Lecture 07: Convolutional Neural Networks

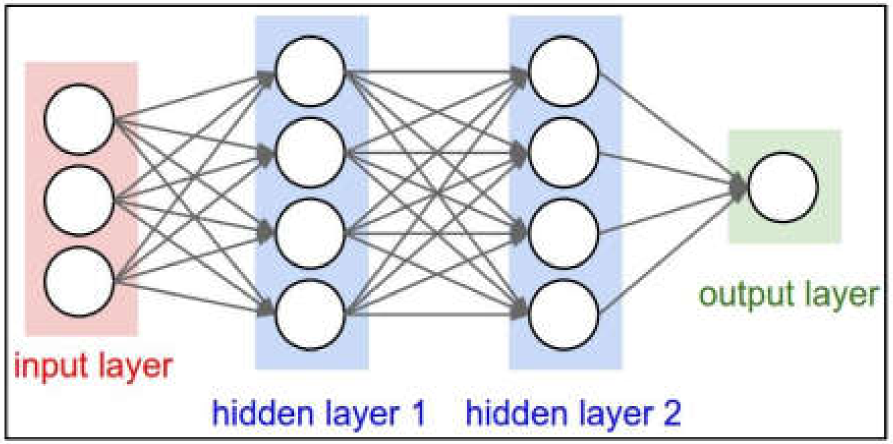
박사과정 김성빈 chengbinjin@inha.edu,
지도교수 김학일 교수 hikim@inha.ac.kr
인하대학교 컴퓨터비전 연구실

Recall From the Last Class

Mini-batch SGD

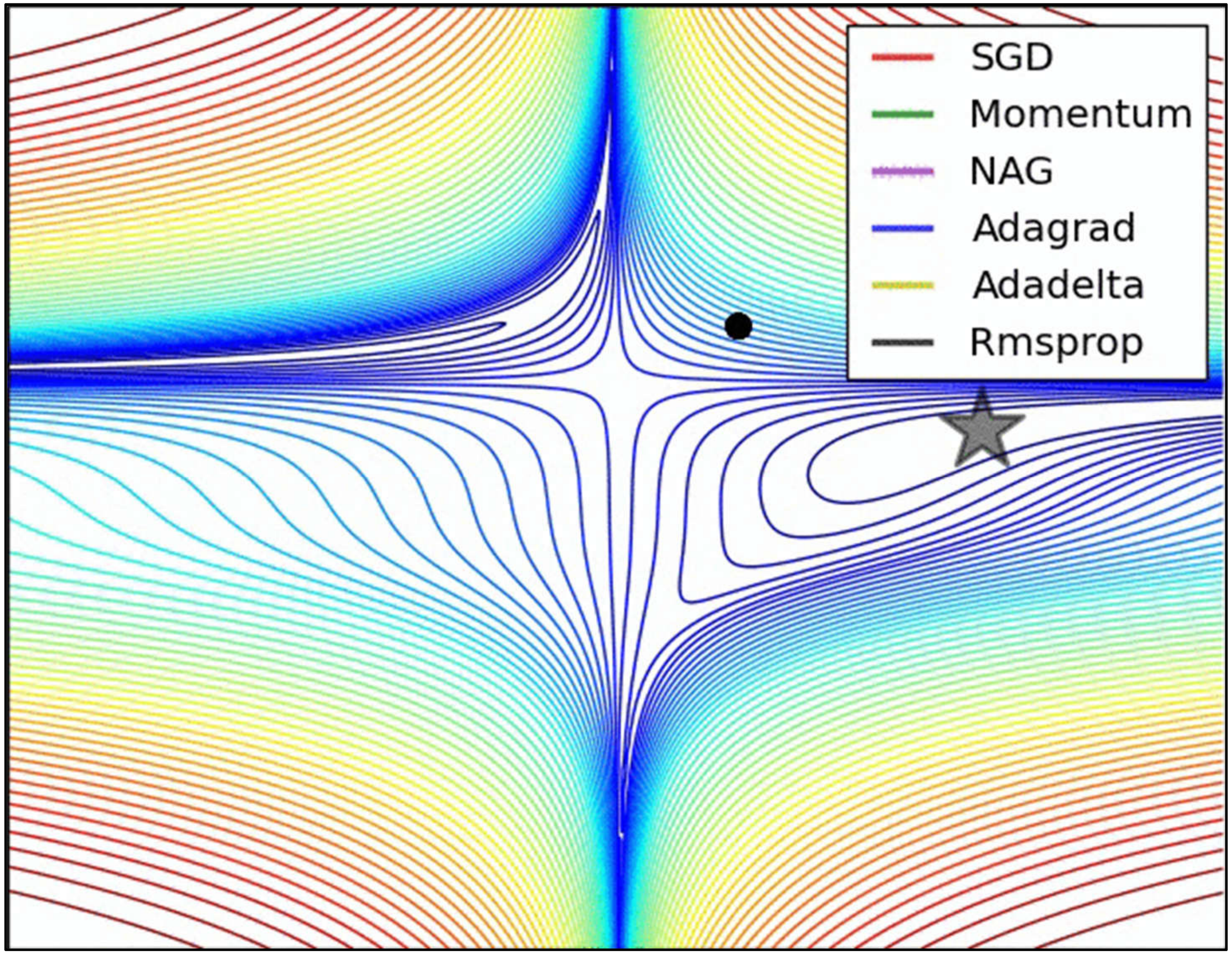
Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backward** to calculate the gradients
4. **Update** the parameters using the gradient



Slide from CS231n

Parameter Updates



We covered:

