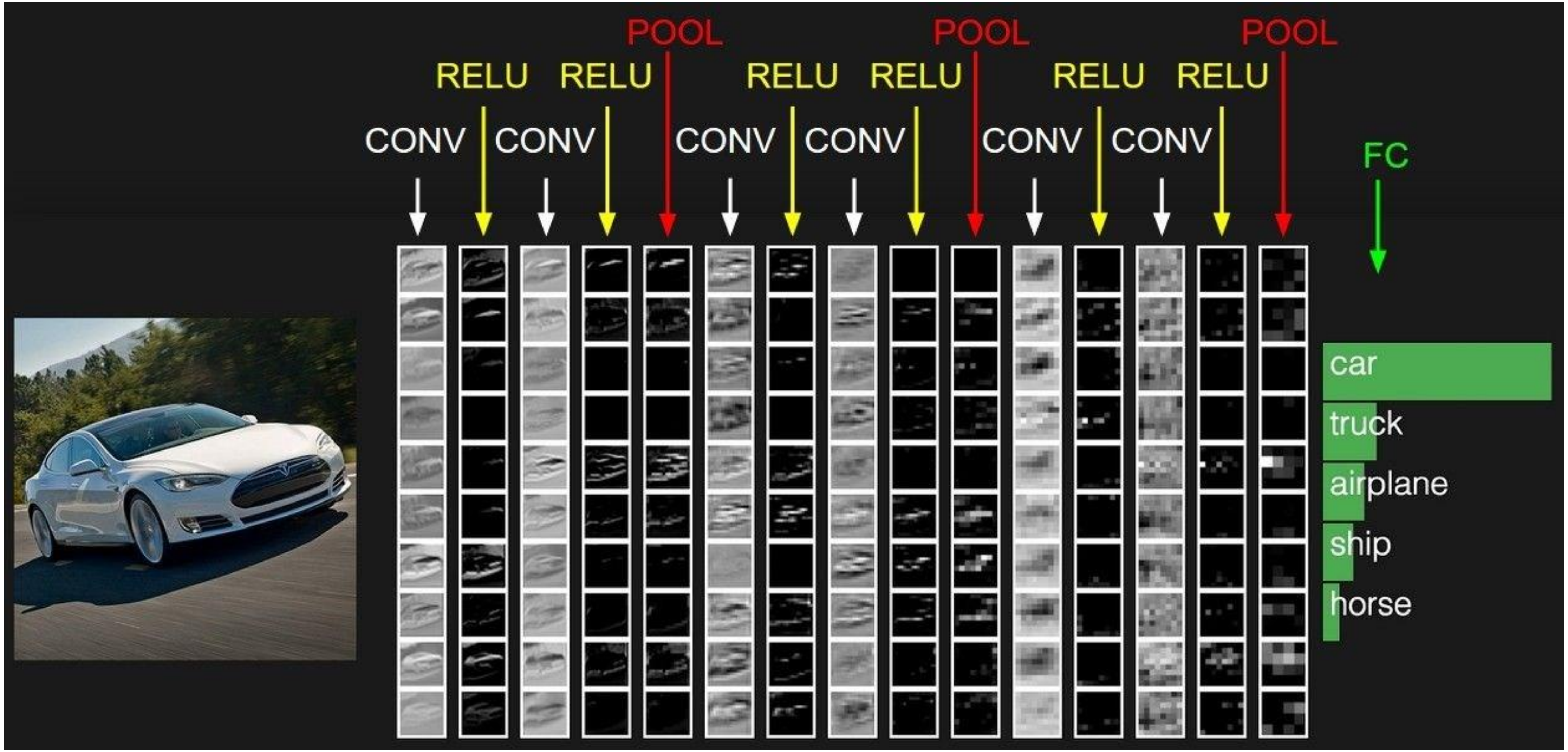
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



