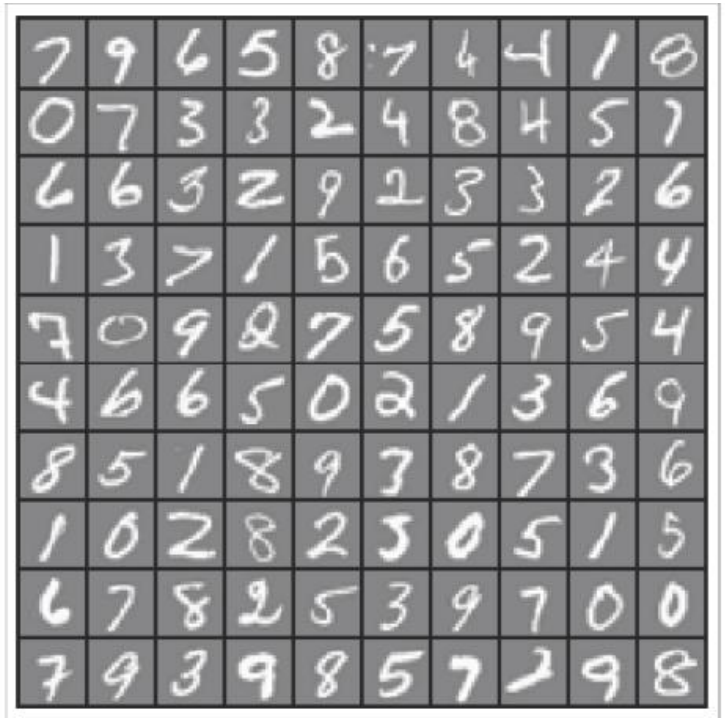
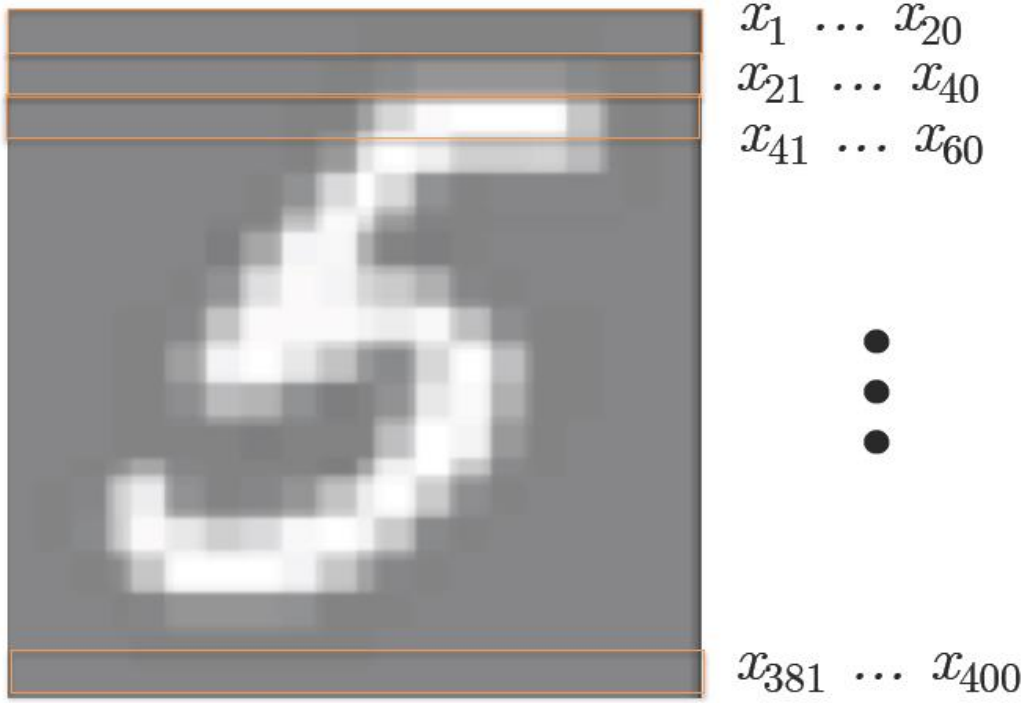
DEEP LEARNING

Convolutional Neural Networks

Layering Representations (1/2)

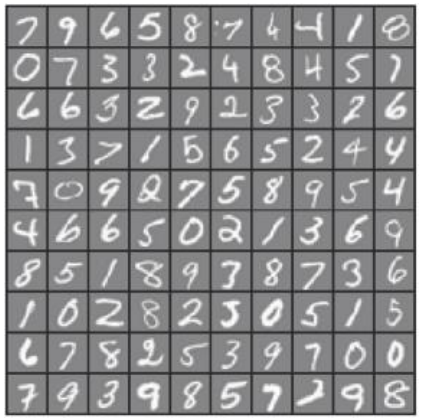


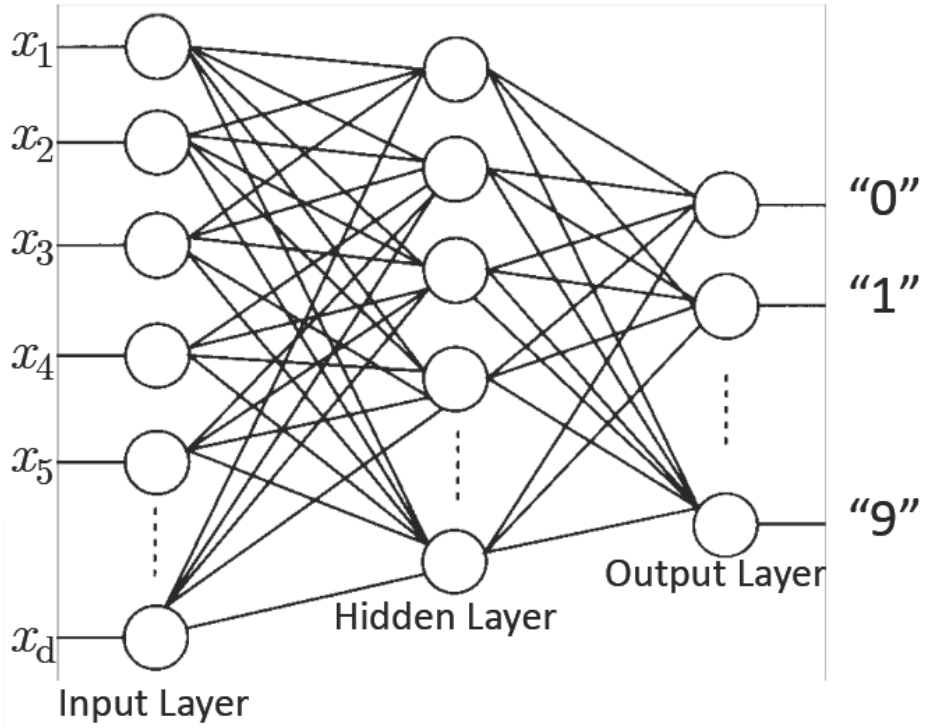
20x20 pixel images
d = 400, 10 classes

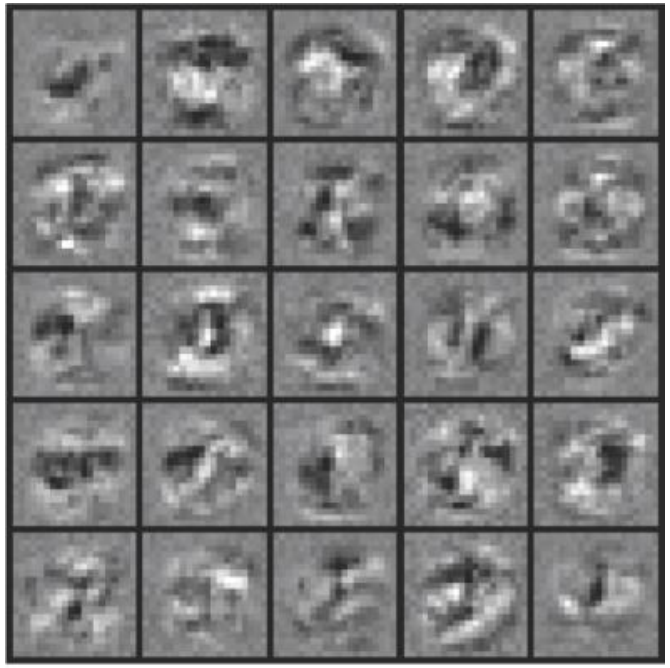


Each image is “unrolled” into a vector x of pixel intensities

Layering Representations (2/2)

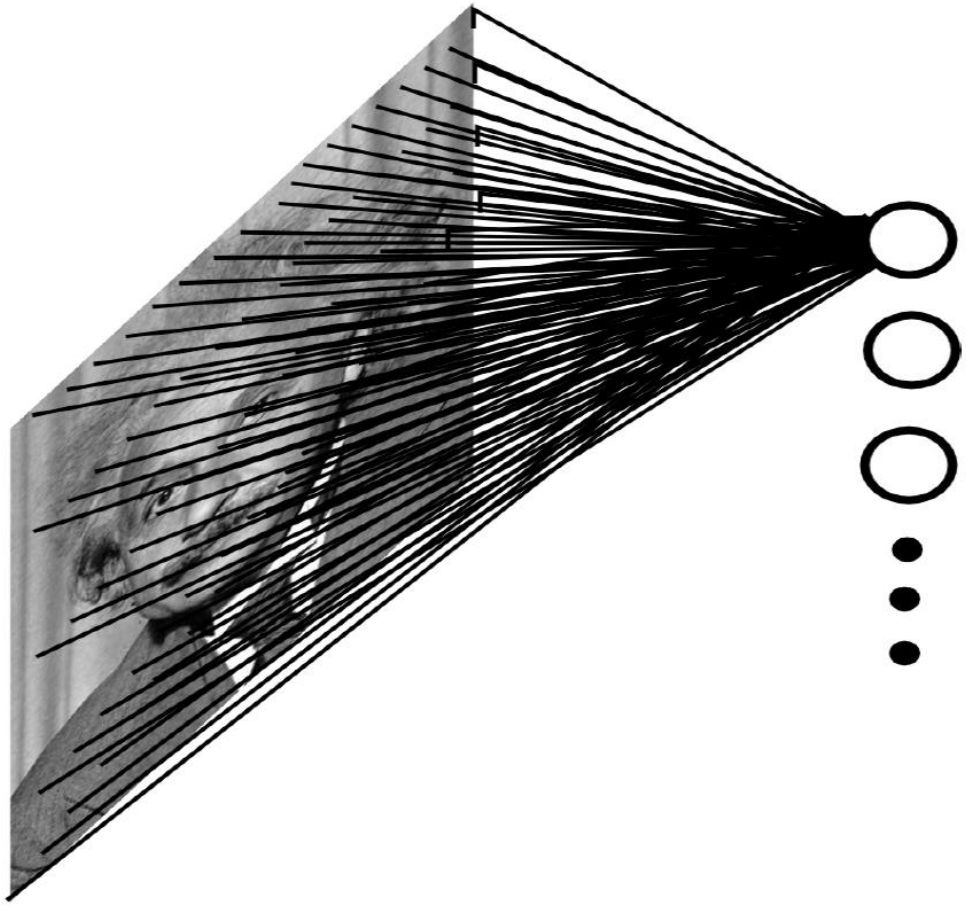






Visualization of Hidden Layer

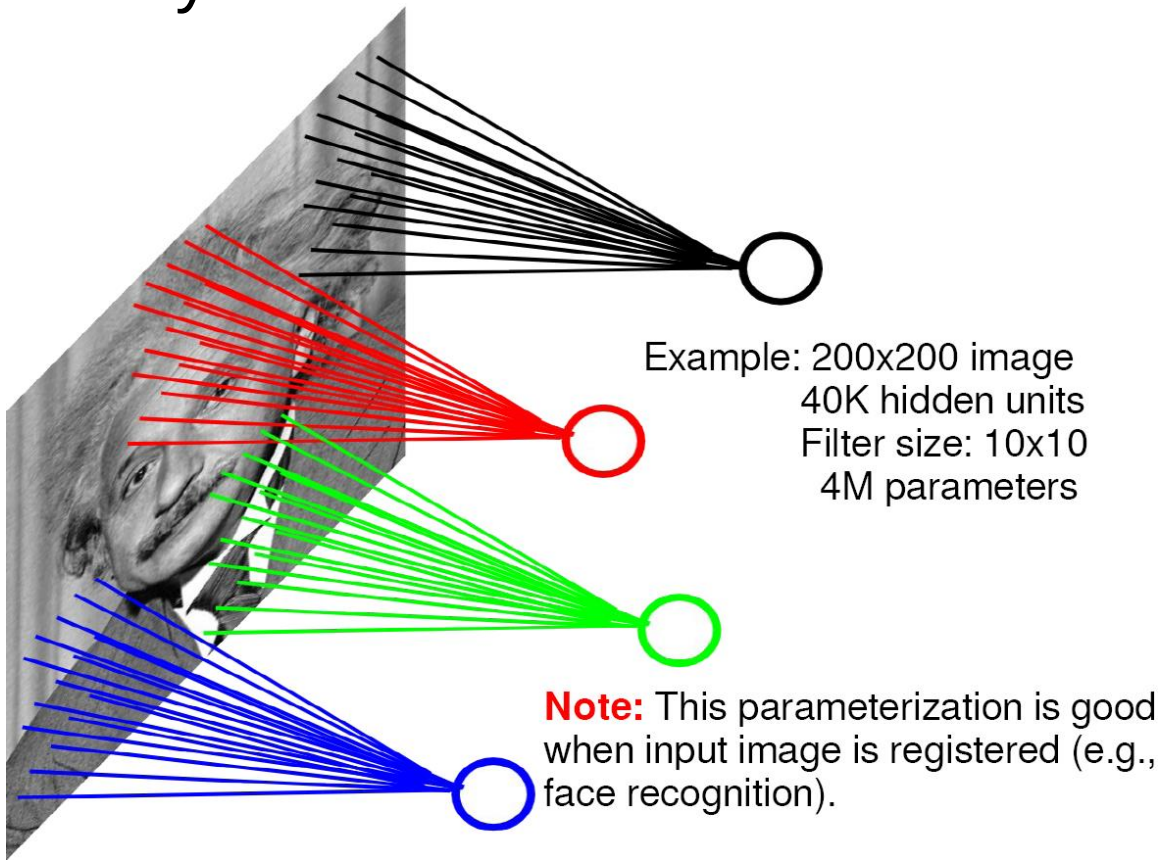
Why Convolution? (1/4)



- Fully connected layer
- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway.
- Example: 200x200 image
  40K hidden units
  ~2B parameters!

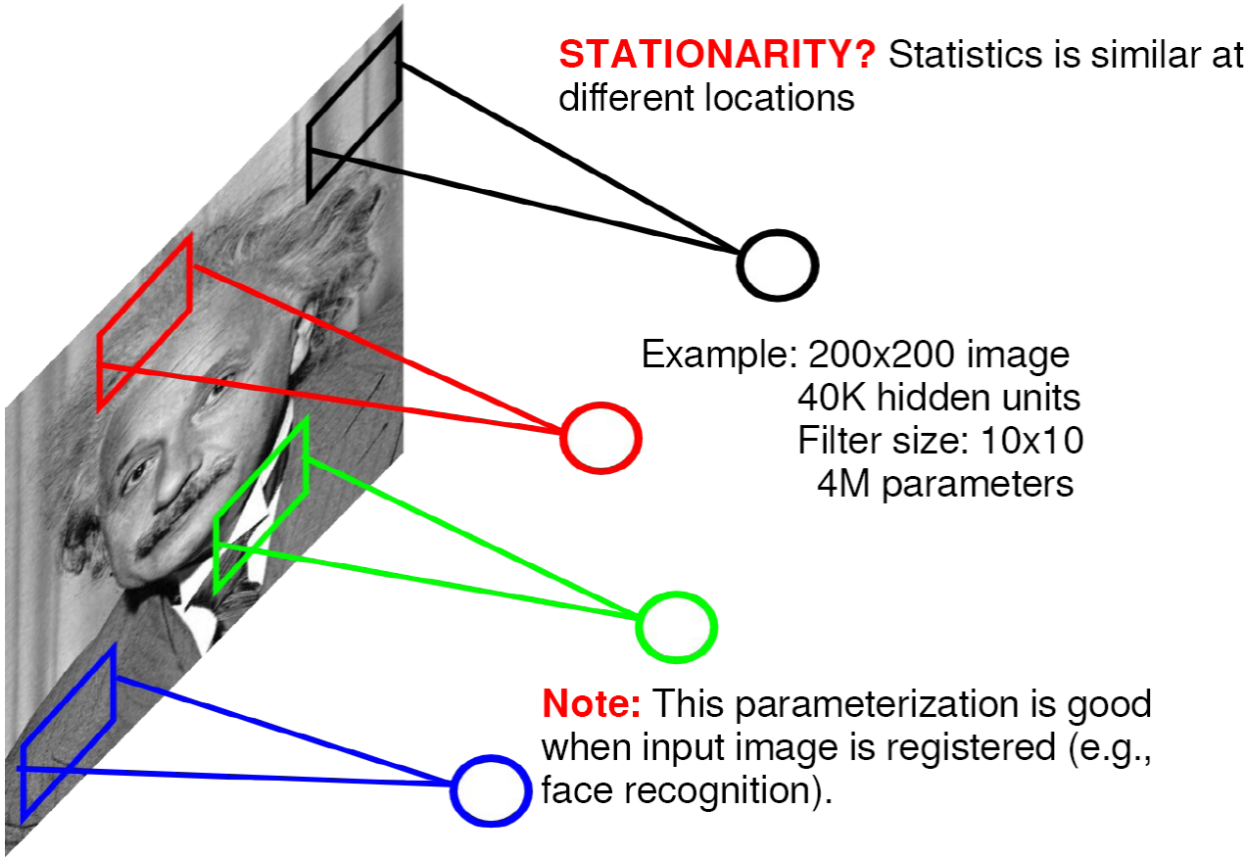
Why Convolution? (2/4)