- SGD
- Momentum,
- NAG,
- Adagrad,
- Rmsprop,
- Adam (not in this vis),

We did not cover

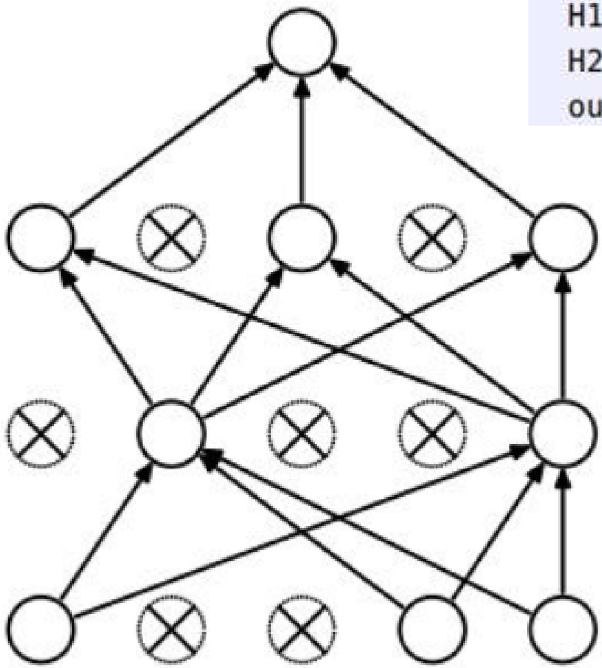
adadelta

Regularization: Dropout

Inverted dropout:

```
U1 = (np.random.rand(*H1.shape) < p) / p
```

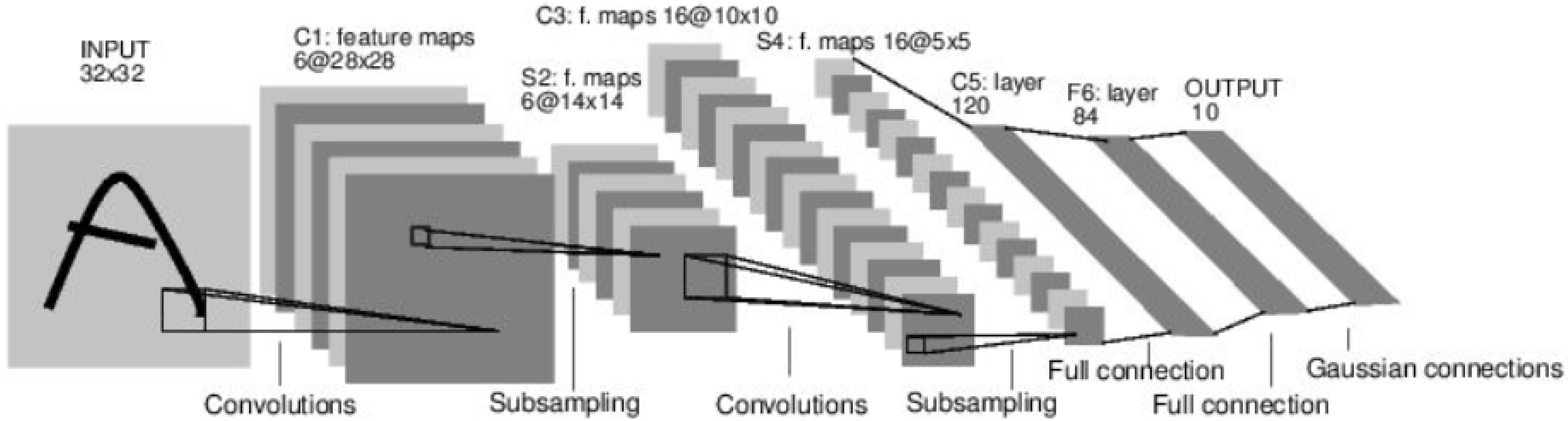
```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```



- Forces the network to have a **redundant representation**

Slide from CS231n

Convolutional Neural Networks



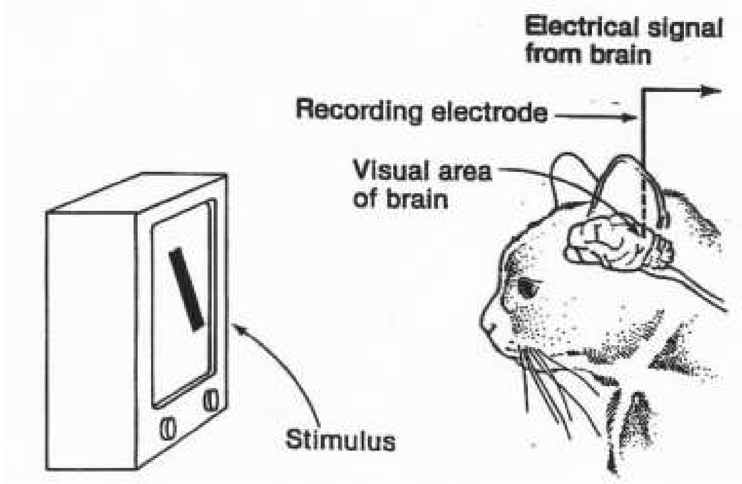
[LeNet-5, LeCun 1980s]

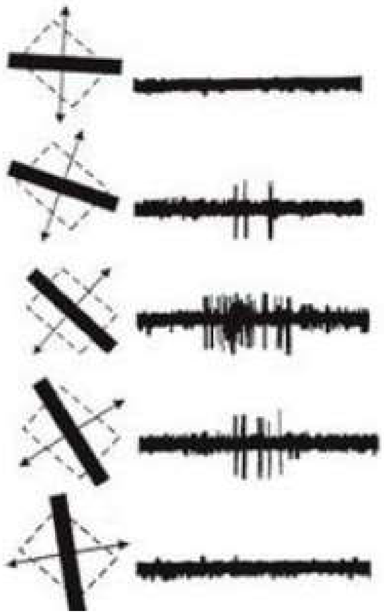
Hubel & Wiesel, 1960s

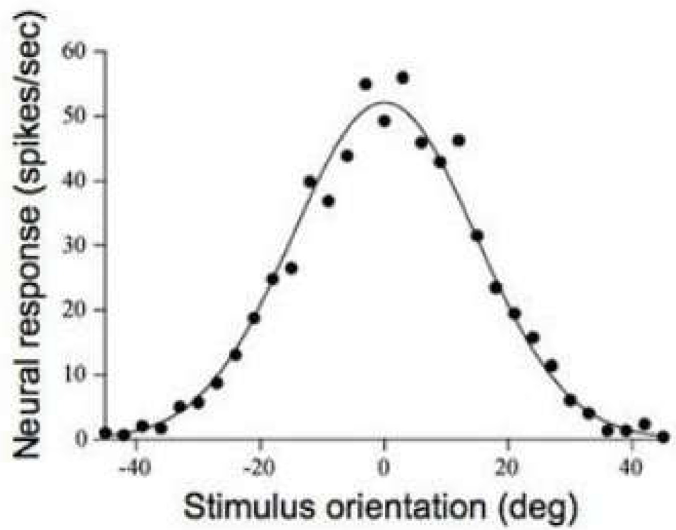
- Recording visual cortex of cat in the first visual area of processing in the back of brain called **v1**

- Winning a Nobel Prize

- The cells excited for edges of particular orientation



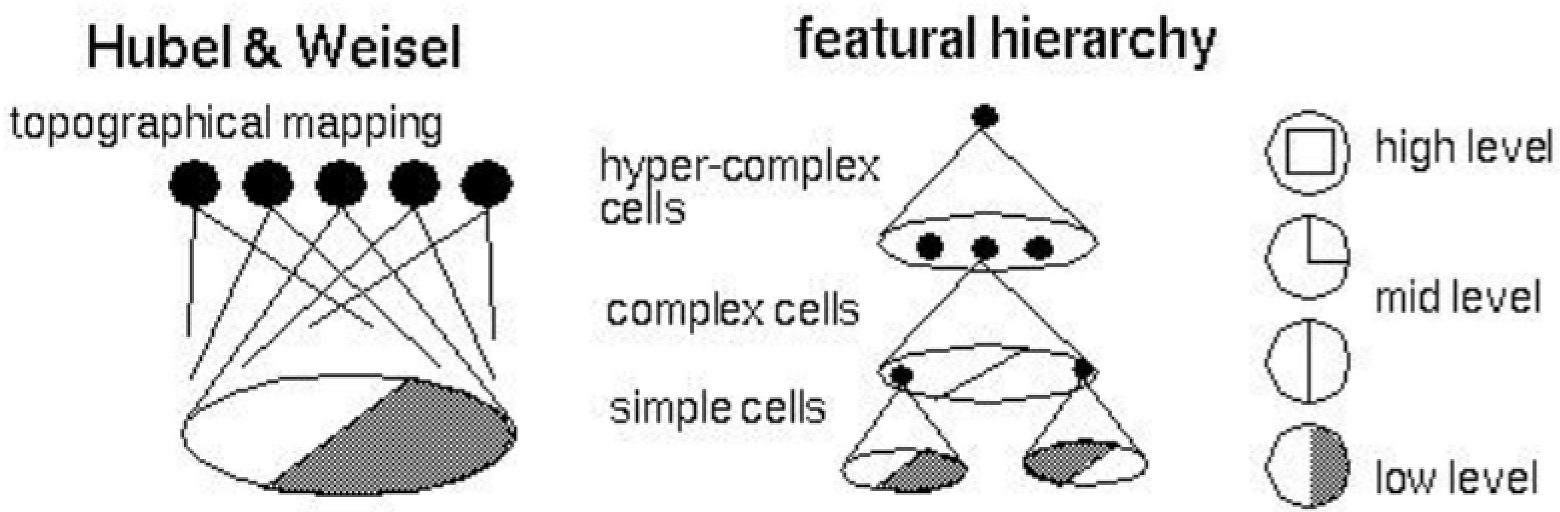




Slide from CS231n

Hierarchical Organization

- Cortex has a kind of <u>hierarchical organization</u>
- A simple cells that <u>are feeding to other cells</u> called complex cells and etc.
- These cells are <u>building on top of each other</u>
- The simple cells <u>have some local receptive fields</u> and then are built up <u>more and more complex representations</u> in the brain

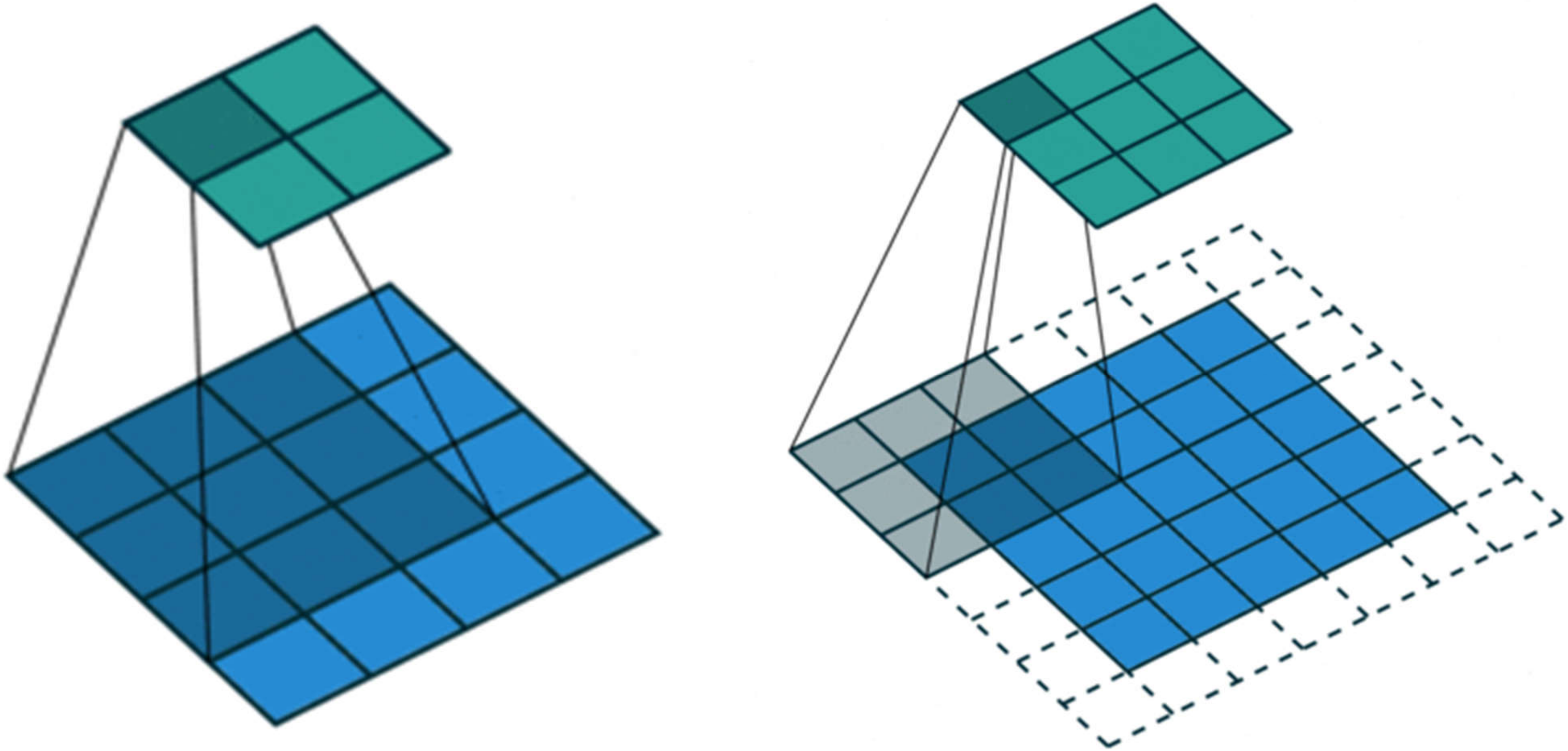


Slide from CS231n

Convolutional Neural Networks

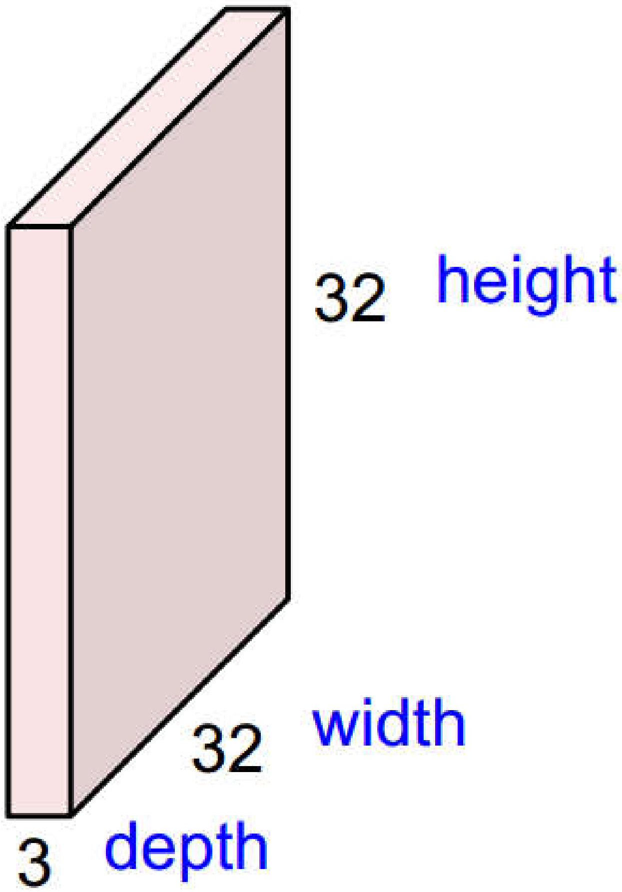
First without the brain stuff

Convolution Layer



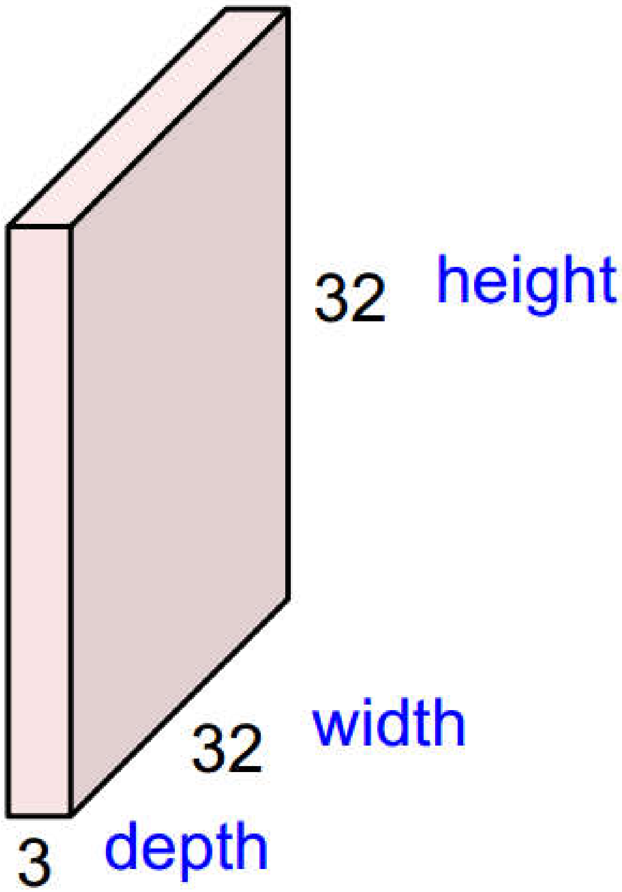
Convolution Layer

32x32x3 image

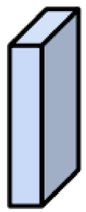


Convolution Layer

32x32x3 image



5x5x3 filter (kernel)



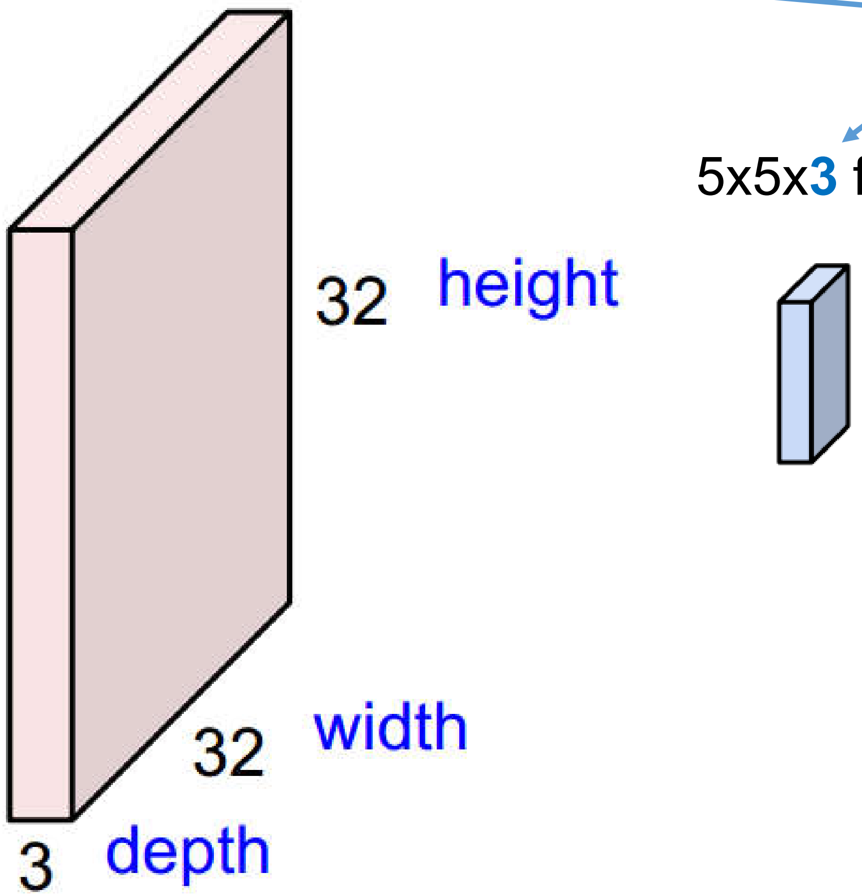
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Convolution Layer

32x32x**3** image

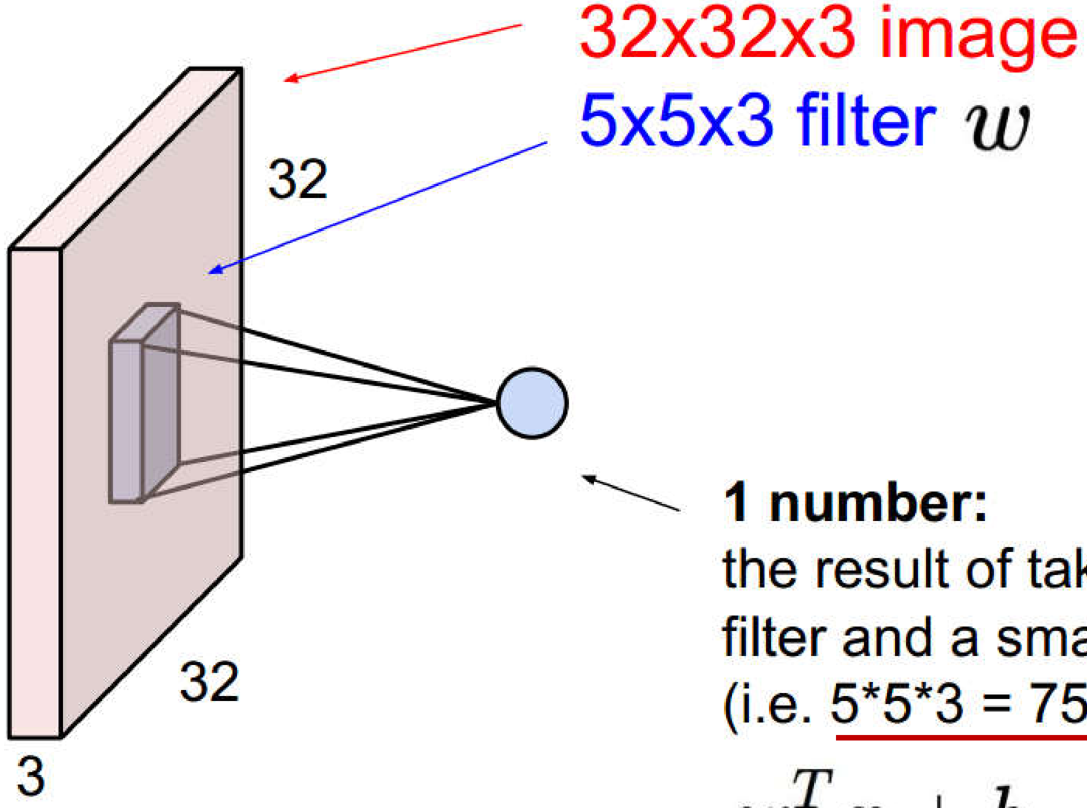
Filters always extend the full depth of the input volume

5x5x**3** filter



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

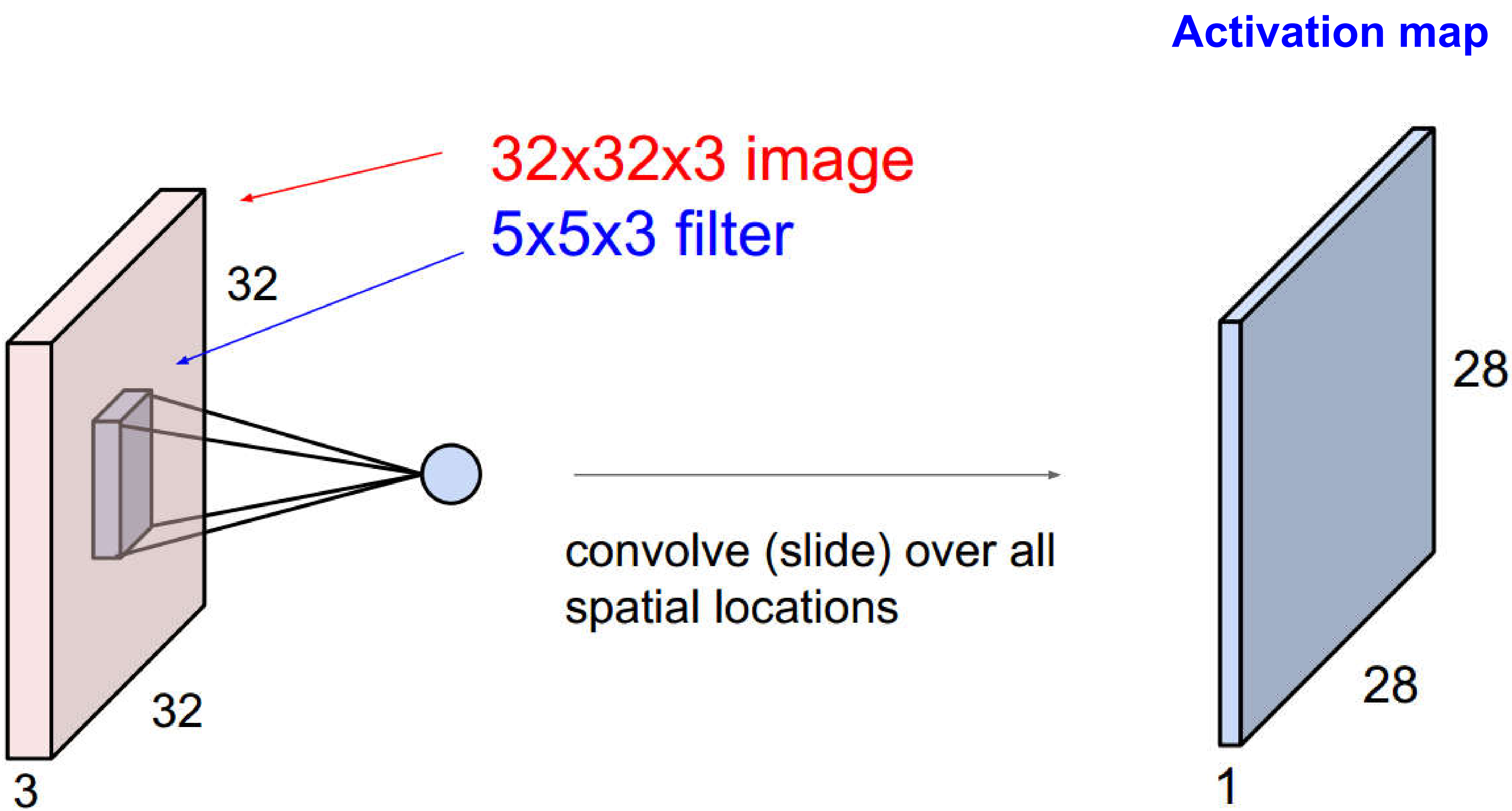
Convolution Layer



