Deep Neural Networks

Machine Learning and Pattern Recognition

(Largely based on slides from Fei-Fei Li & Justin Johnson & Serena Yeung)

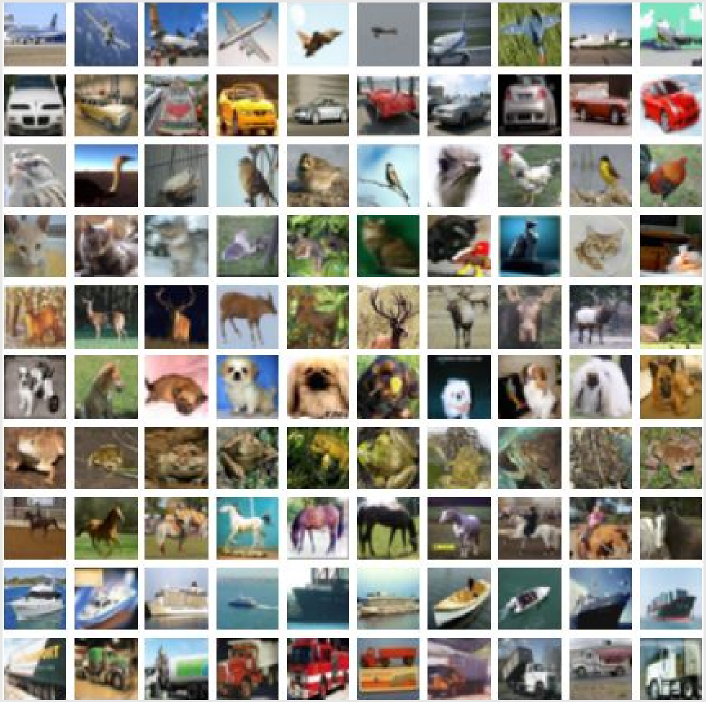
Prof. Sandra Avila

Institute of Computing (IC/Unicamp)

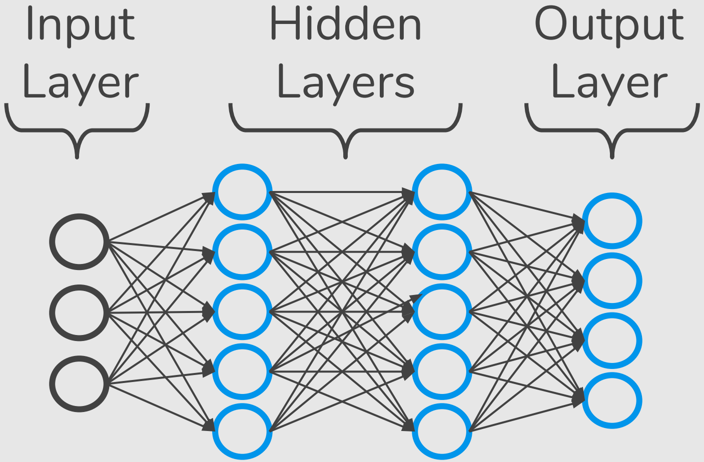
MC886/MO444, October 24, 2017

Convolutional Neural Networks (CNNs)

Fully Connected Layer

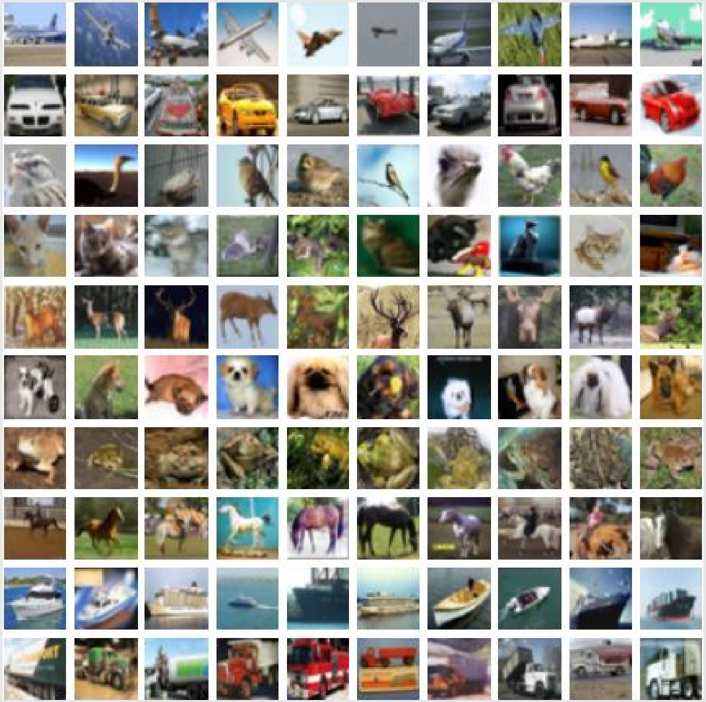


CIFAR-10

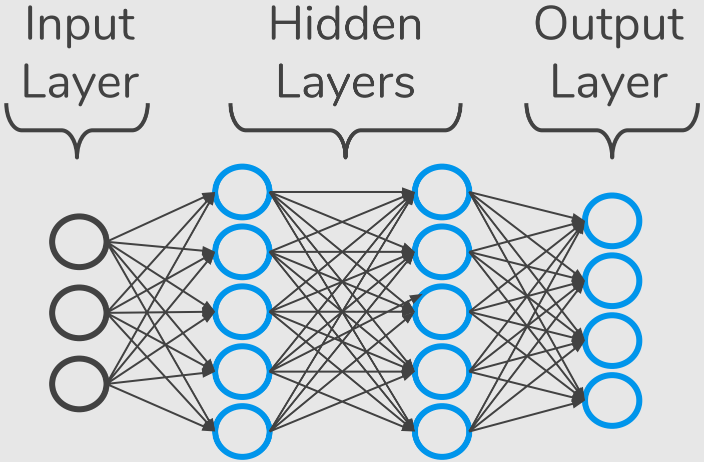


$32 \times 32 \times 3$ image $\Rightarrow$ stretch to 3072×1

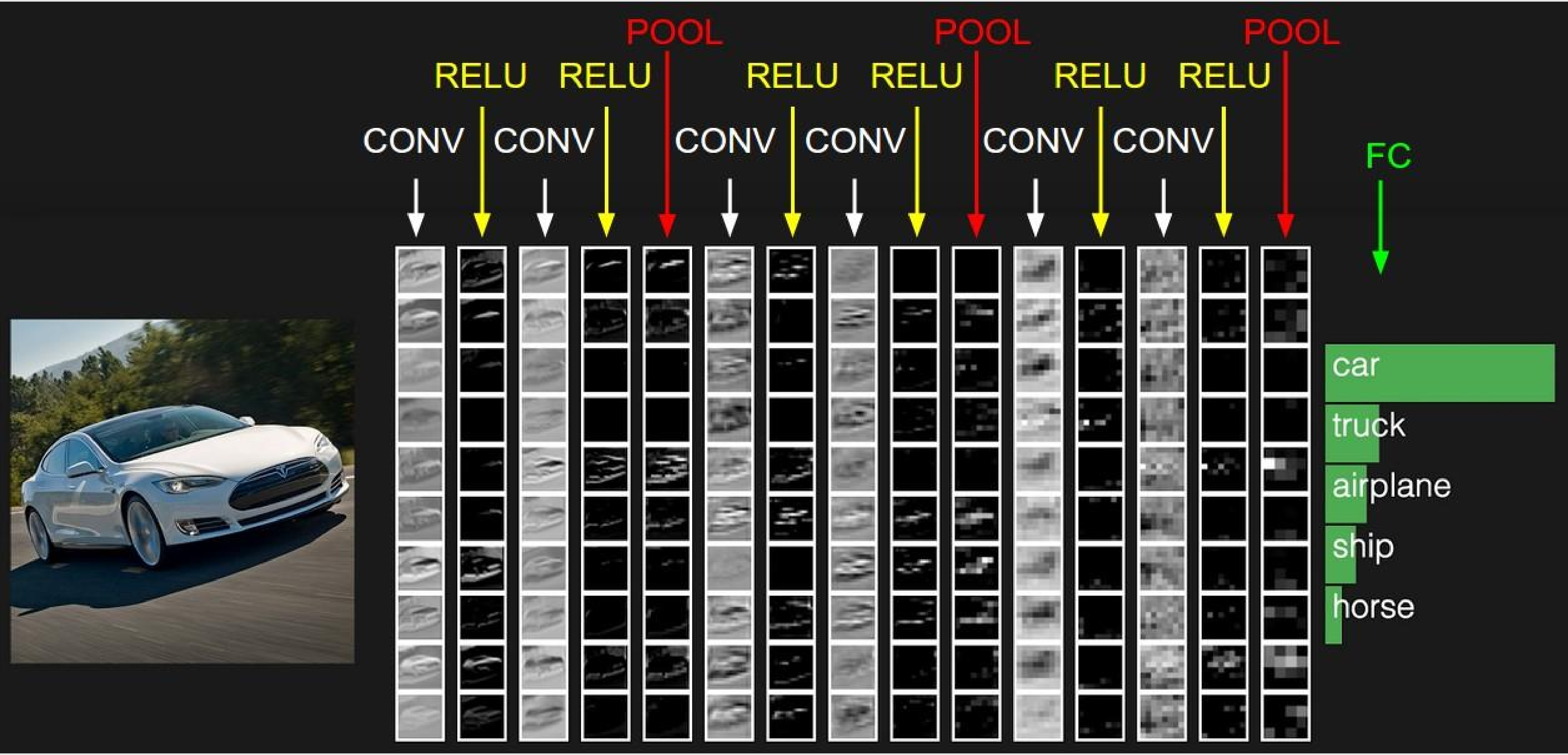
Fully Connected Layer



CIFAR-10



32 × 32 × 3 image ⇒ stretch to 3072 × 1



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

What is a Convolution?

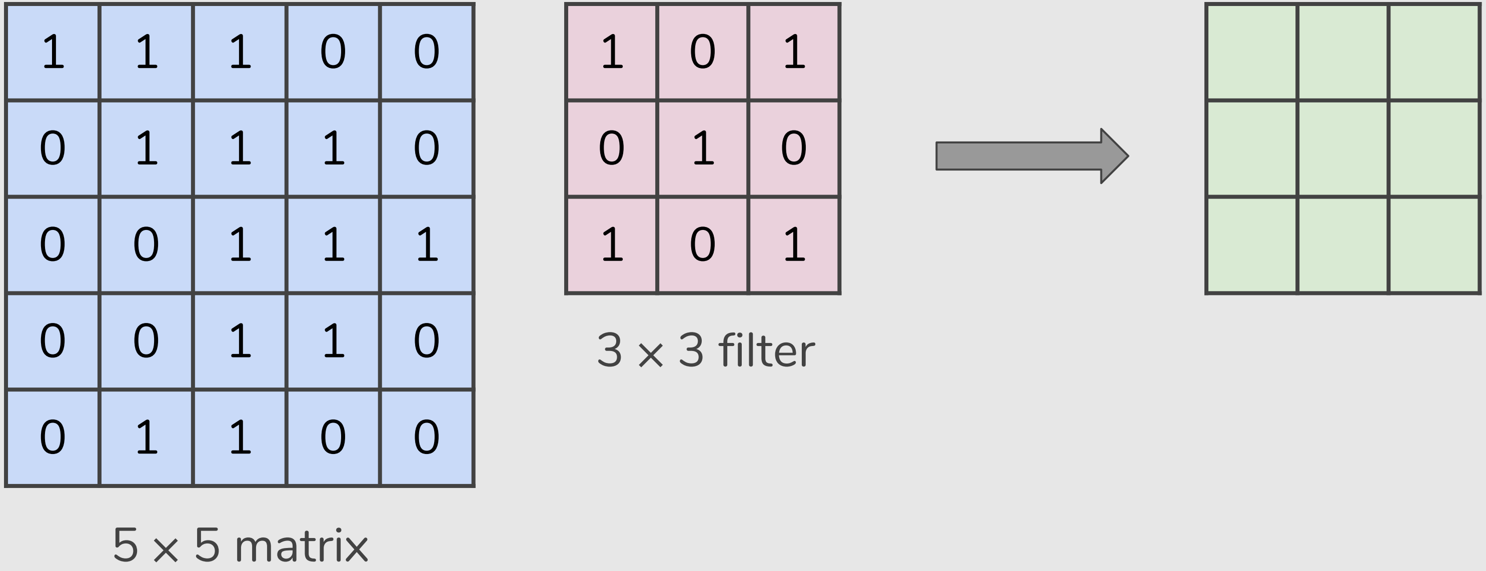
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5×5 matrix

1	0	1
0	1	0
1	0	1

3×3 filter

What is a Convolution?