- Locally connected layer



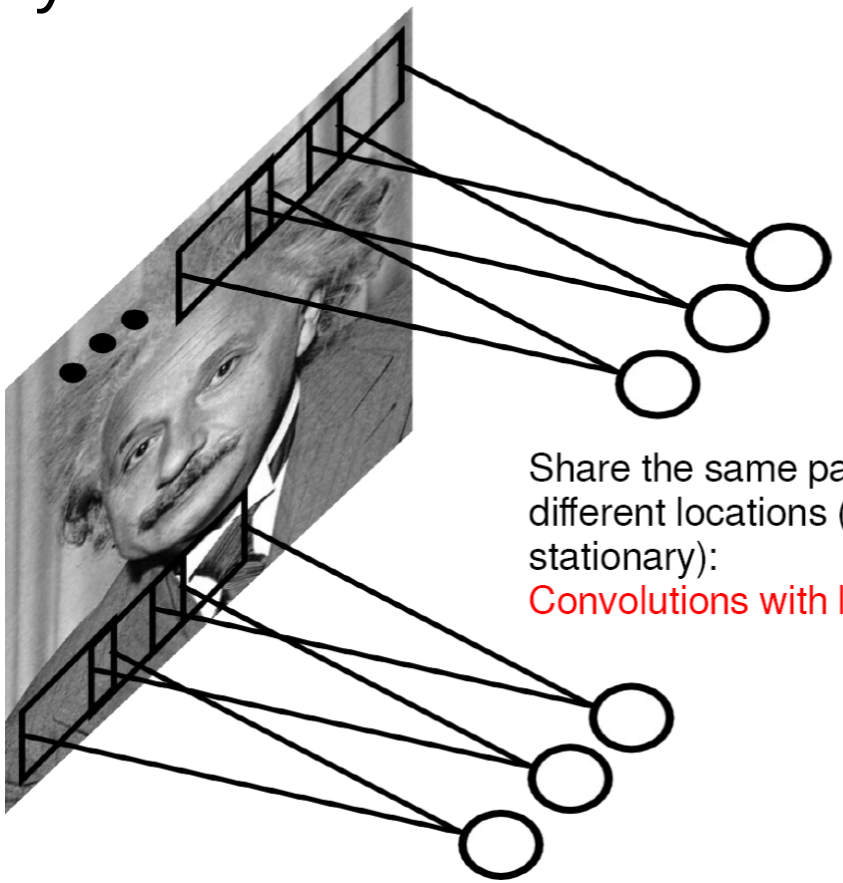
Why Convolution? (3/4)

- Locally connected layer



Why Convolution? (4/4)

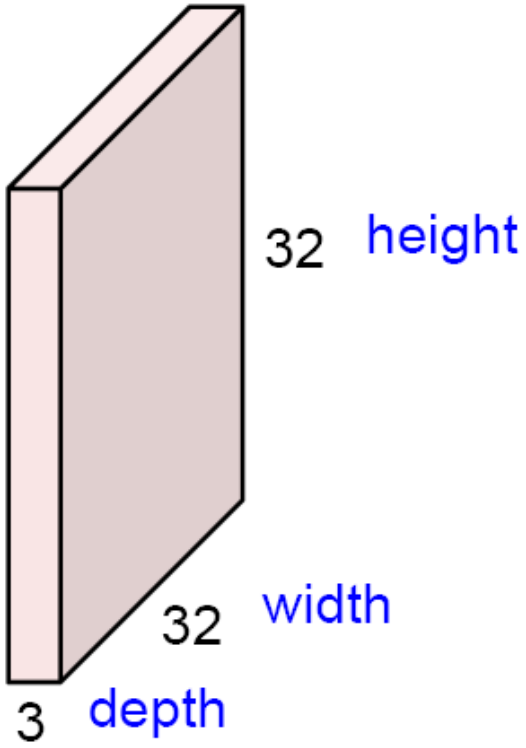
- Convolutional layer



Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

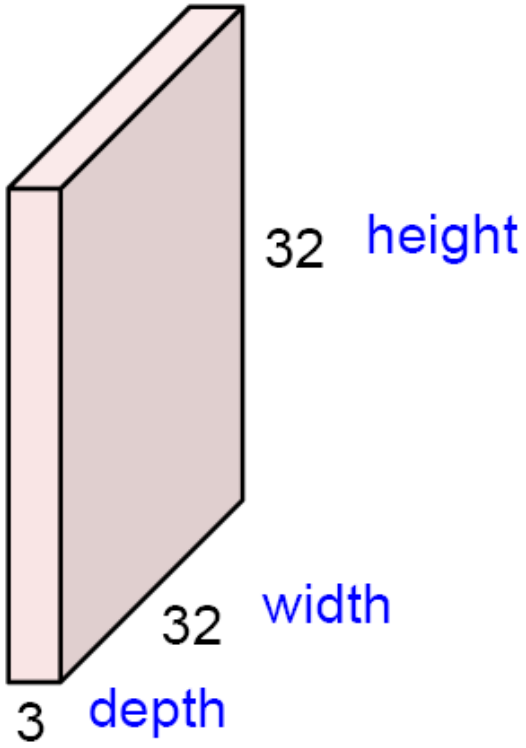
Convolution Layer (1/9)

32x32x3 image

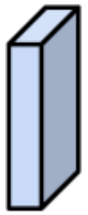


Convolution Layer (2/9)

32x32x3 image

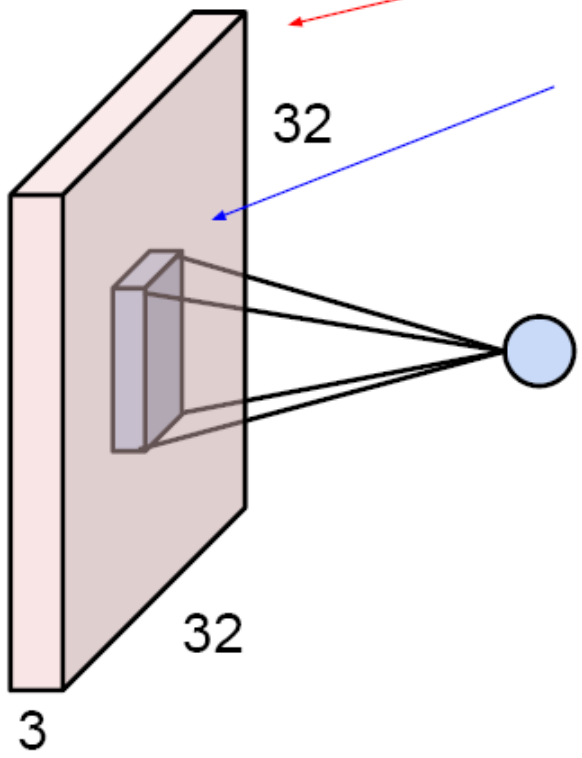


5x5x3 filter



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

Convolution Layer (3/9)



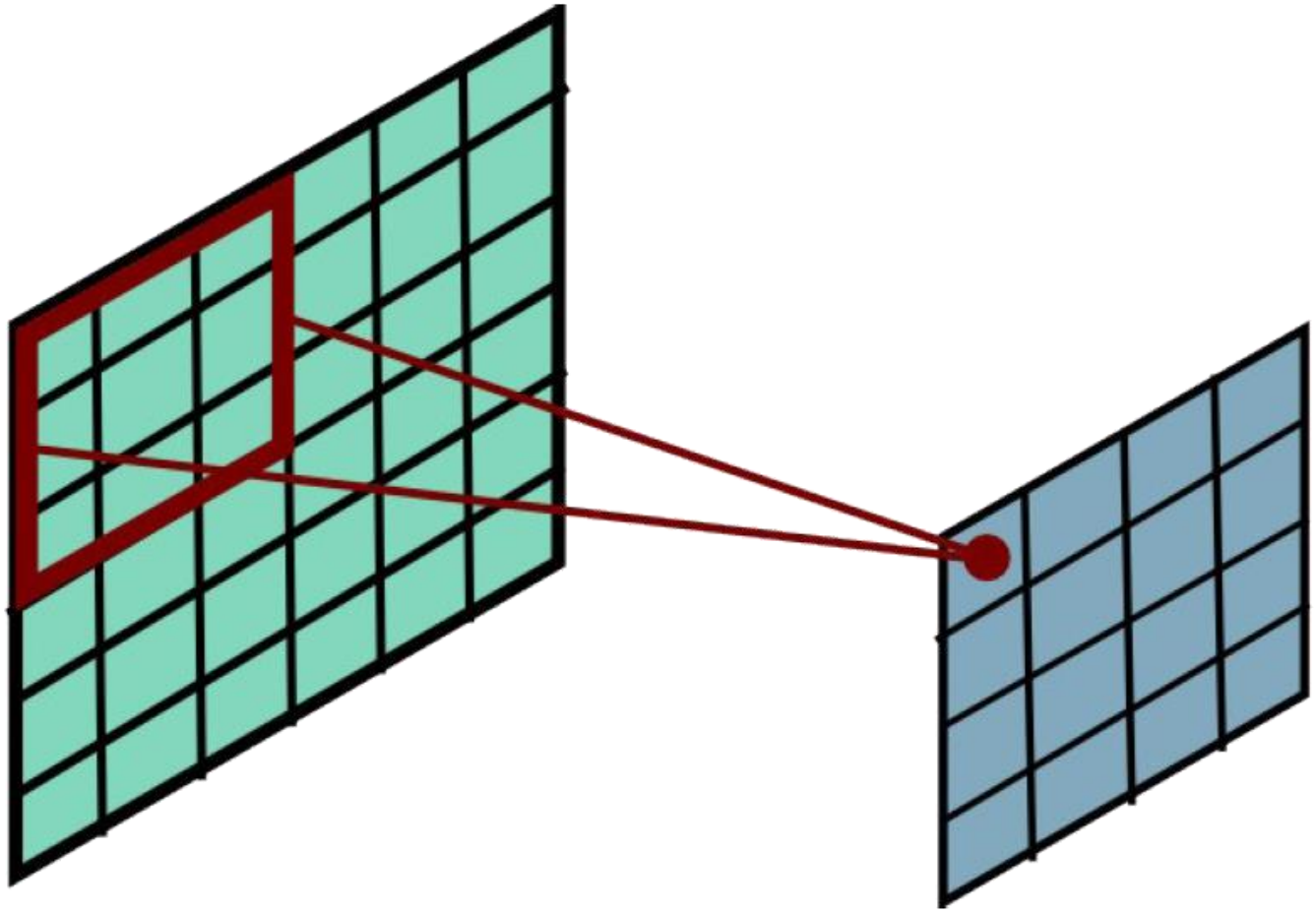
32x32x3 image

5x5x3 filter w

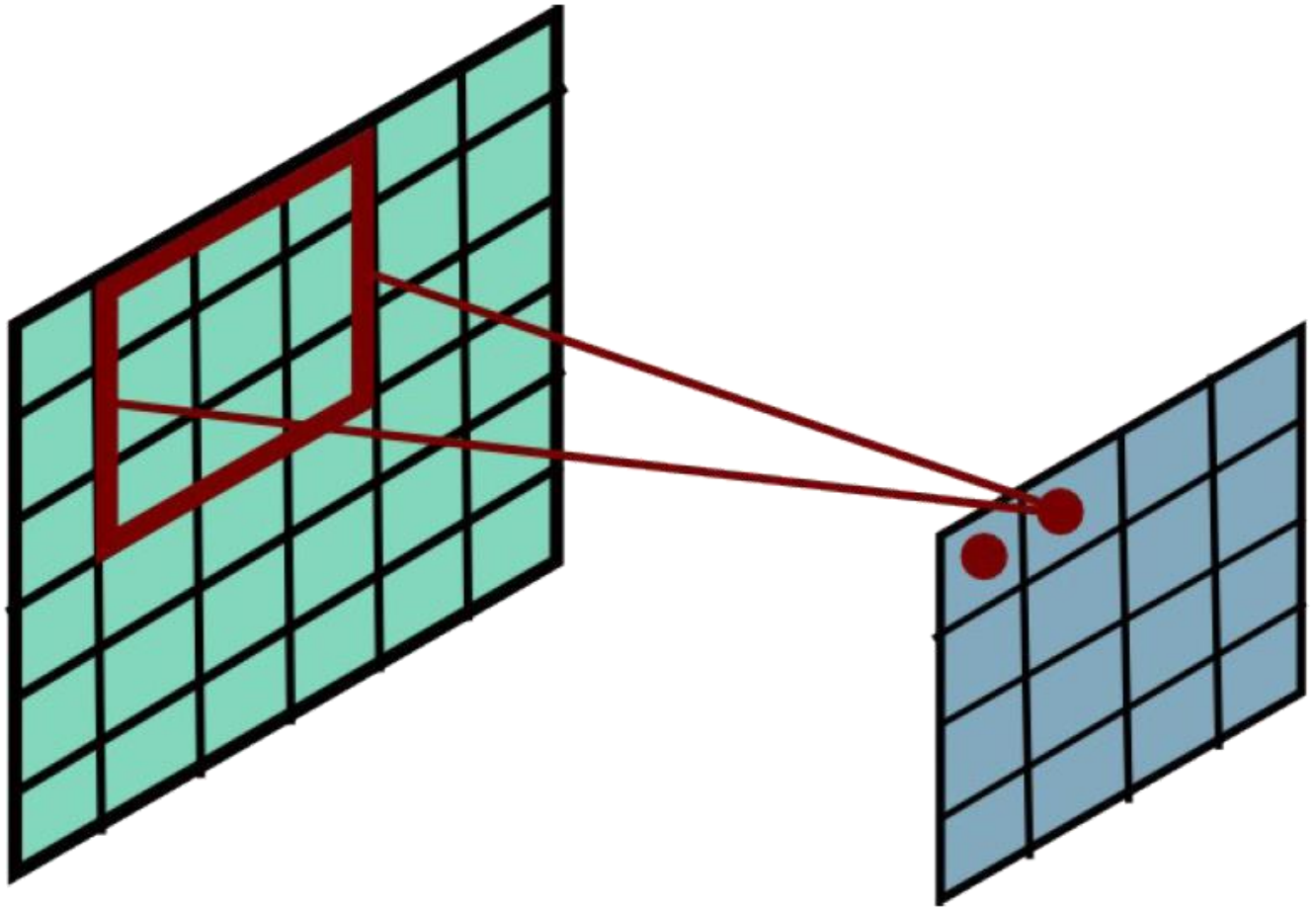
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

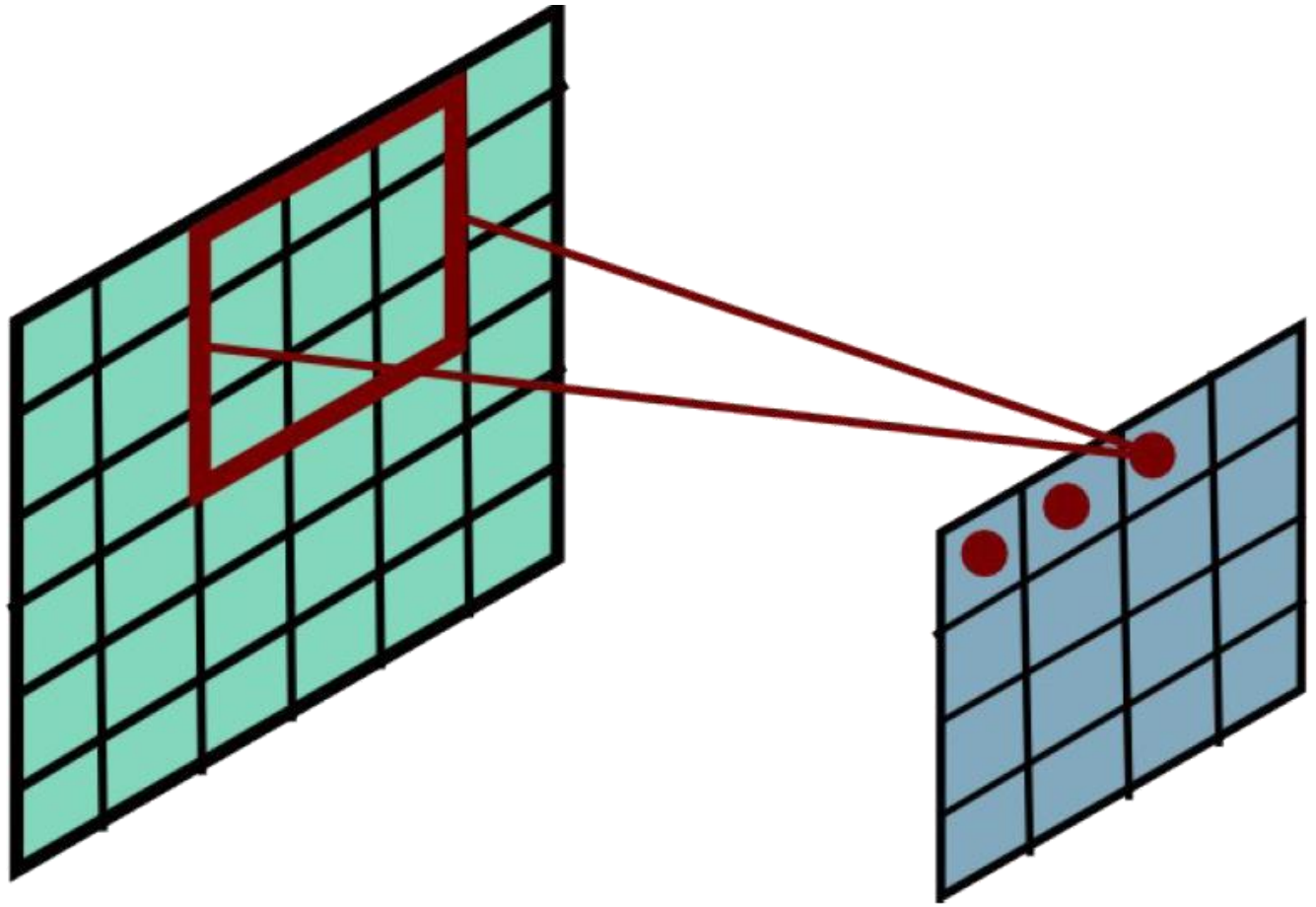
Convolution Layer (4/9)