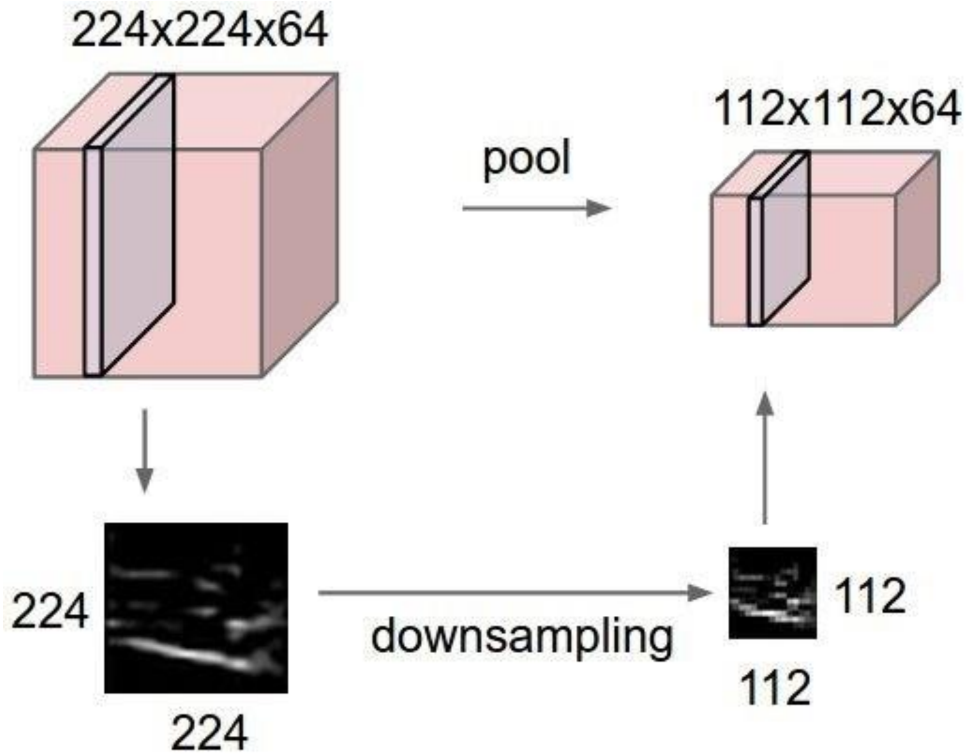
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Сверточная нейронная сеть: **Pooling**

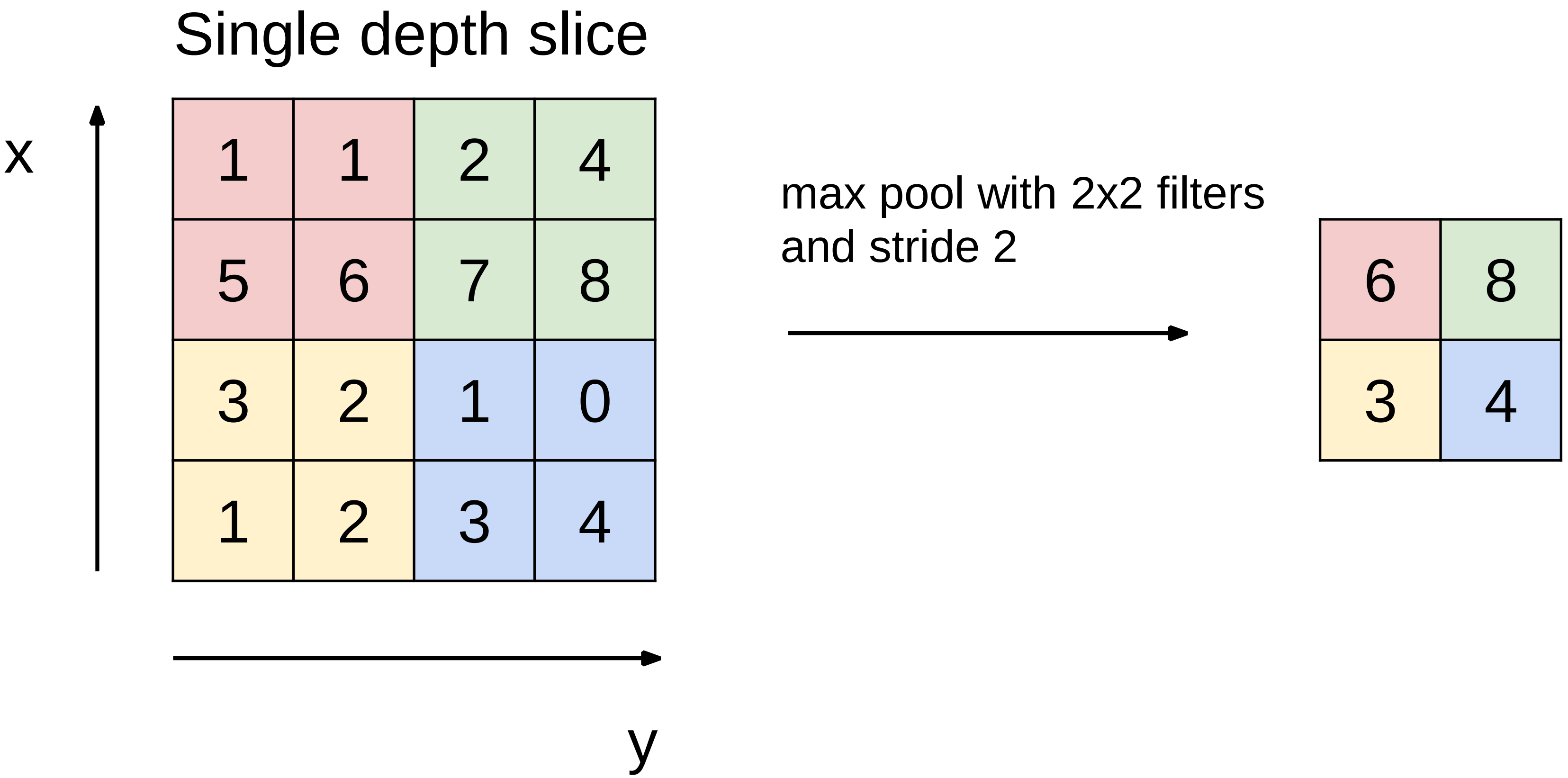


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX 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