What is a Convolution?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 × 5 matrix

1	0	1
0	1	0
1	0	1

3 × 3 filter

4		

1*1 + 1*0 + 1*1 +
0*0 + 1*1 + 1*0 +
0*1 + 0*0 + 1*1 = 4

What is a Convolution?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 × 5 matrix

1	0	1
0	1	0
1	0	1

3 × 3 filter

→

4	3	

$$1*1 + 1*0 + 0*1 + \\ 1*0 + 1*1 + 1*0 + \\ 0*1 + 1*0 + 1*1 = 3$$

What is a Convolution?

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

5 × 5 matrix

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

3 × 3 filter

$$\longrightarrow \begin{bmatrix} 4 & 3 & 4 \\ & & \\ & & \end{bmatrix}$$

$$1*1 + 0*0 + 0*1 + 1*0 + 1*1 + 0*0 + 1*1 + 1*0 + 1*1 = 4$$

What is a Convolution?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 × 5 matrix

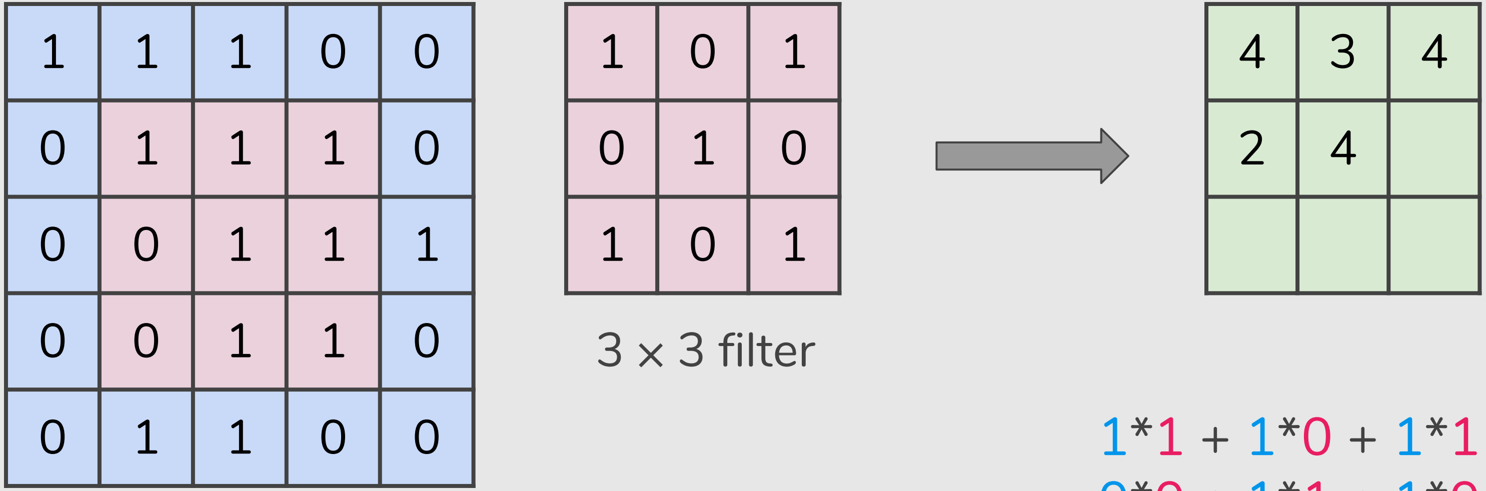
1	0	1
0	1	0
1	0	1

3 × 3 filter

4	3	4
2		

$$0*1 + 1*0 + 1*1 +$$
$$0*0 + 0*1 + 1*0 +$$
$$0*1 + 0*0 + 1*1 = 2$$

What is a Convolution?



5 × 5 matrix

3 × 3 filter

$$1*1 + 1*0 + 1*1 +$$
$$0*0 + 1*1 + 1*0 +$$
$$0*1 + 1*0 + 1*1 = 4$$

What is a Convolution?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 × 5 matrix

1	0	1
0	1	0
1	0	1

3 × 3 filter

→

4	3	4
2	4	3

$$1*1 + 1*0 + 0*1 + \\ 1*0 + 1*1 + 1*0 + \\ 1*1 + 1*0 + 0*1 = 3$$

What is a Convolution?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 × 5 matrix

1	0	1
0	1	0
1	0	1

3 × 3 filter

4	3	4
2	4	3
2		

$$0*1 + 0*0 + 1*1 +$$
$$0*0 + 0*1 + 1*0 +$$
$$0*1 + 1*0 + 1*1 = 2$$

What is a Convolution?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 × 5 matrix

1	0	1
0	1	0
1	0	1

3 × 3 filter

4	3	4
2	4	3
2	3	

0*1 + 1*0 + 1*1 +
0*0 + 1*1 + 1*0 +
1*1 + 1*0 + 0*1 = 3

What is a Convolution?

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 × 5 matrix

1	0	1
0	1	0
1	0	1

3 × 3 filter

$\rightarrow$

4	3	4
2	4	3
2	3	4

$$1*1 + 1*0 + 1*1 +$$
$$1*0 + 1*1 + 0*0 +$$
$$1*1 + 0*0 + 0*1 = 4$$

What is a Convolution?

0	1	0
1	-4	1
0	1	0

What is a Convolution?

Edge Detection

0	1	0
1	-4	1
0	1	0

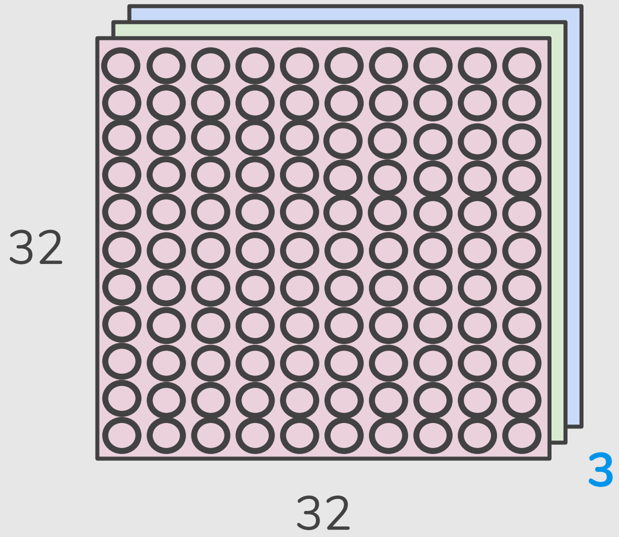
What is a Convolution?

Emboss

-2	-1	0
-1	1	1
0	1	2

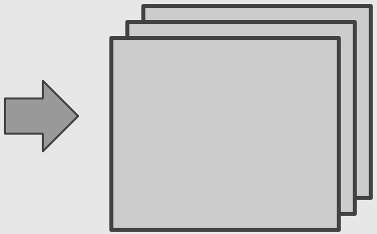
Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure



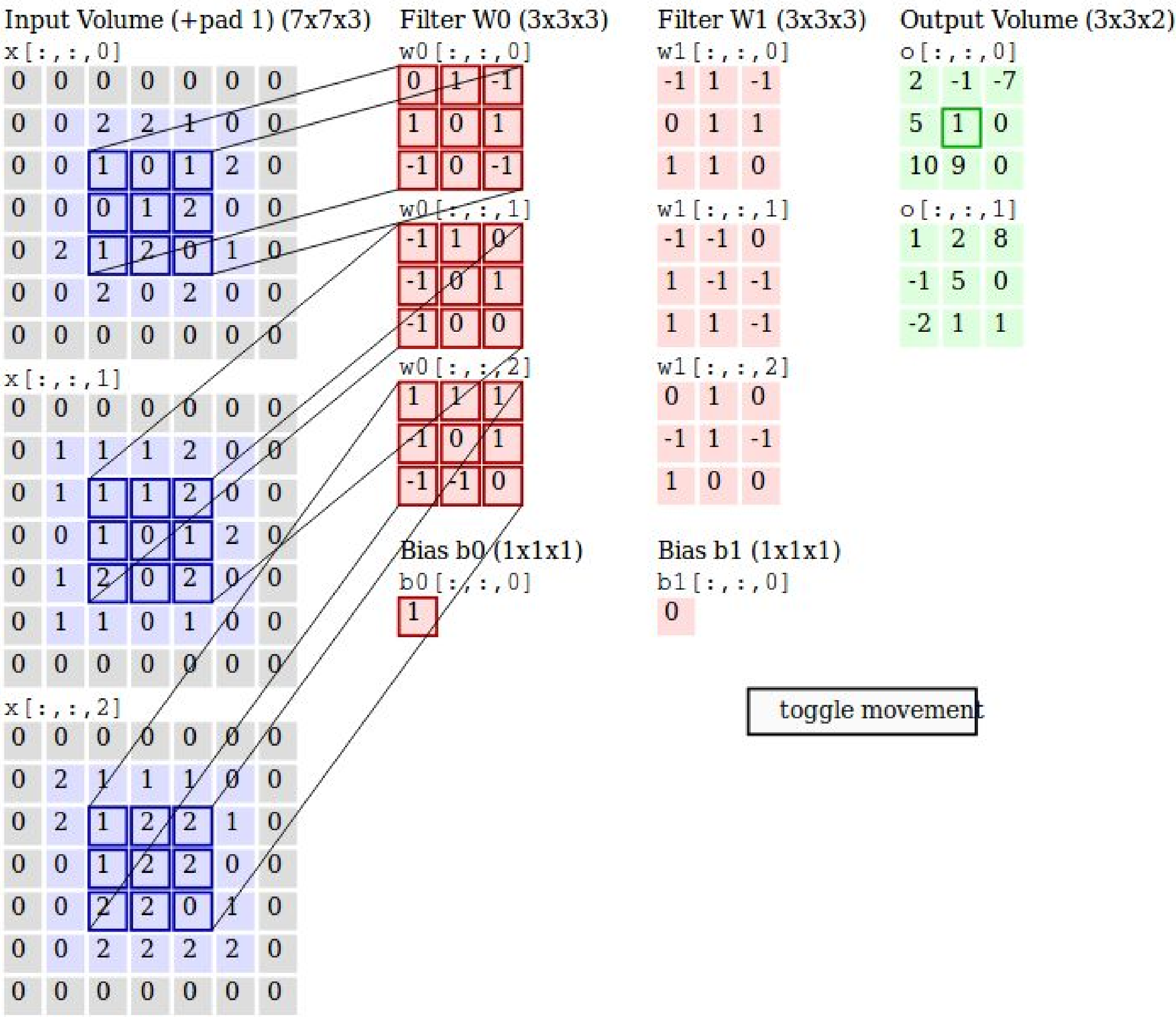
Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

$5 \times 5 \times$ **3** filter



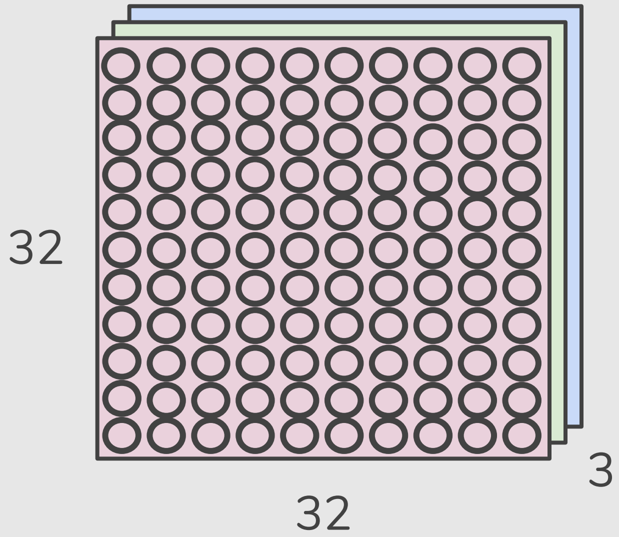
Filters always extend the full depth of the input volume

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



Convolution Layer

$32 \times 32 \times 3$ image $\Rightarrow$ preserve spatial structure

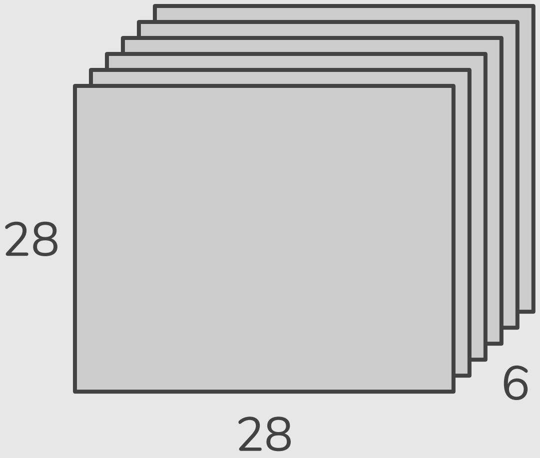


Convolve (slide) over
all spatial locations

$32 \times 32 \times 3$ image
$5 \times 5 \times 3$ filter

If we had 6 $5 \times 5 \times 3$ filters ...