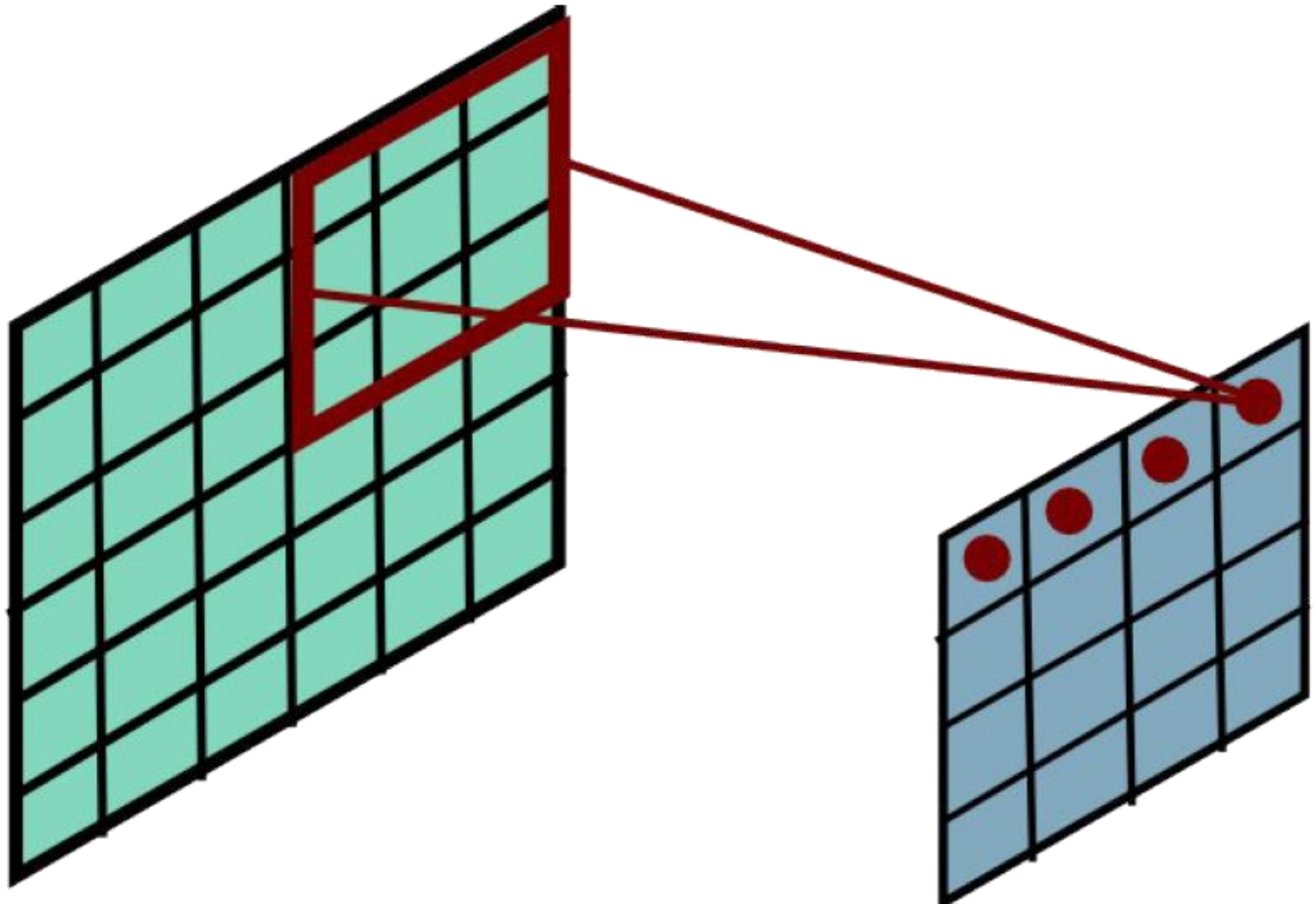
Convolution Layer (5/9)



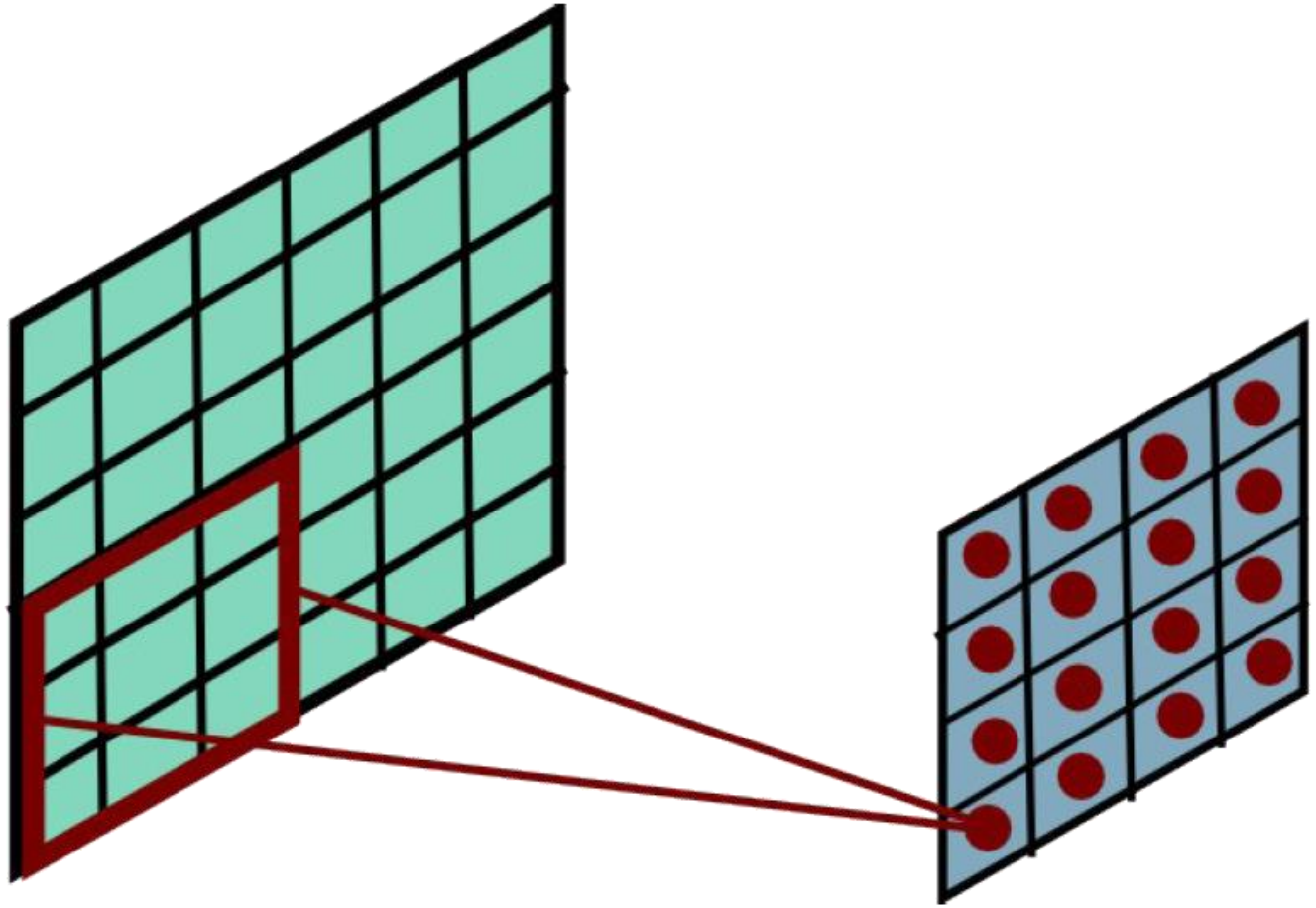
Convolution Layer (6/9)



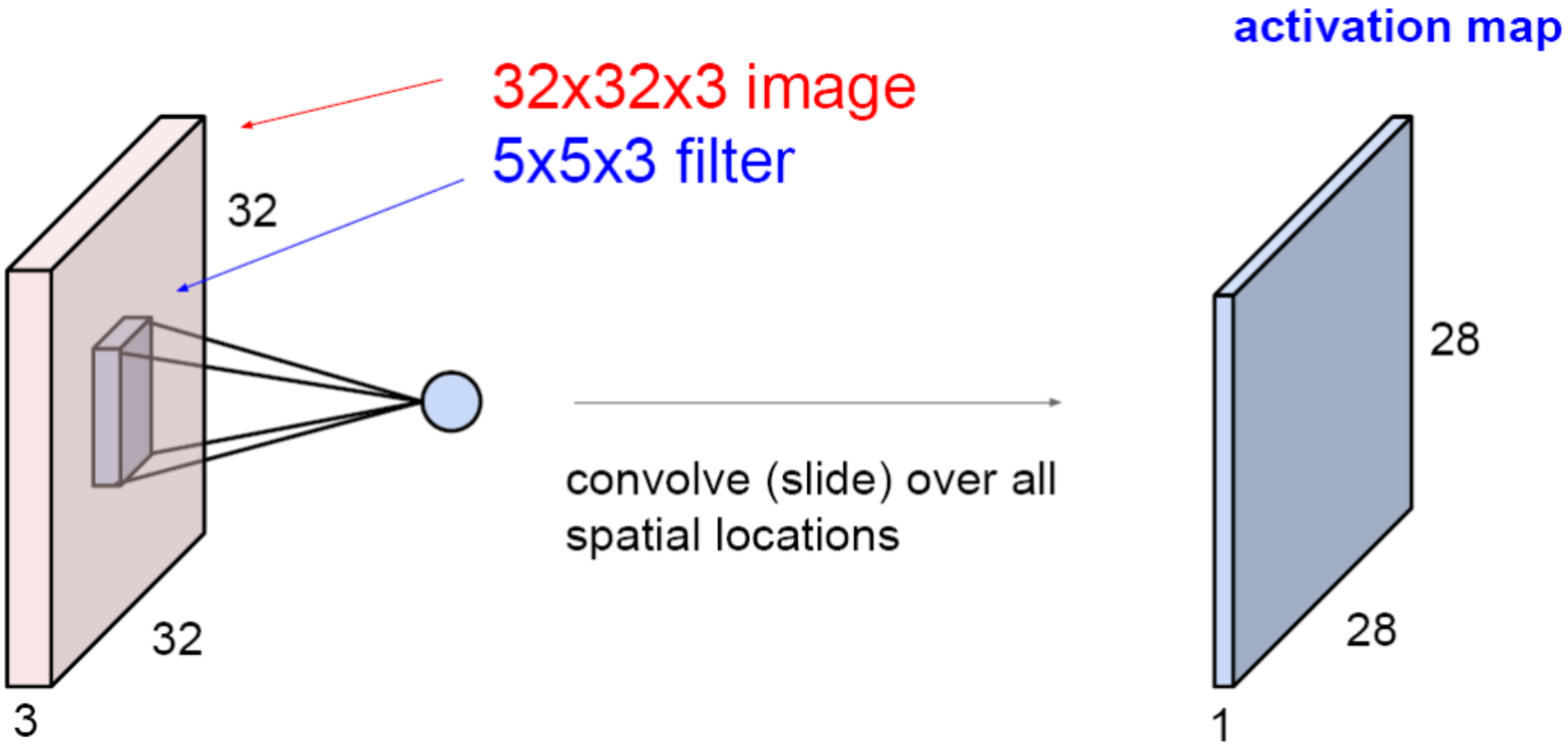
Convolution Layer (7/9)



Convolution Layer (8/9)

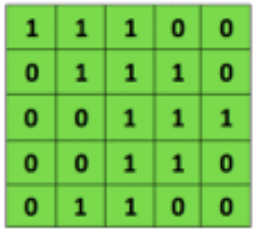


Convolution Layer (9/9)

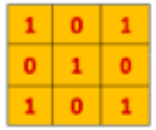


Convolutions In Deep Learning

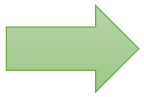
- Loosely mimics neuronal responses
- To compute:
  - Slide small kernel over input
  - Element-wise multiplication and sum