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)

$$w^T x + b$$

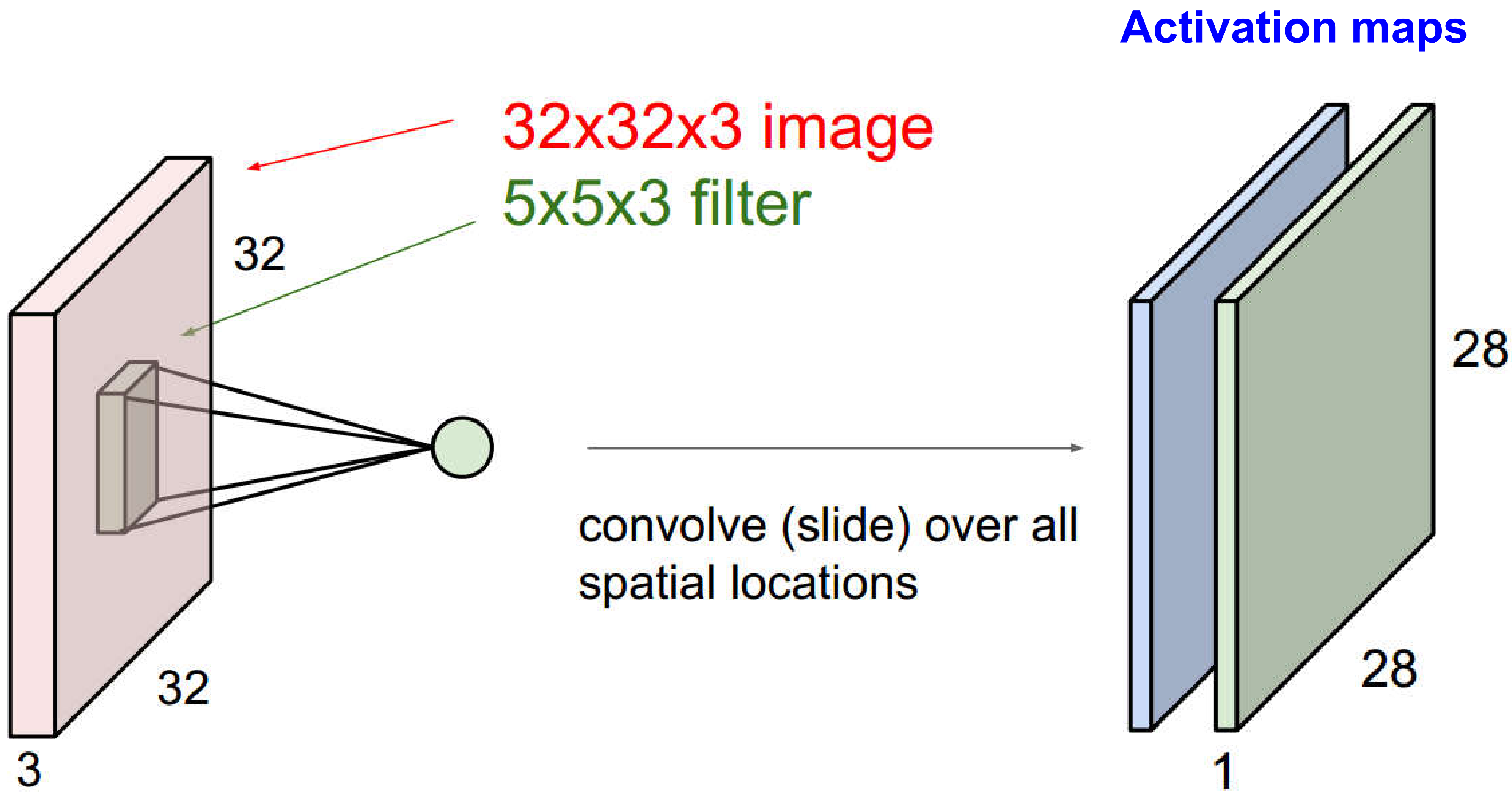
Convolution Layer



Slide from CS231n

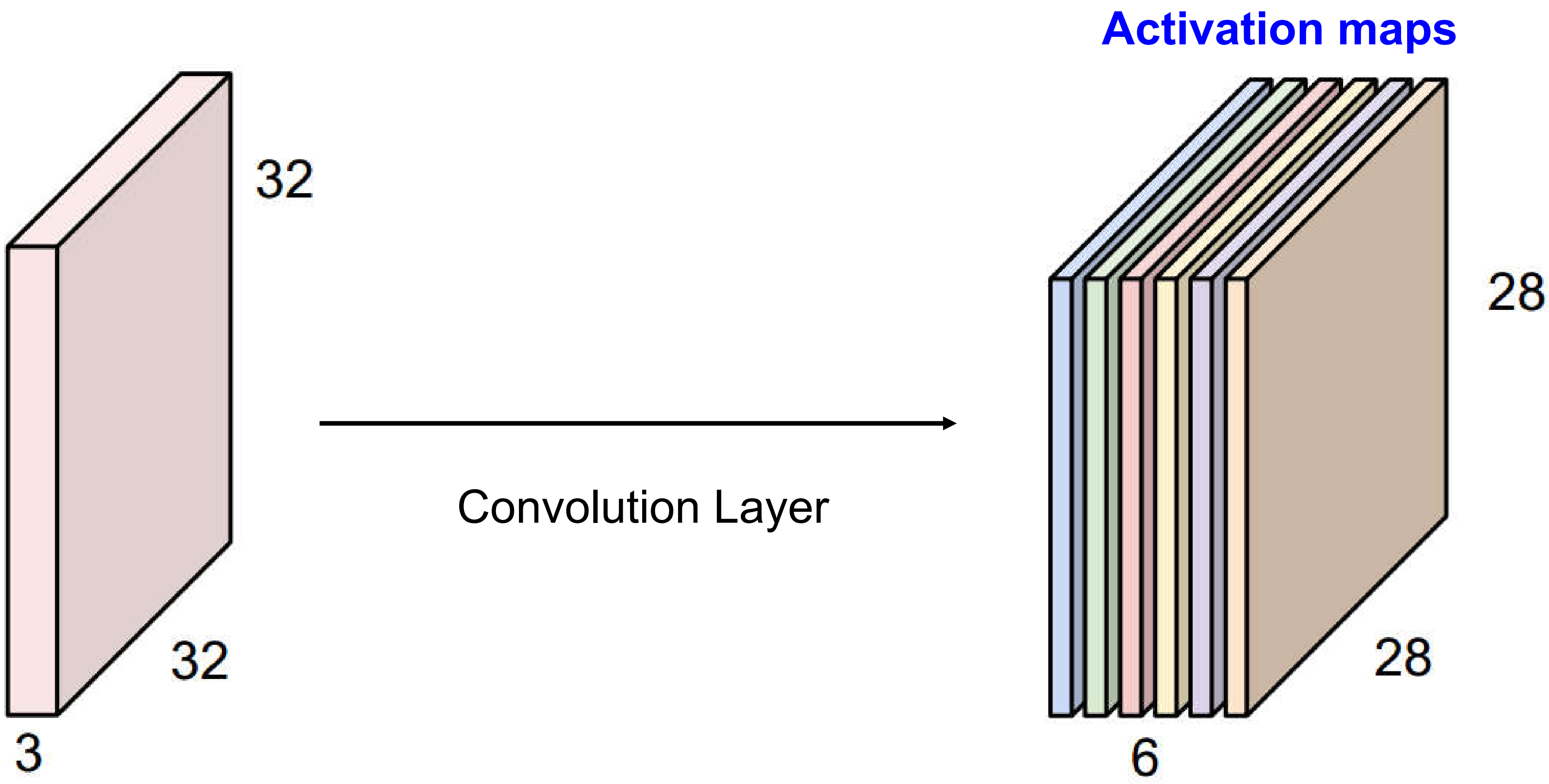
Convolution Layer

consider a second, green filter



Convolution Layer

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

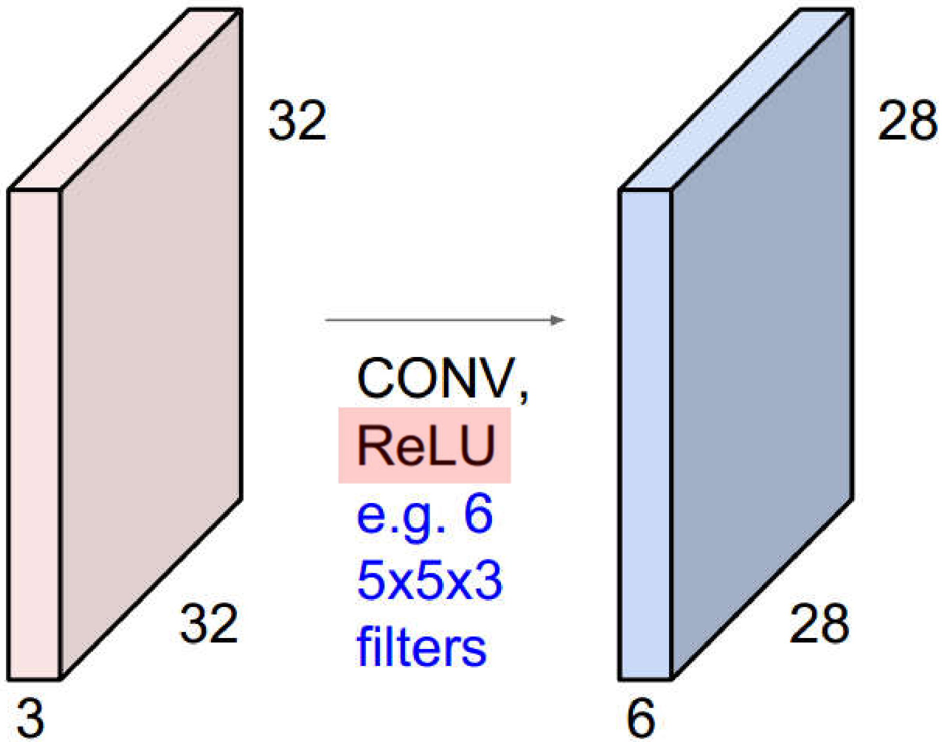


- We stack these up to get a "new image" of size 28x28x6!

Slide from CS231n

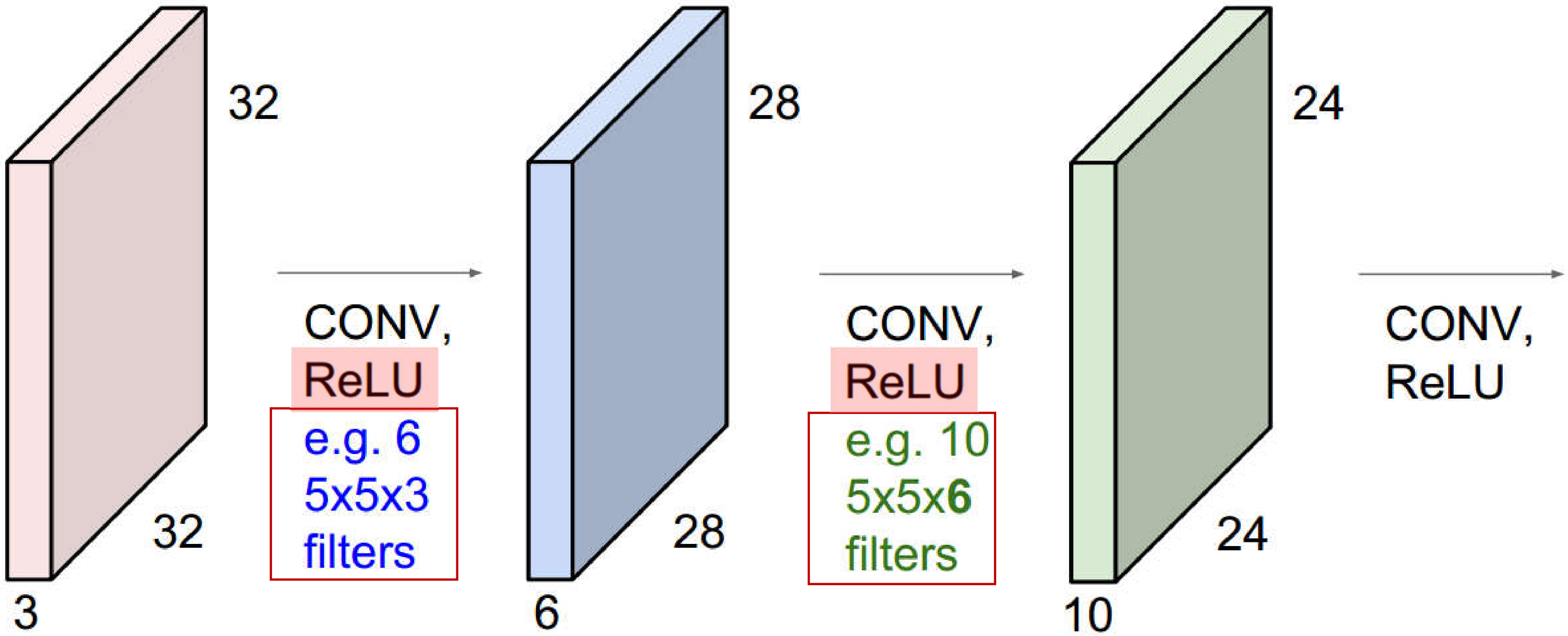
Convolution Layer

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



Convolution Layer

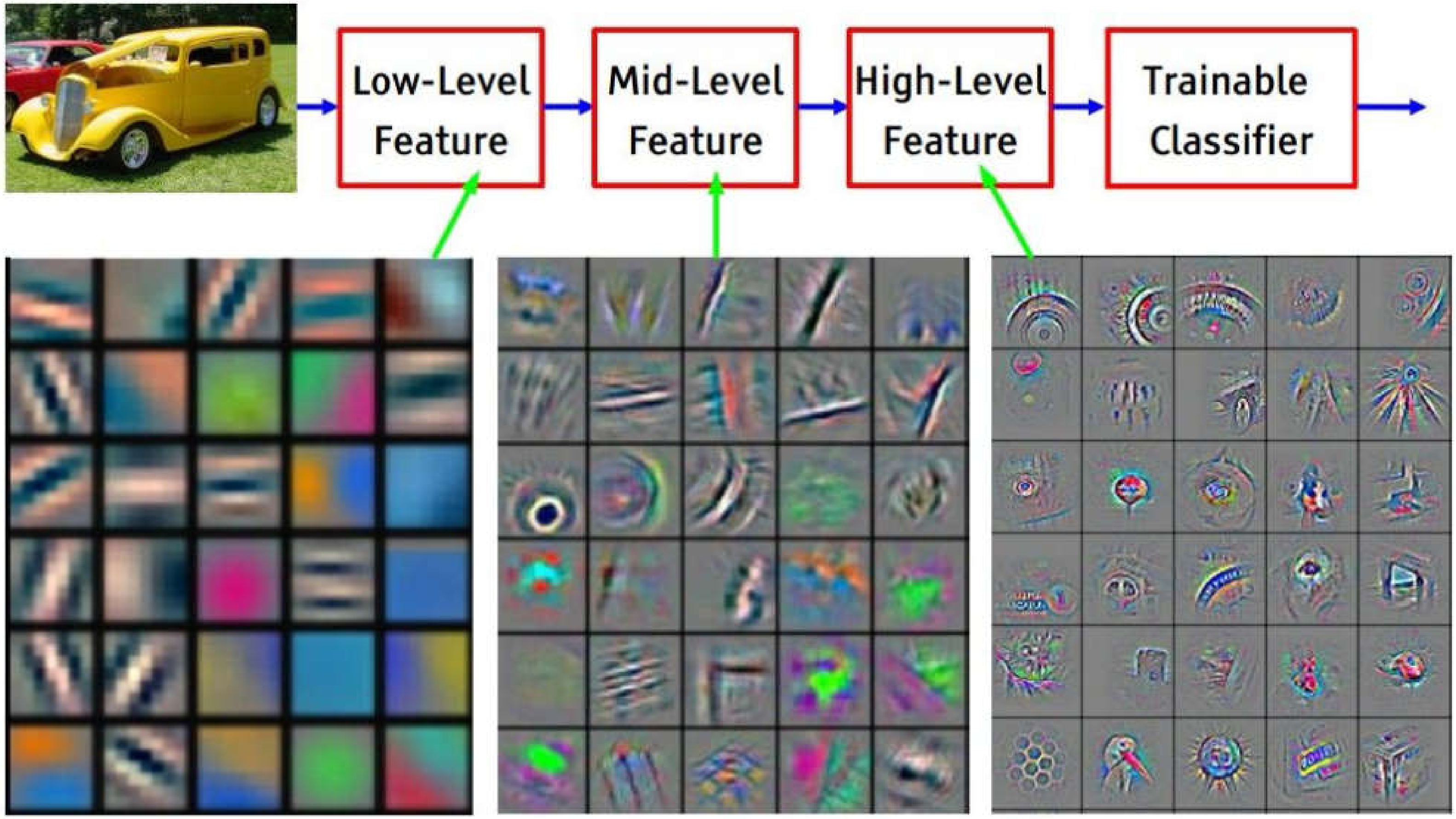
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Slide from CS231n

Convolution Layer

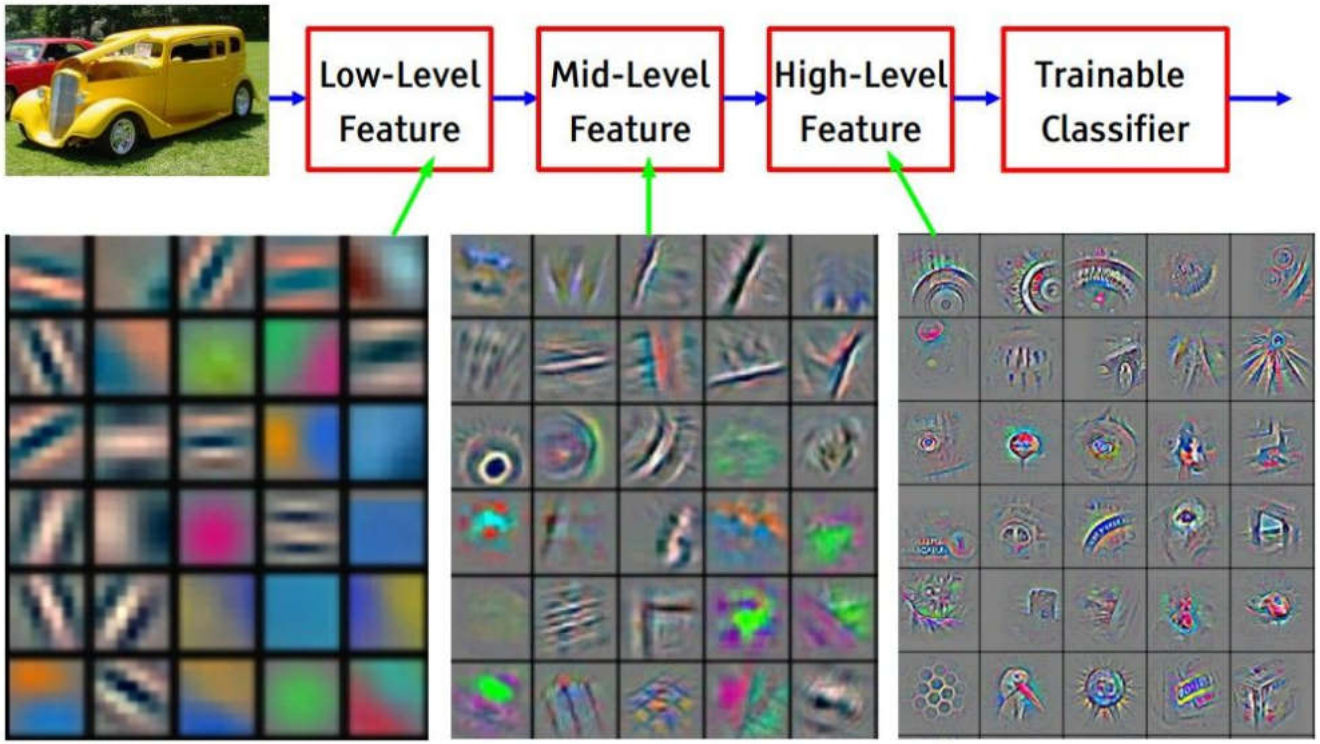
Preview

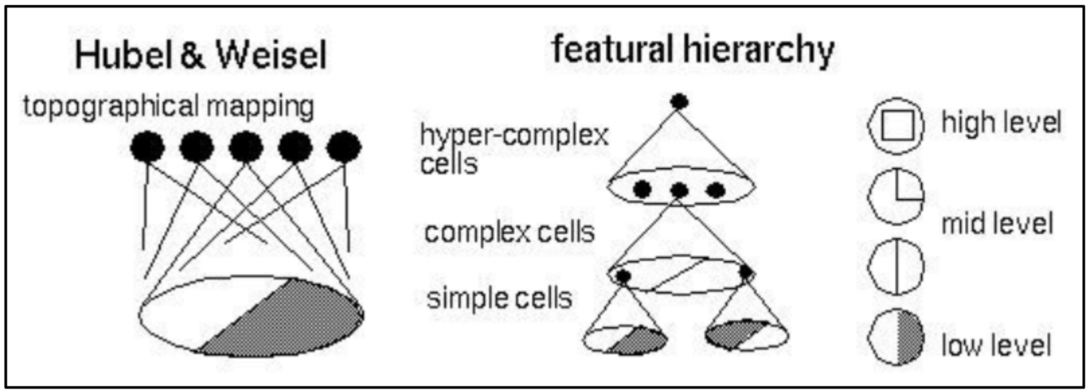


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide from Yann LeCun

Convolution Layer

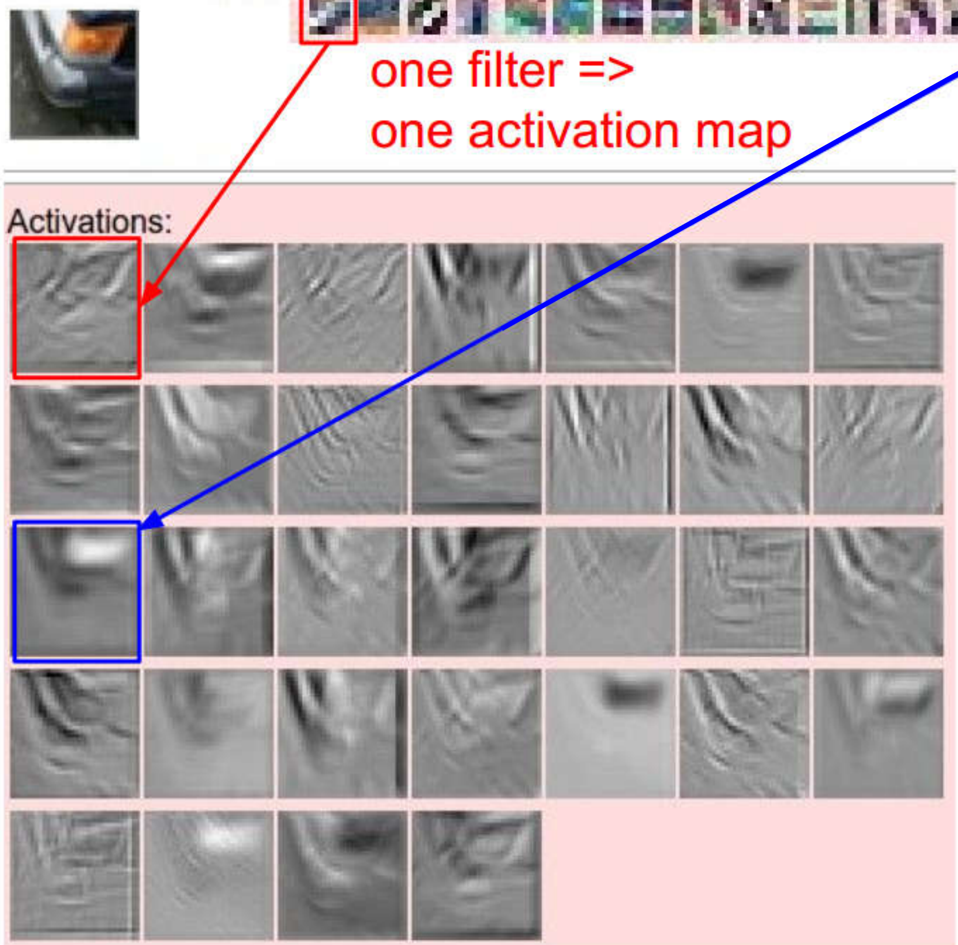




Hubel & Wiesel, 1960s

Slide from Yann LeCun

Convolution Layer



Example 5x5 filters

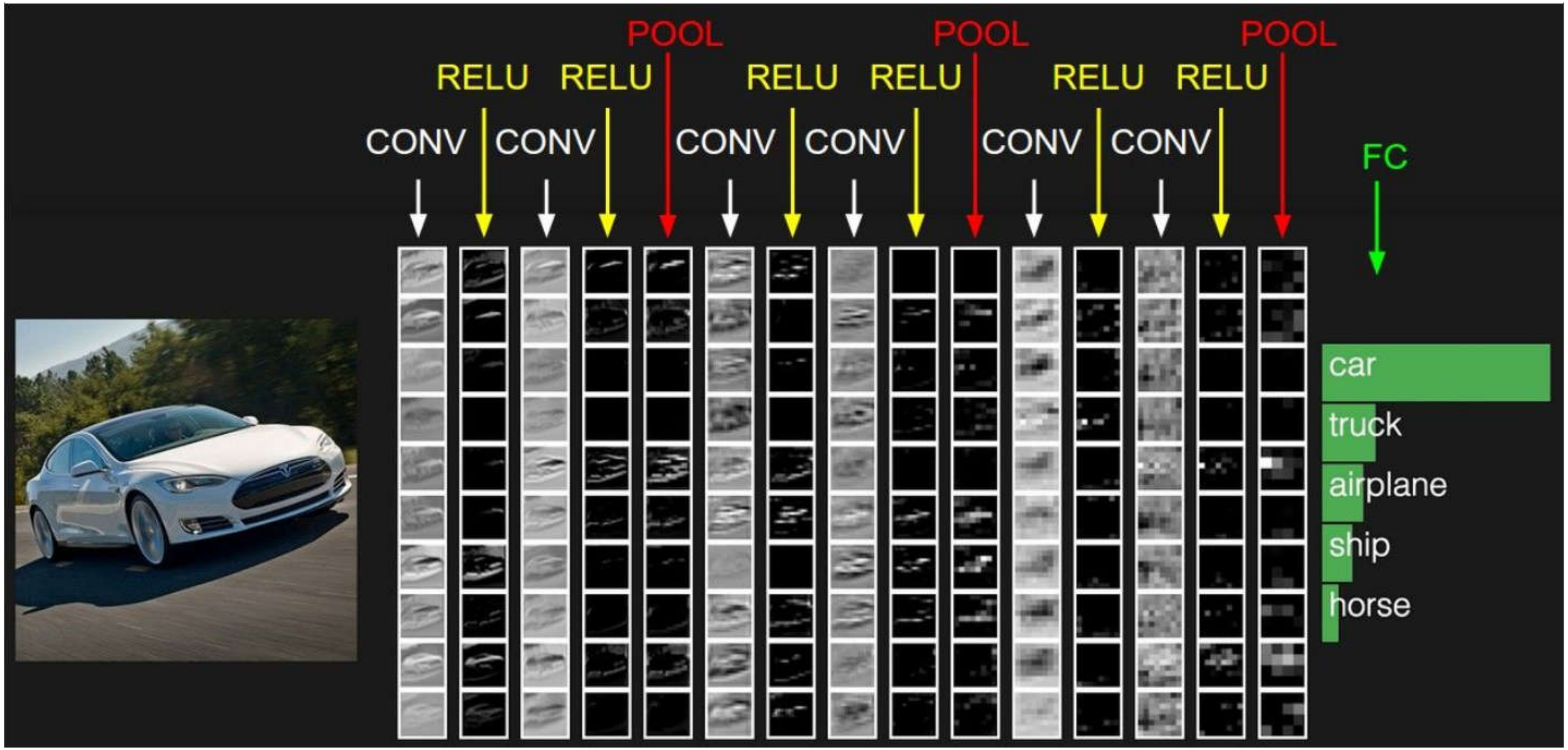
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

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

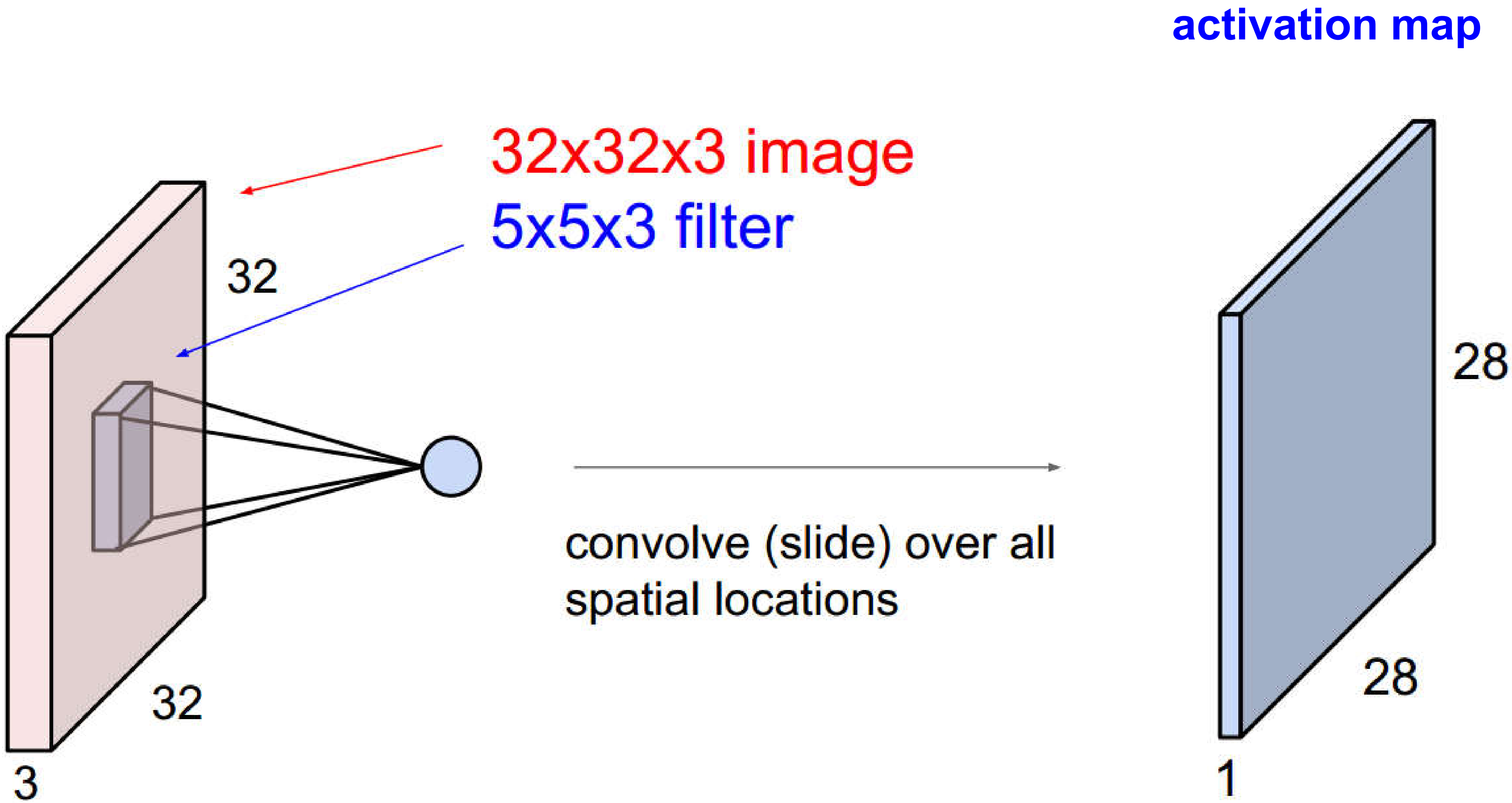
elementwise multiplication and sum of a filter and the signal (image)

Convolution Layer



Convolution Layer

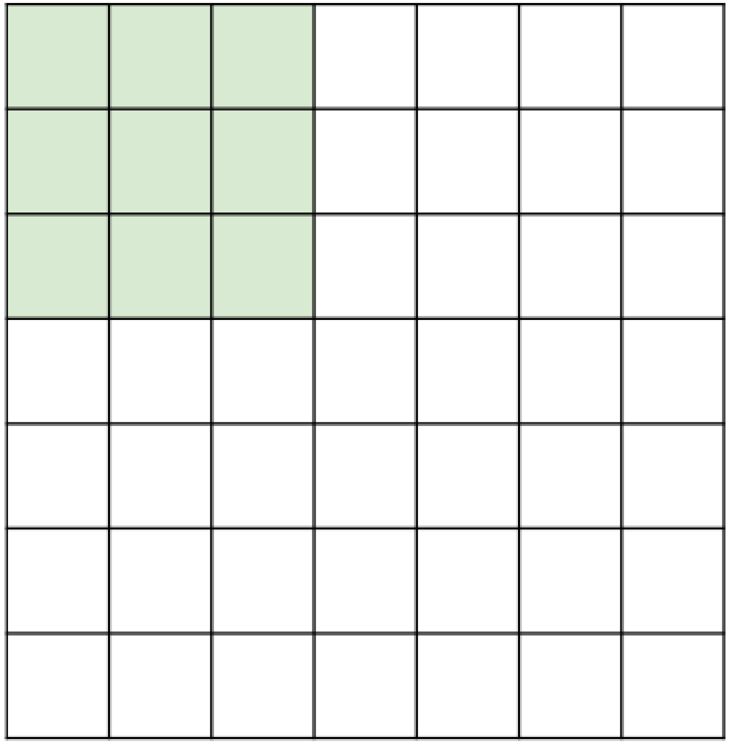
- See a closer look at spatial dimensions:



Convolution Layer

See a closer look at spatial dimensions:

7

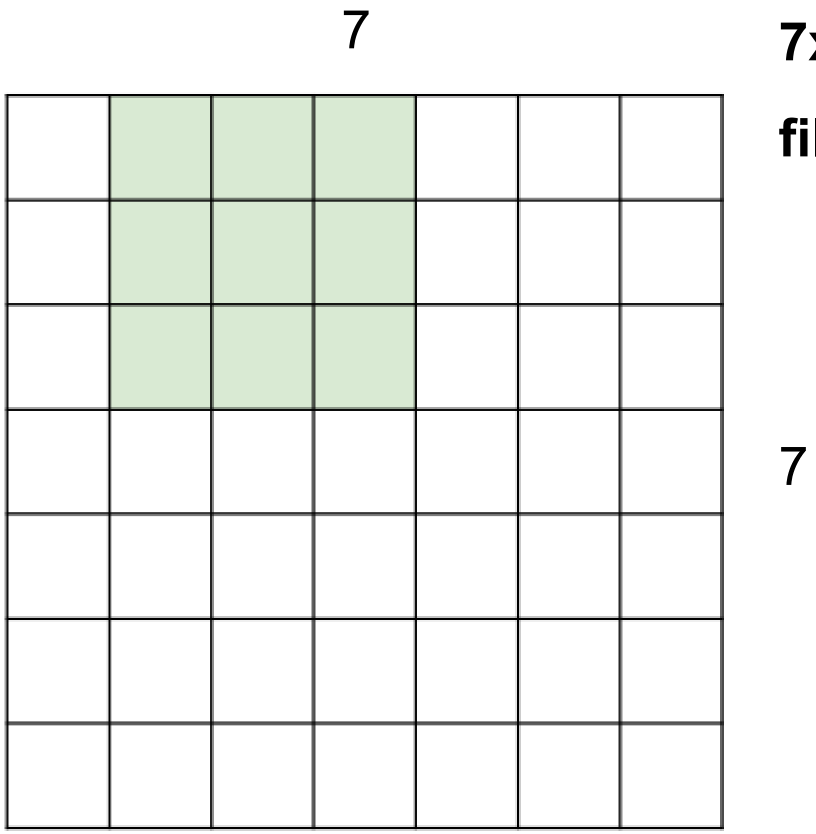


7

7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 1**

Convolution Layer

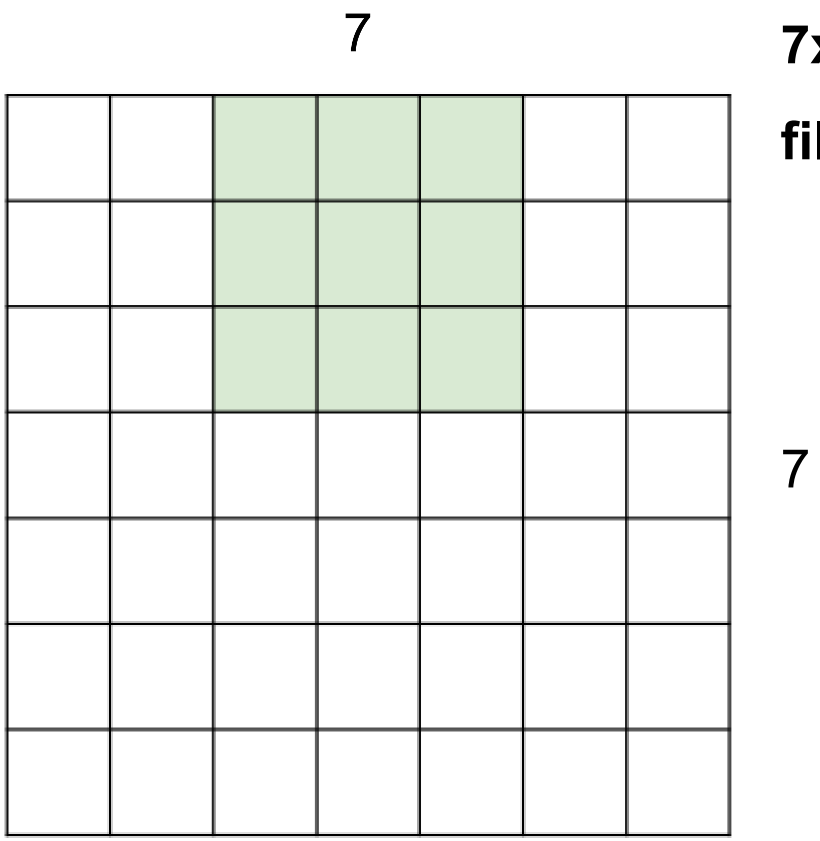
See a closer look at spatial dimensions:



7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 1**

Convolution Layer

See a closer look at spatial dimensions:

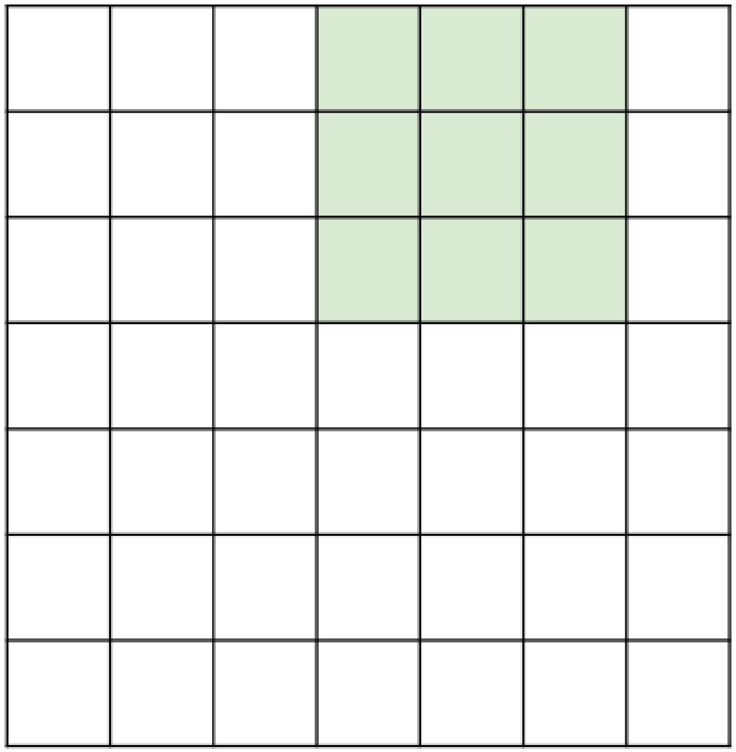


7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 1**

Convolution Layer

See a closer look at spatial dimensions:

7



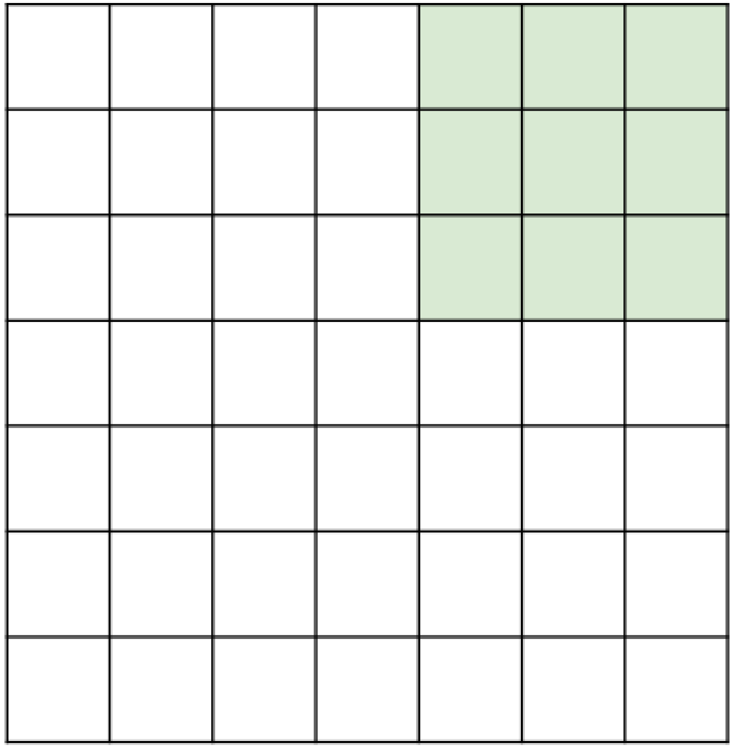
7

7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 1**

Convolution Layer

See a closer look at spatial dimensions:

7



7

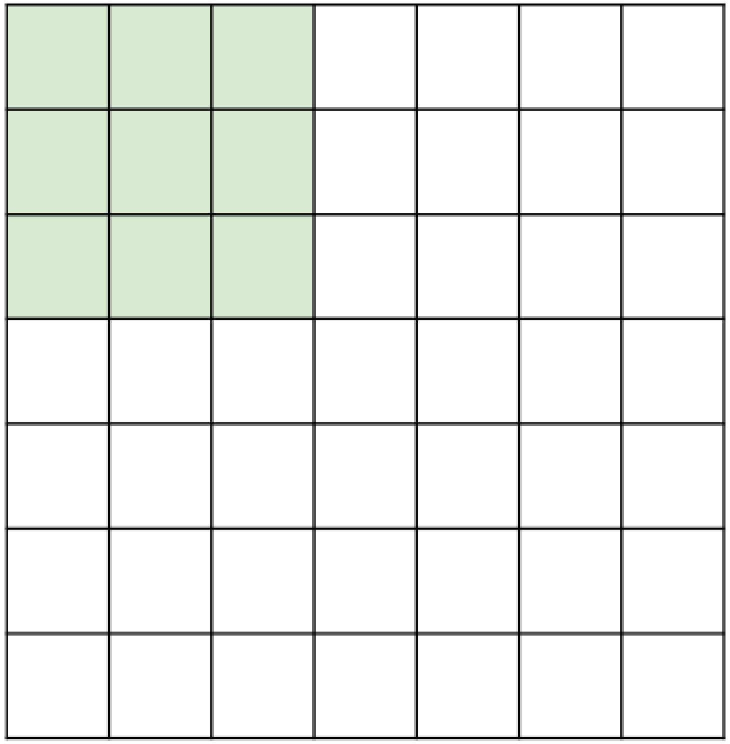
7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 1**

=> 5x5xk output

Convolution Layer

See a closer look at spatial dimensions:

7



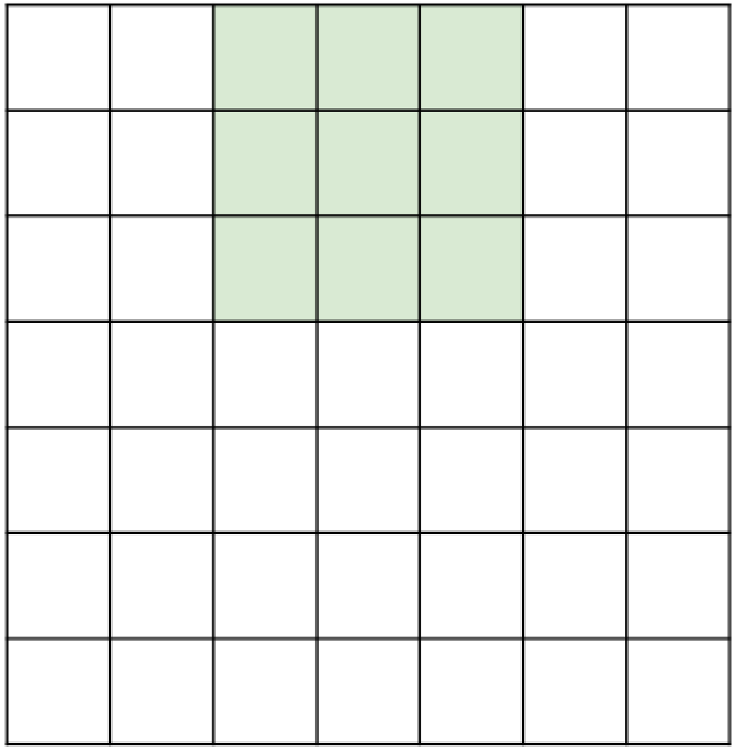
7

7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 2**

Convolution Layer

See a closer look at spatial dimensions:

7



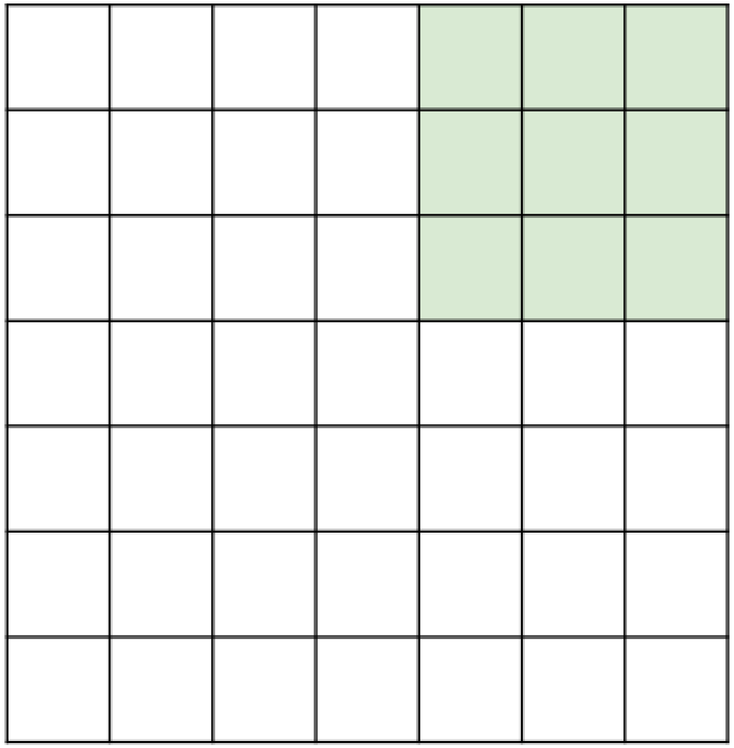
7

7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 2**

Convolution Layer

See a closer look at spatial dimensions:

7

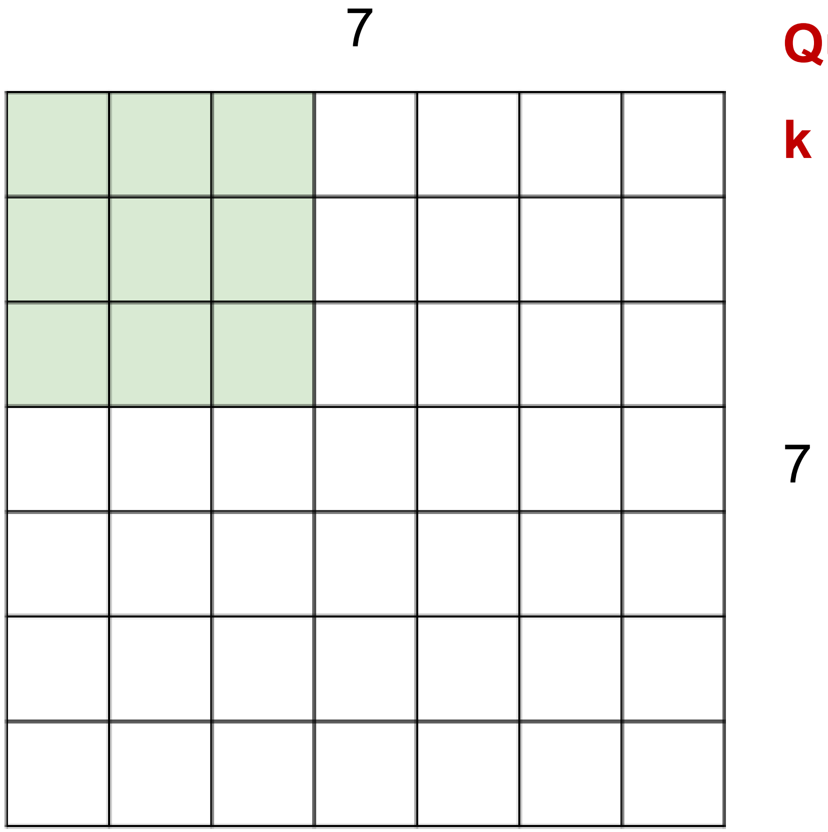


7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 2**

7 **=> 3x3xk output**

Convolution Layer

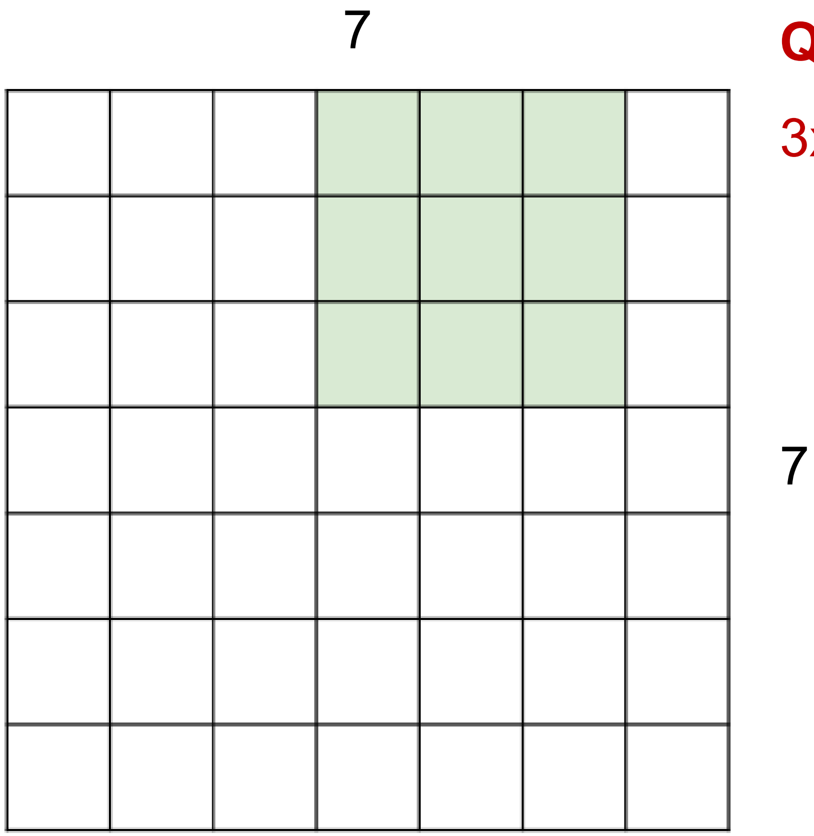
See a closer look at spatial dimensions:



Question: 7x7xd input (spatially) assume **k 3x3xd filters** applied **with stride 3?**

Convolution Layer

See a closer look at spatial dimensions:

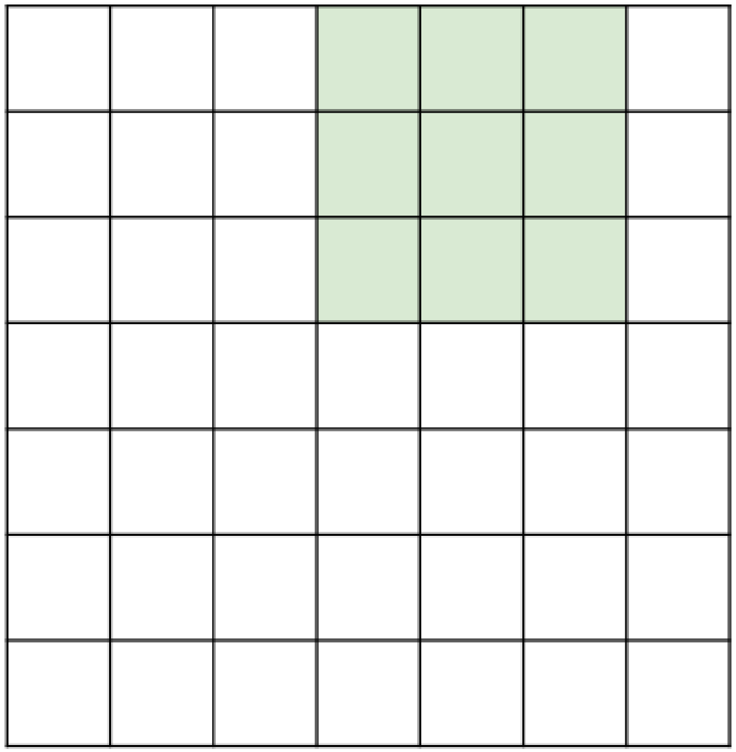


Question: 7x7xd input (spatially) assume 3x3xd filter applied **with stride 3?**

Convolution Layer

See a closer look at spatial dimensions:

7



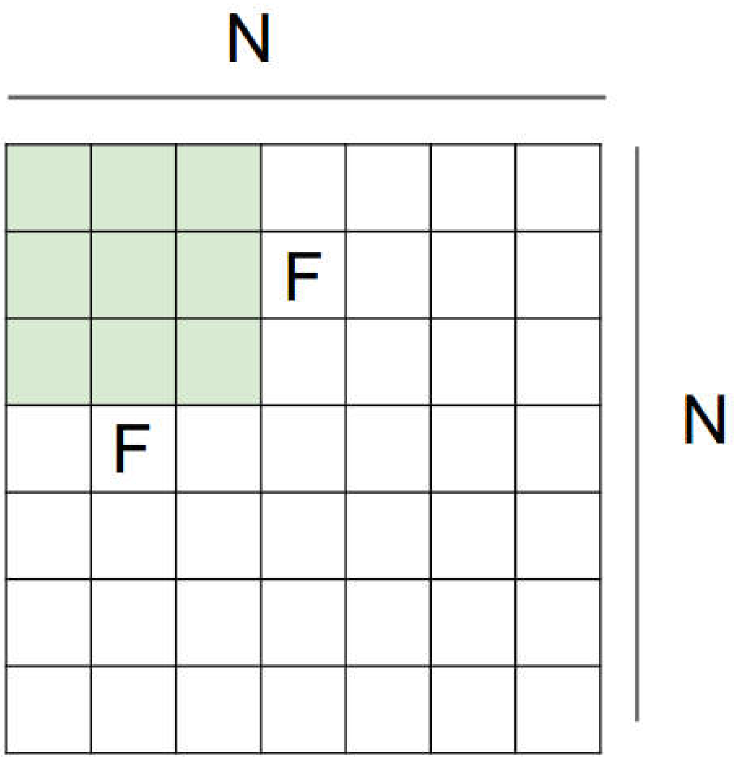
7

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

Doesn’t fit!

Cannot apply 3x3xd filter on 7x7xd input with stride 3.

Convolution Layer



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 - -!

- In practice different implementation might deal with this differently, some might throw an exception, some might give you a two by two output and ignore some parts of the input.

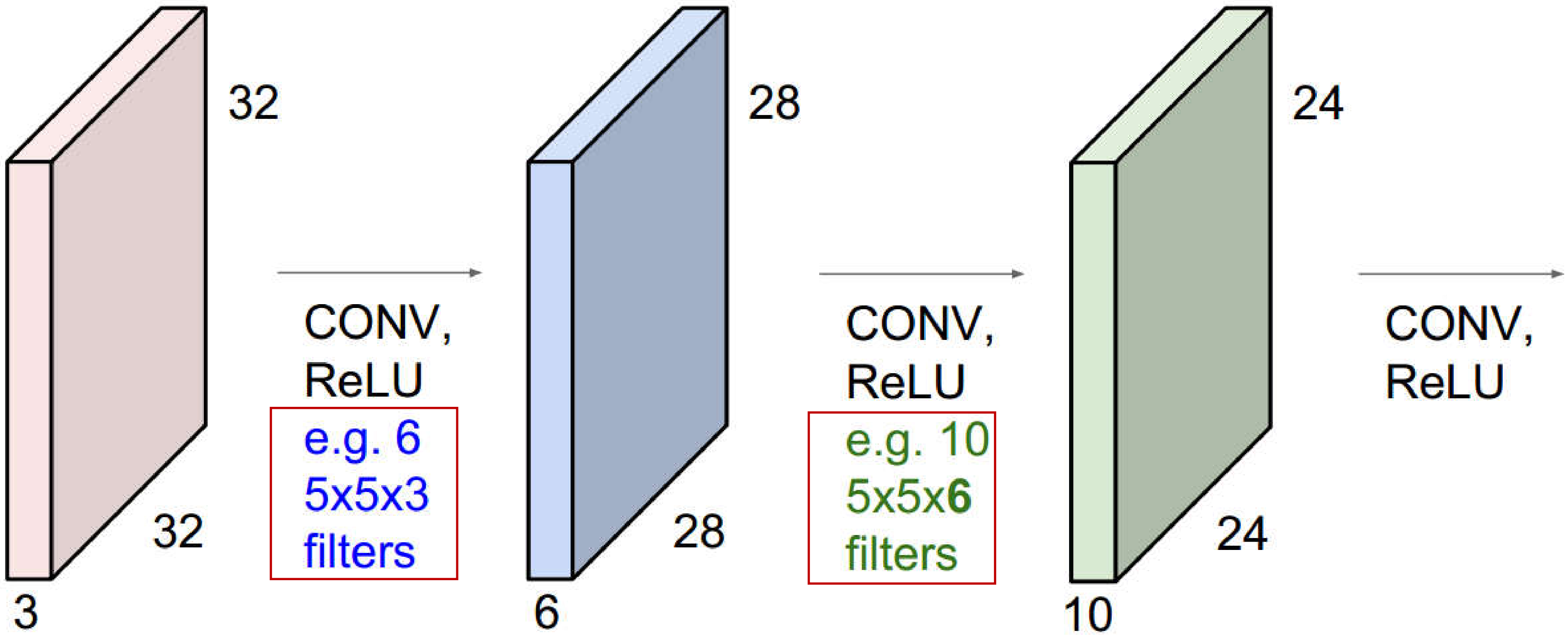
Slide from CS231n

Convolution Layer

Remember back to…

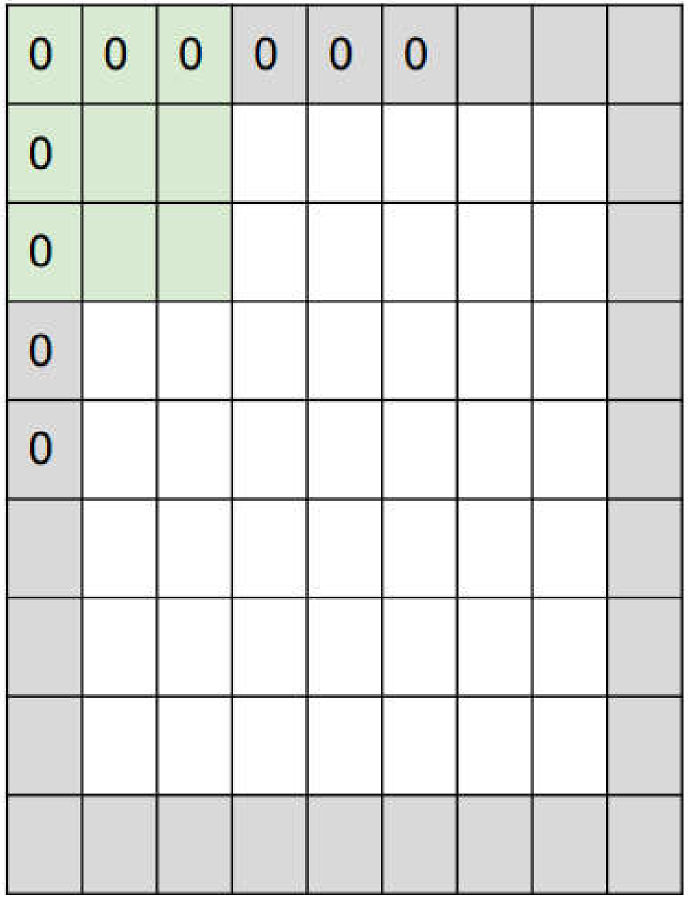
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!

(32 -> 28 -> 24 …). Shrinking too fast is not good, doesn't work well.