6 activation maps



Convolutional Layer

The size of the **Activation Map** (or Feature Map or Convolved Feature) is controlled by three parameters:

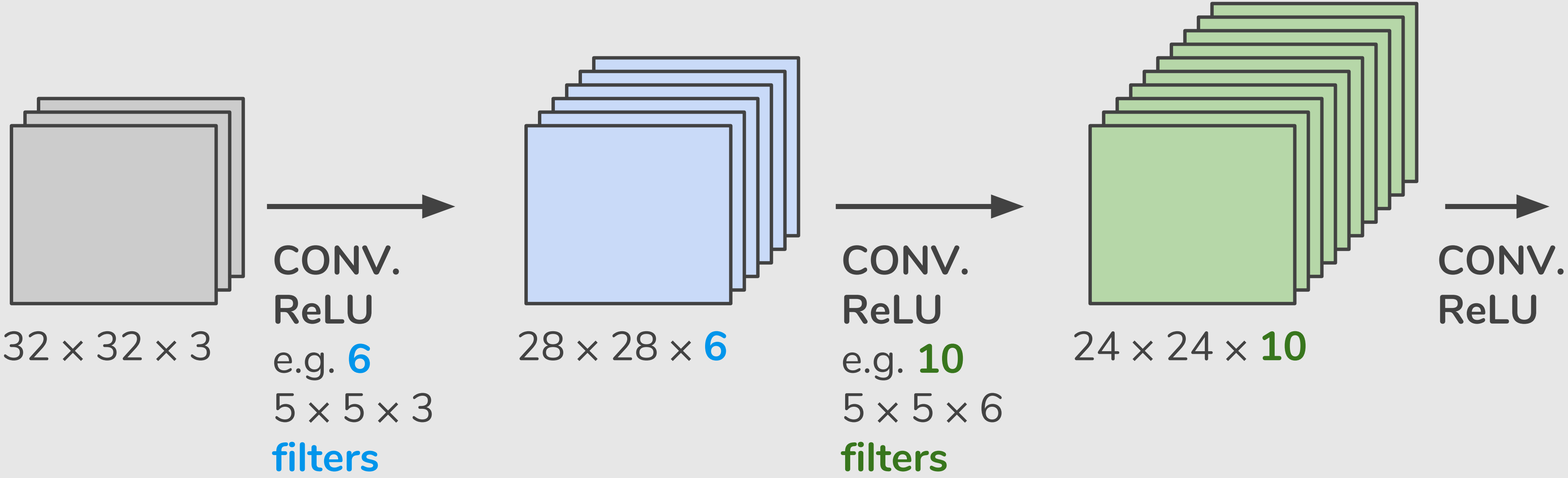
Convolutional Layer

The size of the **Activation Map** (or Feature Map or Convolved Feature) is controlled by three parameters:

- **Depth**: corresponds to the **number of filters** we use for the convolution operation.

Convolutional Layers

Sequence of Convolutional Layers, interspersed with activation functions.

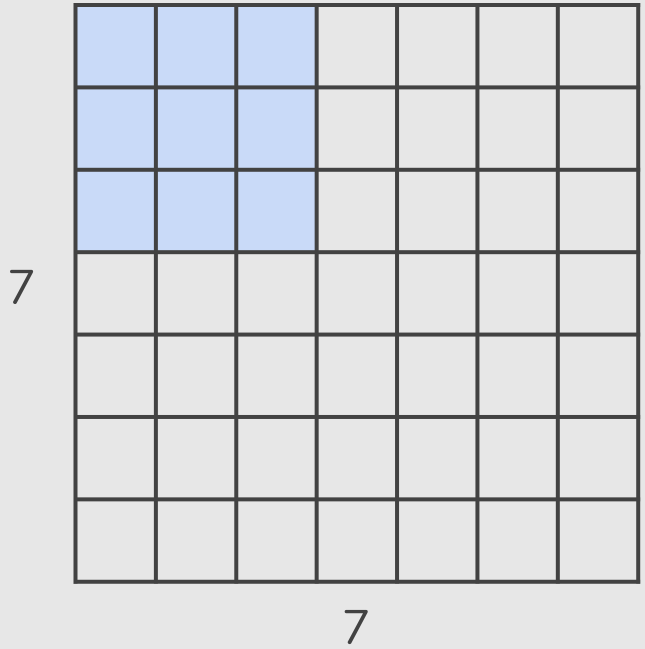


Convolutional Layer

The size of the **Activation Map** (or Feature Map or Convolved Feature) is controlled by three parameters:

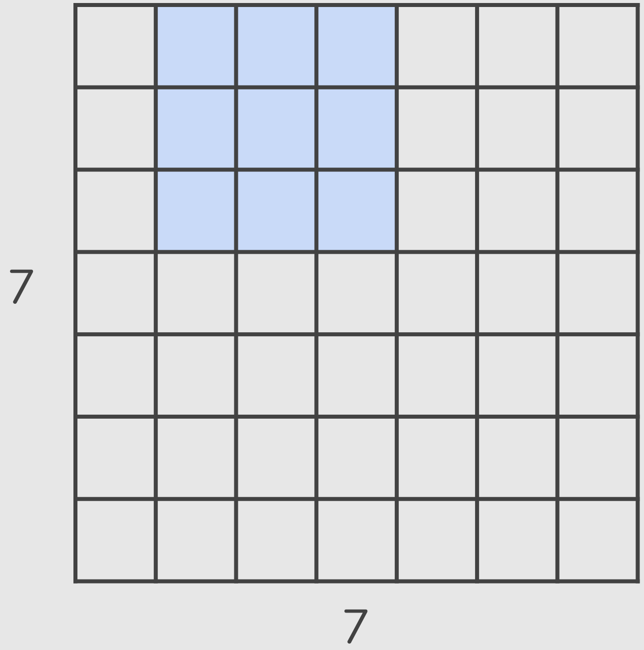
- **Depth**: corresponds to the **number of filters** we use for the convolution operation.
- **Stride**: the **number of pixels by which we slide** our filter matrix over the input matrix.

Convolutional Layer: Stride



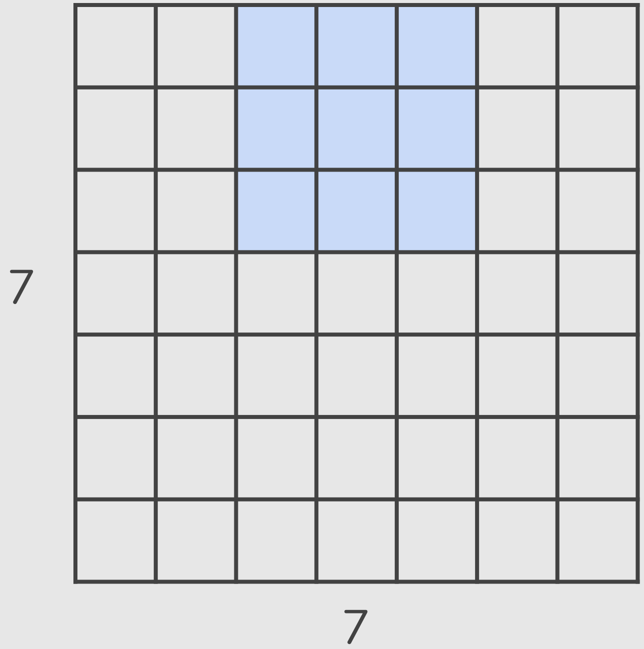
7×7 input (spatially)
assume 3×3 filter
applied with **stride 1**

Convolutional Layer: Stride



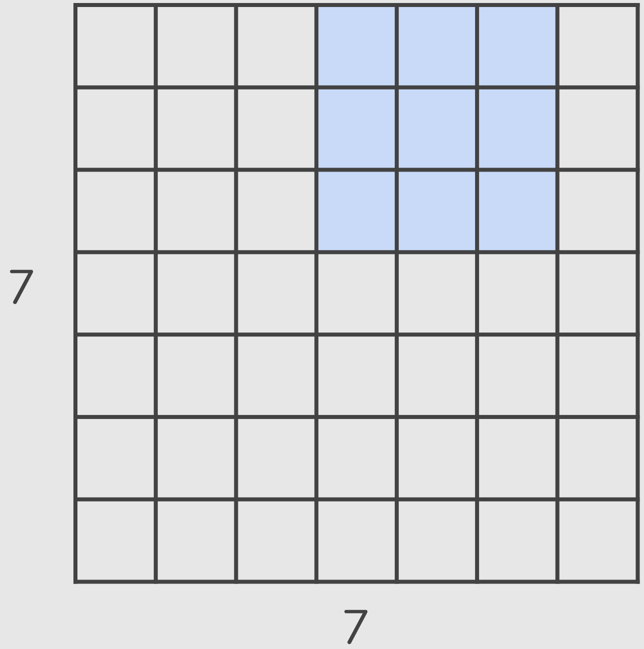
7×7 input (spatially)
assume 3×3 filter
applied with **stride 1**

Convolutional Layer: Stride



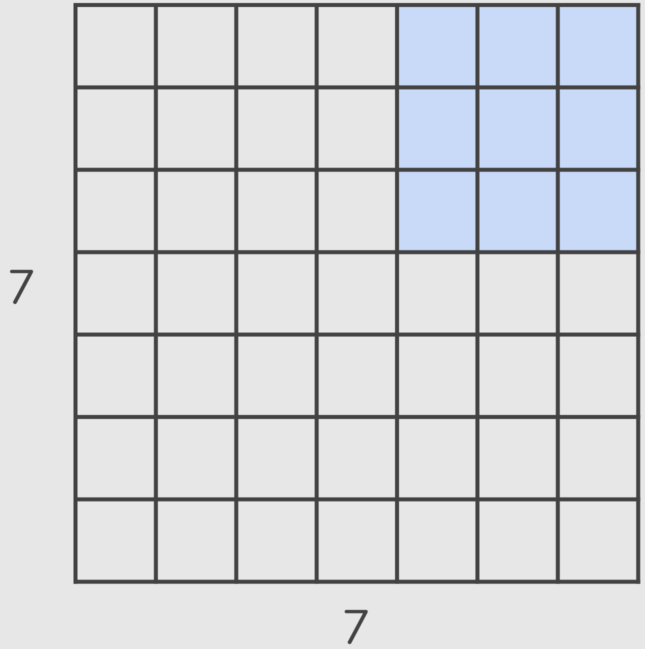
7×7 input (spatially)
assume 3×3 filter
applied with **stride 1**

Convolutional Layer: Stride



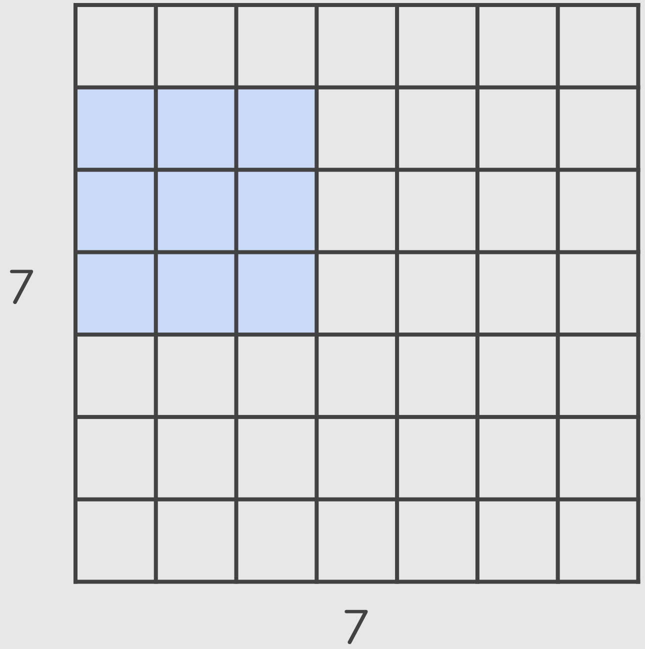
7×7 input (spatially)
assume 3×3 filter
applied with **stride 1**