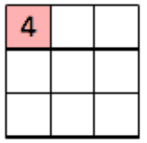
Input matrix



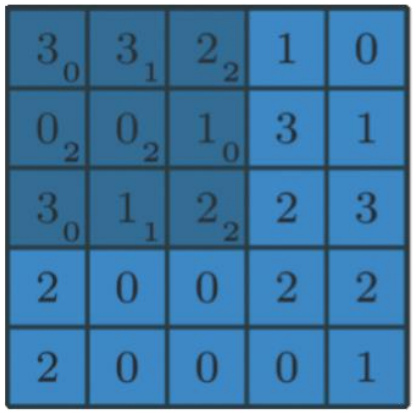
Convolutional 3x3 filter



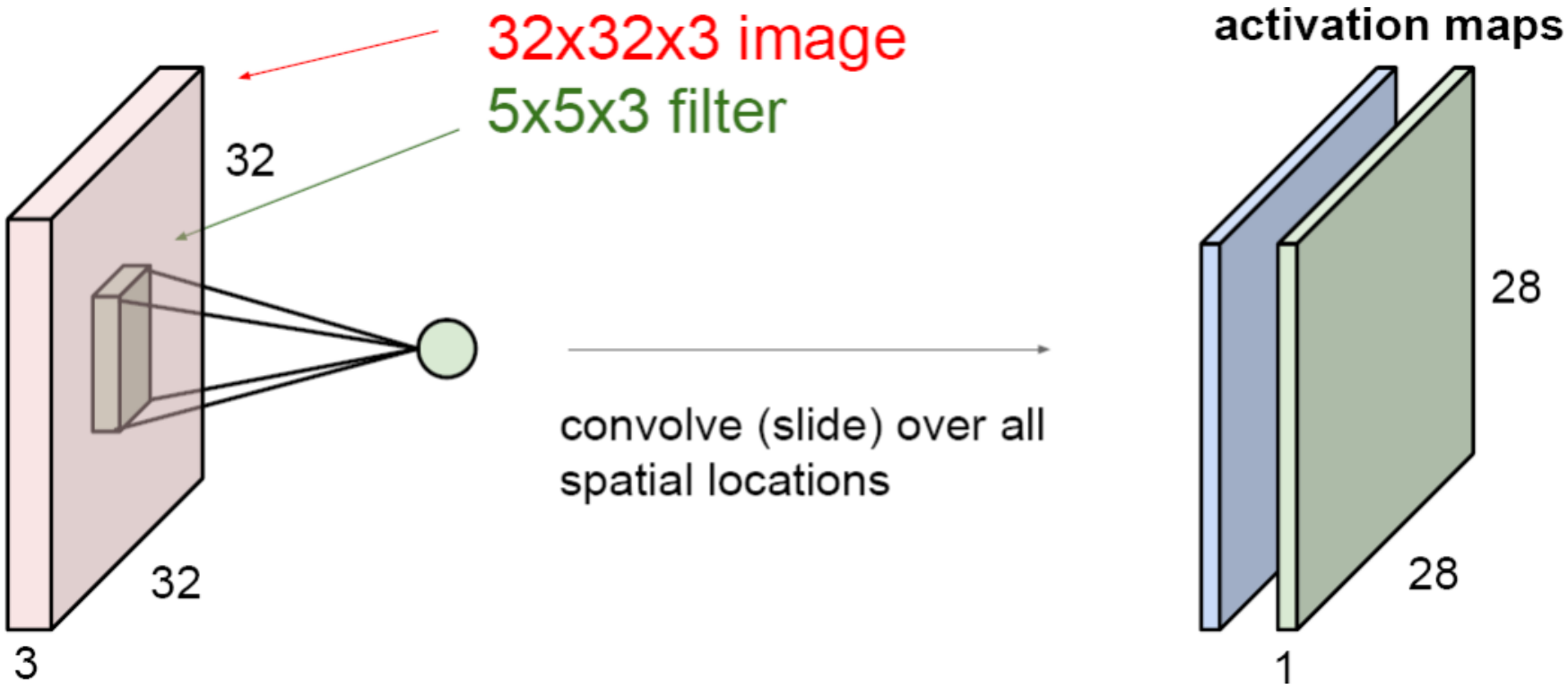
Image



Convolved Feature



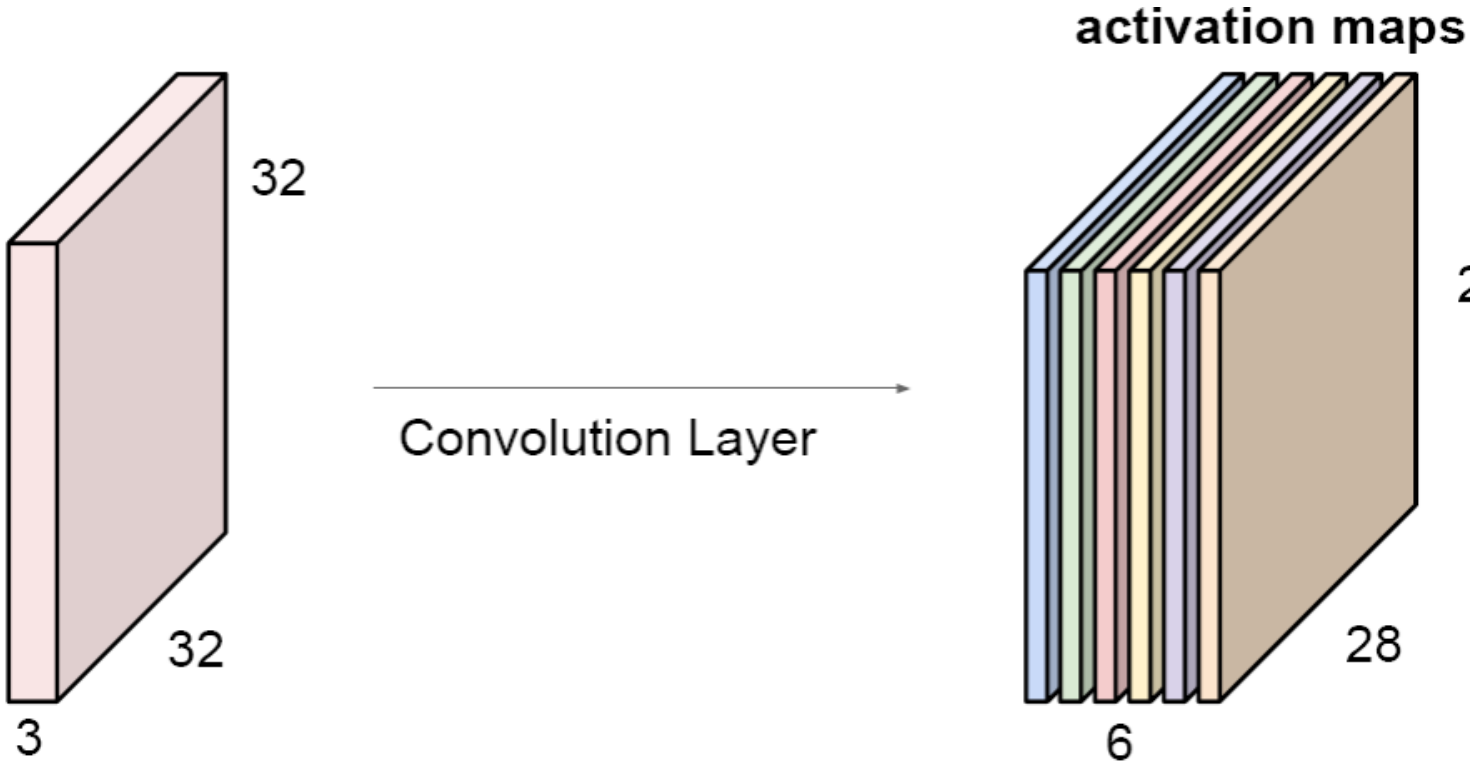
Convolution Layer (1/2)



consider a second, green filter

Convolution Layer (2/2)

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!

Motivation: Convolution for Edge Detection

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

Vertical lines

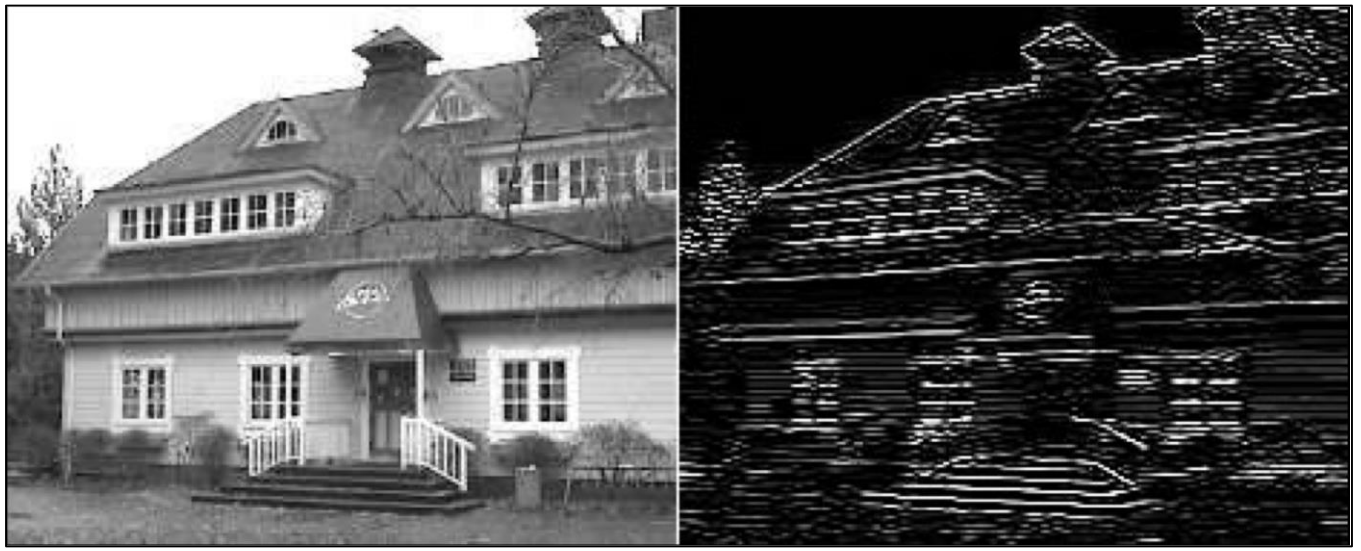
-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines

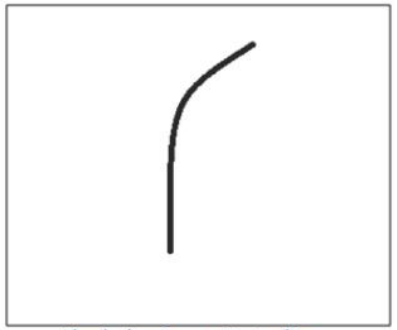
Horizontal lines:



Cross-correlation In Two Dimensions

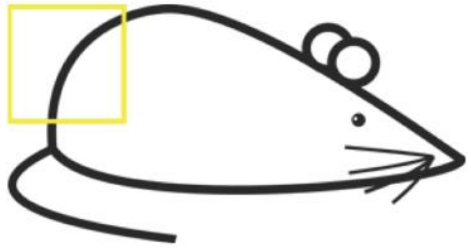
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

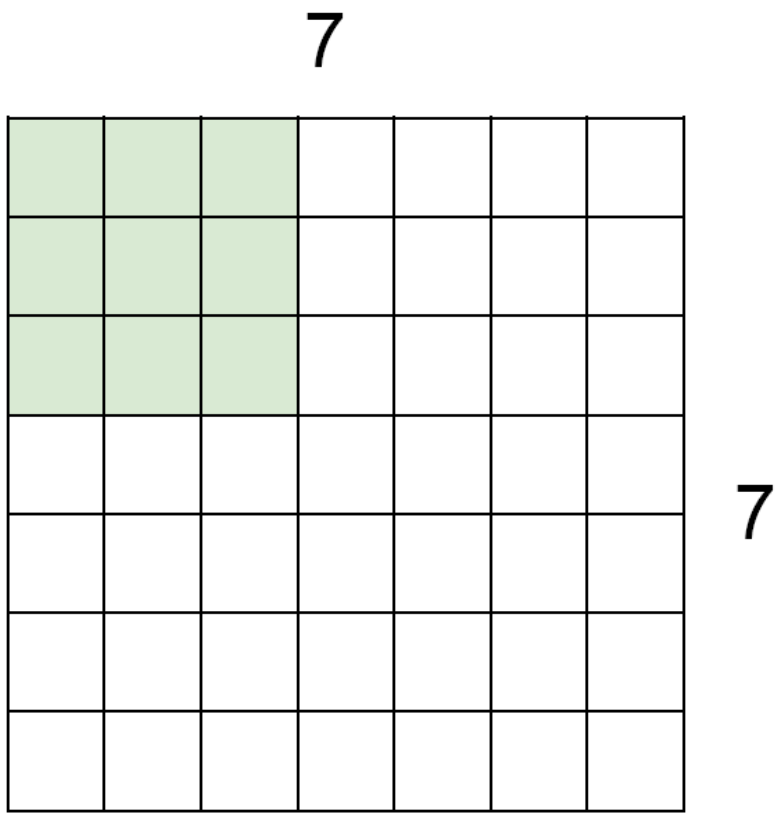
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

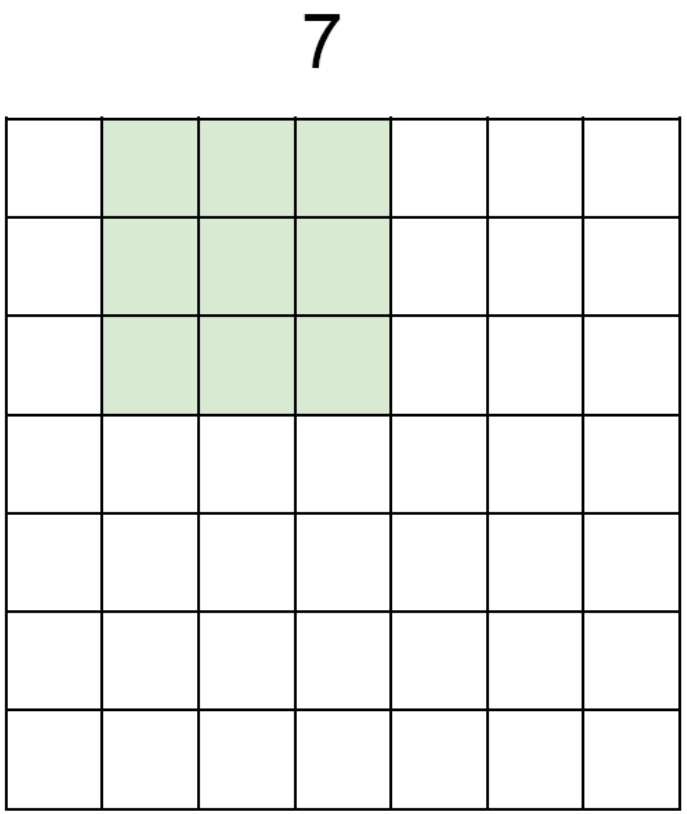
Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Spatial Dimensions (1/11)



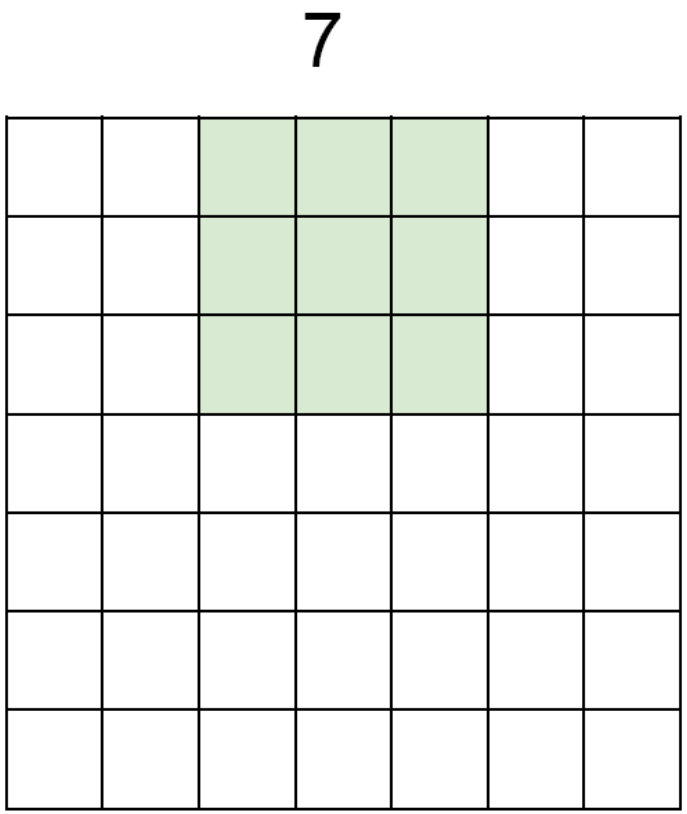
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions (2/11)



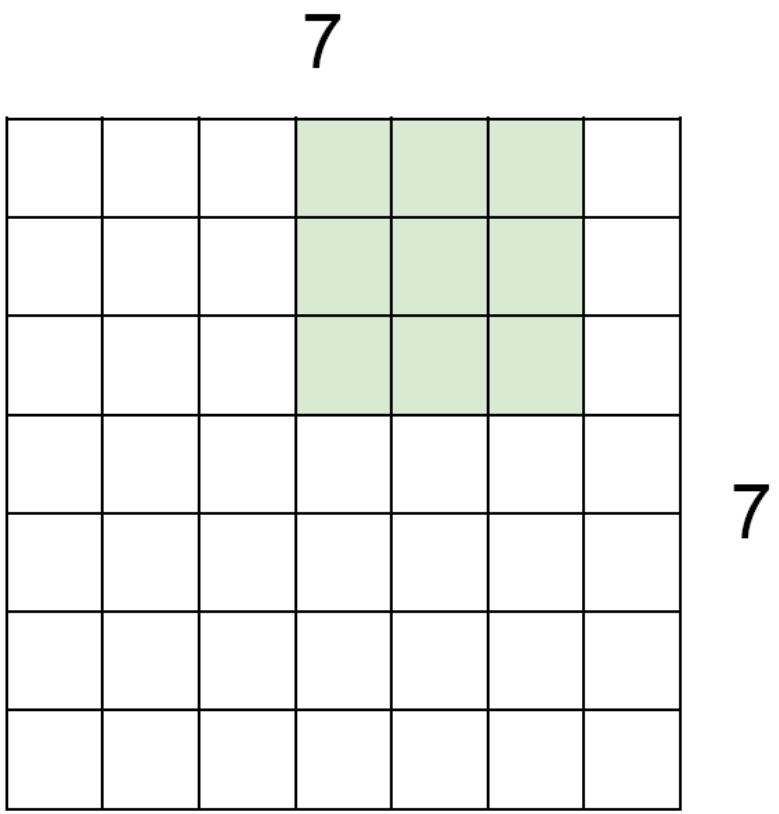
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions (3/11)



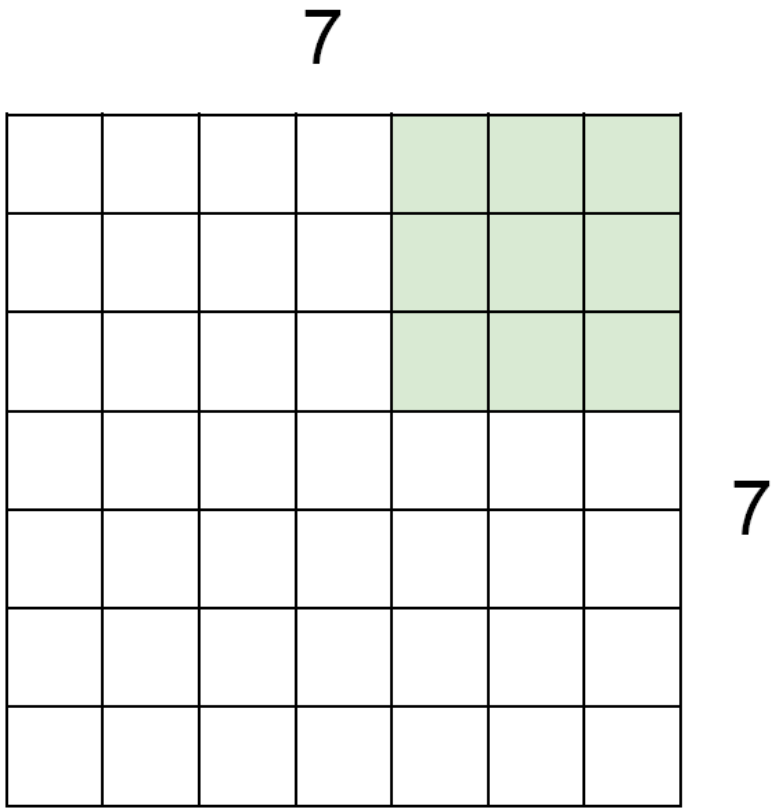
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions (4/11)



7x7 input (spatially)
assume 3x3 filter

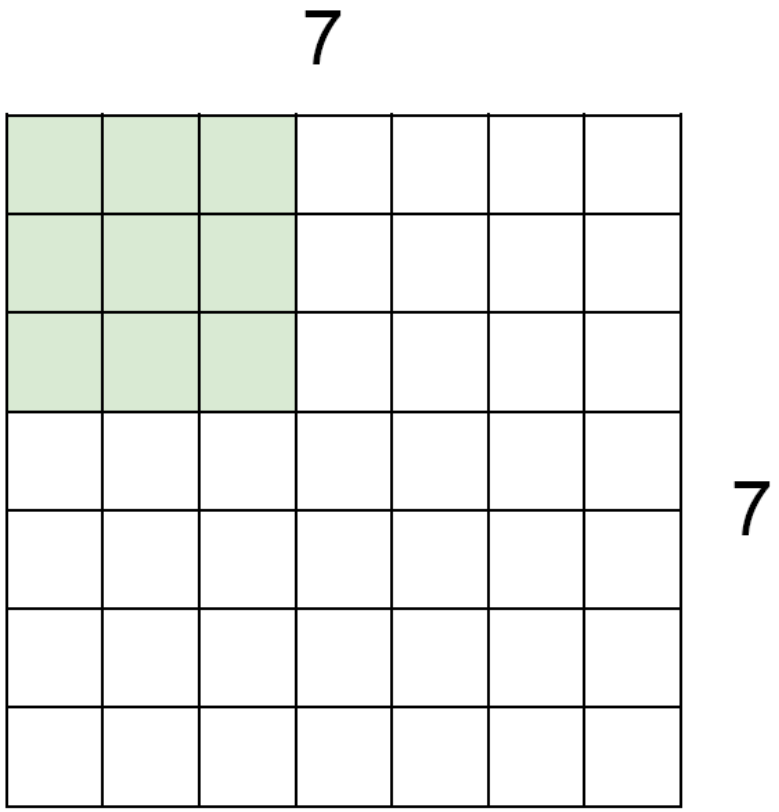
Spatial Dimensions (5/11)