Convolution Layer

<u>In practice: Common to **zero pad** the border</u>



e.g. input 7x7xd

k 3x3xd filters, applied with **stride 1**

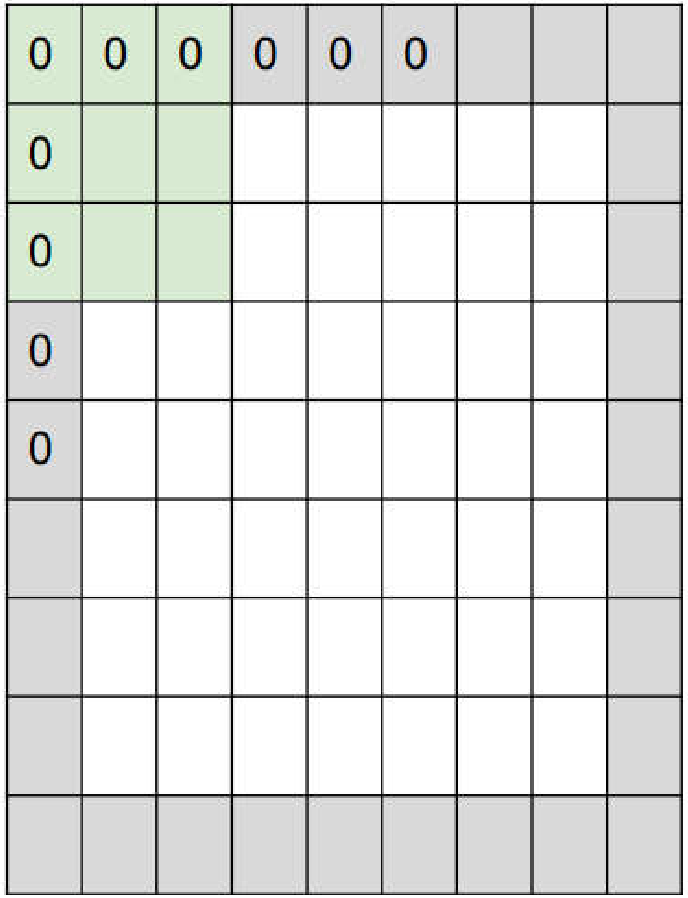
pad with 1 pixel border => what is the output?

Recall:

(N – F) / stride + 1

Convolution Layer

In practice: Common to **zero pad** the border



e.g. input 7x7xd

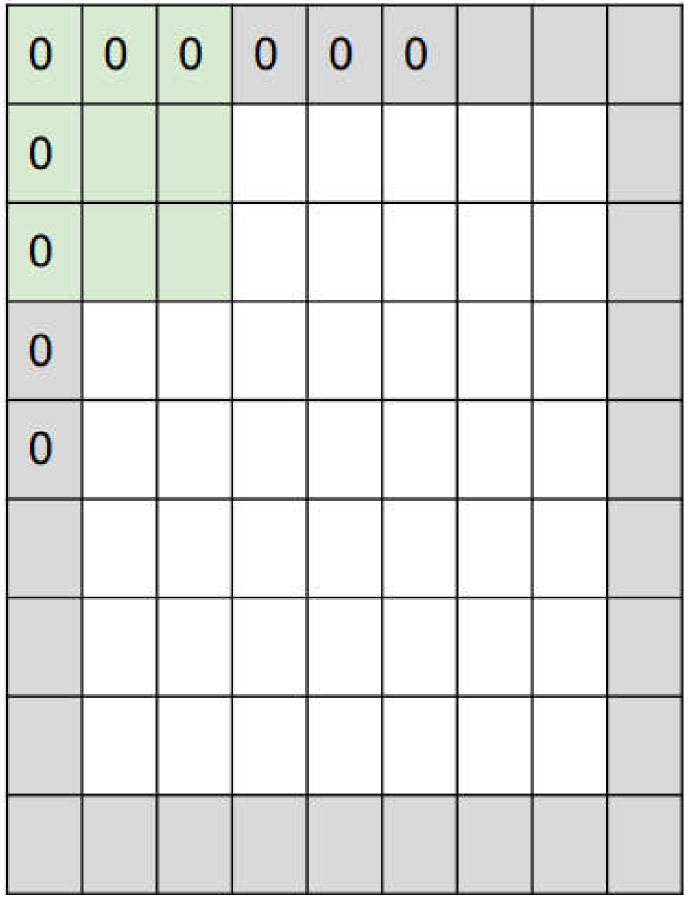
k 3x3xd filters, applied with **stride 1**

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

7x7xk output!

Convolution Layer

In practice: Common to **zero pad** the border



e.g. input 7x7xd

k 3x3xd filters, applied with **stride 1**

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

7x7xk output!

- In general, common to see CONV layers with stride 1, filters to 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

Question: why pad zero in the boundary? Not other values.

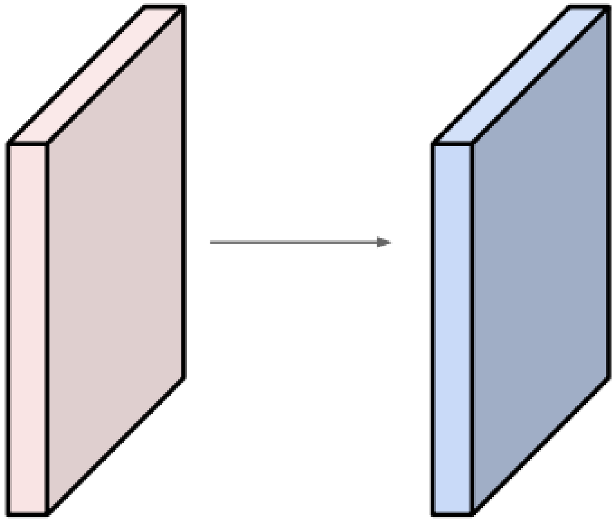
Convolution Layer

Examples time:

Input volume: **32x32x3**

10 5x5 filters with **stride 1, pad 2**

Question: output volume size?

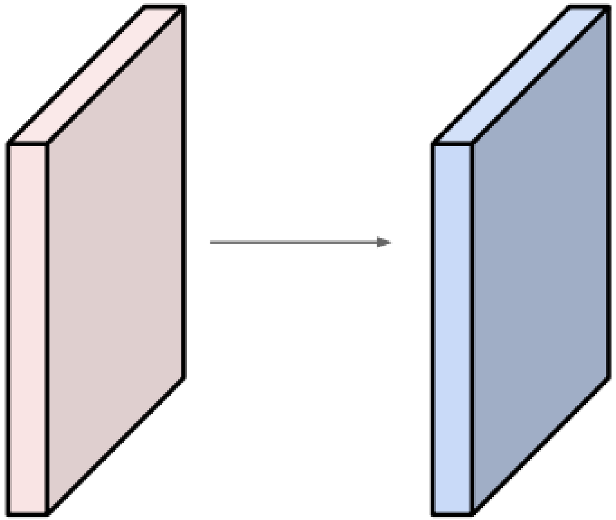


Convolution Layer

Examples time:

Input volume: **32x32x3**

10 5x5 filters with **stride 1, pad 2**



Question: output volume size?

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

32x32x10

$$(N + 2 * P - F) / S + 1$$

Slide from CS231n

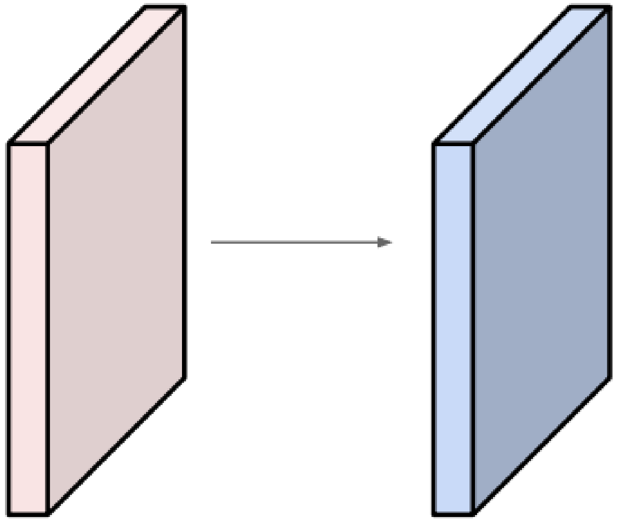
Convolution Layer

Examples time:

Input volume: **32x32x3**

10 5x5 filters with **stride 1, pad 2**

Question: Number of parameters in this layer?

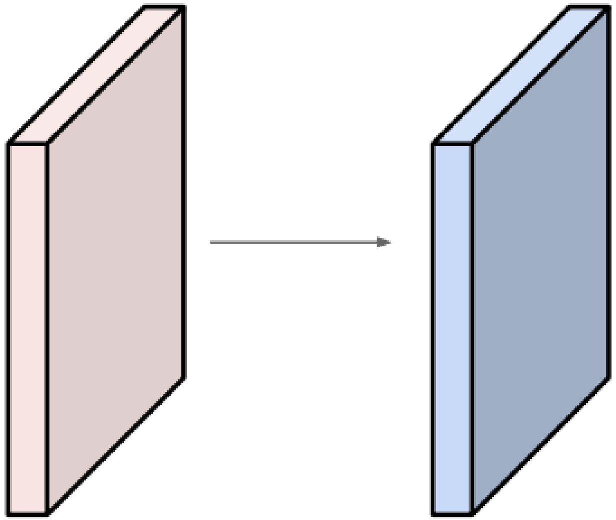


Convolution Layer

Examples time:

Input volume: **32x32x3**

10 5x5 filters with **stride 1, pad 2**



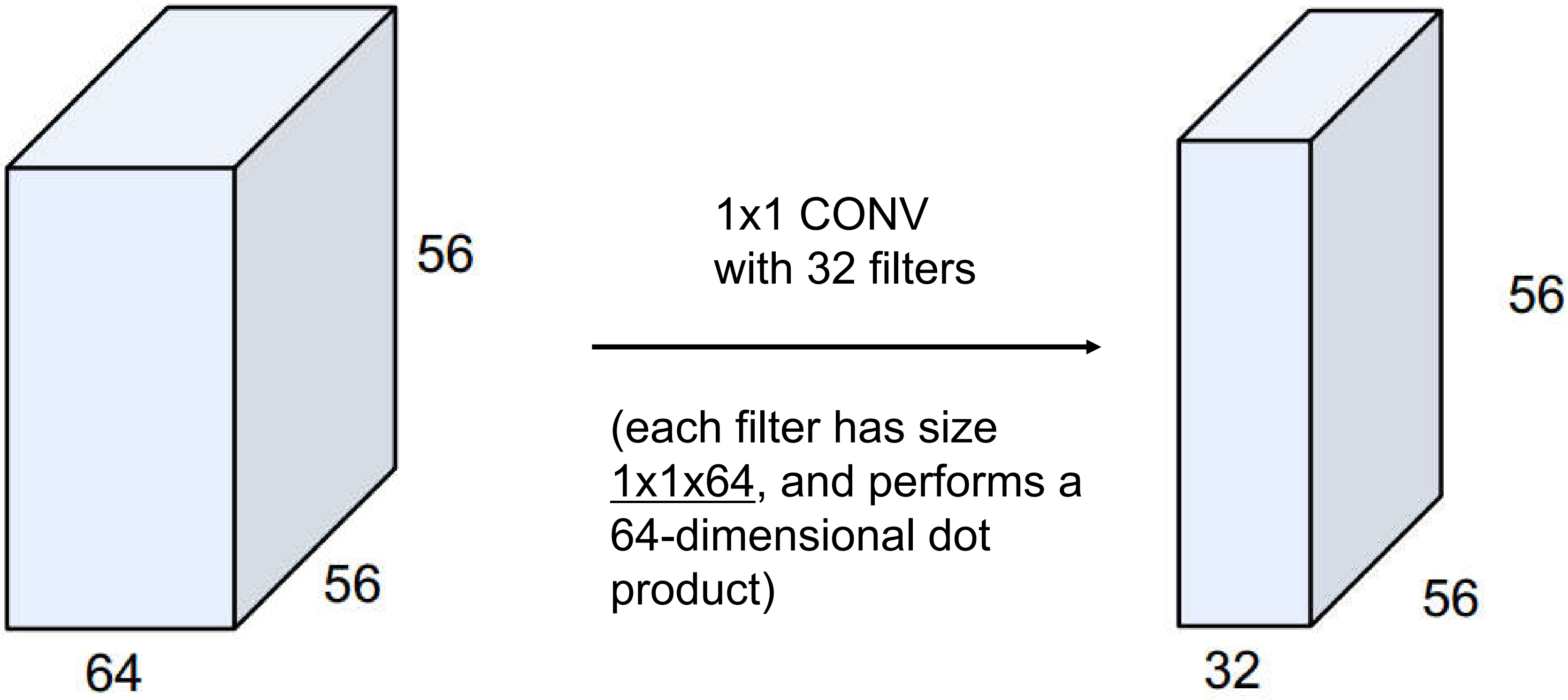
Question: Number of parameters in this layer?

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

=> 76 * 10 = **760**

Convolution Layer

(btw, 1x1 convolution layers make perfect sense)



Summary to Convolution Layer