Convolutional Layer: Stride



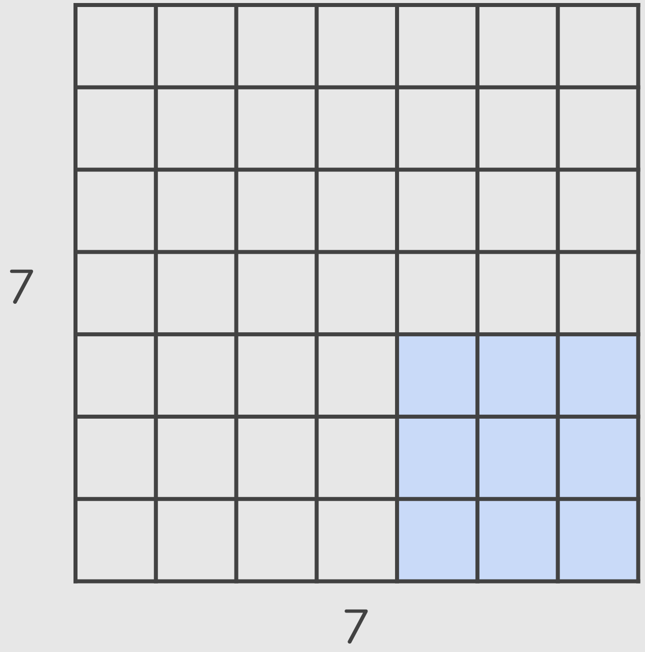
7×7 input (spatially)
assume 3×3 filter
applied with **stride 1**

Convolutional Layer: Stride



7×7 input (spatially)
assume 3×3 filter
applied with **stride 1**

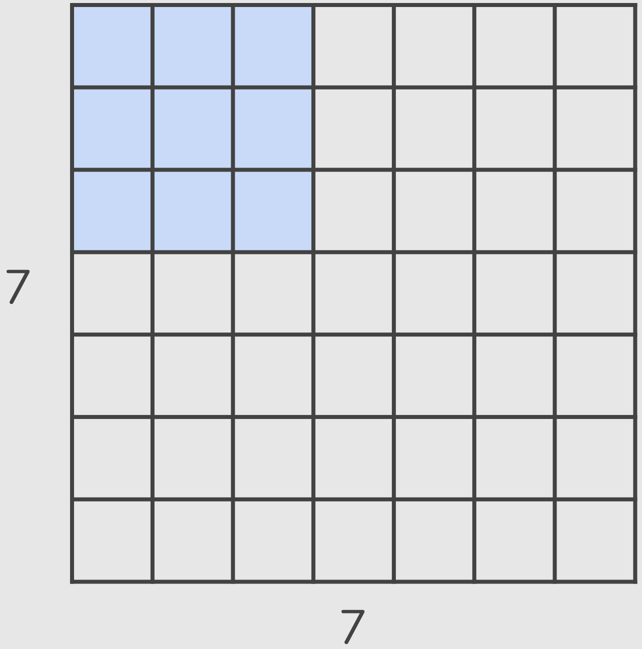
Convolutional Layer: Stride



7×7 input (spatially)
assume 3×3 filter
applied with **stride 1**

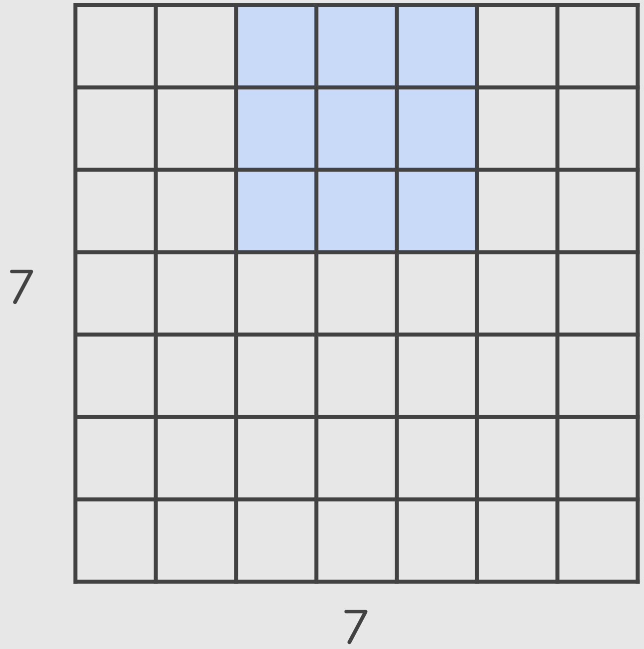
⇒ 5×5 output

Convolutional Layer: Stride



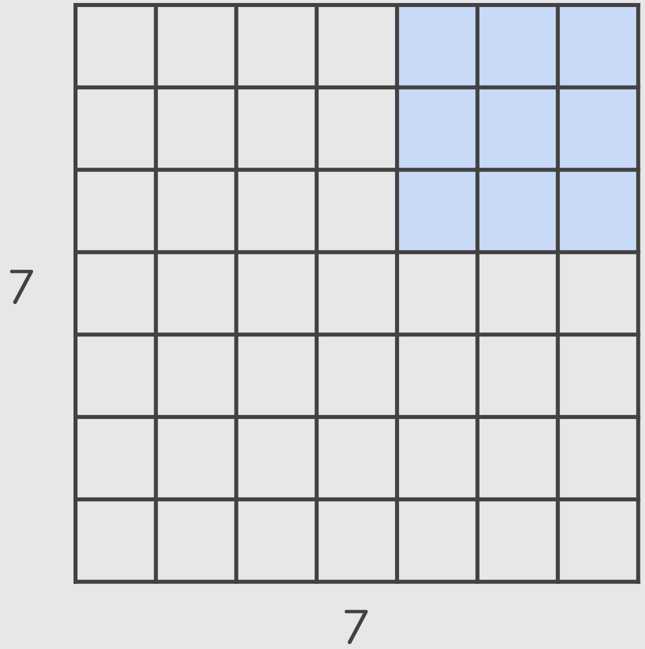
7×7 input (spatially)
assume 3×3 filter
applied with **stride 2**

Convolutional Layer: Stride



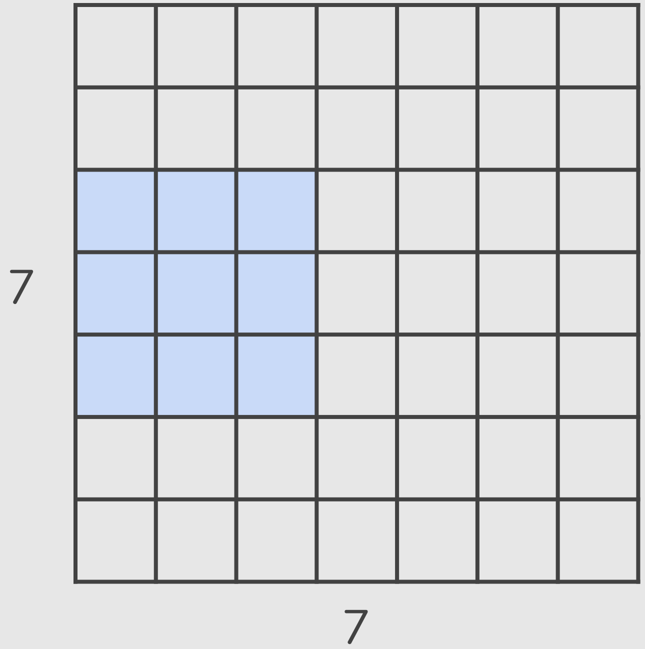
7×7 input (spatially)
assume 3×3 filter
applied with **stride 2**

Convolutional Layer: Stride



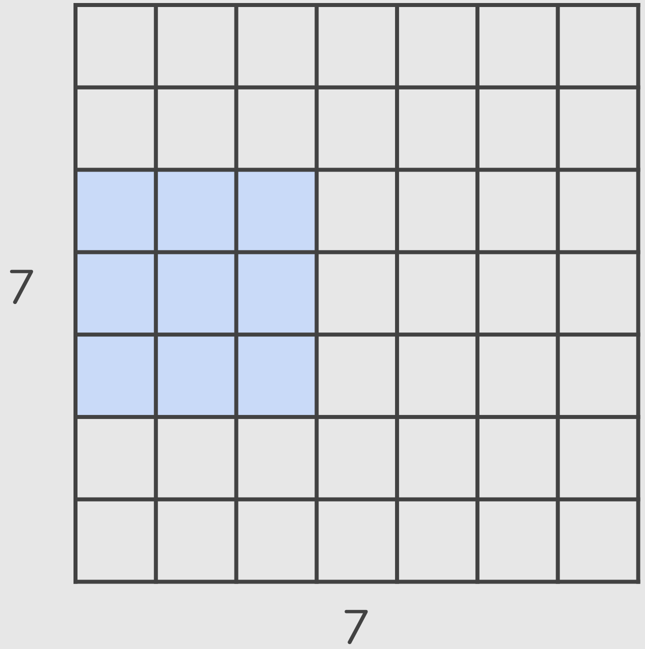
7×7 input (spatially)
assume 3×3 filter
applied with **stride 2**

Convolutional Layer: Stride



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

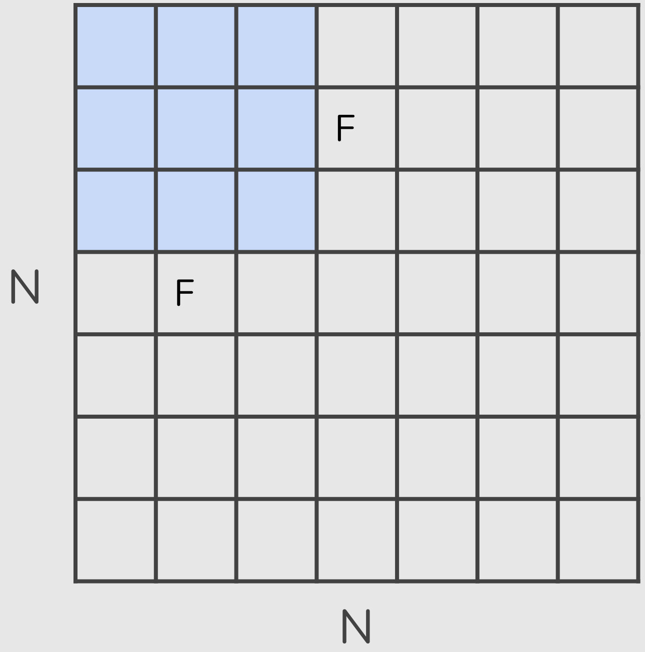
Convolutional Layer: Stride



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

⇒ 3×3 output

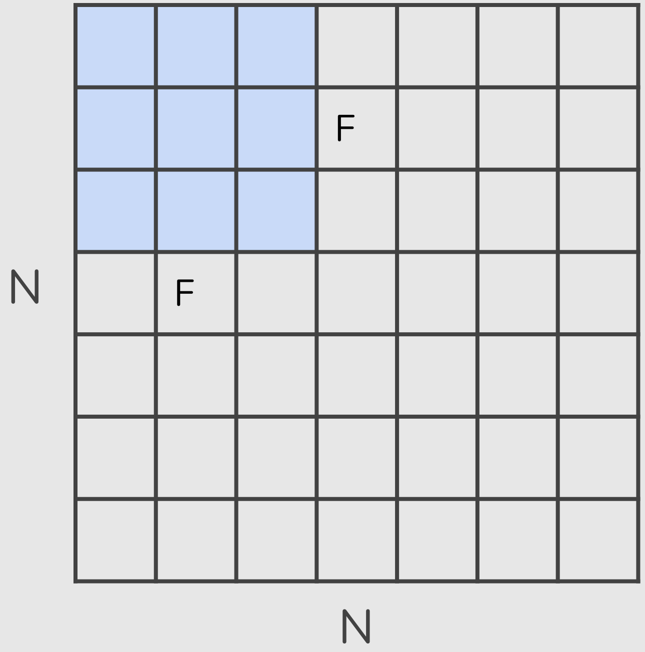
Convolutional Layer: Stride



Output size:

(N - F) / stride + 1

Convolutional Layer: Stride



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

Convolutional Layer

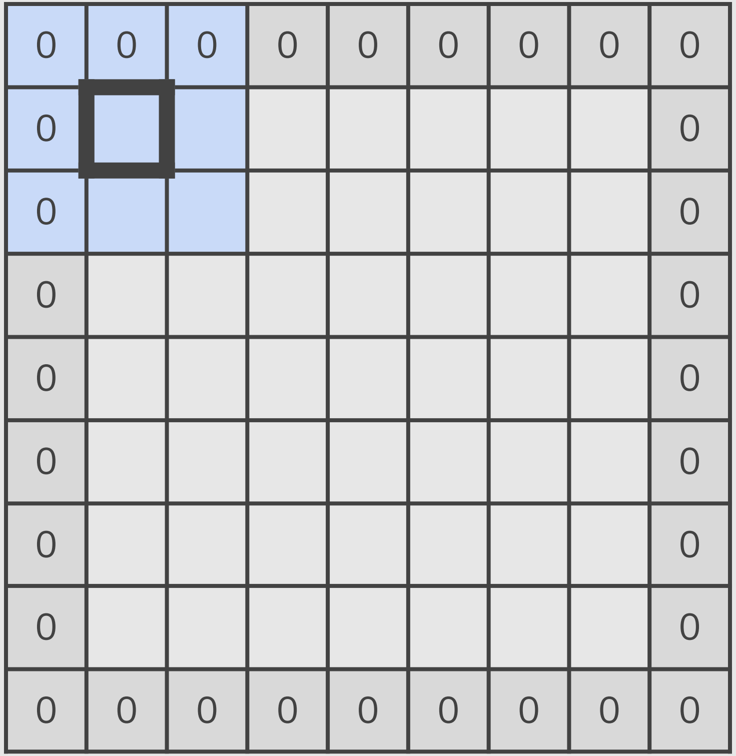
The size of the **Activation Map** (or Feature Map or Convolved Feature) is controlled by three parameters:

- **Depth**: corresponds to the **number of filters** we use for the convolution operation.
- **Stride**: the **number of pixels by which we slide** our filter matrix over the input matrix.
- **Zero-padding**: sometimes, it is convenient to pad the input matrix with **zeros around the border**.

Convolutional Layer: Zero-Padding

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

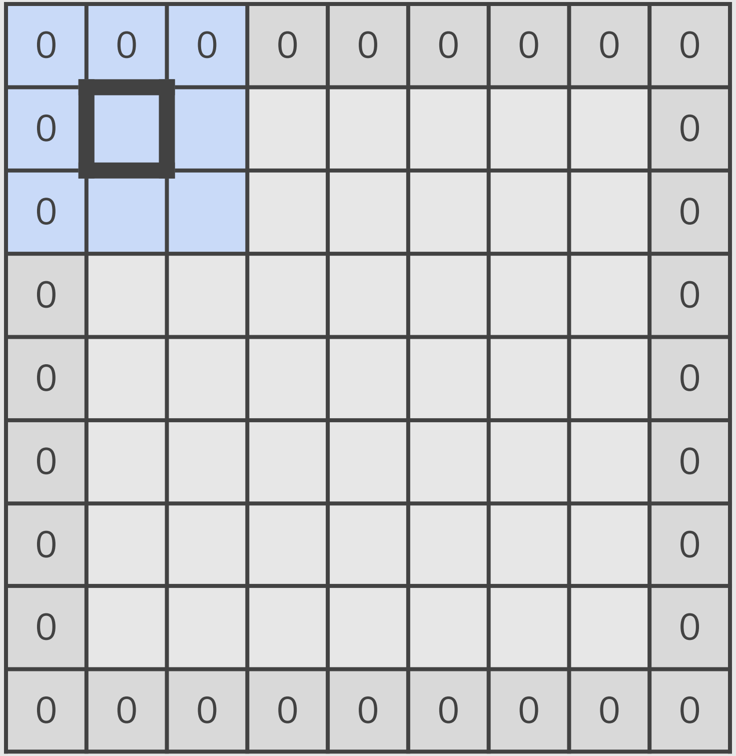
Convolutional Layer: Zero-Padding



7 × 7 input,
3 × 3 filter applied
with **stride 1 with pad 1**

What is the output?
7 × 7 output

Convolutional Layer: Zero-Padding

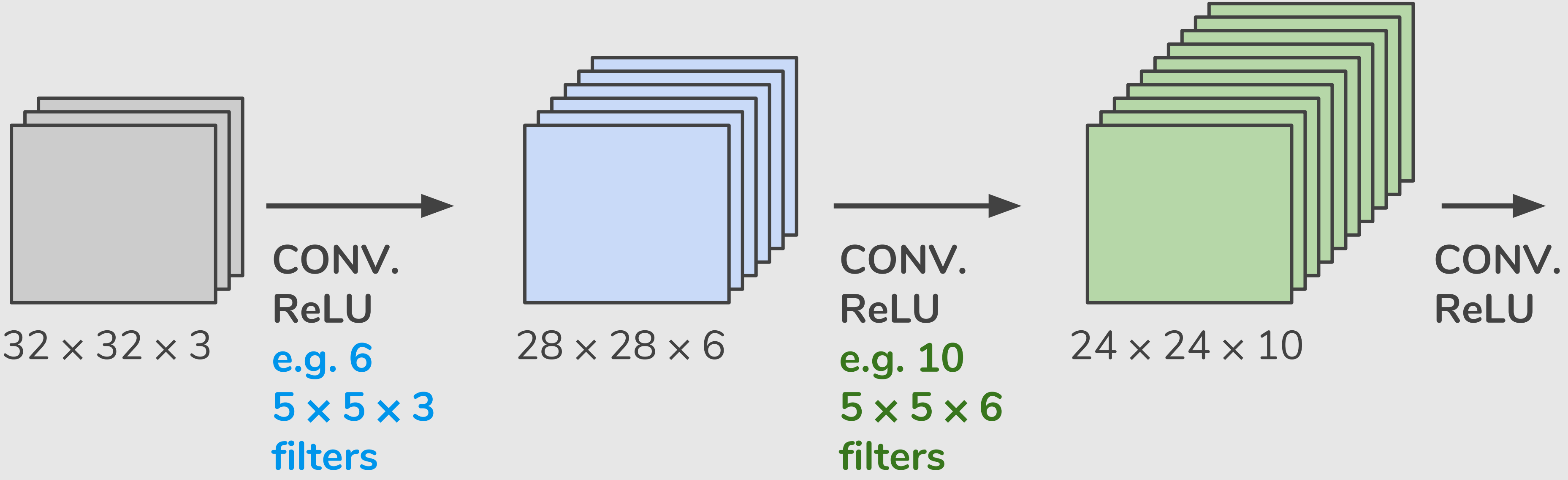


In general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$ **(will preserve size spatially)**.

e.g. $F = 3 \Rightarrow$ zero pad with 1
$F = 5 \Rightarrow$ zero pad with 2
$F = 7 \Rightarrow$ zero pad with 3

Shrinking too fast is not good, doesn't work well.

$32 \rightarrow 28 \rightarrow 24 \rightarrow \ldots$



Number of Parameters

Input volume: $\mathbf{32 \times 32 \times 3}$

10 5×5 filters with stride 1, pad 2

Number of parameters in this layer?

Number of Parameters

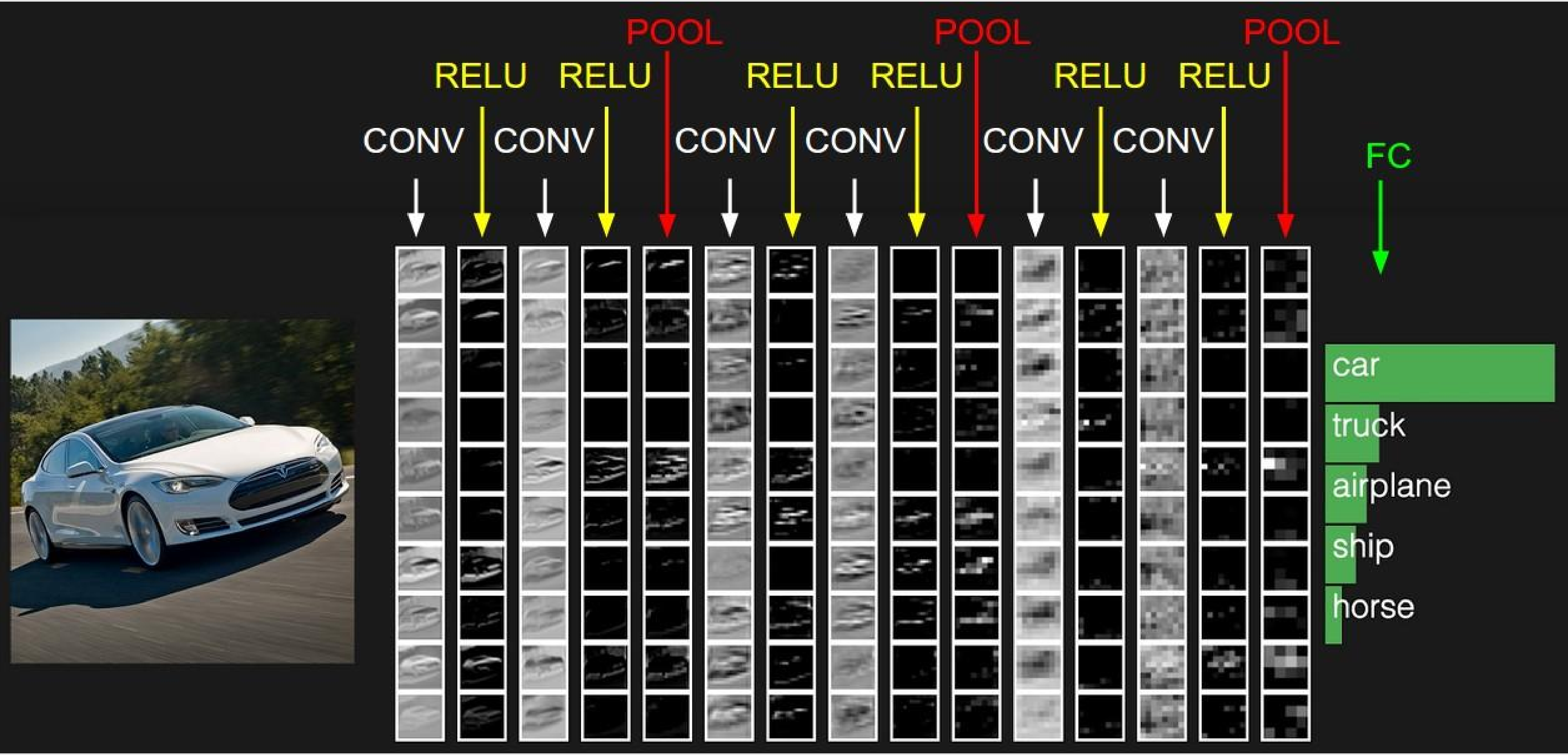
Input volume: **32** $\times$ **32** $\times$ **3**

10 **5** $\times$ **5** filters with stride 1, pad 2

Number of parameters in this layer?