7x7 input (spatially)
assume 3x3 filter

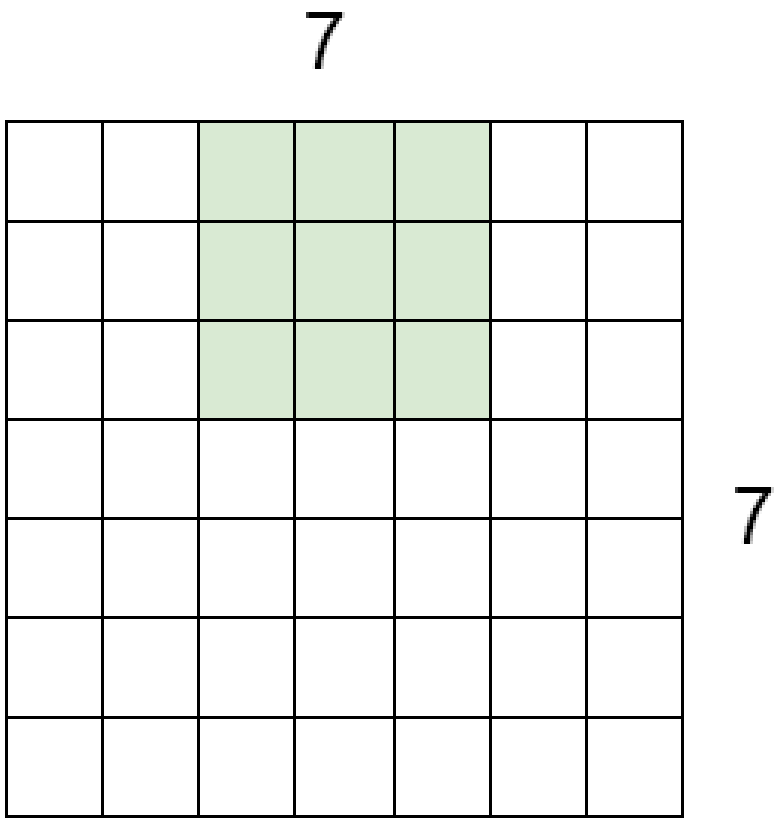
=> 5x5 output

Spatial Dimensions (6/11)



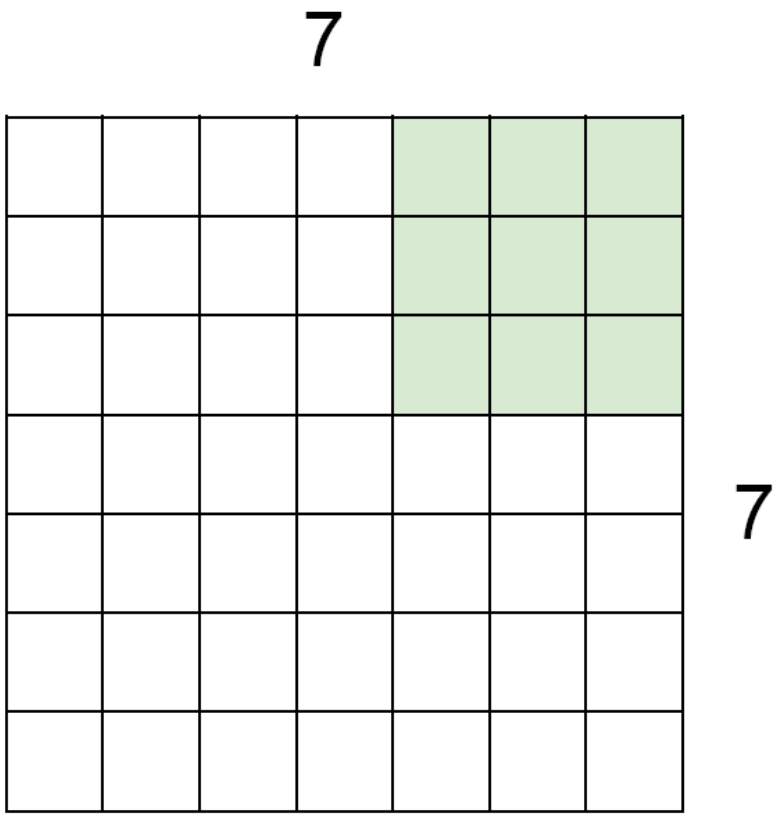
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Spatial Dimensions (7/11)



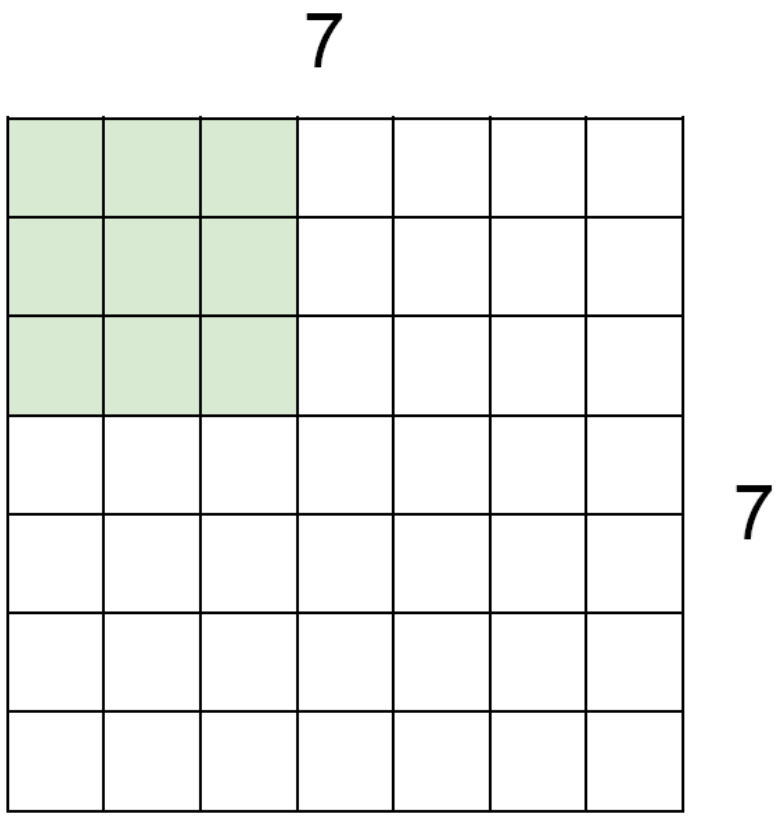
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Spatial Dimensions (8/11)



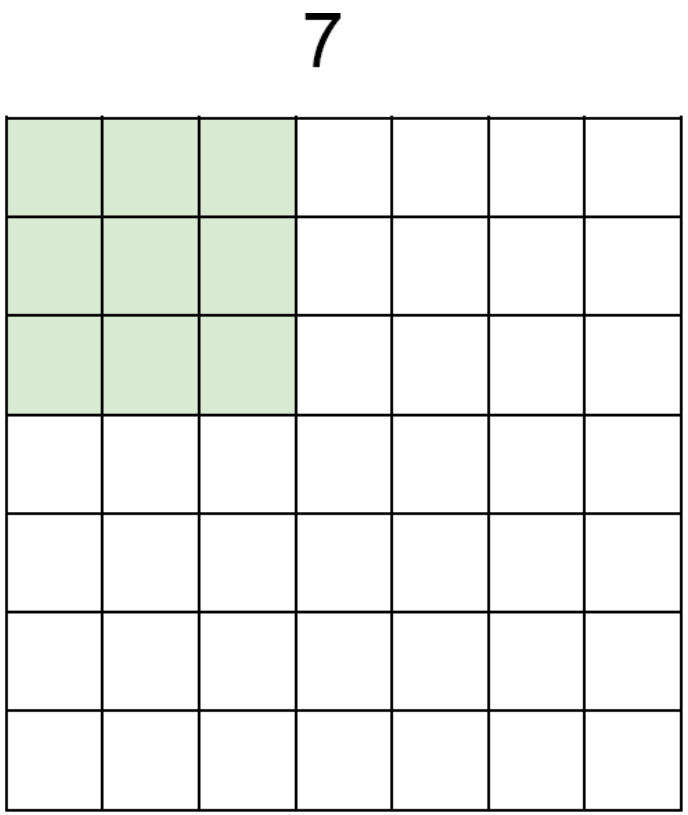
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Spatial Dimensions (9/11)



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

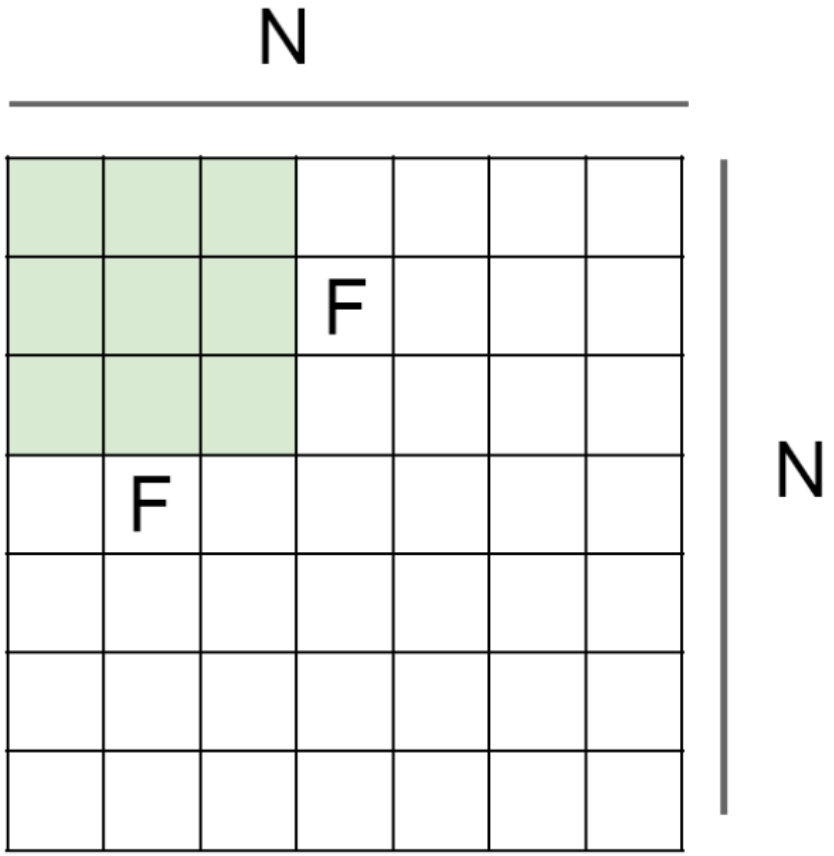
Spatial Dimensions (10/11)



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

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

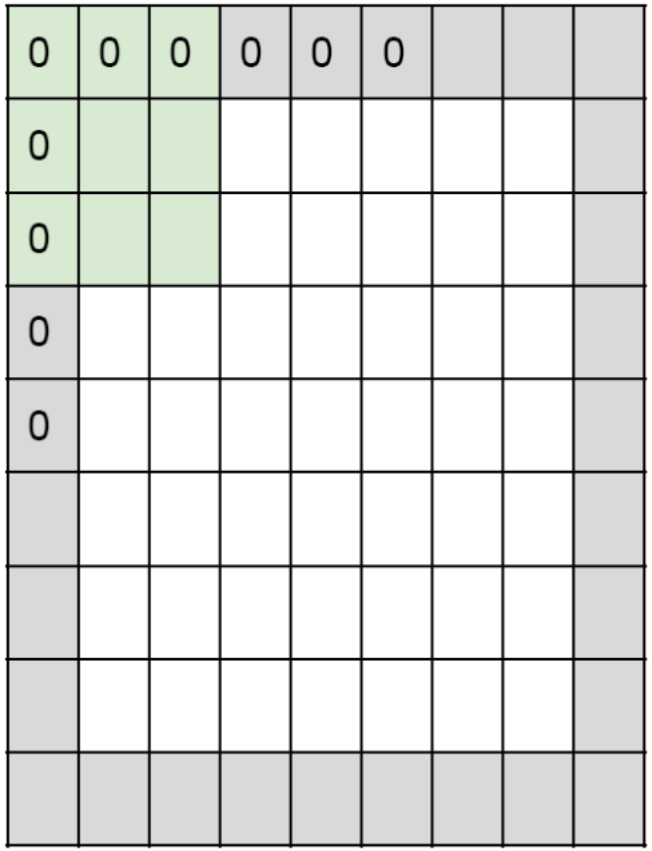
Spatial Dimensions (11/11)



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 :\

Common to Zero Pad the Border

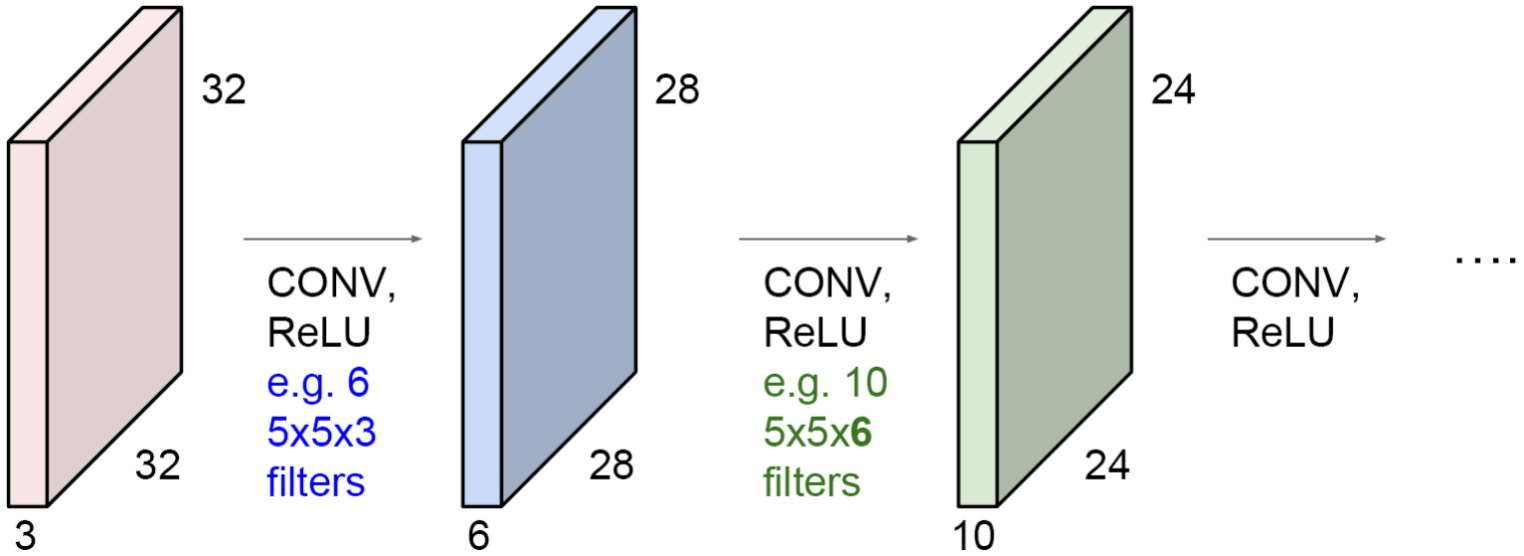


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Multiple Convolution Layer

- Example:
  - 32 x 32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 ➔ 28 ➔ 24 ...). Shrinking too fast is not good, doesn't work well.



Convolution Summary (1/2)

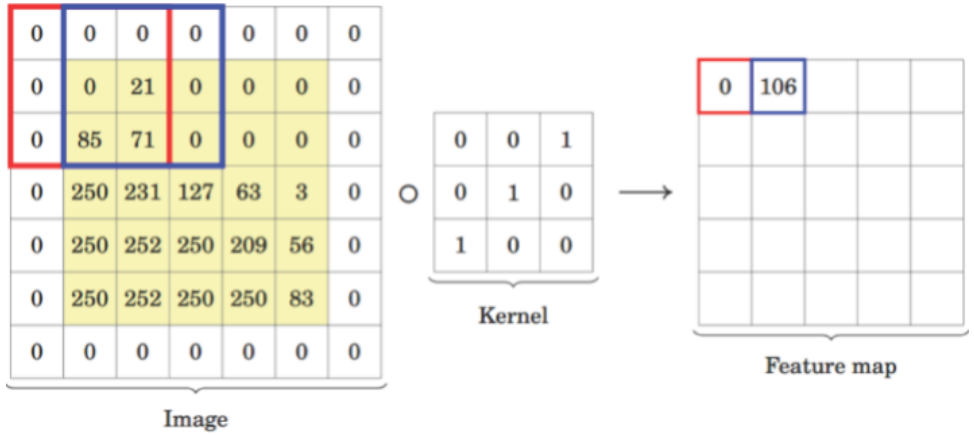
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (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.