```
tf.nn.conv2d(
    input,
    filter,
    strides,
    padding,
    use_cudnn_on_gpu=True,
    data_format='NHWC',
    dilations=[1, 1, 1, 1],
    name=None
)
```

- 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 $W_2 \times H_2 \times \mathrm{D}_2$ 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)
- 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.

Slide from CS231n

Summary to Convolution 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.

Common settings:
K = (powers of 2, e.g. 32, 64, 128, 256, …)
- 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

- Produces a volume of size $W_2 \times H_2 \times \mathrm{D}_2$ 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)
- 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.

Convolution Layer

Example: CONV layer in Torch

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.

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

Slide from CS231n

Convolution Layer

Example: CONV layer in Caffe

```
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
    }
  }
}
```

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.

Convolution Layer

Example: CONV layer in Lasagne

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.

class `lasagne.layers.Conv2DLayer`(*incoming, num_filters, filter_size, stride=(1, 1), pad=0, untie_biases=False, W=lasagne.init.GlorotUniform(), b=lasagne.init.Constant(0.), nonlinearity=lasagne.nonlinearities.rectify, flip_filters=True, convolution=theano.tensor.nnet.conv2d, **kwargs*) [source]

2D convolutional layer

Performs a 2D convolution on its input and optionally adds a bias and applies an elementwise nonlinearity.

Parameters:

incoming : a `Layer` instance or a tuple

The layer feeding into this layer, or the expected input shape. The output of this layer should be a 4D tensor, with shape `(batch_size, num_input_channels, input_rows, input_columns)`.

num_filters : int

The number of learnable convolutional filters this layer has.

filter_size : int or iterable of int

An integer or a 2-element tuple specifying the size of the filters.

stride : int or iterable of int

An integer or a 2-element tuple specifying the stride of the convolution operation.

pad : int, iterable of int, 'full', 'same' or 'valid' (default: 0)

By default, the convolution is only computed where the input and the filter fully overlap (a valid convolution). When `stride=1`, this yields an output that is smaller than the input by `filter_size - 1`. The *pad* argument allows you to implicitly pad the input with zeros, extending the output size.

A single integer results in symmetric zero-padding of the given size on all borders, a tuple of two integers allows different symmetric padding per dimension.

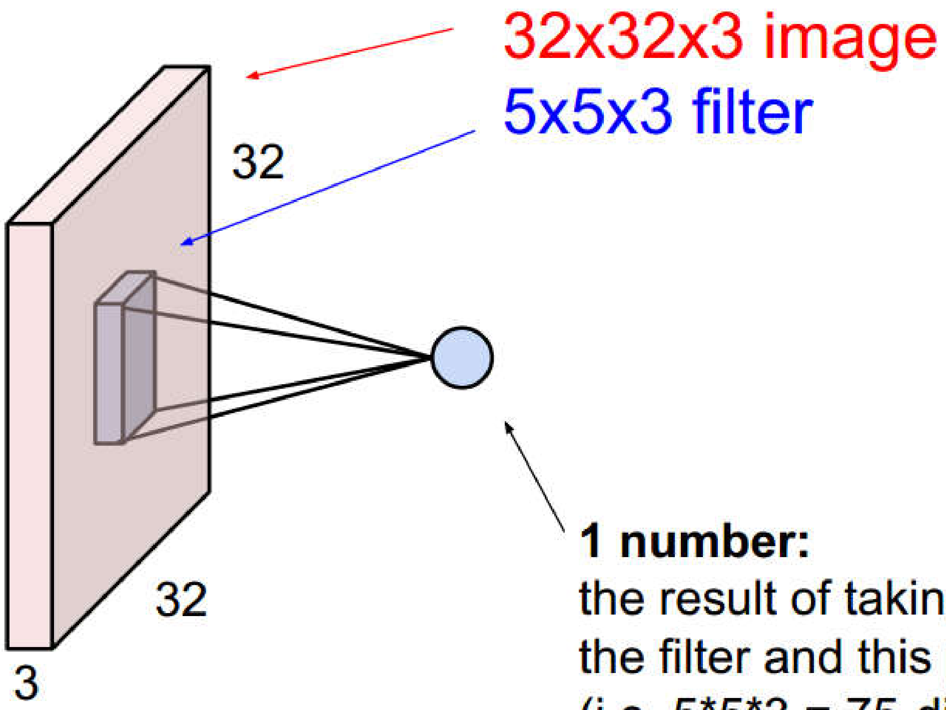
`'full'` pads with one less than the filter size on both sides. This is equivalent to computing the convolution wherever the input and the filter overlap by at least one position.

`'same'` pads with half the filter size (rounded down) on both sides. When `stride=1` this results in an output size equal to the input size. Even filter size is not supported.

`'valid'` is an alias for `0` (no padding / a valid convolution).

Convolution Layer

The brain/neuron view of CONV Layer

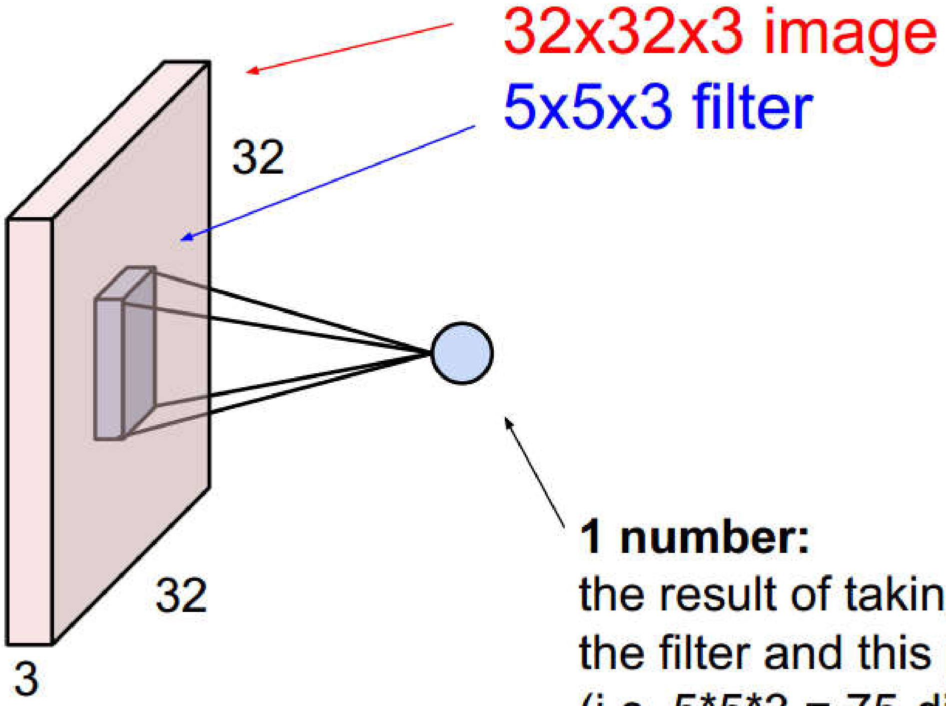


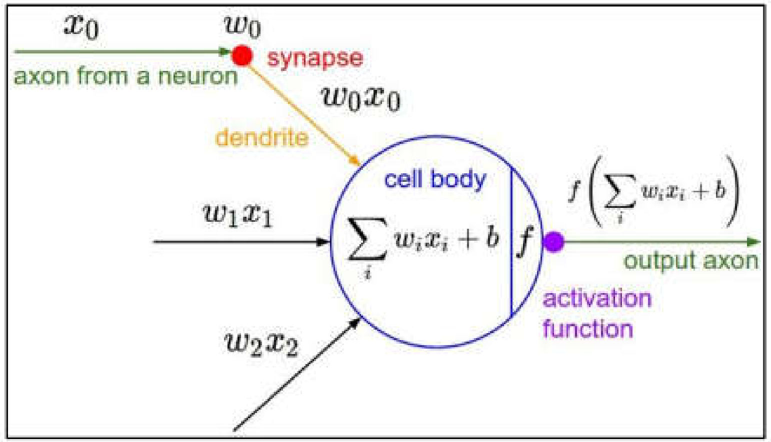
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

Slide from CS231n

Convolution Layer

The brain/neuron view of CONV Layer





It's just a neuron with local connectivity…

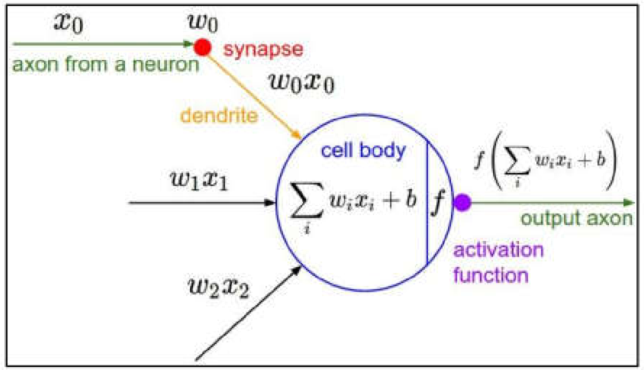
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

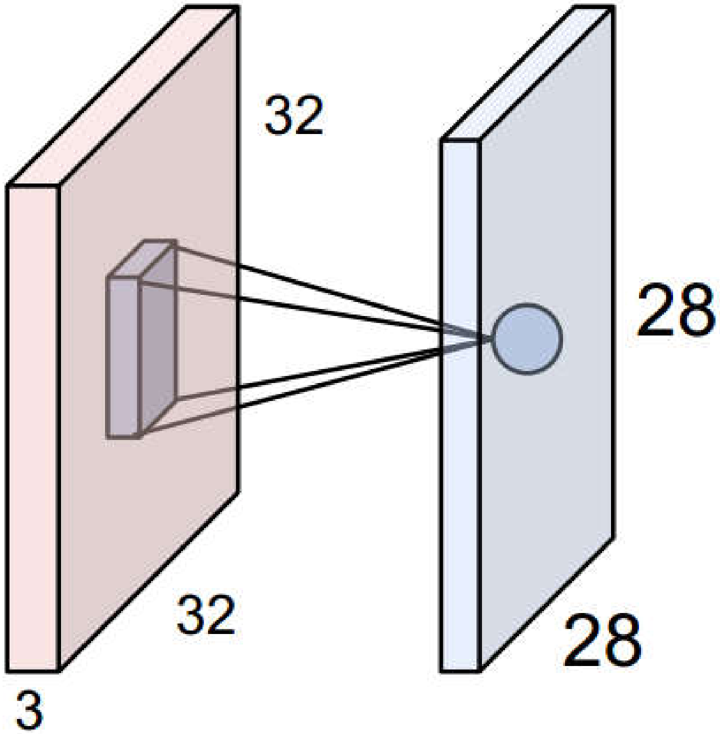
- <u>The neurons **receptive fields** is 5 by 5</u>

Slide from CS231n

Convolution Layer

The brain/neuron view of CONV Layer





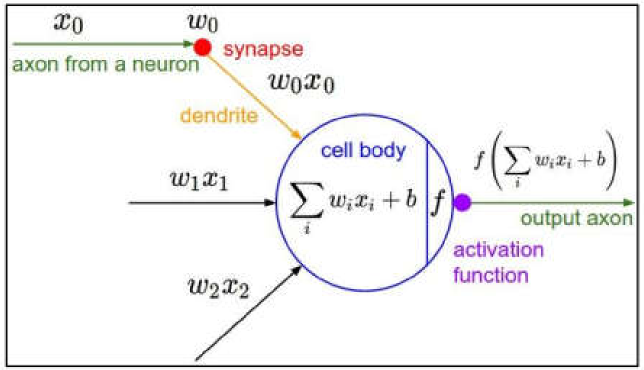
An activation map is a 28x28 sheet of neuron outputs:

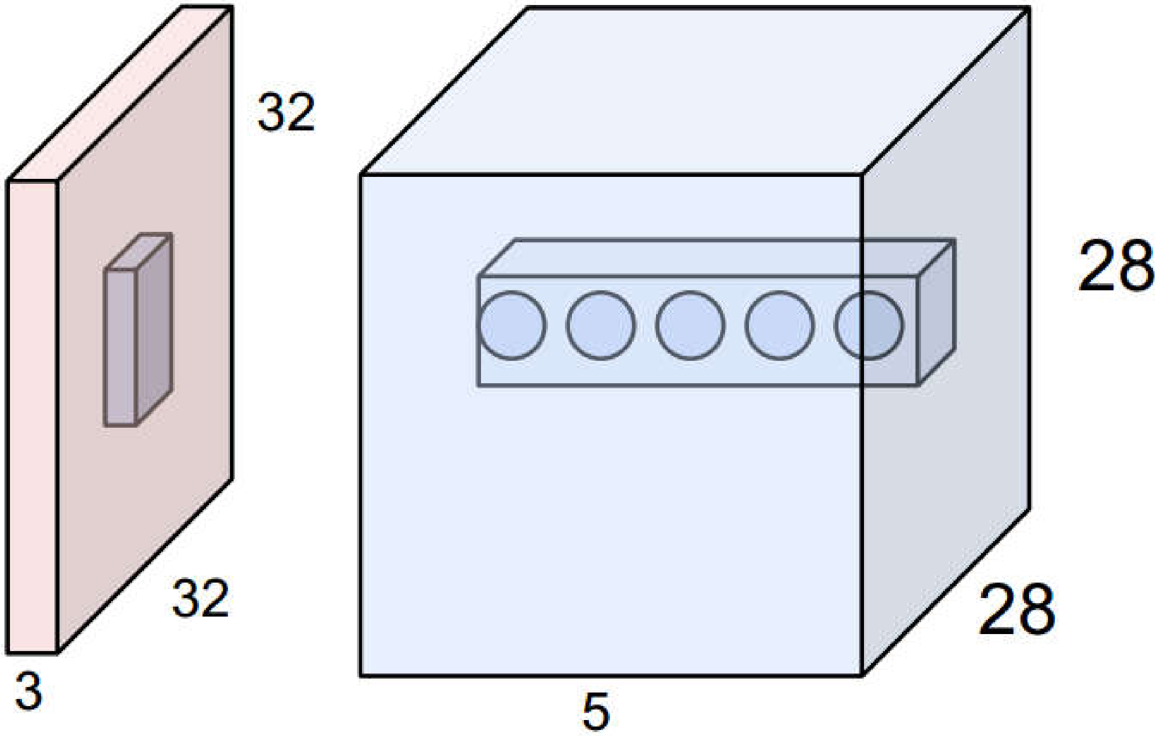
1. Each is connected to a <u>small region in the input</u>
2. All of them <u>share parameters</u>

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

Convolution Layer

The brain/neuron view of CONV Layer





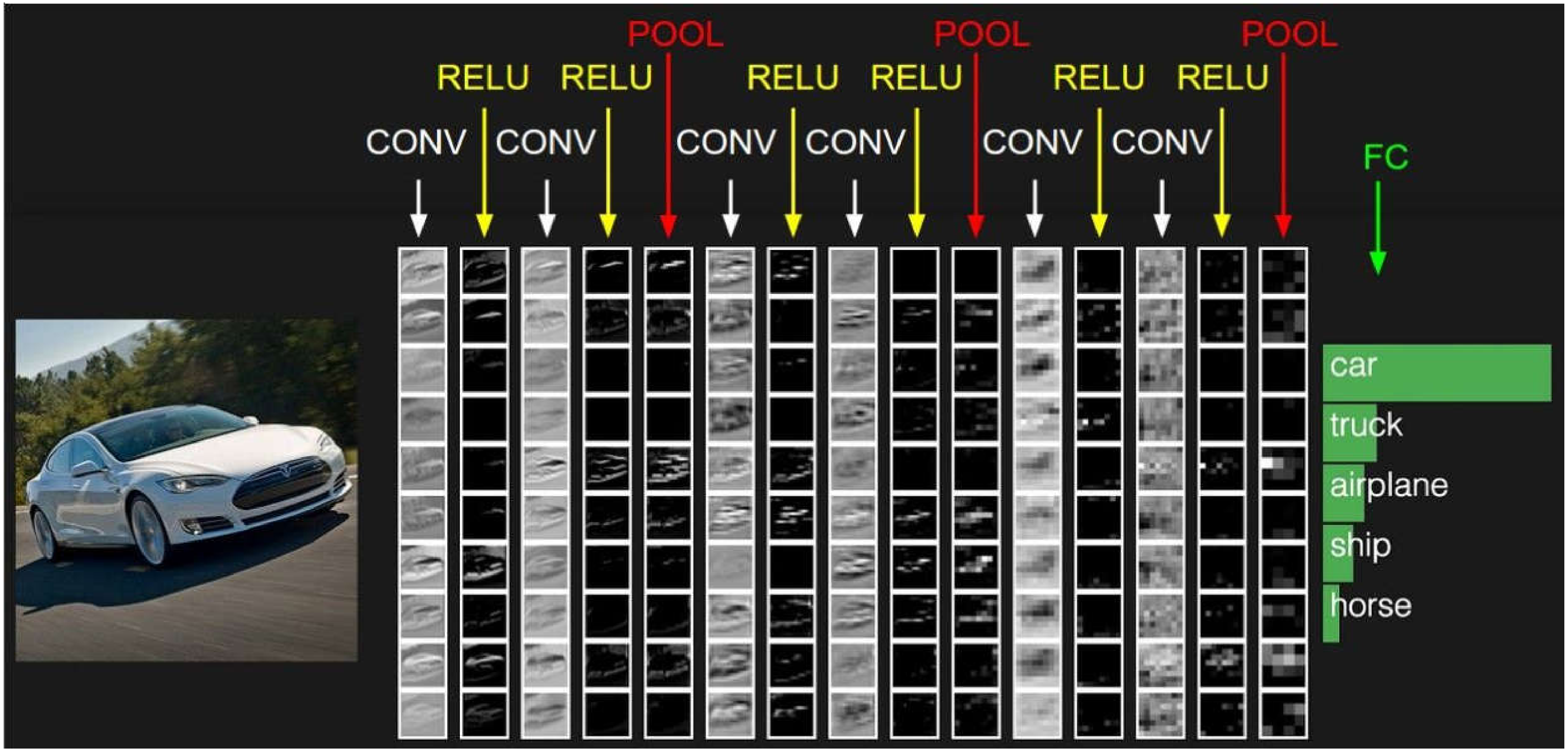
E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

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

- Sharing weights can constrain capacity to control overfitting

Slide from CS231n

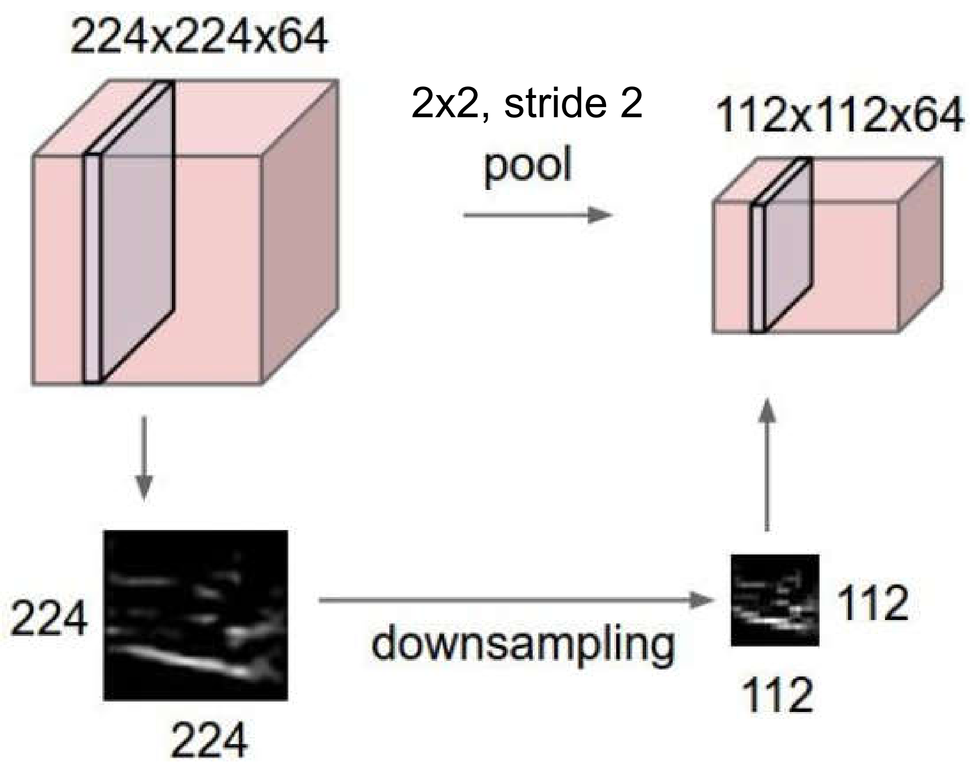
Pooling Layer