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

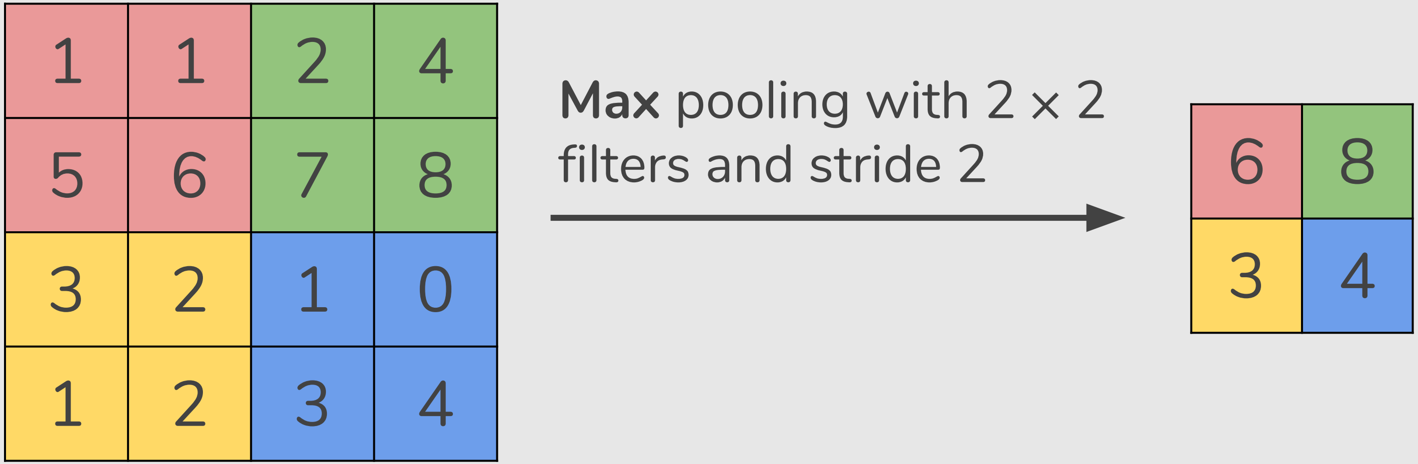
=> **76***10* = 760



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

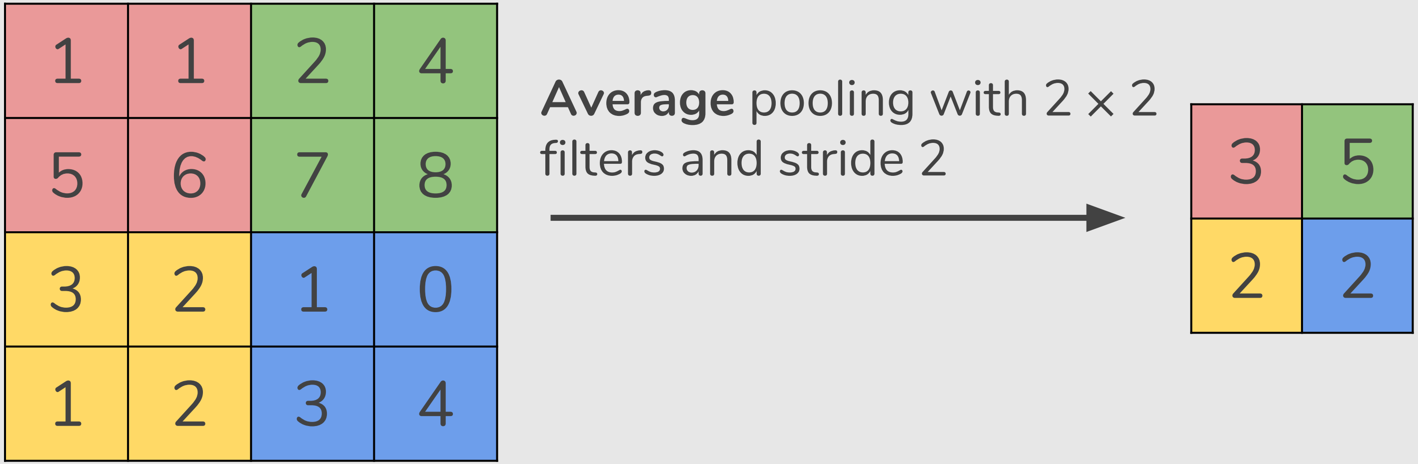
Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently



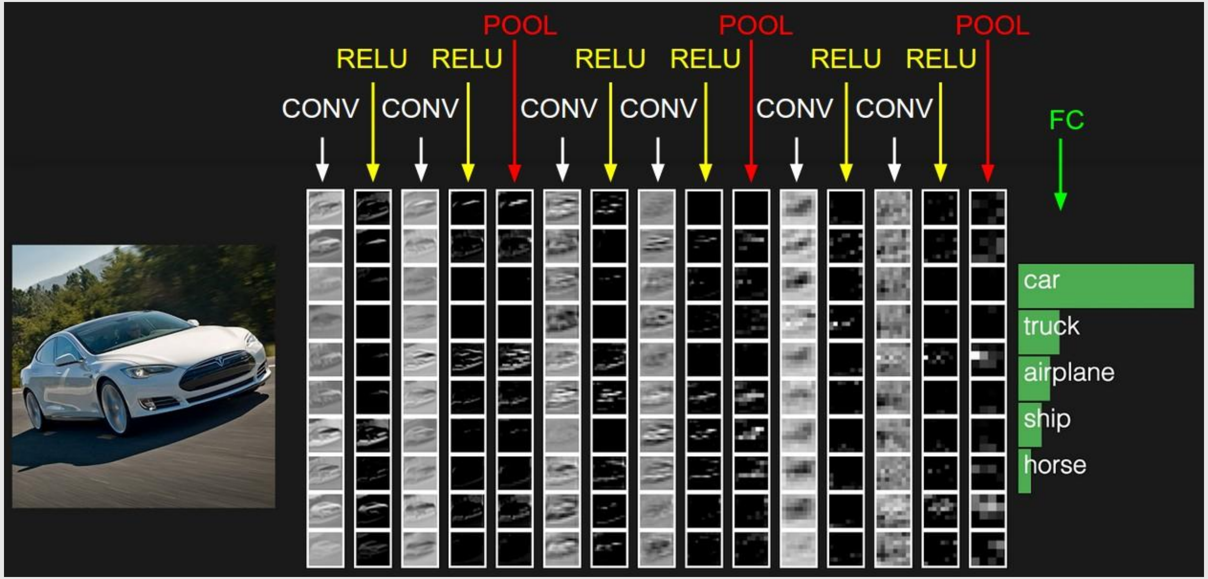
Pooling Layer

The function of Pooling is **to progressively reduce the spatial size of the input representation**. In particular, pooling

- makes the input representations (feature dimension) smaller and more manageable
- reduces the number of parameters and computations in the network, therefore, controlling overfitting
- makes the network invariant to small transformations, distortions and translations in the input image

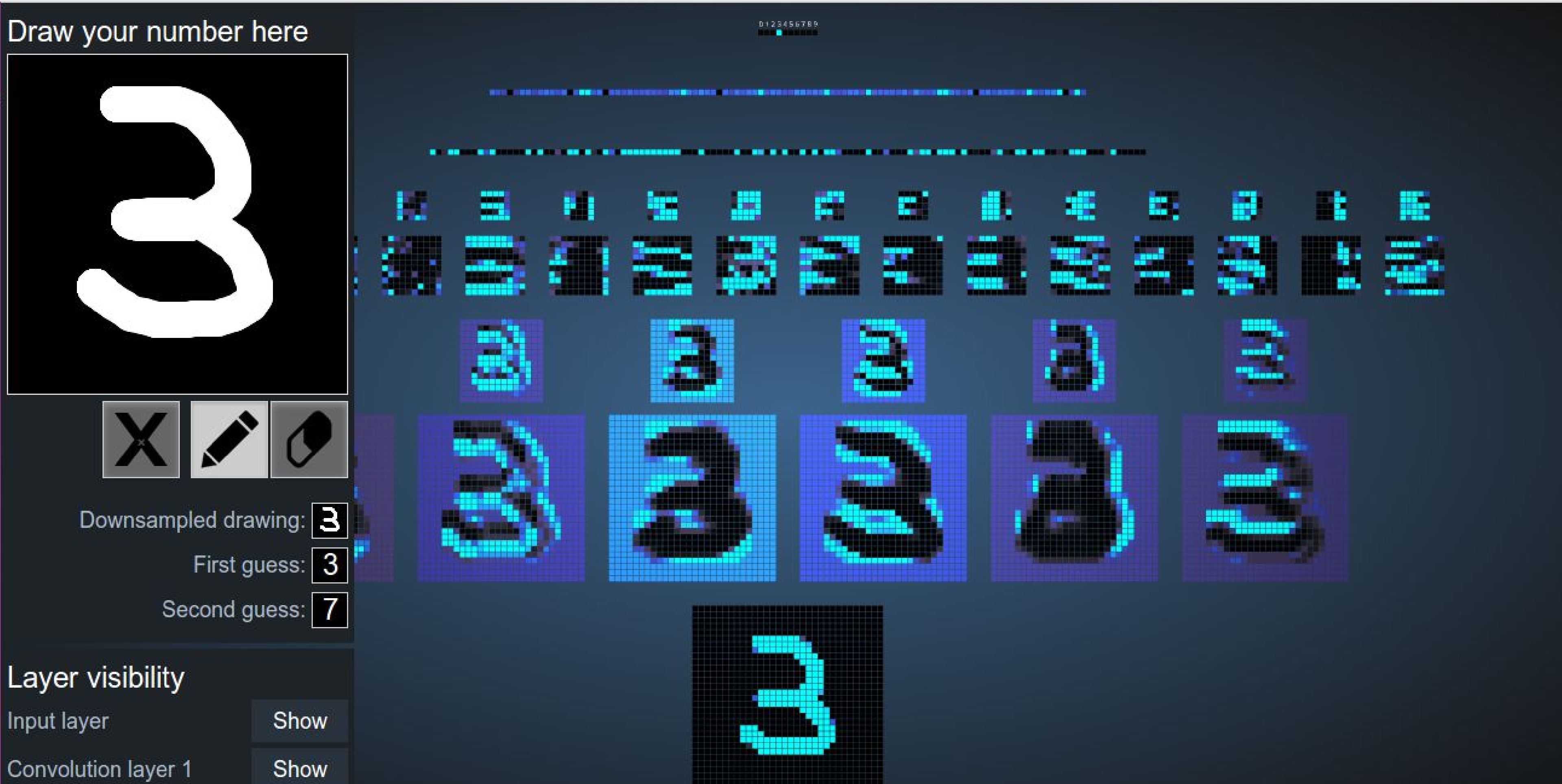
Fully Connected Layer

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



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

http://scs.ryerson.ca/~aharley/vis/conv/flat.html



Visualizing a CNN trained on Handwritten Digits

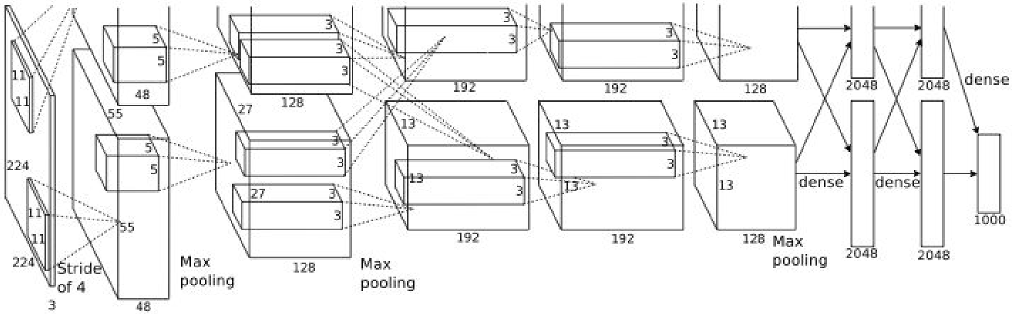
- Input image: 1024 pixels (32 x 32 image)
- CONV 1 (+ RELU): 6 5 × 5 (stride 1) filters
- POOL 1: 2 × 2 max pooling (with stride 2)
- CONV 2 (+ RELU): 16 5 × 5 (stride 1) filters
- POOL 2: 2 × 2 max pooling (with stride 2)
- 3 FC layers:
  - 120 neurons in the first FC layer
  - 100 neurons in the second FC layer
  - 10 neurons in the third FC (Output layer)