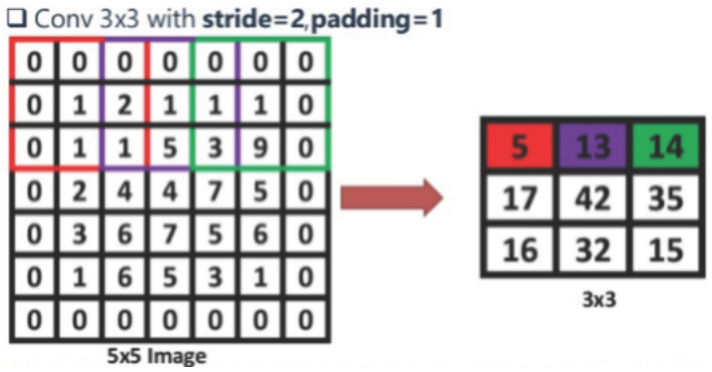
Common settings:

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

Convolution Summary (2/2)

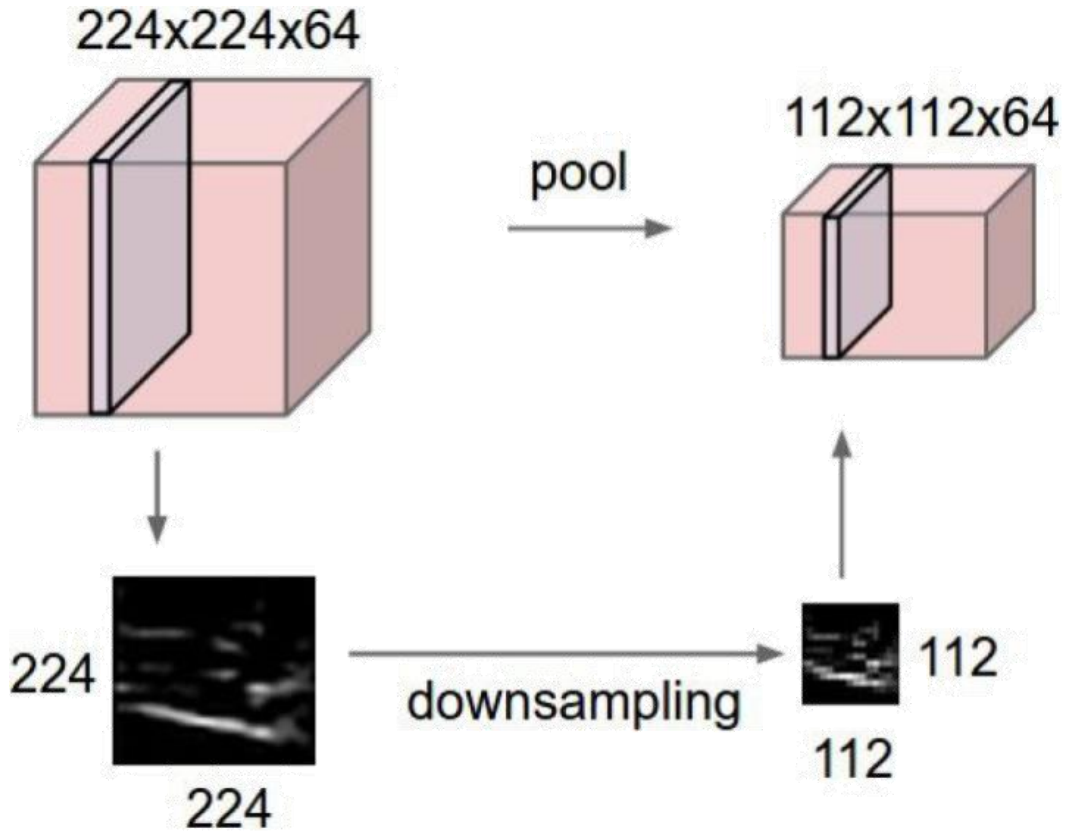
- Layers structured as matrices, not vectors
- Convolution of matrices by small (e.g. 3 x 3) matrices called **kernels**
- The kernels are the learned weights
- Depending on **padding** and **stride**, image may decrease slightly in size
- Intuitively, conv layers learn how to relate neighboring pixels
- Image layers in a CNN typically have many **features**
- That means kernels are 4D, e.g. 3 x 3 **x 16 x 32**





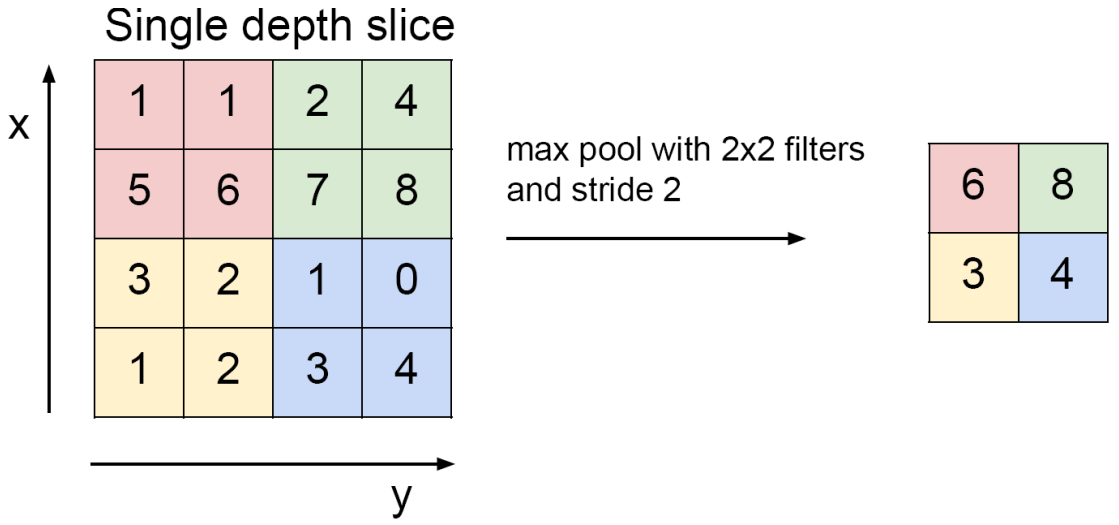
Pooling Layer

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



Max Pool (1/2)

- Intuitively, **Max Pool** is used because nearby pixels will give very similar output
- Over the course of a network, pooling shifts layers from spatial dimensions to features
- e.g.:

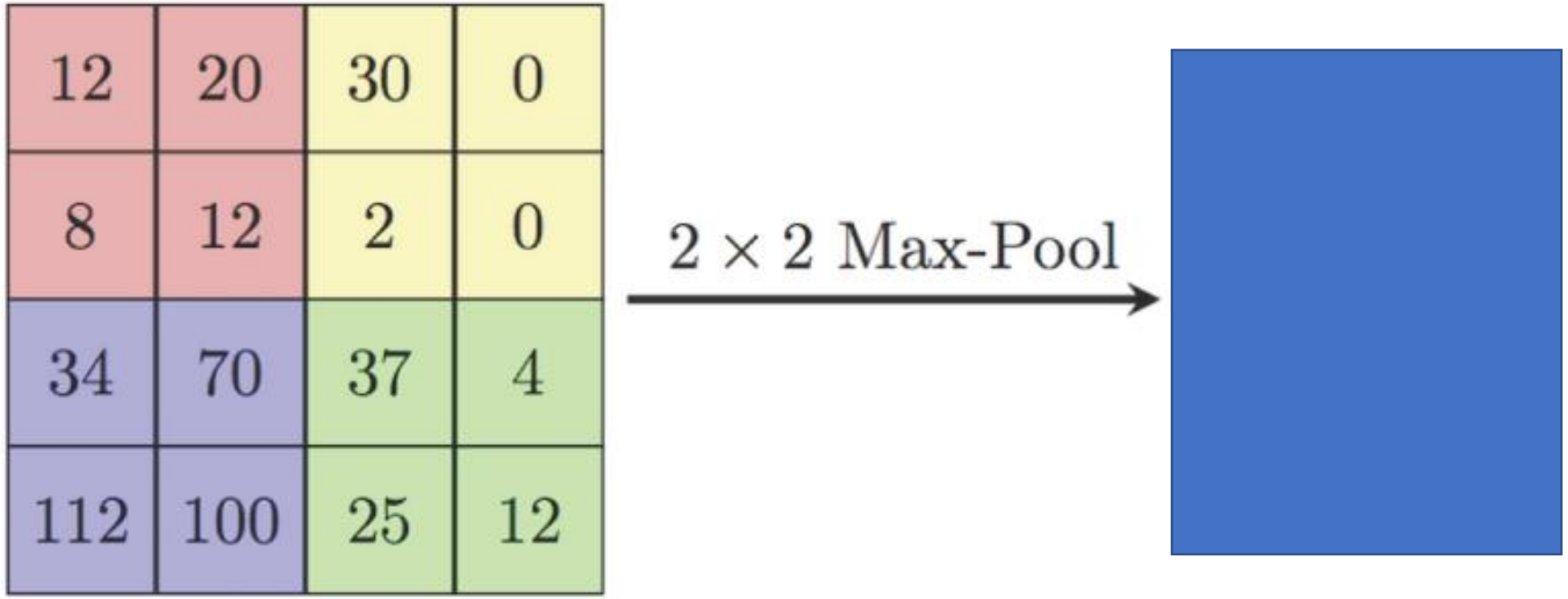


Conv **Pool** **Conv** **Pool**

28 x 28 x 1 => 26 x 26 x 16 => 13 x 13 x 16 => 11 x 11 x 64 => 6 x 6 x 64 => ...

Max Pool (2/2)

- Practice

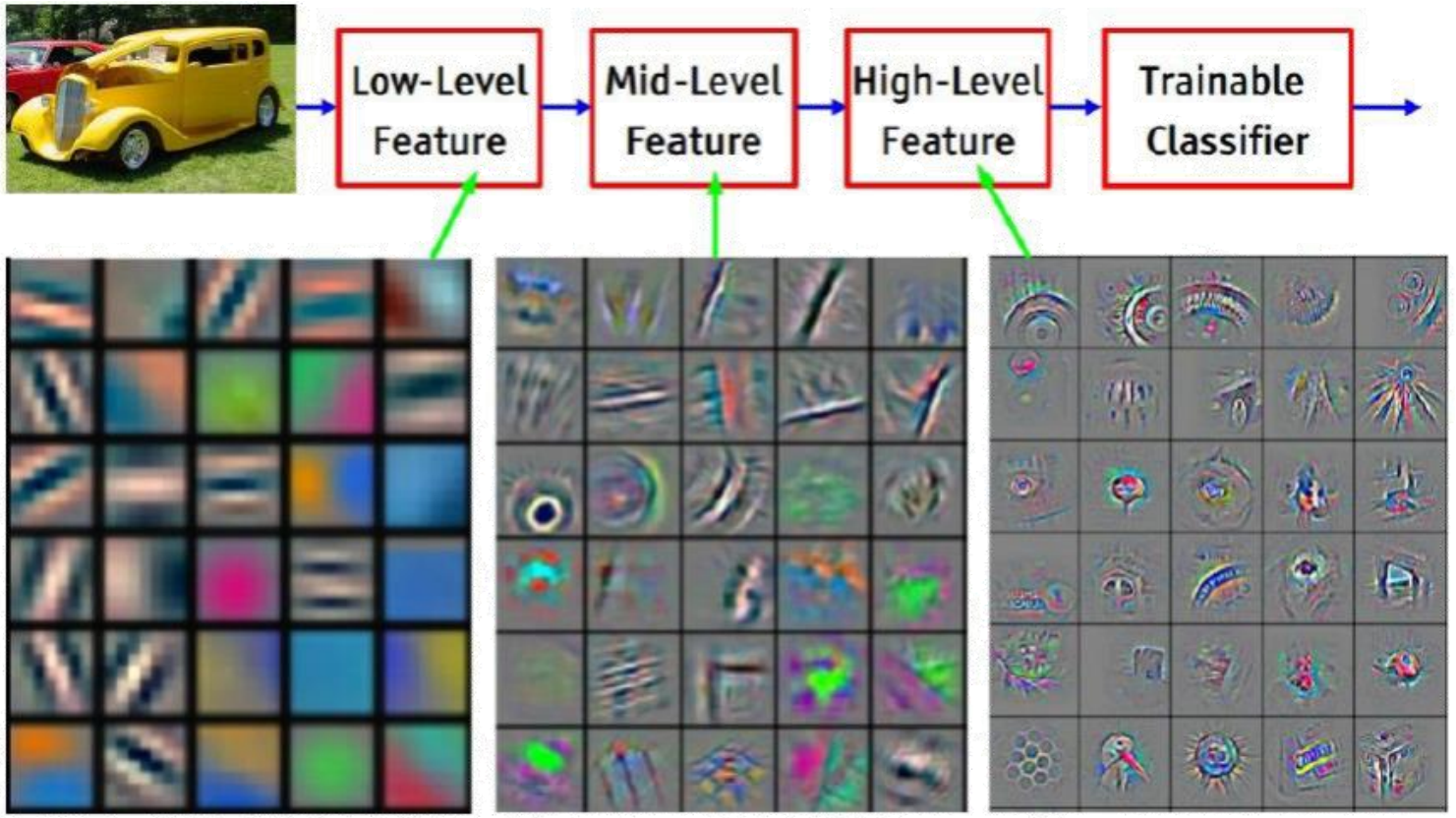


Pooling Summary

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters
  - Their spatial extent $\boldsymbol{F}$,
  - The stride S,
- Produces a volume of the size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F) / S + 1$

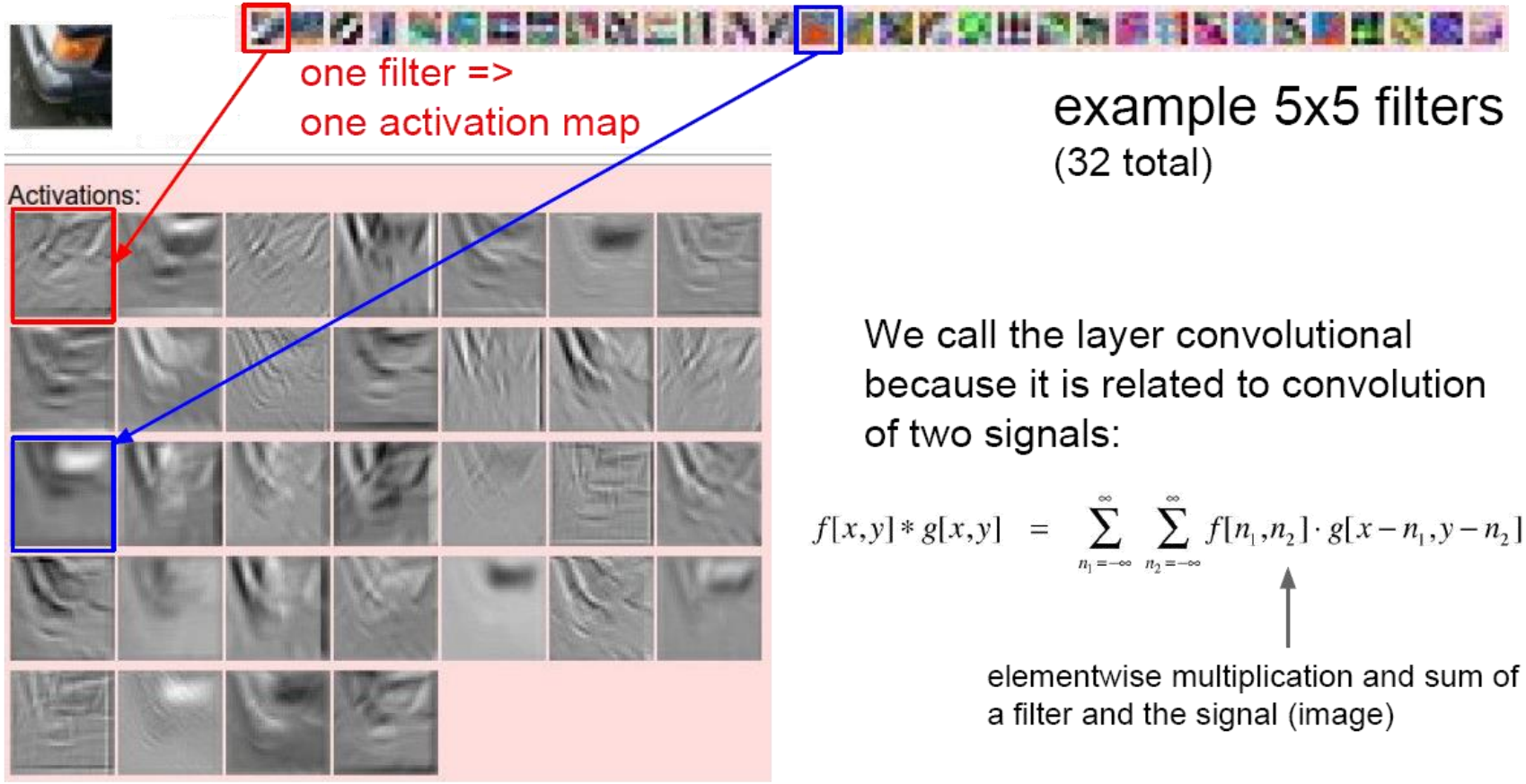
Common settings:
F=2, S=2
F=3, S=2

What CNNs Learn (1/3)



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

What CNNs Learn (2/3)