Pooling

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Pooling

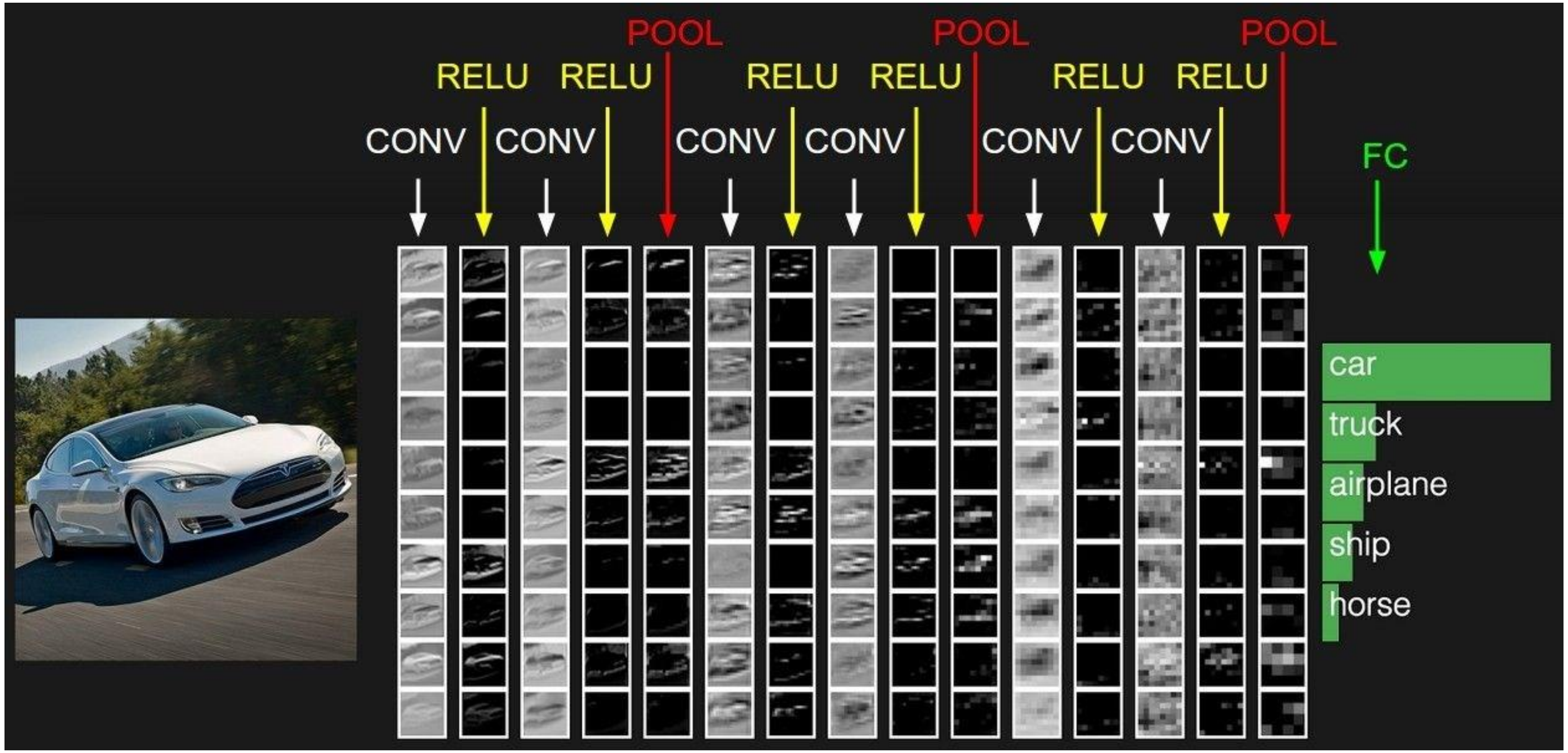
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

Сверточная нейронная сеть:
Fully Connected Layer (FC layer)

Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



[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

Резюме

- Сверточная нейронная сеть:
  CONV + POOL + FC слои
- Типичная архитектура классической сверточной сети
  [(CONV-RELU)*N-POOL]*M-(FC-RELU)*K,SOFTMAX
  где N до ≈5, M большое до ≈15, 0 <= K <= 2.

  современные нейронные сети ResNet, DenseNet имеют более сложные архитектуры

В следующий раз

- Активационные функции
- Предобработка изображений
- Инициализация весов
- Подбор гиперпараметров