Slide from CS231n

Pooling Layer

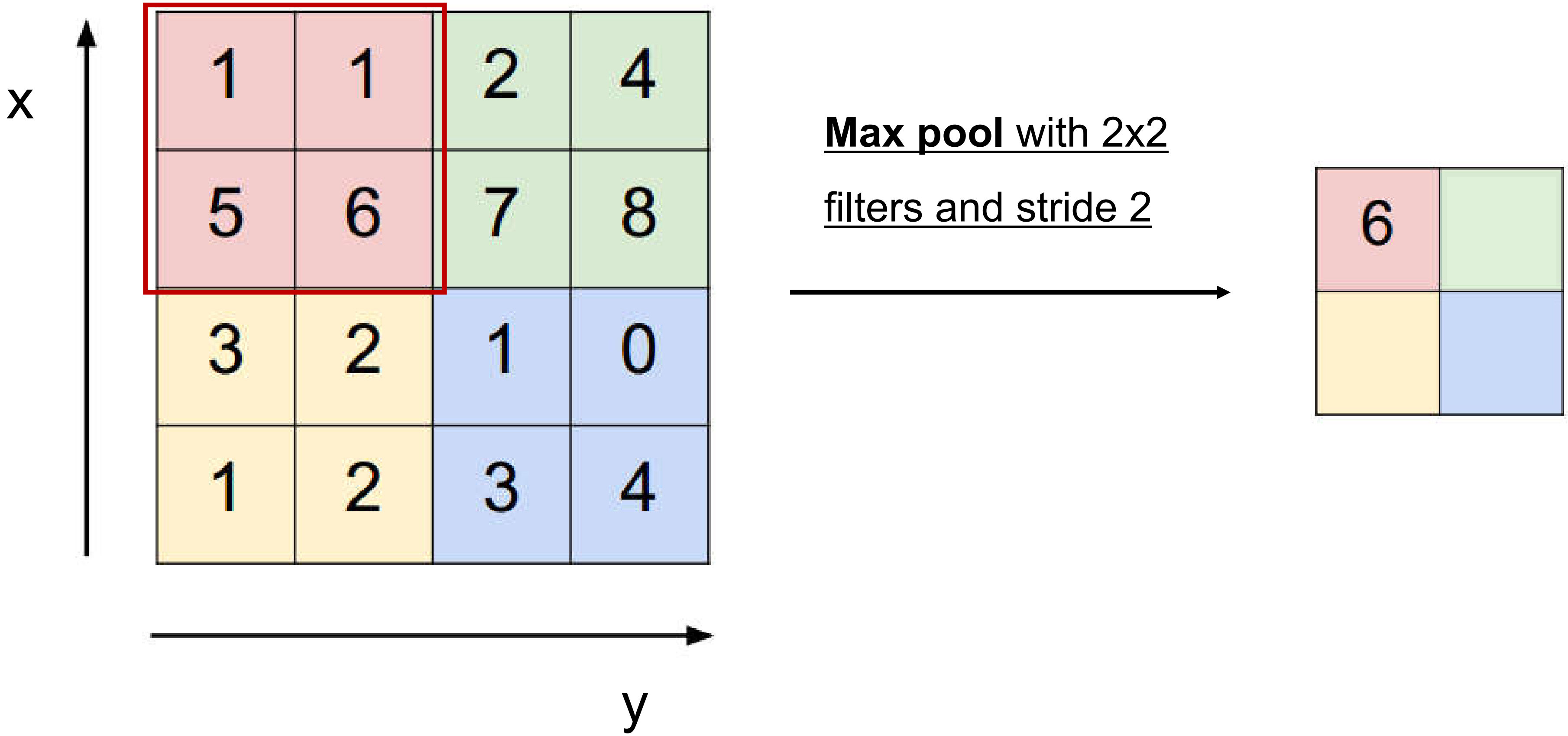
- <u>Makes the representations smaller and more manageable</u>
- Operates over each activation map <u>independently</u>:



Slide from CS231n

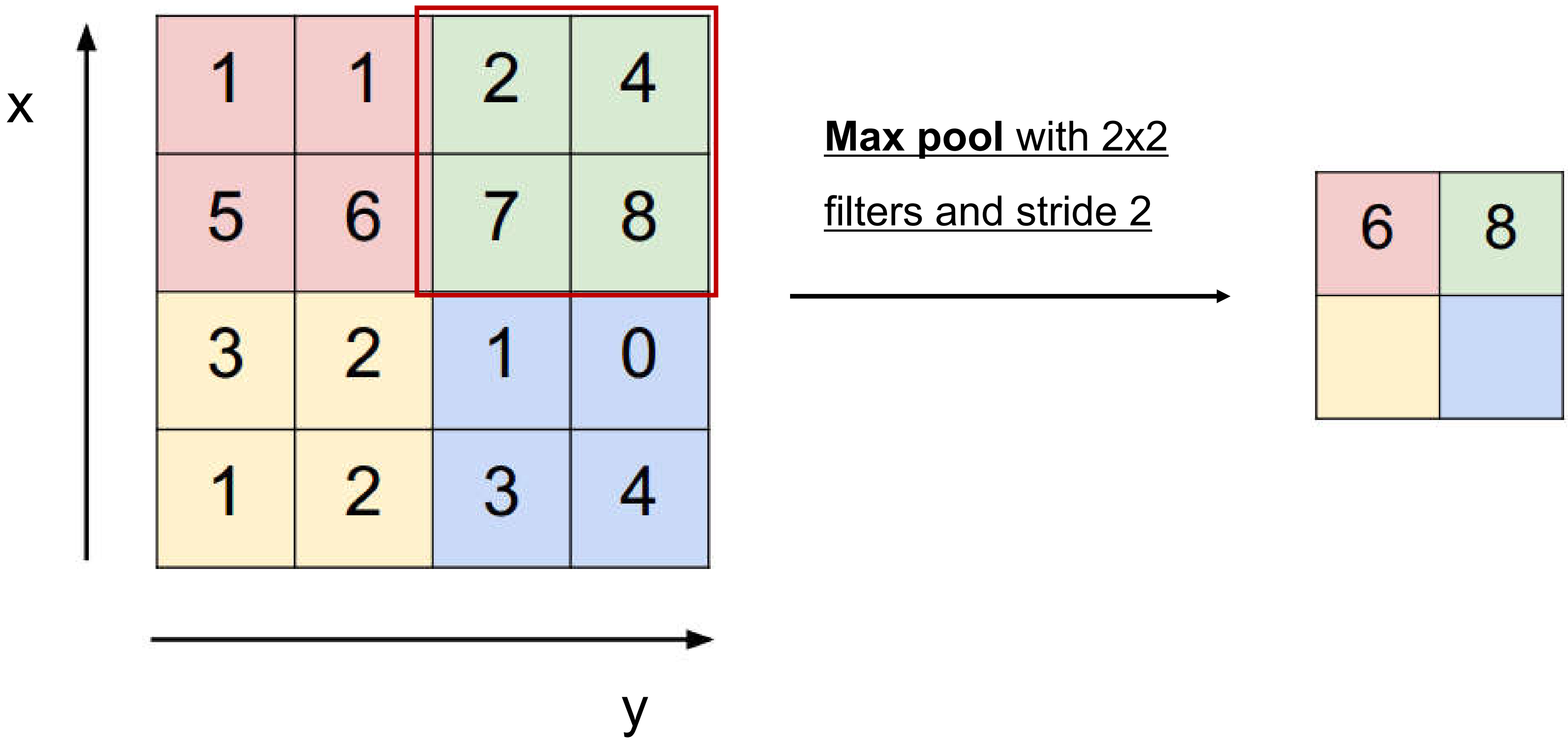
Pooling Layer

- **Max pooling**
- Average pooling



Pooling Layer

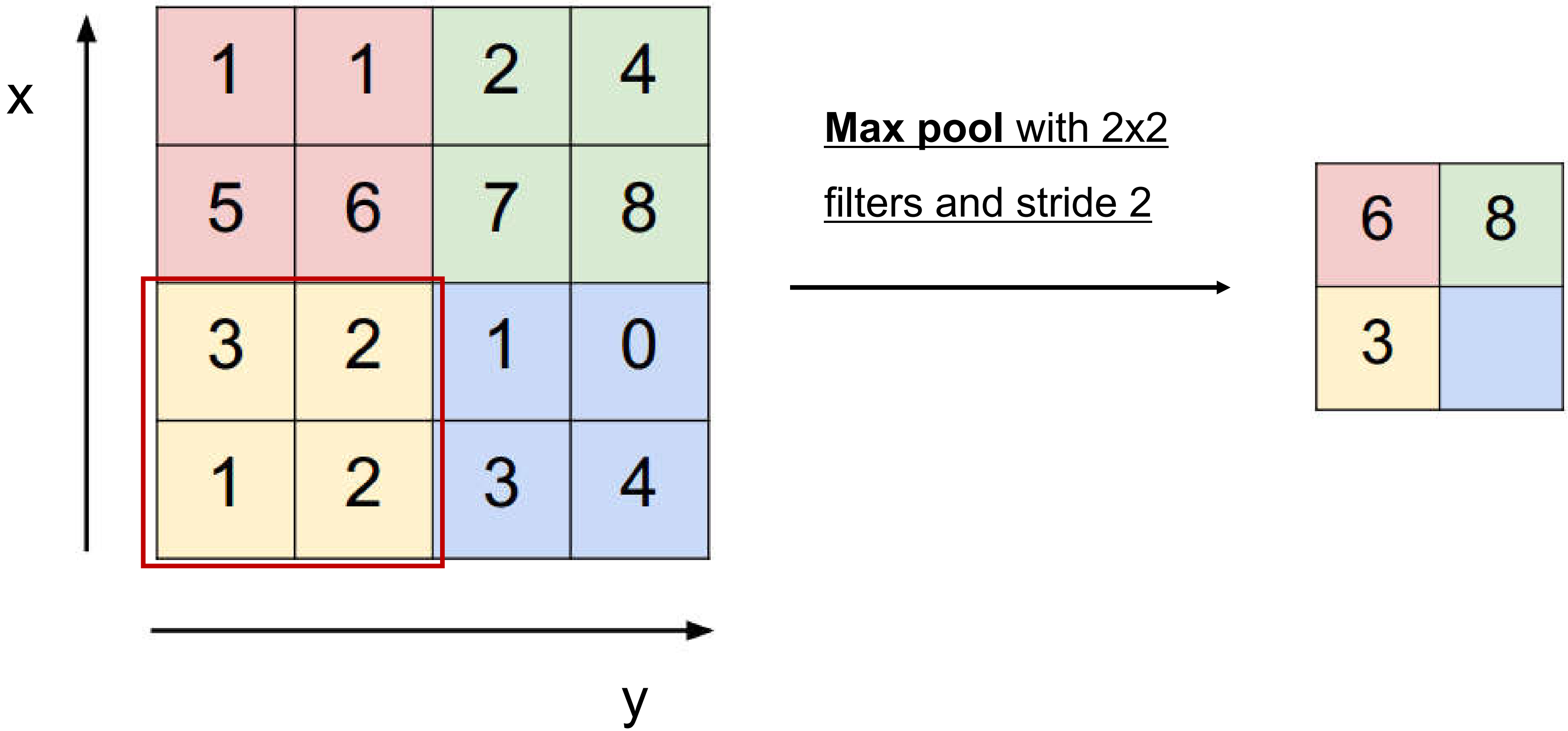
- **Max pooling**
- Average pooling



Slide from CS231n

Pooling Layer

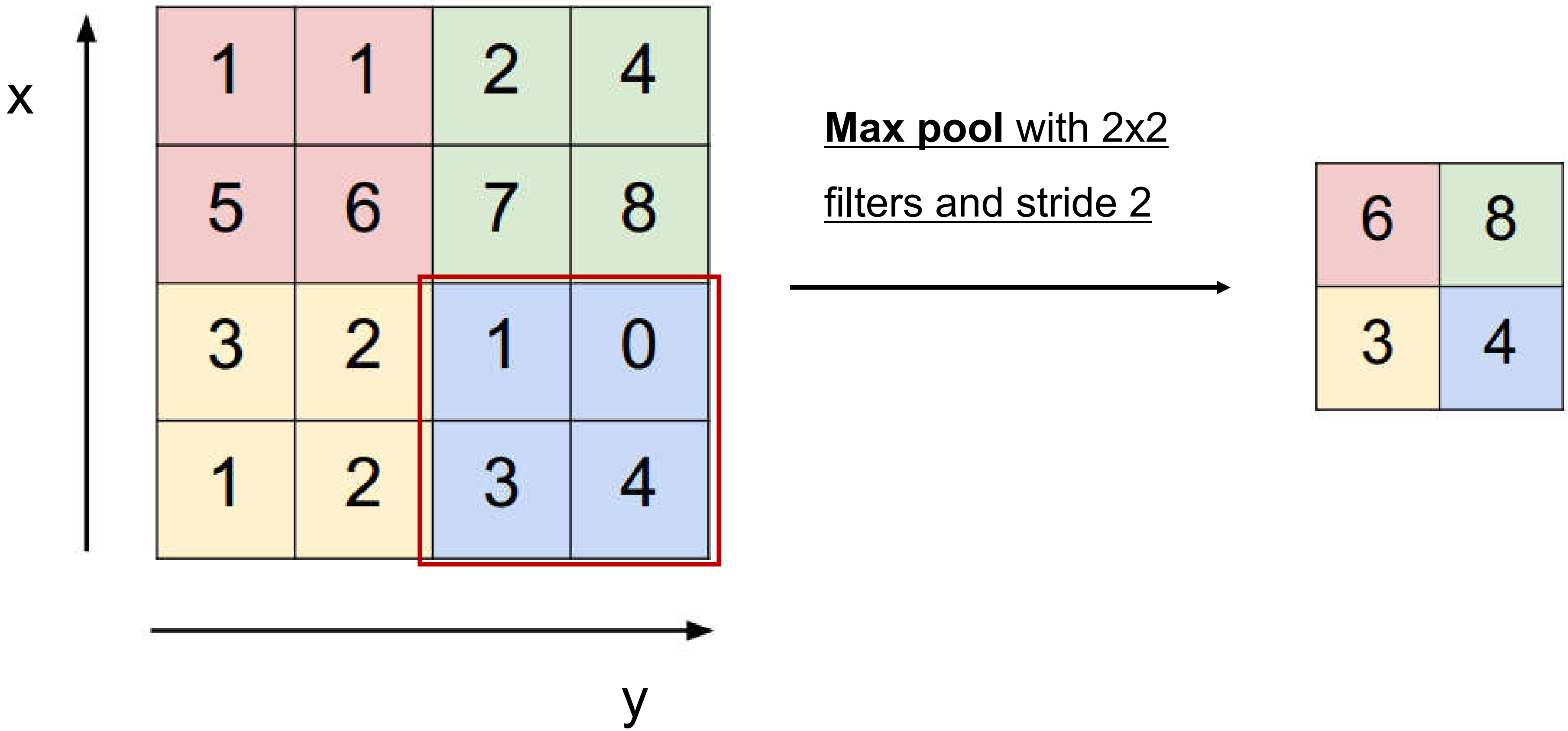
- **Max pooling**
- Average pooling



Slide from CS231n

Pooling Layer

- **Max pooling**
- Average pooling



Summary to Pooling Layer

- 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:
  - $W_2 = (W_1 - F) / S + 1$
  - $H_2 = (H_1 - F) / S + 1$
  - $D_2 = D_1$
- <u>Introduces zero parameters (weights) since it computes a fixed function of the input</u>
- <u>Note that it is not common to use **zero-padding** for pooling layers</u>

Slide from CS231n

Summary to Pooling Layer

- 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 (weights) since it computes a fixed function of the input
- Note that it is not common to use zero-padding for pooling layers

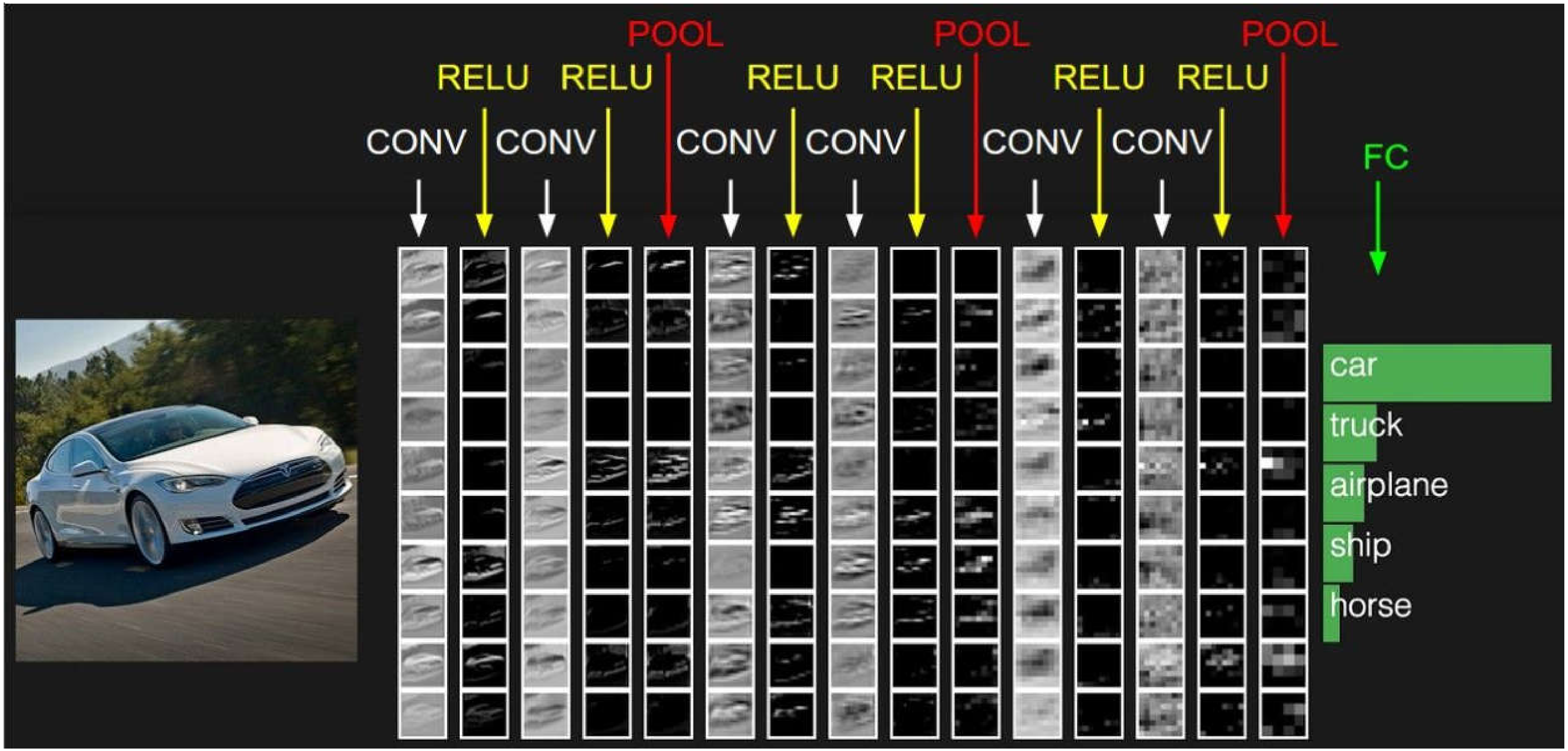
Common settings:

F = 2, S = 2

F = 3, S = 2

Fully Connected Layer (FC Layer)

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



Slide from CS231n

Demo

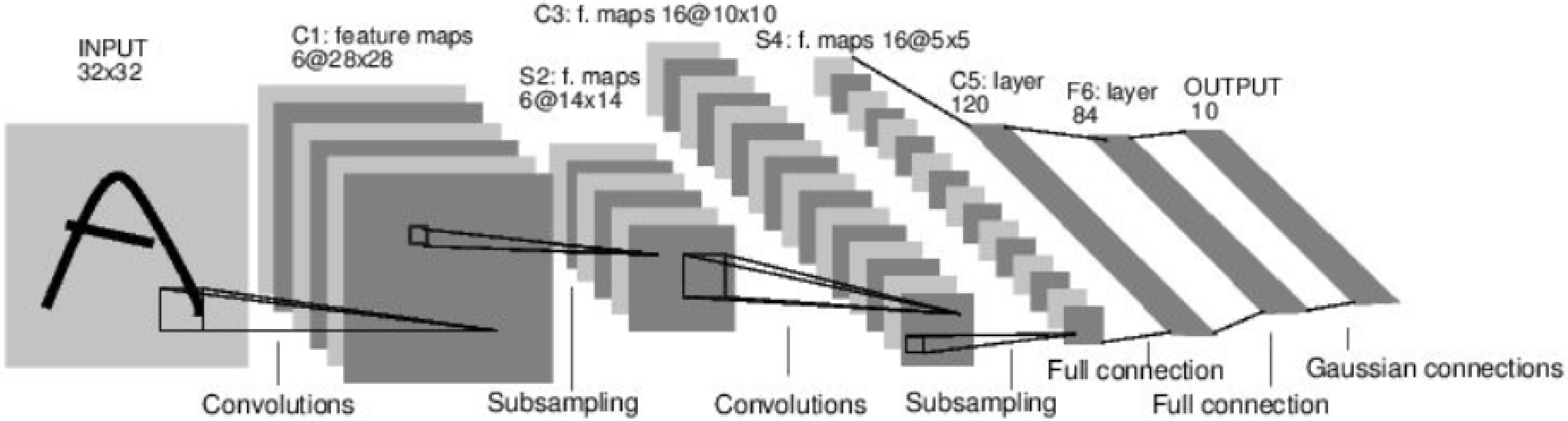
- ConvNetJS demo: training on CIFAR-10

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

Case Study

First without the brain stuff

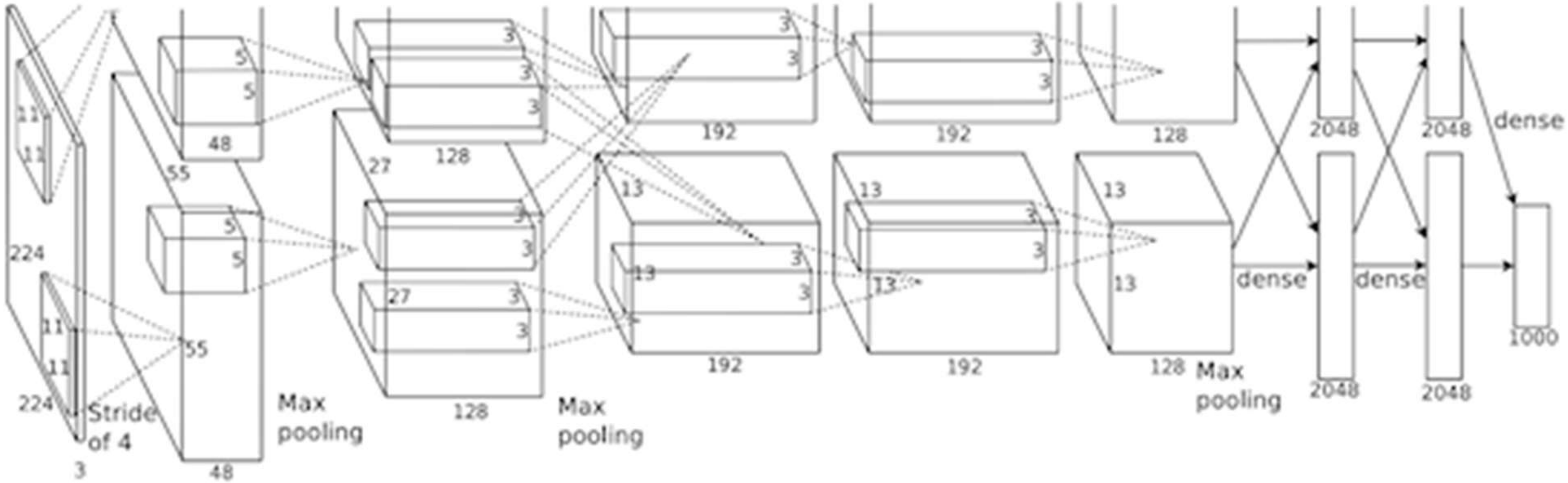
Case Study: LeNet-5



- Conv filters were 5x5, applied a stride 1
- Subsampling (pooling) layers were 2x2 applied at stride 2
- i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC-FC]

[LeCun et al., 1998]

Case Study: AlexNet



- Two steams (historical reason)

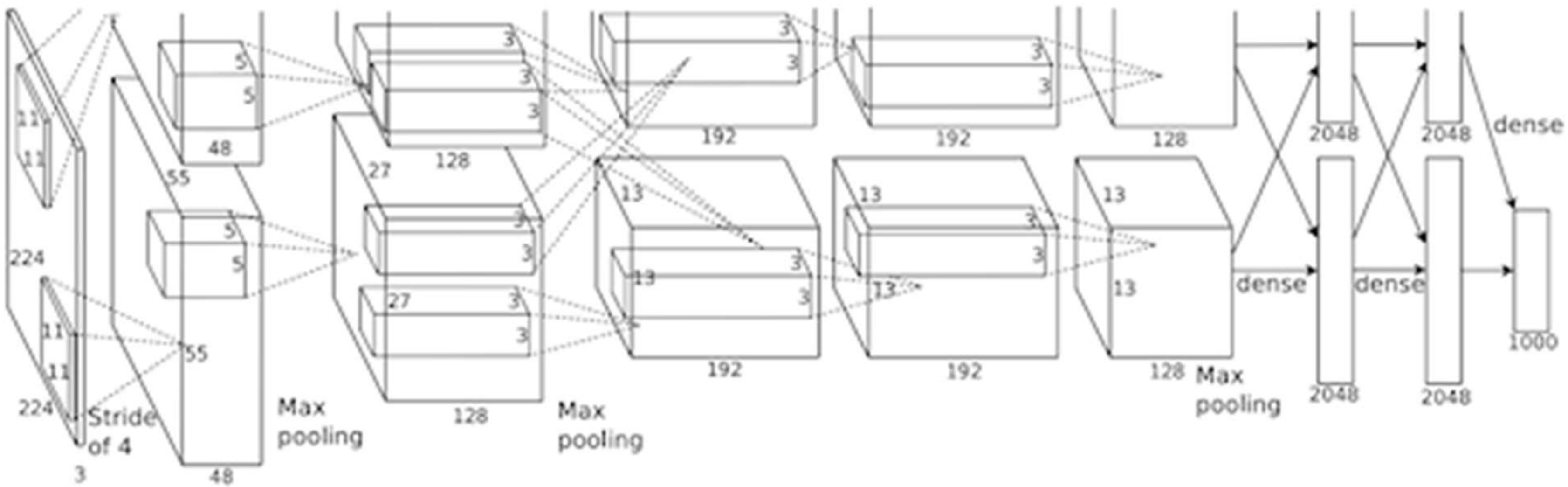
Input 227x227x3 images

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

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

[Krizhevsky et al. 2012]

Case Study: AlexNet



Input 227x227x3 images

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

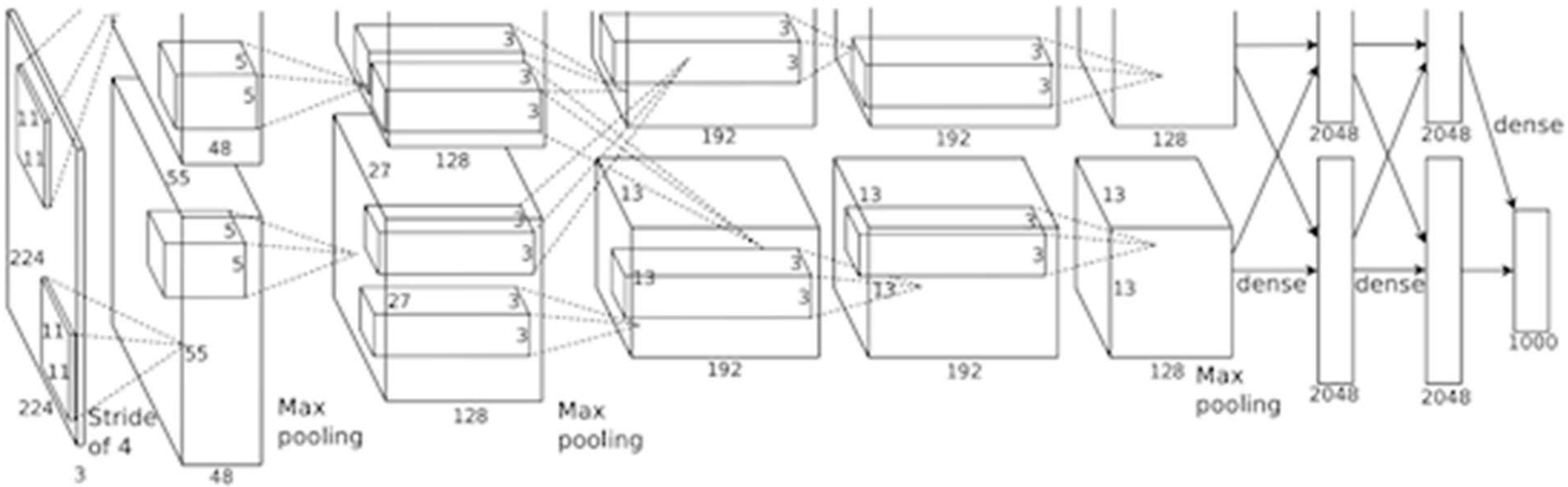
Question: what is the output volume size? Hint: (227 – 11) / 4 + 1 = 55

⇒ Output volume **[55x55x96]**

Question: what is the total number of parameters in this layers?

[Krizhevsky et al. 2012]

Case Study: AlexNet



Input 227x227x3 images

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

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

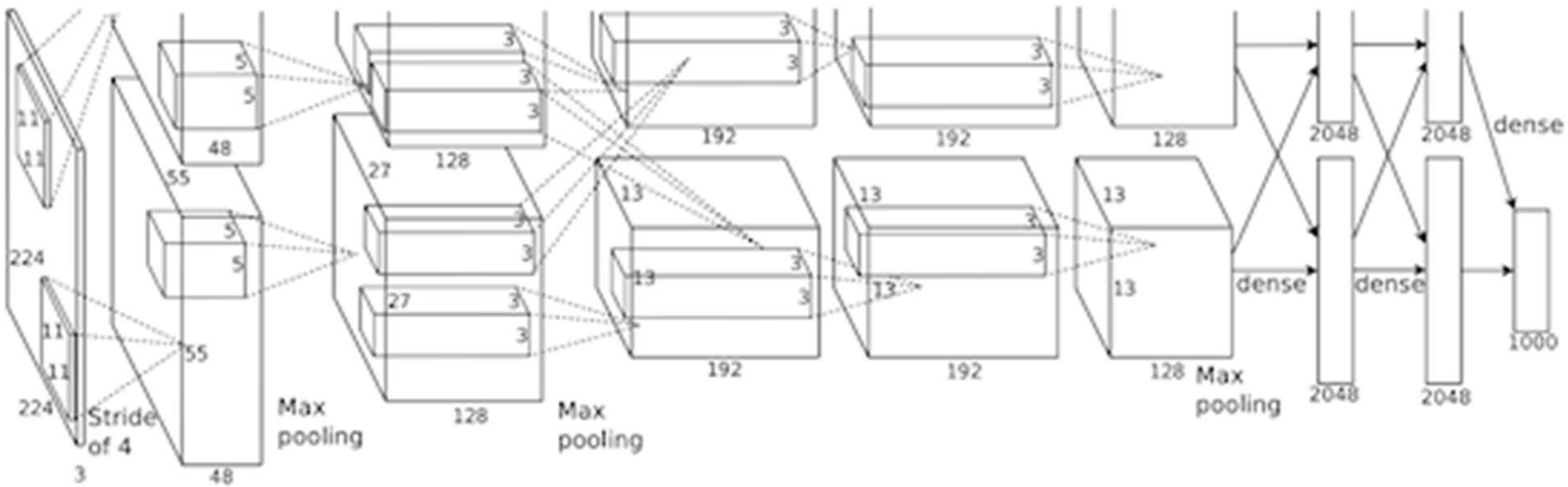
⇒ Output volume **[55x55x96]**

Question: what is the total number of parameters in this layers?

=> (11 * 11 * 3) * 96 + 96 = **35K**

[Krizhevsky et al. 2012]

Case Study: AlexNet



Input 227x227x3 images

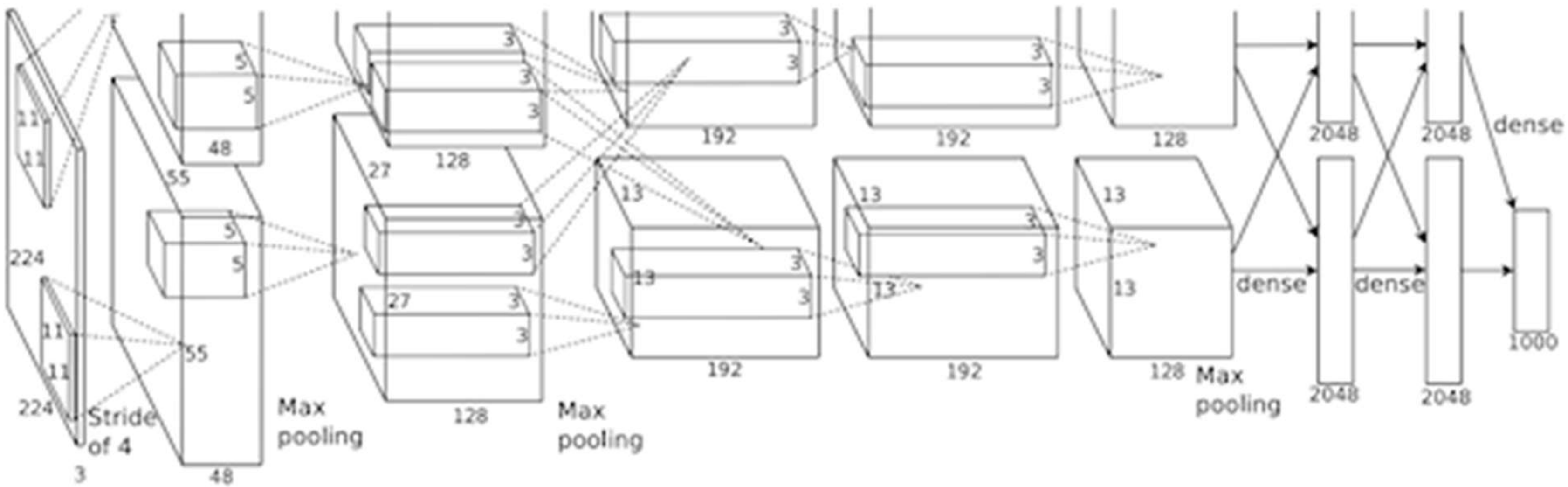
After CONV1: 55x55x96

Second layer (POOL1): <u>3x3 filters applied at stride 2</u>

Question: what is the output volume size? Hint: (55 – 3) / 2 + 1

[Krizhevsky et al. 2012]

Case Study: AlexNet



Input 227x227x3 images

After CONV1: 55x55x96

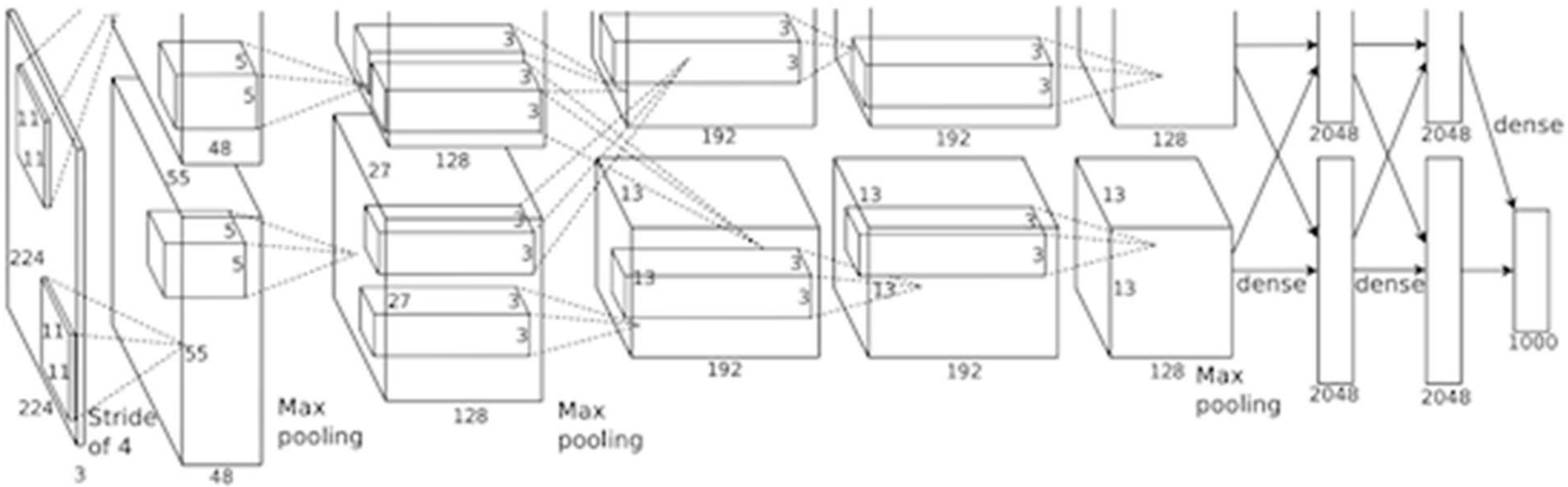
Second layer (POOL1): 3x3 filters applied at stride 2

Question: what is the output volume size? Hint: (55 – 3) / 2 + 1

Output volume: 27x27x96

[Krizhevsky et al. 2012]

Case Study: AlexNet



Input 227x227x3 images

After CONV1: 55x55x96

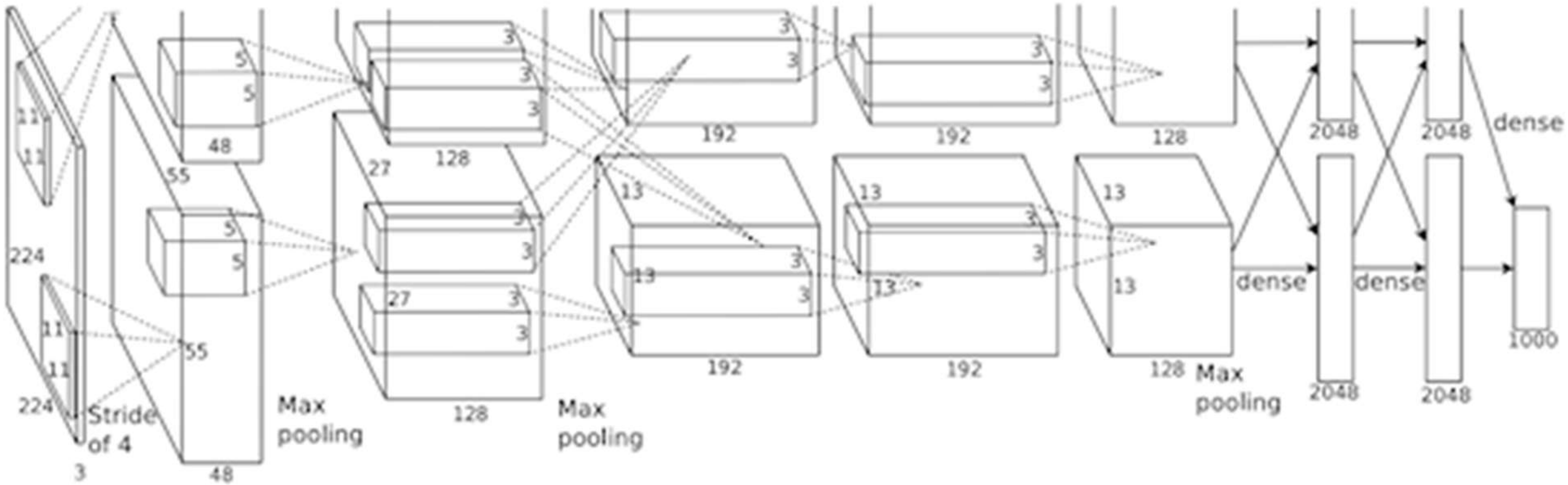
Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

Question: what is the number of parameters in this layer?

[Krizhevsky et al. 2012]

Case Study: AlexNet



Input 227x227x3 images

After CONV1: 55x55x96

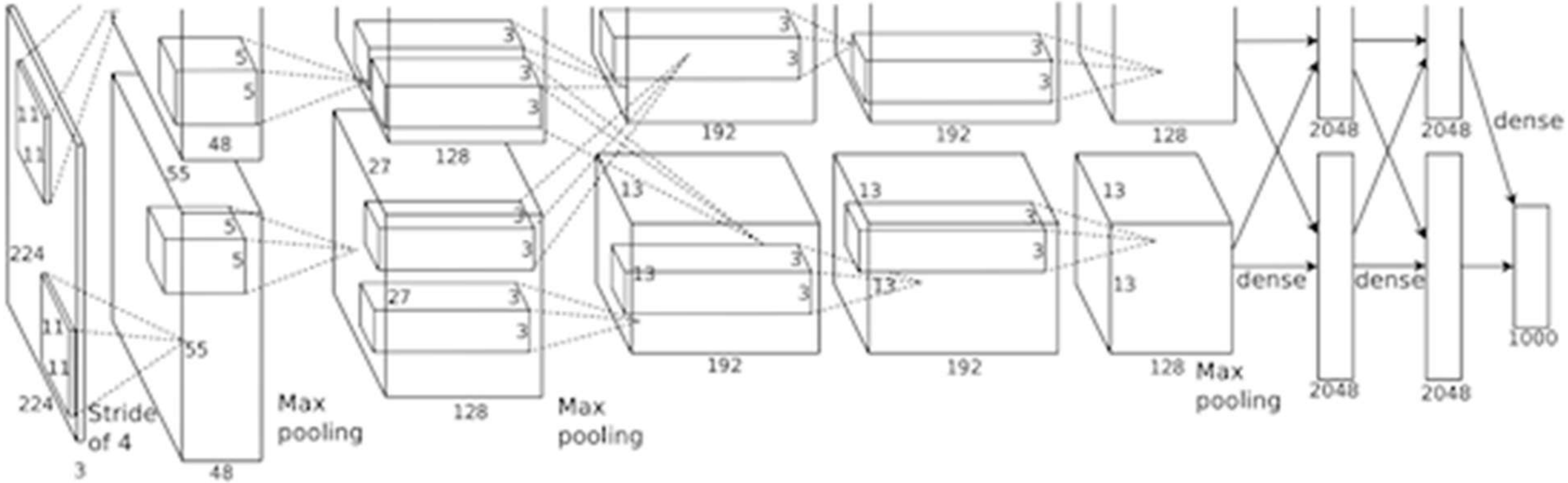
Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

Parameters: 0!

[Krizhevsky et al. 2012]

Case Study: AlexNet



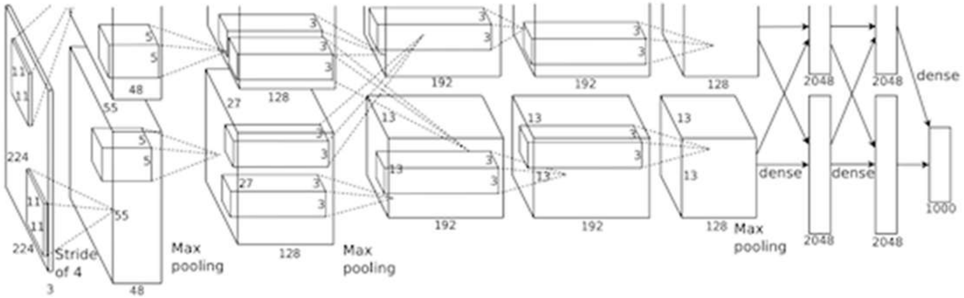
Input 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

[Krizhevsky et al. 2012]

Cause Study: AlexNet

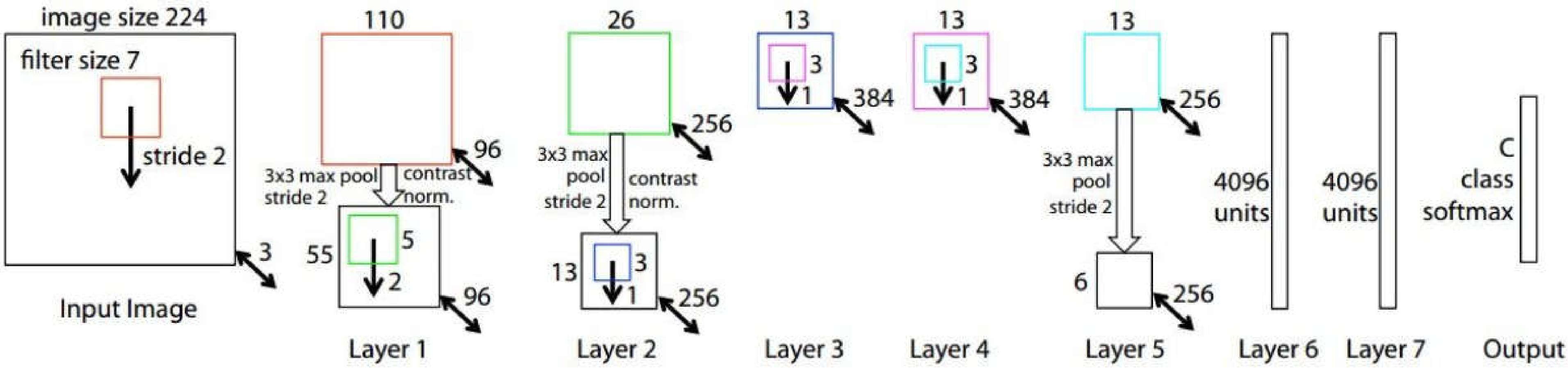


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 91 11x11 filters at stride 4, ad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- First use of ReLU
- Used Norm layers (not common anymore)
- Heavy data augmentation
- Dropout 0.5
- Batch size 128
- SGD momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when validation accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

[Krizhevsky et al. 2012]

Case Study: ZFNet