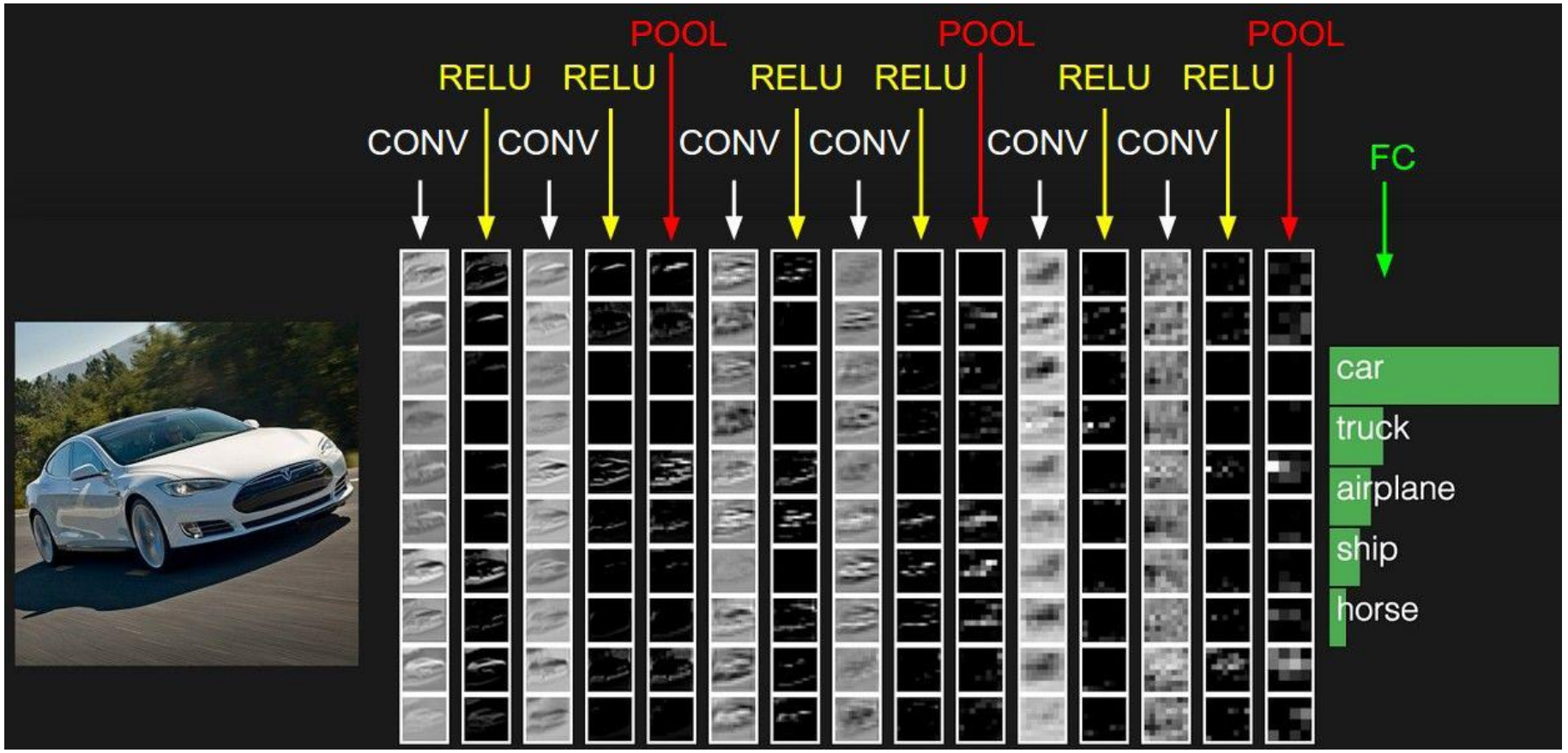
example 5x5 filters
(32 total)

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

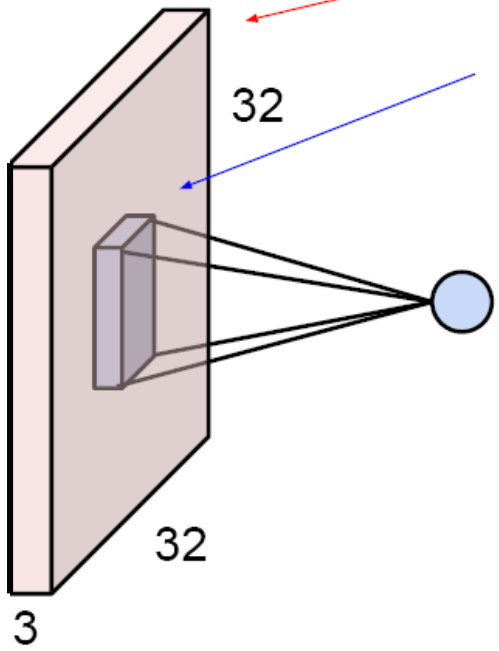
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

What CNNs Learn (3/3)



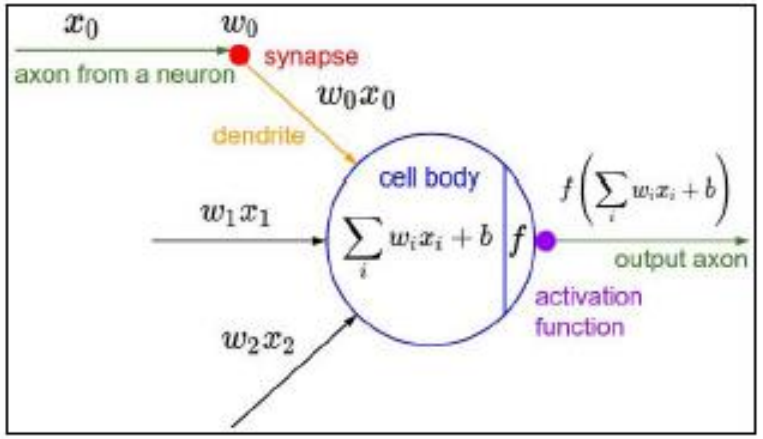
The Brain/Neuron View of CONV Layer (1/3)



32x32x3 image

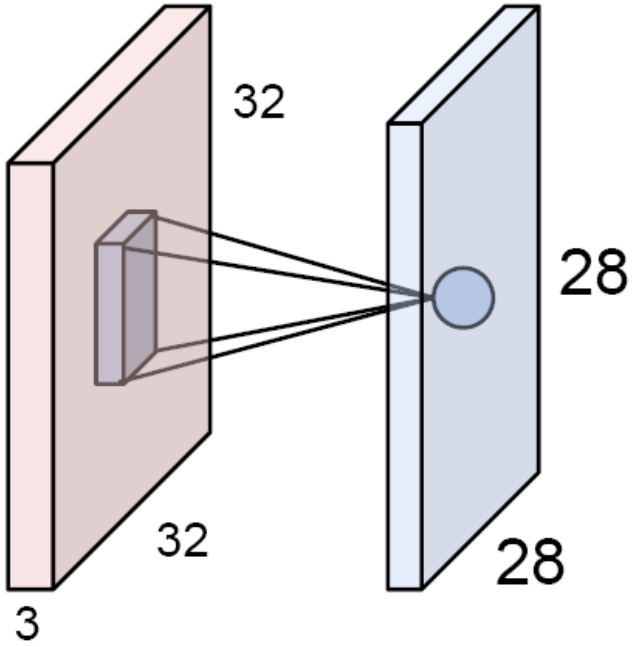
5x5x3 filter

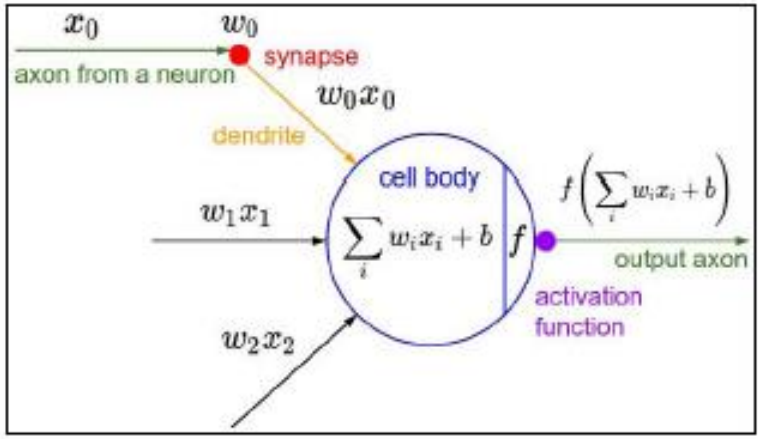
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)



It's just a neuron with local connectivity…

The Brain/Neuron View of CONV Layer (2/3)



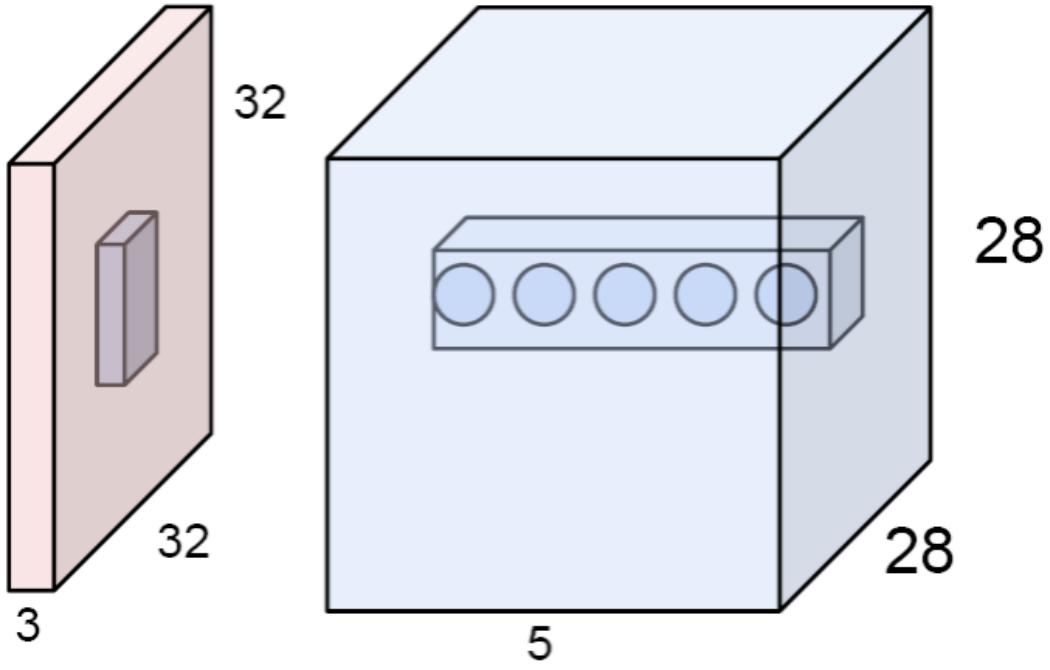


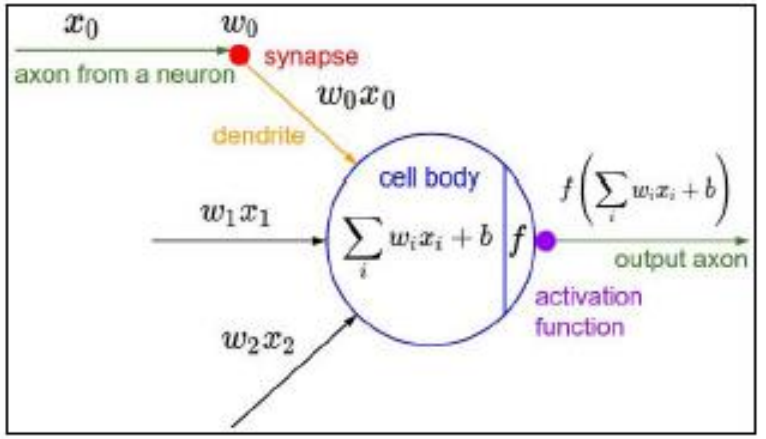
An activation map is a 28x28 sheet of neuron outputs:

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 (3/3)

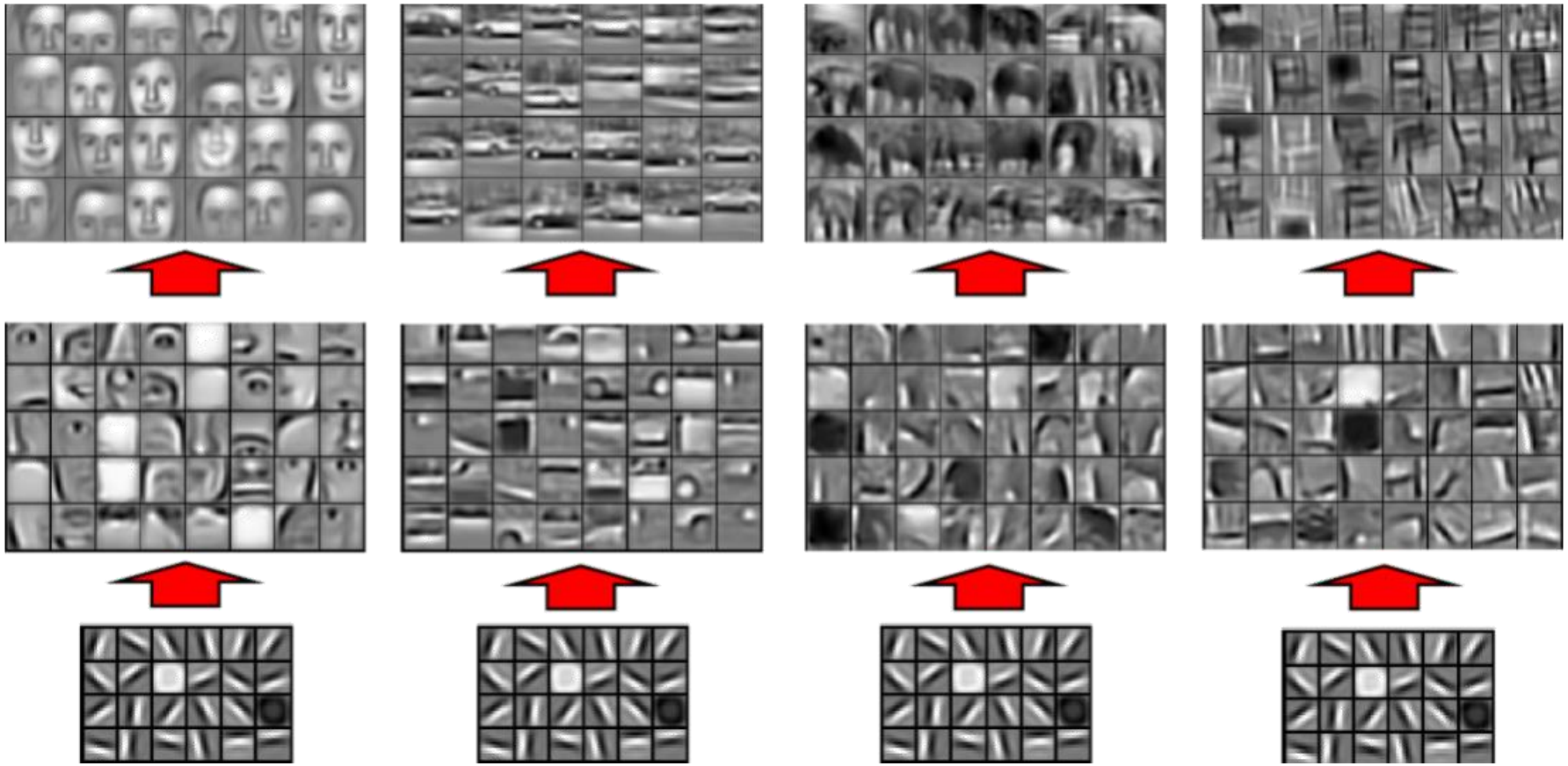




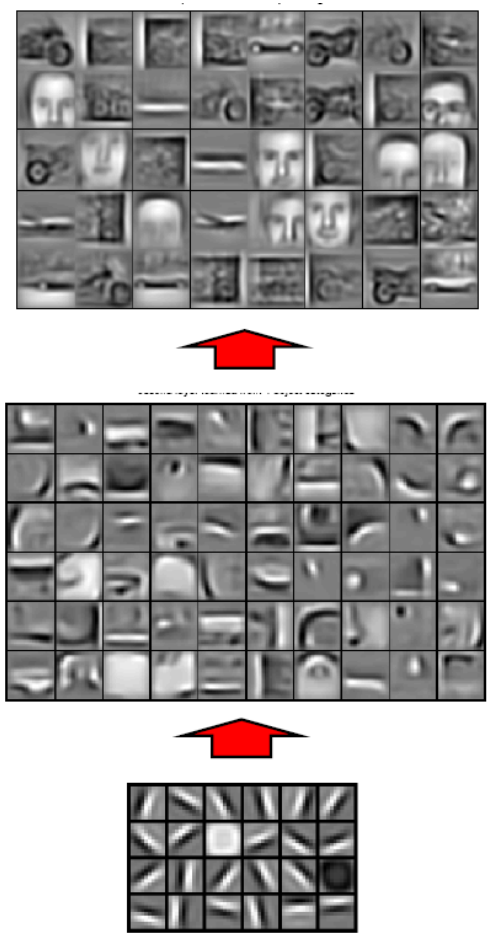
E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

Learning of Object Parts



Training on Multiple Objects



Trained on 4 classes (cars, faces, motorbikes, airplanes).

Second layer: Shared-features and object-specific features.