DNNs Architectures

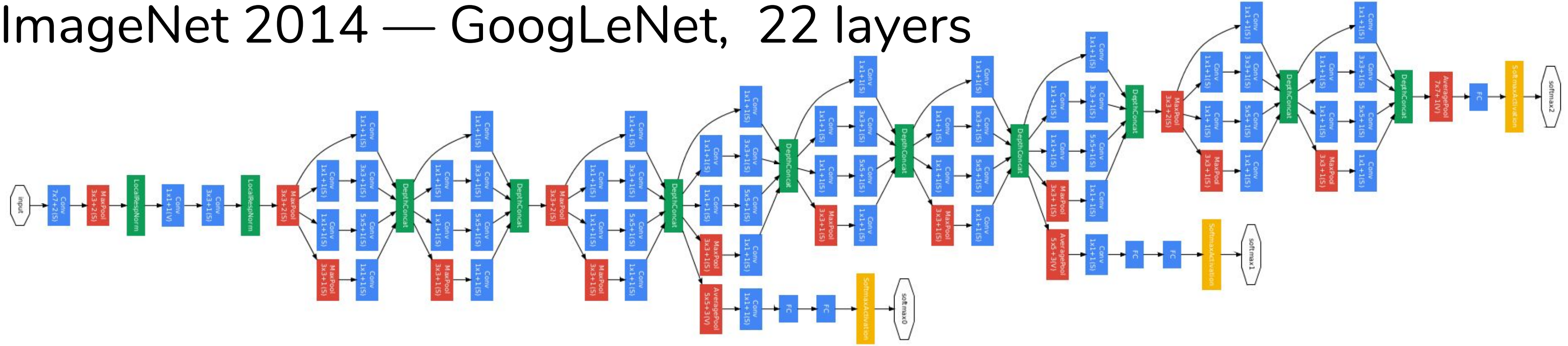
DNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **GoogLeNet** by Szegedy et al. (2014)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **ResNet** by Kaiming He et al. (2015)

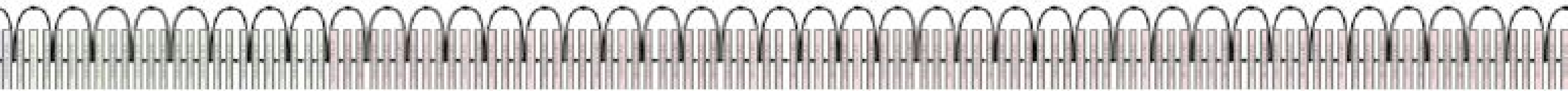
ImageNet 2012 — AlexNet, 8 layers



ImageNet 2014 — GoogLeNet, 22 layers



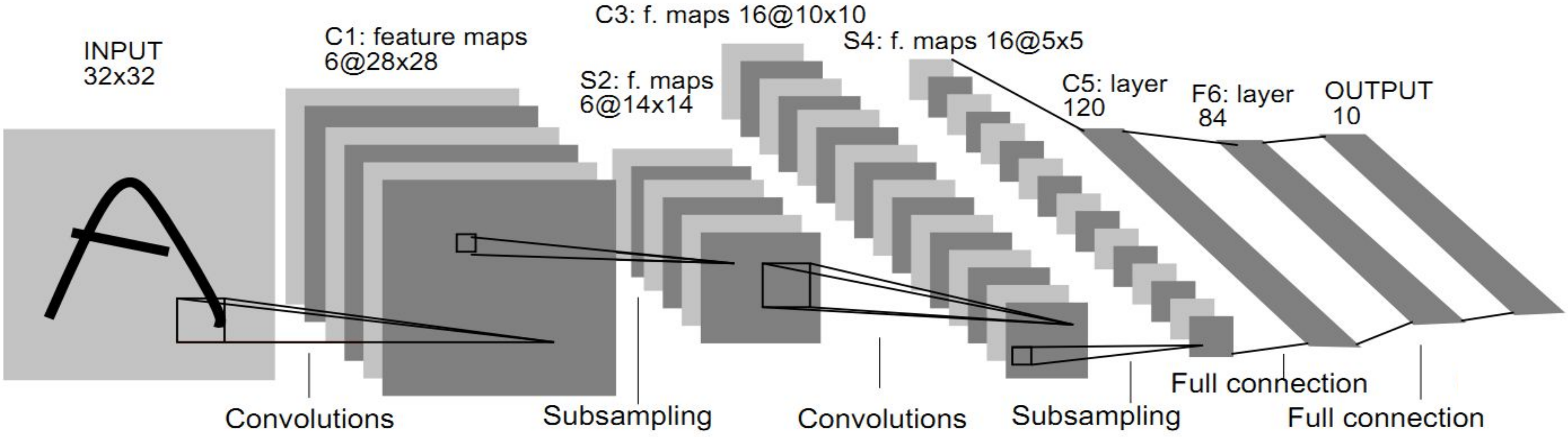
ImageNet 2015 — ResNet, 152 layers



DNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **GoogLeNet** by Szegedy et al. (2014)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **ResNet** by Kaiming He et al. (2015)

LeNet-5 [LeCun et al., 1998]

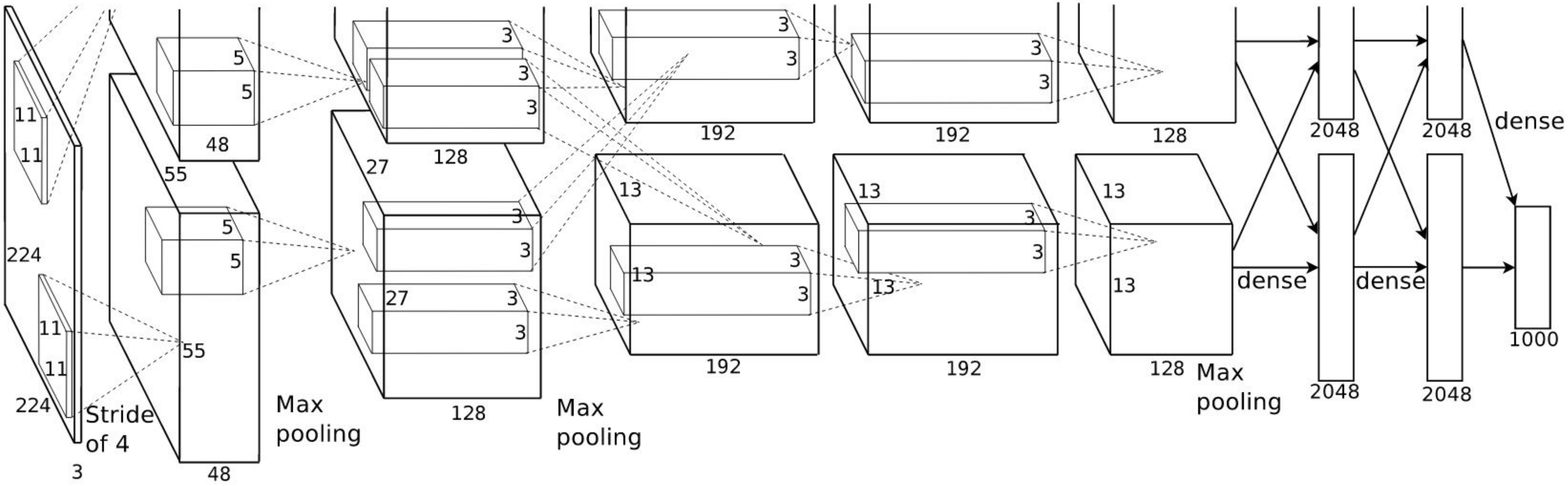


Convolution filters: 5x5 with stride 1

Subsampling (Pooling) layers: 2x2 with stride 2

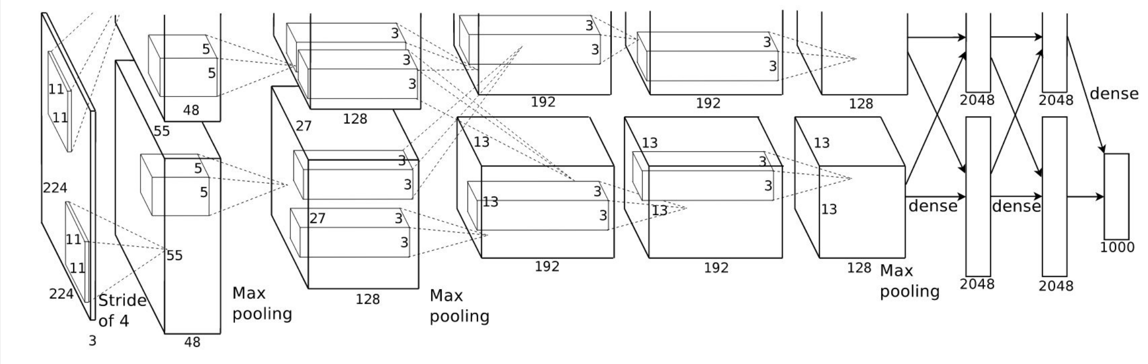
[CONV-POOL-CONV-POOL-FC-FC]

AlexNet [Krizhevsky et al., 2012]



AlexNet [Krizhevsky et al., 2012]

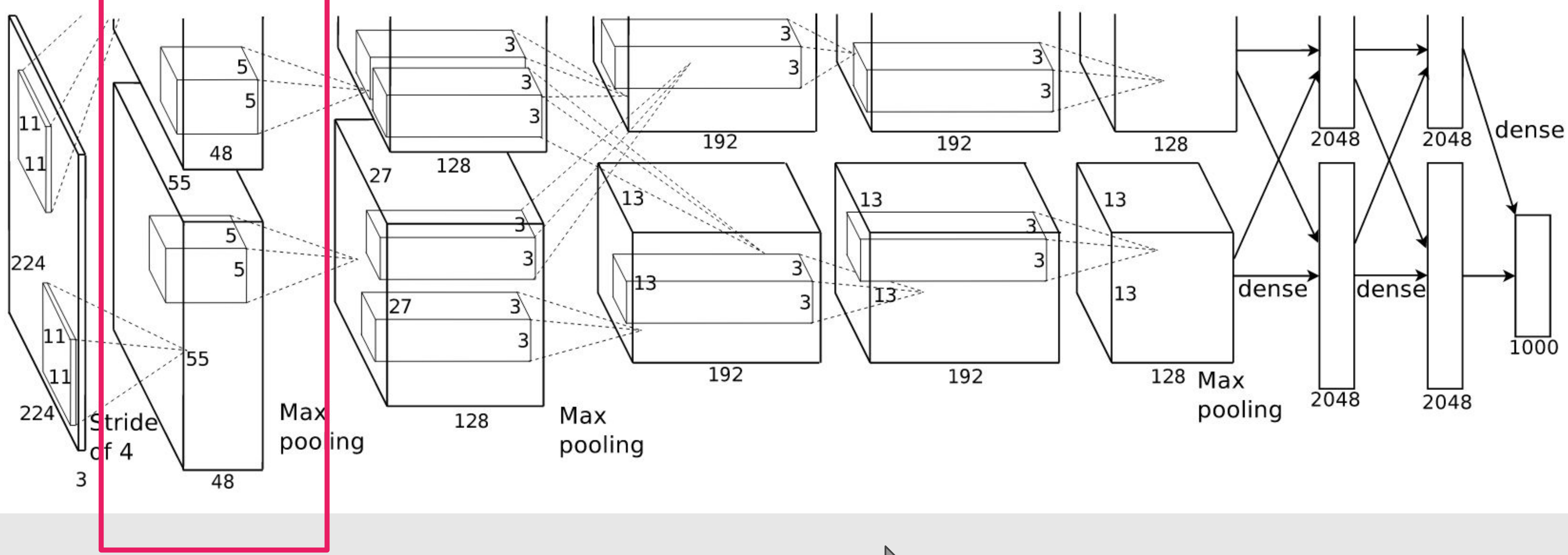
Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
MAX POOL3
FC6
FC7
FC8



AlexNet [Krizhevsky et al., 2012]