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3, 4, 5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

- "FC7" features
- Zeiler created a company ,and worked on this a bit more and ended up with **11.7%**

[Zeiler and Fergus, 2013]

Case Study: VGGNet

Only 3x3 CONV stride 1, pad 1 and 2x2

MAX POOL stride 2

Best model

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					

11.2% top 5 error in ILSVRC 2013 -> **7.3** top 5 error

[Simonyan and Zisserman 2014]

Case Study: VGGNet

Most memory is in early CONV

INPUT: [224x224x3] memory: 224*224*3=150k
CONV3-64: [224x224x64] memory: **224*224*64=3.2M**
CONV3-64: [224x224x64] memory: **224*224*64=3.2M**
POOL2: [112x1112x64] memory: 112*112*64=800K
CONV3-128: [112x112x128] memory: 112*112*128=1.6M
CONV3-128: [112x112x128] memory: 112*112*128=1.6M
POOL2: [56x56x128] memory: 56*56*128=400K
CONV3-256: [56x56x256] memory: 56*56*256=800K
CONV3-256: [56x56x256] memory: 56*56*256=800K
CONV3-256: [56x56x256] memory: 56*56*256=800K
POOL2: [28x28x256] memory: 28*28*256=200K
CONV3-512: [28x28x512] memory: 28*28*512=400K
CONV3-512: [28x28x512] memory: 28*28*512=400K
CONV3-512: [28x28x512] memory: 28*28*512=400K
POOL2: [14x14x512] memory: 14*14*512=100K
CONV3-512: [14x14x512] memory: 14*14*512=100K
CONV3-512: [14x14x512] memory: 14*14*512=100K
CONV3-512: [14x14x512] memory: 14*14*512=100K
POOL2: [7x7x512] memory: 7*7*512=25K
FC: [1x1x4096] memory: 4096
FC: [1x1x4096] memory: 4096
FC: [1x1x1000] memory: 1000
TOTAL memory: 24M * 4 bytes ~= 93MB/image (only forward! ~*2 for backward)

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

[Simonyan and Zisserman 2014]

Case Study: VGGNet

INPUT: [224x224x3] params: 0 **(not counting biases)**
CONV3-64: [224x224x64] params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] params: (3*3*64)*64=36,864
POOL2: [112x1112x64] params: 0
CONV3-128: [112x112x128] params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] params: 0
CONV3-256: [56x56x256] params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] params: 0
CONV3-512: [28x28x512] params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] params: 0
CONV3-512: [14x14x512] params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] params: 0
FC: [1x1x4096] params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] params: 4096*4096 = 16,777,216
FC: [1x1x1000] params: 4096 * 1000 = 4,096,000

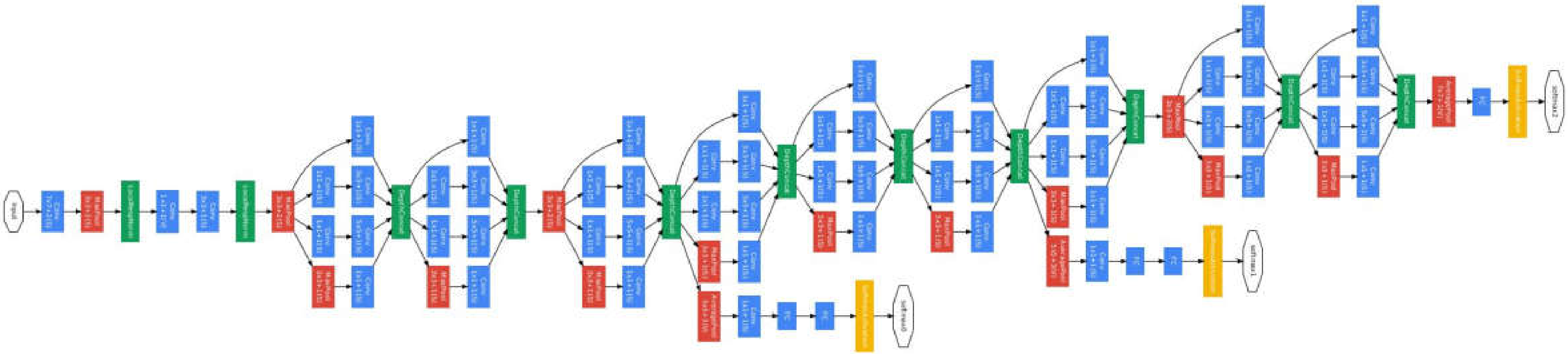
TOTAL params: 138M parameters

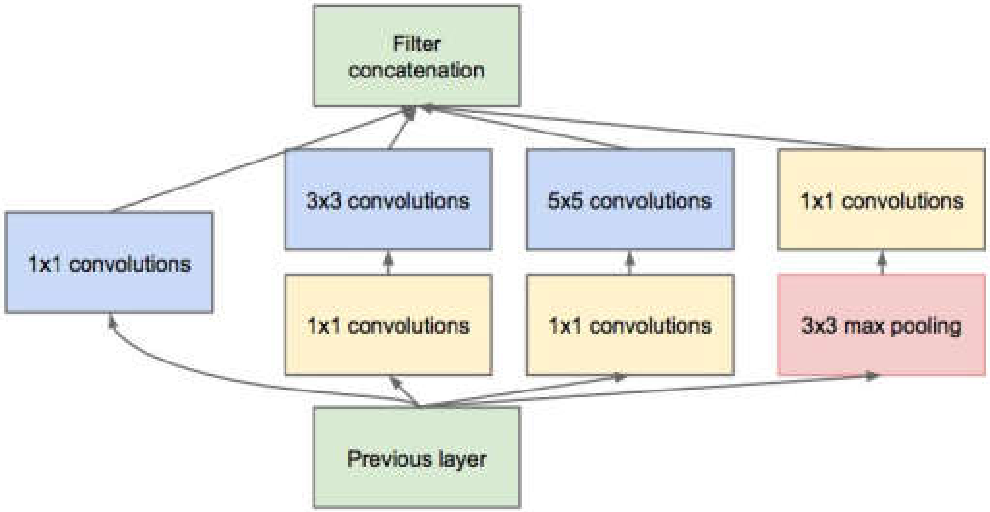
ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			

Most params are in late FC

[Simonyan and Zisserman 2014]

Case Study: GoogLeNet





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

[Szegedy et al., 2014]

Case Study: 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 most heavy FC layer)

Compared to AlexNet:

- 12x less params
- 2x more compute
- 6.67% (vs. 15.4%)

Slide from CS231n

Case Study: ResNet

ILSVRC 2015 winner (3.6% top 5 error)

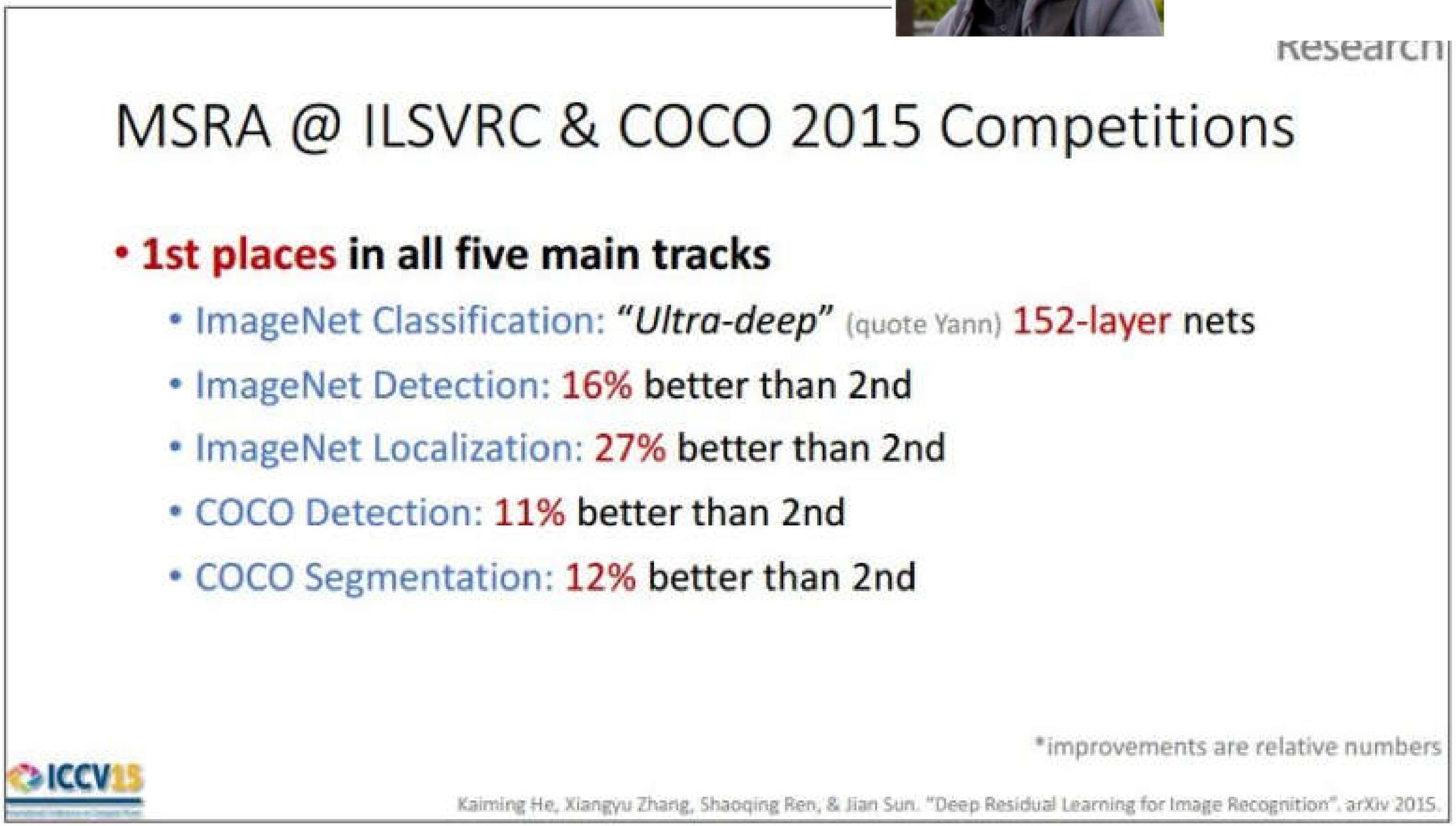
Microsoft Research Asia

Kaiming He 何恺明

Research Scientist

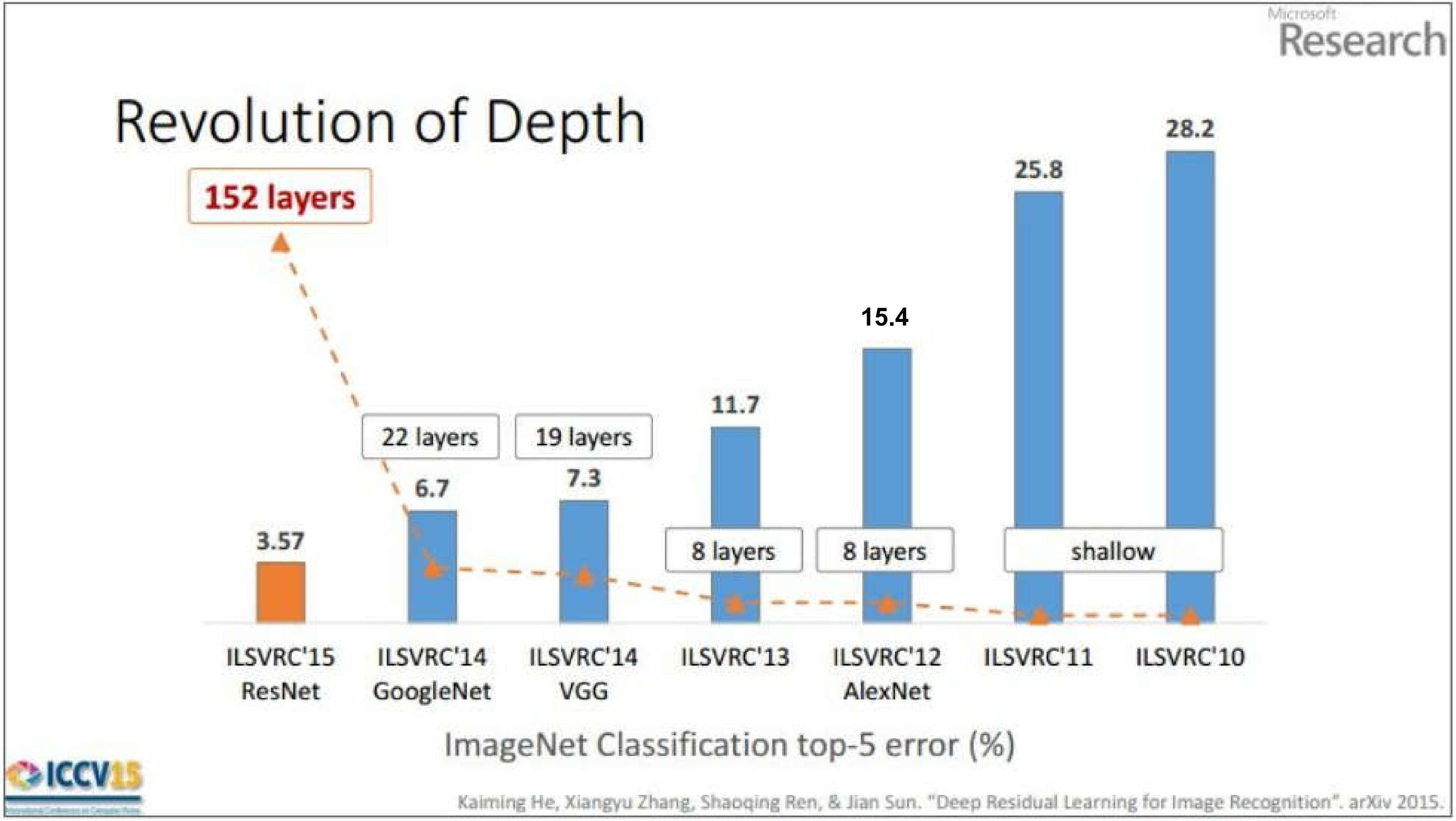
Facebook AI Research (FAIR), Menlo Park, CA

kaiminghe@fb.com



Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

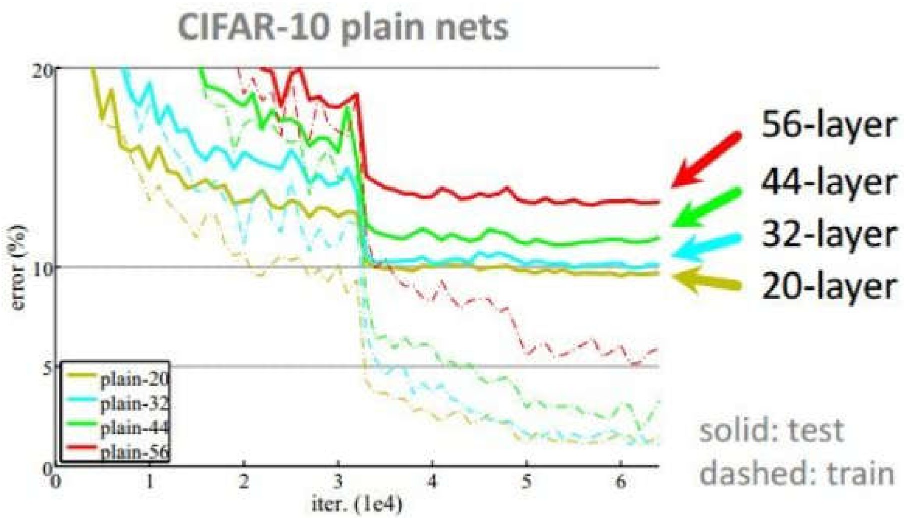
Case Study: ResNet

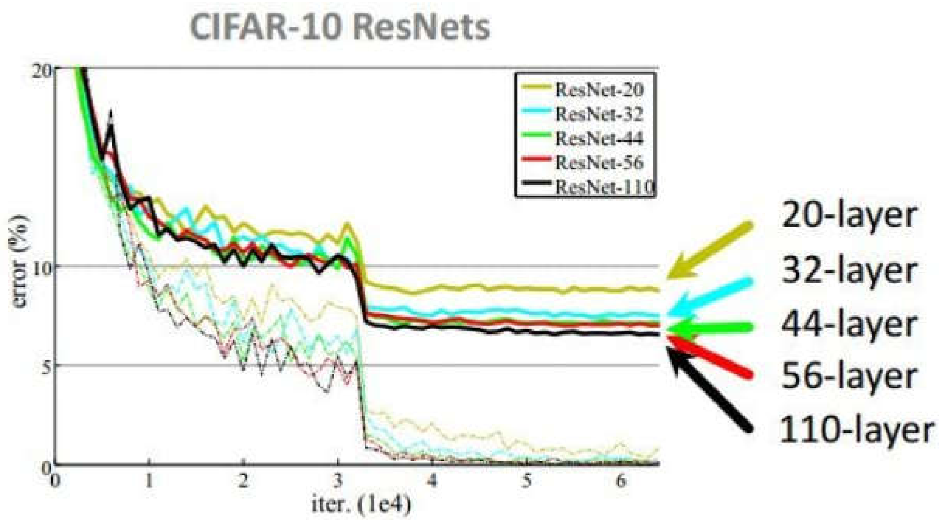


Slide from Kaiming He's presentation

Case Study: ResNet

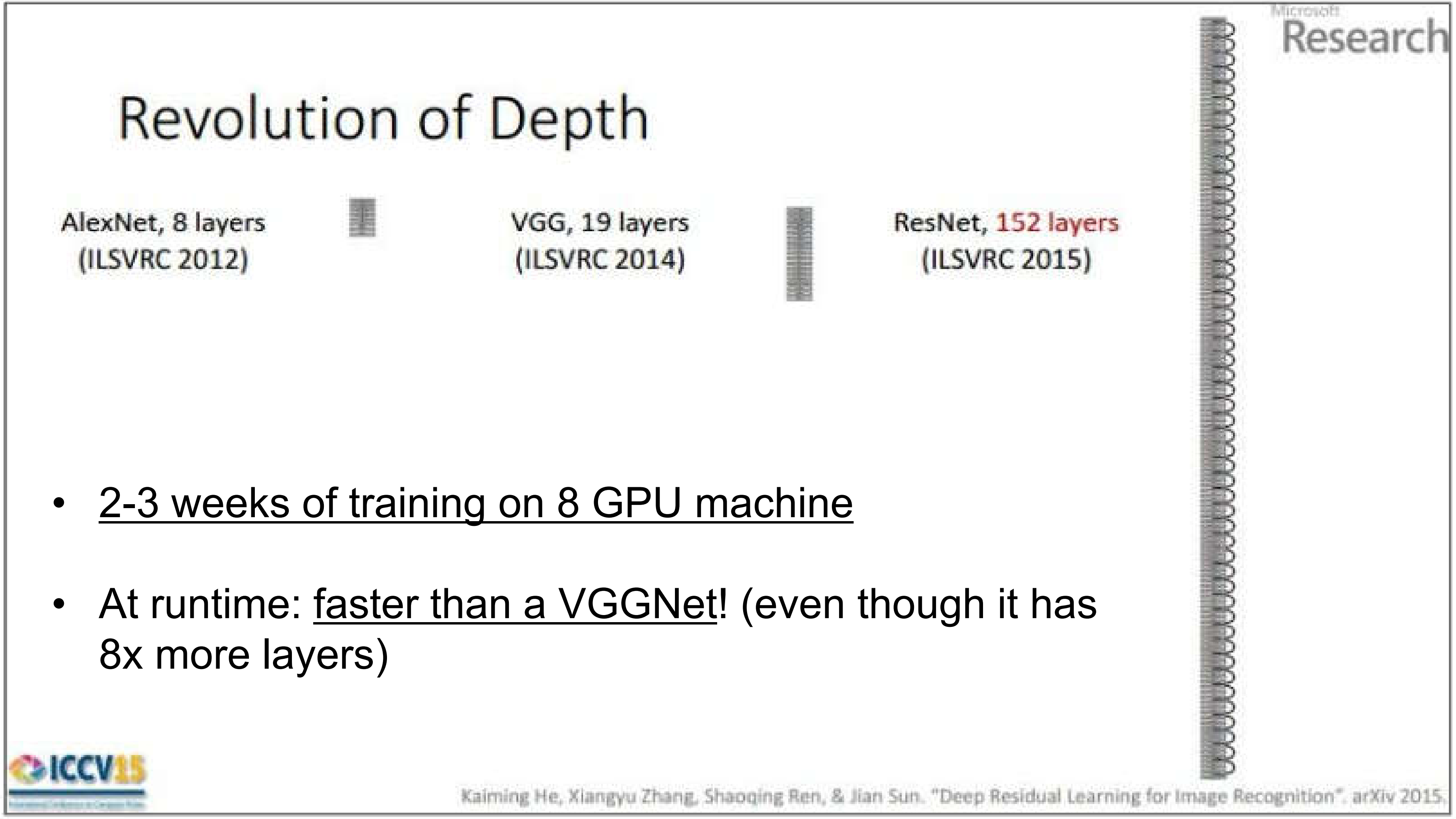
CIFAR-10 experiments





Case Study: ResNet

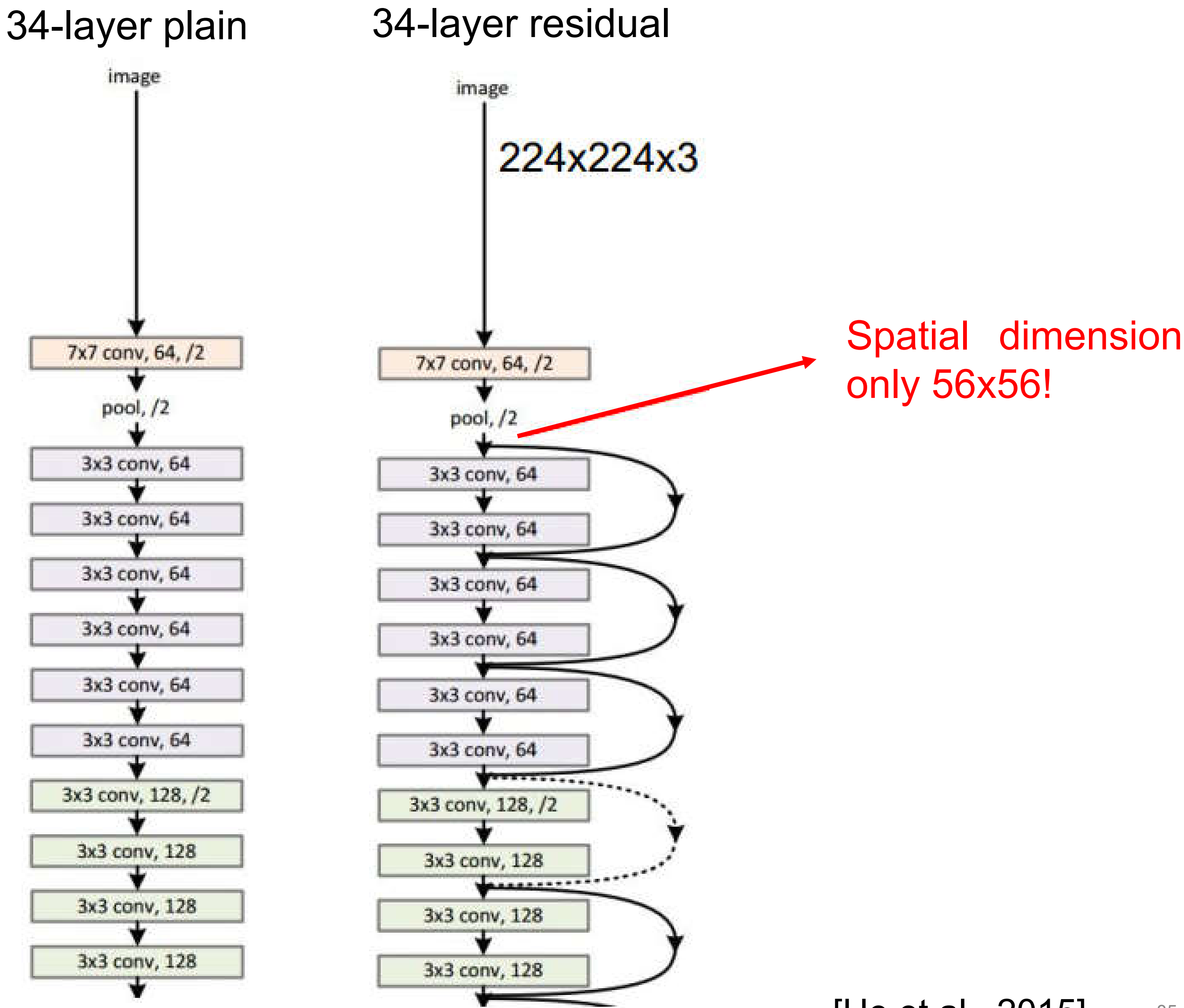
ILSVRC 2015 winner (3.6% top 5 error)



- 2-3 weeks of training on 8 GPU machine
- At runtime: faster than a VGGNet! (even though it has 8x more layers)

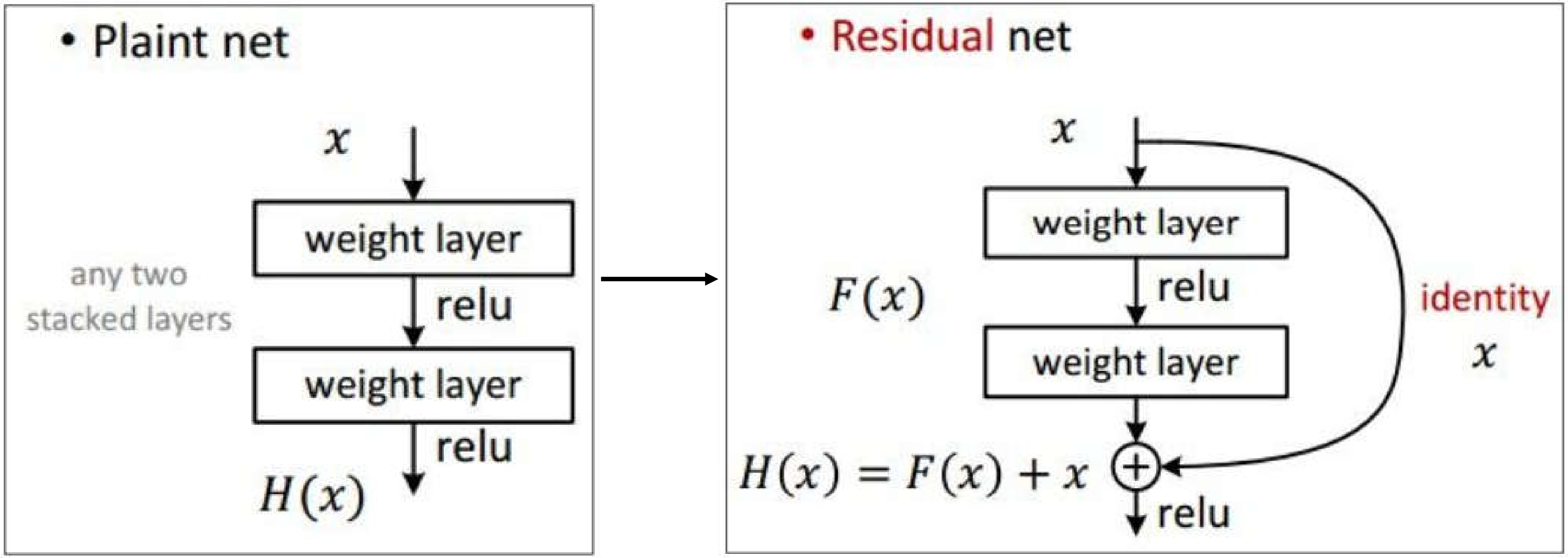
Slide from Kaiming He's recent presentation

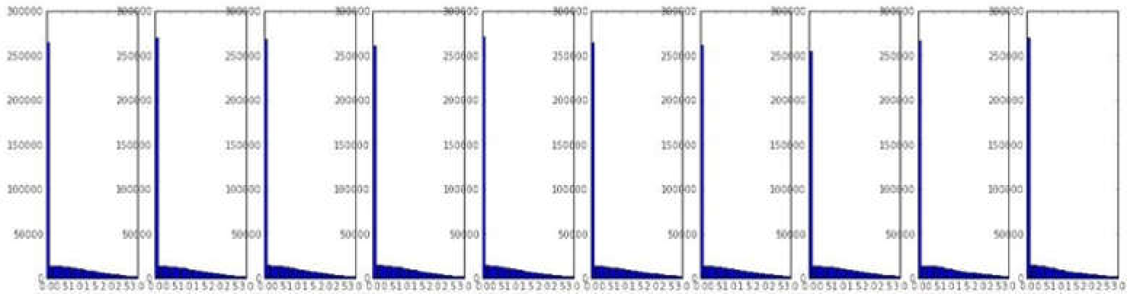
Case Study: ResNet



[He et al., 2015]

Case Study: ResNet





He et al., 2015

Case Study: ResNet

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- **No dropout used**

In Batch-norm paper claims that if you use bath-norm, there's less of a need for dropout…

Slide from CS231n

Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- **Trend towards getting rid of POOL/FC layers (just CONV)**
- Typical architecture look like
  - [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX
  - Where N is usually up to ~5, M is large, 0<=K<=2. (but recent advances such as ResNet/GooLeNet challenge this paradigm)

Slide from CS231n

Thank you for your attention!