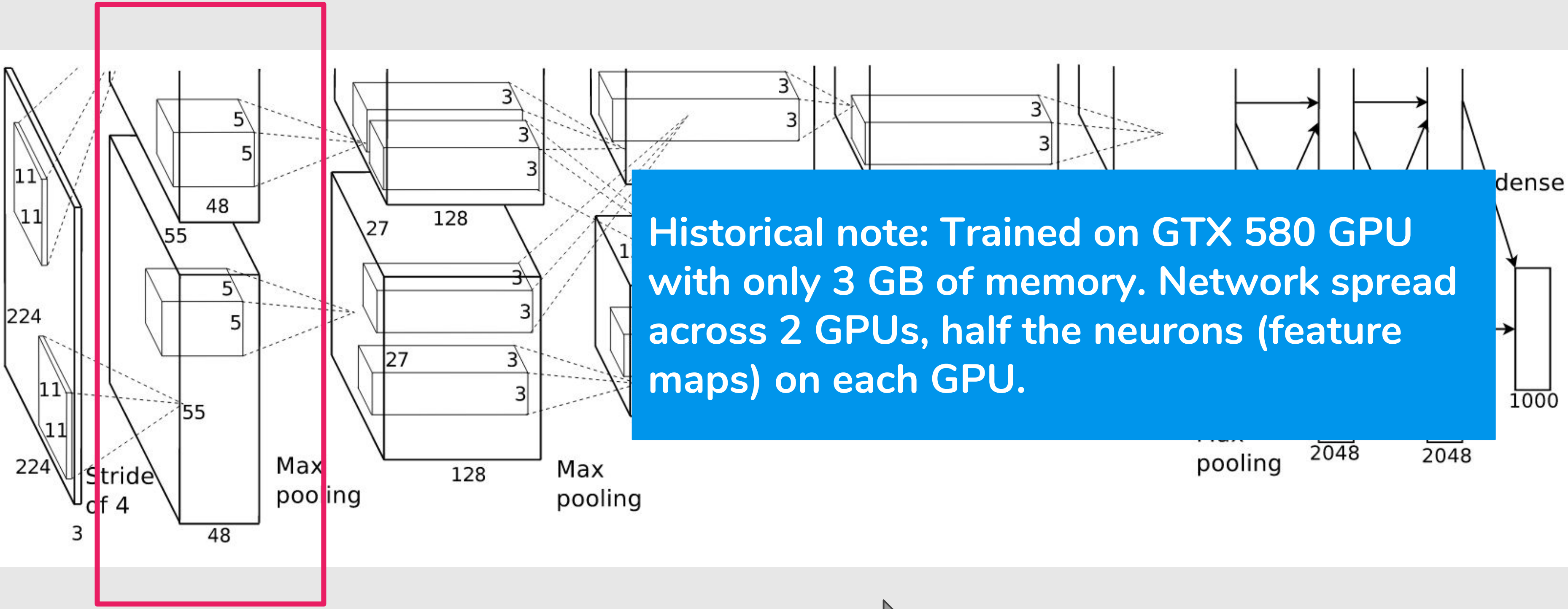
[227x227x3] INPUT
[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0
[27x27x96] **MAX POOL1**: 3x3 filters at stride 2
[27x27x96] **NORM1**: Normalization layer
[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2
[13x13x256] **MAX POOL2**: 3x3 filters at stride 2
[13x13x256] **NORM2**: Normalization 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)

AlexNet [Krizhevsky et al., 2012]



[55x55x96] CONV1 → [55x55x48] x 2

AlexNet [Krizhevsky et al., 2012]



Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

[55x55x96] **CONV1** ➡ [55x55x48] x 2

AlexNet [Krizhevsky et al., 2012]

Details:

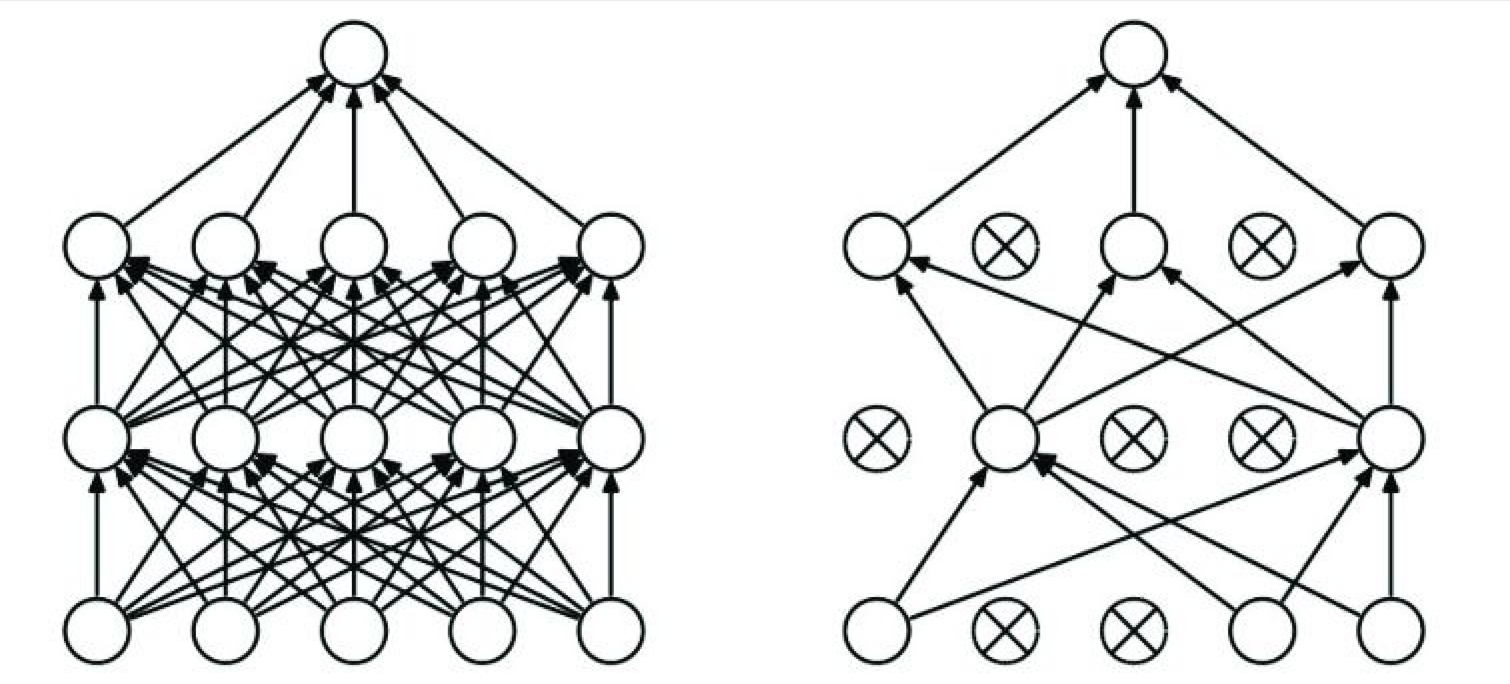
- 60 million learned parameters
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble: 18.2% -> 15.4%

Dropout

- Dropout is a radically different technique for regularization.
- Dropout doesn't rely on modifying the cost function. Instead, **in dropout we modify the network itself**.

[Srivastava et al., 2014] Dropout: A Simple Way to Prevent Neural Networks from Overfitting
https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Dropout



Standard Network　　　　After applying dropout

Dropout

- Heuristically, when we dropout different sets of neurons, it's rather like we're training different neural networks.
- The dropout procedure is like averaging the effects of a very large number of different networks.
- The different networks will overfit in different ways, and so, hopefully, the net effect of dropout will be to reduce overfitting.

AlexNet [Krizhevsky et al., 2012]

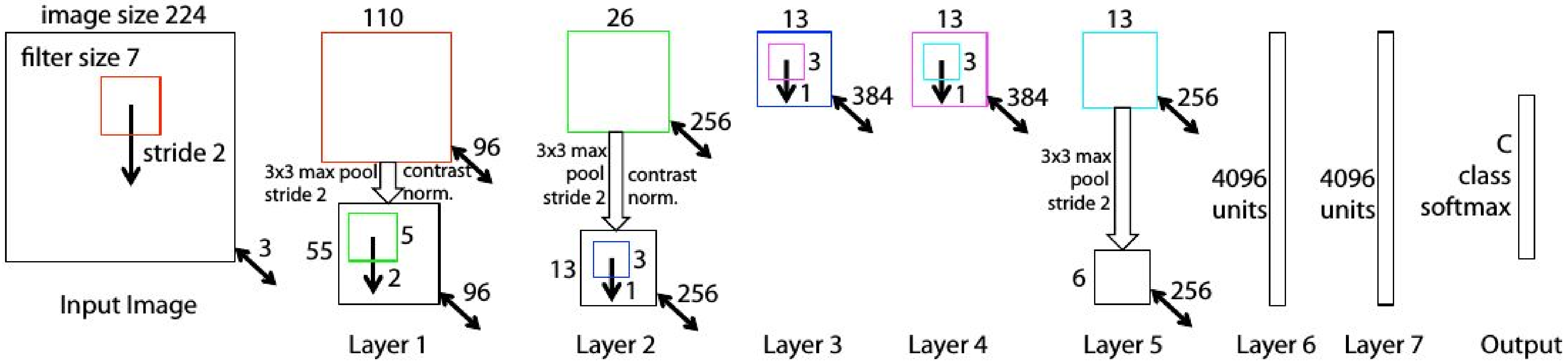
Details:

- 60 million learned parameters
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble: 18.2% -> 15.4%

DNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **GoogLeNet** by Szegedy et al. (2014)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **ResNet** by Kaiming He et al. (2015)

ZFNet [Zeiler & Fergus, 2013]



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

DNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **GoogLeNet** by Szegedy et al. (2014)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **ResNet** by Kaiming He et al. (2015)

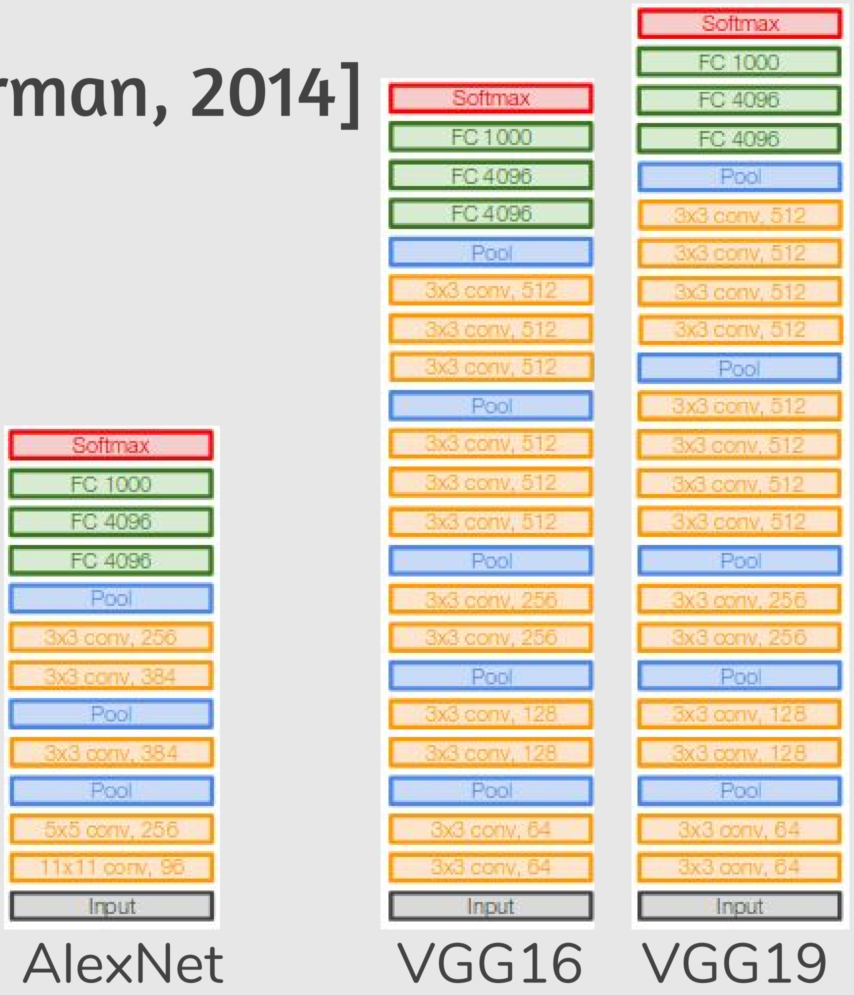
VGGNet [Simonyan & Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
16 - 19 layers (VGG16Net)

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

11.7% in ILSVRC’13 (ZFNet)
7.3% in ILSVRC’14



DNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **GoogLeNet** by Szegedy et al. (2014)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **ResNet** by Kaiming He et al. (2015)

To be continued ...

References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 11 & 13

Machine Learning Courses

- https://www.coursera.org/learn/neural-networks
- “The 3 popular courses on Deep Learning”: https://medium.com/towards-data-science/the-3-popular-courses-for-deeplearning-ai-ac37d4433bd