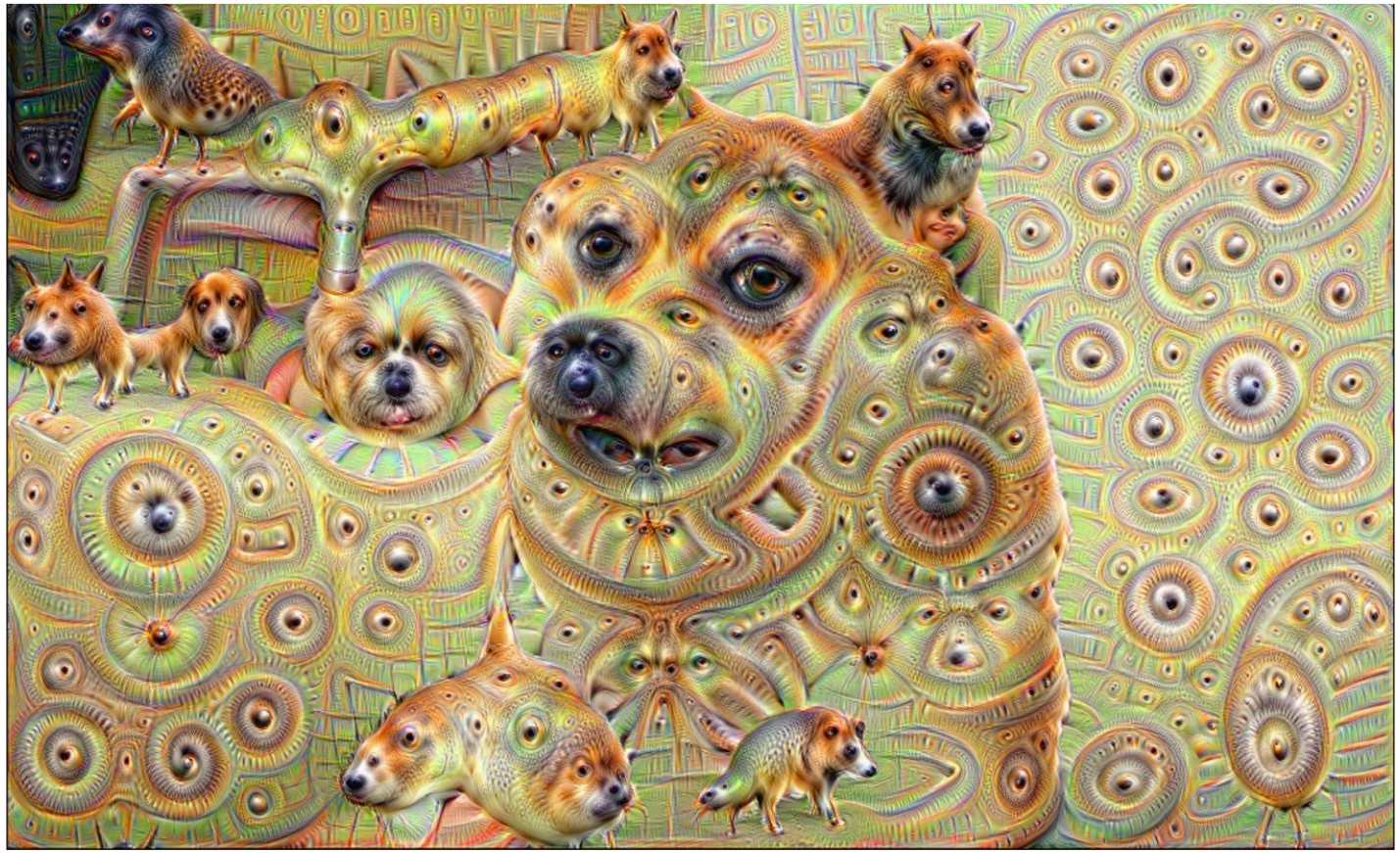
Third layer: More specific features.

Visualizing Neurons In a Network

- https://distill.pub/2017/feature-visualization/
- Visualization by optimizing image to maximize activation at particular neurons/layers.
- How does this work?

DeepDream (1/2)

DeepDream (2/2)



A Convolution Exercise (1/6)

- Suppose we want to find out whether the following image depicts Cartesian axes.
- We do so by convolving the image with two filters (no padding, stride of 1) and applying a max-pool operation with kernel width of 2.
- Compute the output by hand.

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix}$$

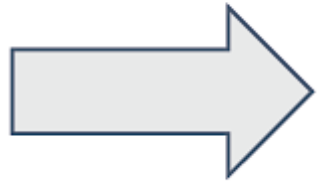
Image

$$\begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 1 & 1 & 1 \\ -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}$$

Filters

A Convolution Exercise (2/6)

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \qquad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$$



$$\begin{bmatrix} 2 & . \\ . & . \end{bmatrix}$$

$$\left(0 \times \frac{-1}{2}\right) + (1 \times 1) + \left(0 \times \frac{-1}{2}\right)$$
$$\left(0 \times \frac{-1}{2}\right) + (1 \times 1) + \left(0 \times \frac{-1}{2}\right)$$
$$\left(0 \times \frac{-1}{2}\right) + (1 \times 1) + \left(0 \times \frac{-1}{2}\right) = 2$$

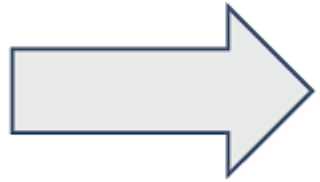
A Convolution Exercise (3/6)

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$$

$$\begin{bmatrix} 2 & -2 \\ . & . \end{bmatrix}$$

A Convolution Exercise (4/6)

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$$



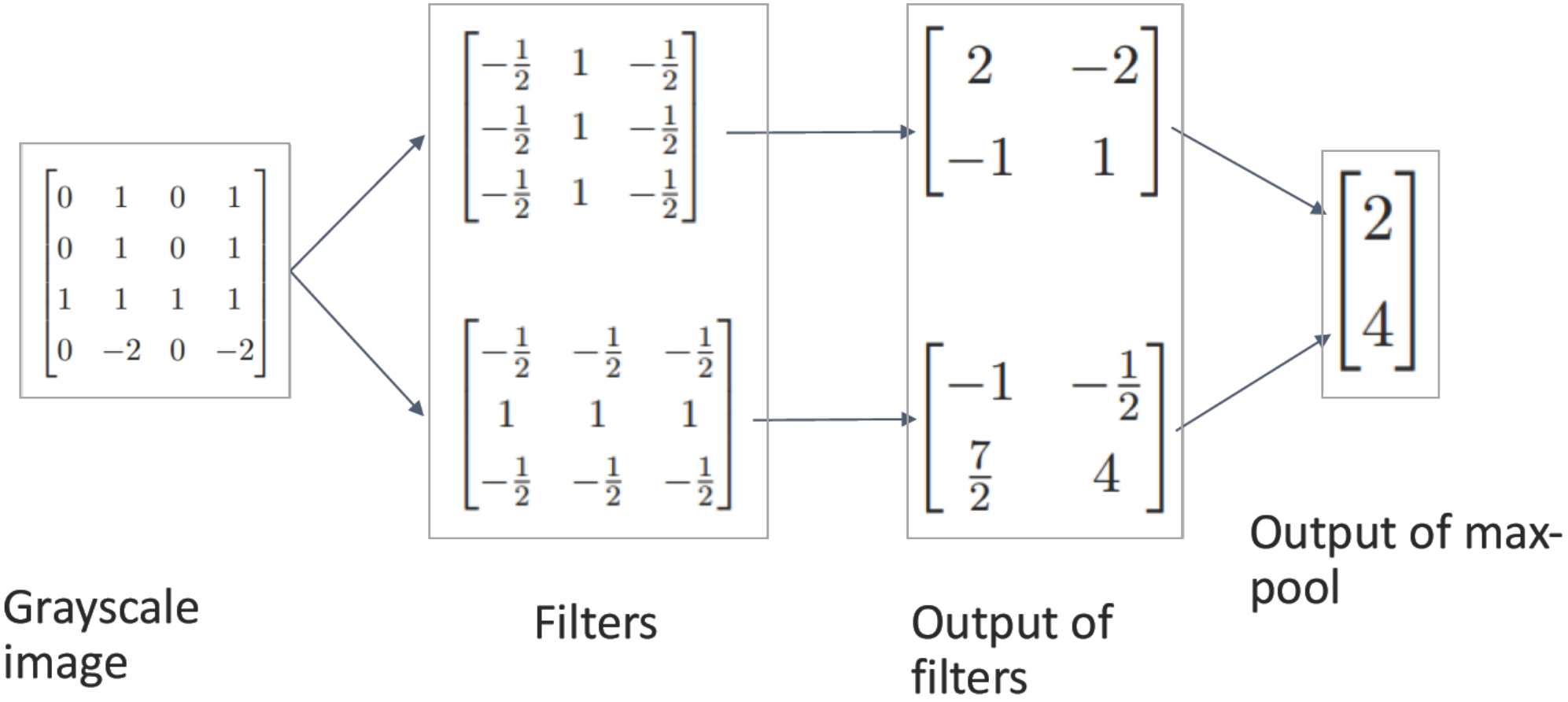
$$\begin{bmatrix} 2 & -2 \\ -1 & . \end{bmatrix}$$

A Convolution Exercise (5/6)

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \end{bmatrix} \quad \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$$

$$\begin{bmatrix} 2 & -2 \\ -1 & 1 \end{bmatrix}$$

Convolution Exercise: Solution (6/6)

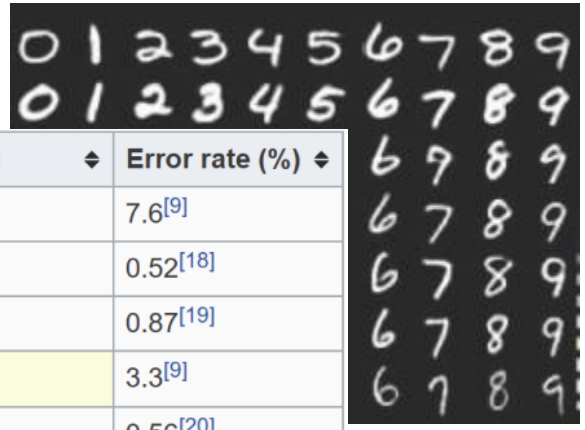


CNNs Are Translation-invariant By Design

- Although we would like many other kinds of invariance, none are baked into the architecture of CNNs.

Some Classic CNNs

CNNs Perform Extremely Well on MNIST



Type	Classifier	Distortion	Preprocessing	Error rate (%)
Linear classifier	Pairwise linear classifier	None	Deskewing	7.6[9]
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	None	Shiftable edges	0.52[18]
Boosted Stumps	Product of stumps on Haar features	None	Haar features	0.87[19]
Non-linear classifier	40 PCA + quadratic classifier	None	None	3.3[9]
Support vector machine	Virtual SVM, deg-9 poly, 2-pixel jittered	None	Deskewing	0.56[20]
Deep Neural network	2-layer 784-800-10	None	None	1.6[21]
Deep Neural network	2-layer 784-800-10	elastic distortions	None	0.7[21]
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	elastic distortions	None	0.35[22]
Convolutional neural network	6-layer 784-40-80-500-1000-2000-10	None	Expansion of the training data	0.31[15]
Convolutional neural network	6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.27[23]
Convolutional neural network	Committee of 35 CNNs, 1-20-P-40-P-150-10	elastic distortions	Width normalizations	0.23[8]
Convolutional neural network	Committee of 5 CNNs, 6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.21[17]

Example Implementation

- Hello World! of object recognition DL is the MNIST dataset for handwritten digit recognition
- MNIST dataset is available at http://yann.lecun.com/exdb/mnist/
- However, many modern DL libraries provide a convenience method for loading the MNIST dataset (no need to separately download!)
- It is stored in your home directory, as 15MB *mnist.pkl.gz* file
- Let's implement a CNN for hand-written digit recognition on MNIST

Example Implementation

- Let’s implement hand-written digit recognition using MLP
- Discuss pros/cons, compare with CNN

MNIST in PyTorch

- MNIST images have size 28x28x1 (black-and-white images have 1 feature / pixel)
- Design 2 conv. layers that reduce images to 7x7 and create 32 features. Specify your filter size, stride, and pooling kernel size.
- Finish classifying using a feedforward net. There are 10 labels.

Now Let's Develop it with PyTorch

- https://pytorch.org/docs/stable/nn.html#convolution-functions

```
# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7*7*32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
```

CNNs Perform Reasonably Well On CIFAR-10

- Feedforward neural nets achieve about 50% accuracy

Research Paper	Error rate (%)	Publication Date
Convolutional Deep Belief Networks on CIFAR-10[6]	21.1	August, 2010
Densely Connected Convolutional Networks[7]	5.19	August 24, 2016
Wide Residual Networks[8]	4.0	May 23, 2016
Neural Architecture Search with Reinforcement Learning[9]	3.65	November 4, 2016

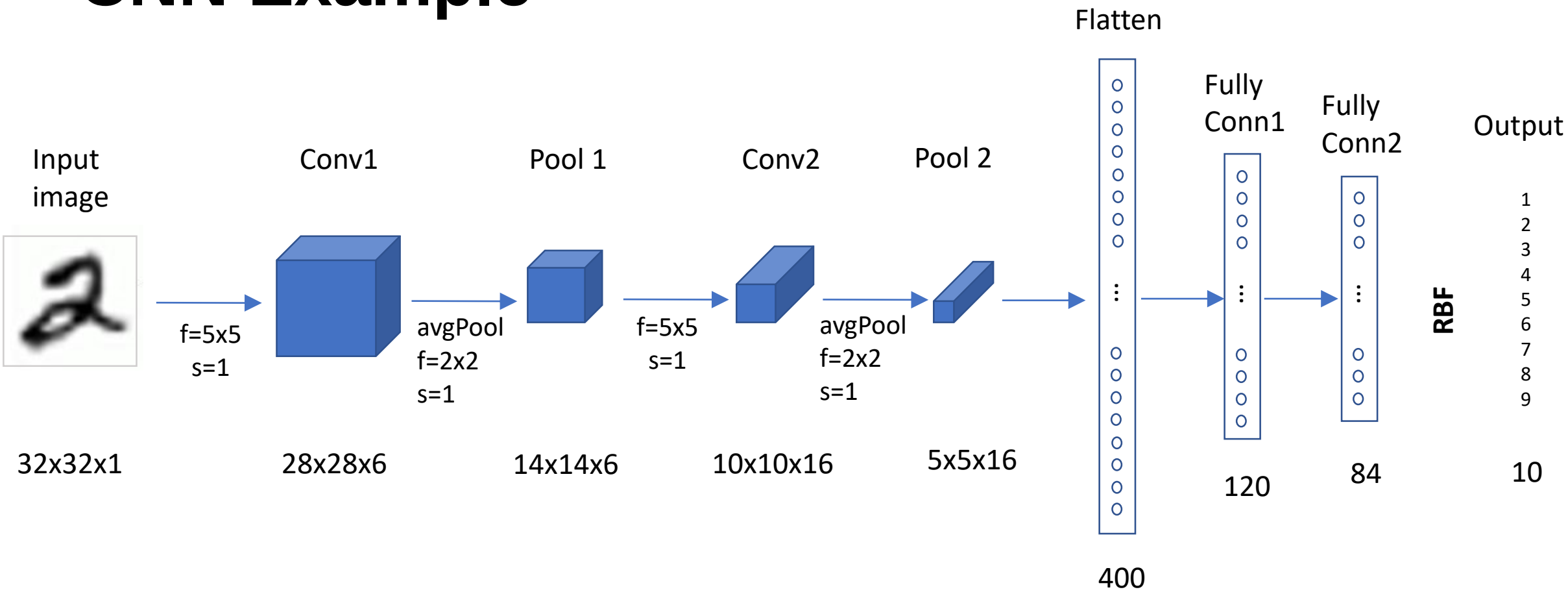
CIFAR-10

- Dataset is available by Canadian Institute for Advanced Research (CIFAR) at http://www.cs.toronto.edu/~kriz/cifar.html
- The CIFAR-10 dataset consists of 60,000 photos divided into 10 classes
- Classes include common objects such as airplanes, automobiles, birds, cats and so on
  - Dataset is split in a standard way, where 50,000 images are used for training a model and the remaining 10,000 for evaluating its performance
  - Photos are in color with red, green and blue channels, but are small measuring 32 x 32 pixel squares

Example Implementation

- Let’s develop a CNN for object recognition on CIFAR-10 dataset
- Evaluate the performance

CNN Example


Input image
Conv1
Pool 1
Conv2
Pool 2
Flatten
Fully Conn1
Fully Conn2
Output
f=5x5 s=1
avgPool f=2x2 s=1
f=5x5 s=1
avgPool f=2x2 s=1
RBF
1 2 3 4 5 6 7 8 9
32x32x1
28x28x6
14x14x6
10x10x16
5x5x16
400
120
84
10

CNN Example

First Layer

Grayscale image

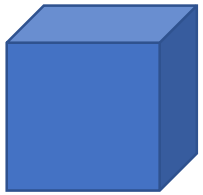
32x32x1

C1: Convolutional layer

f=5
p=0
s=1
channels=6

Trainable parameters:
(5*5+1)*6 = 156

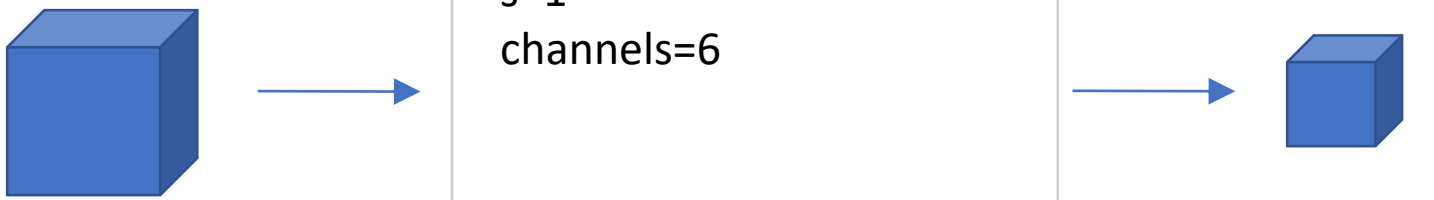
Connections:
(5*5+1)*6*28*28
= 122304



28x28x6

CNN Example

Second Layer



CNN Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X			X	X	X
5				X	X	X			X	X	X	X			X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

C3: Convolutional layer

Third Layer

14x14x6

f=5
p=0
s=1
channels=16

Trainable parameters:
6*(3*5*5+1)+6*(4*5*5+1)+
3*(4*5*5+1)+1*(6*5*5+1) =
1516

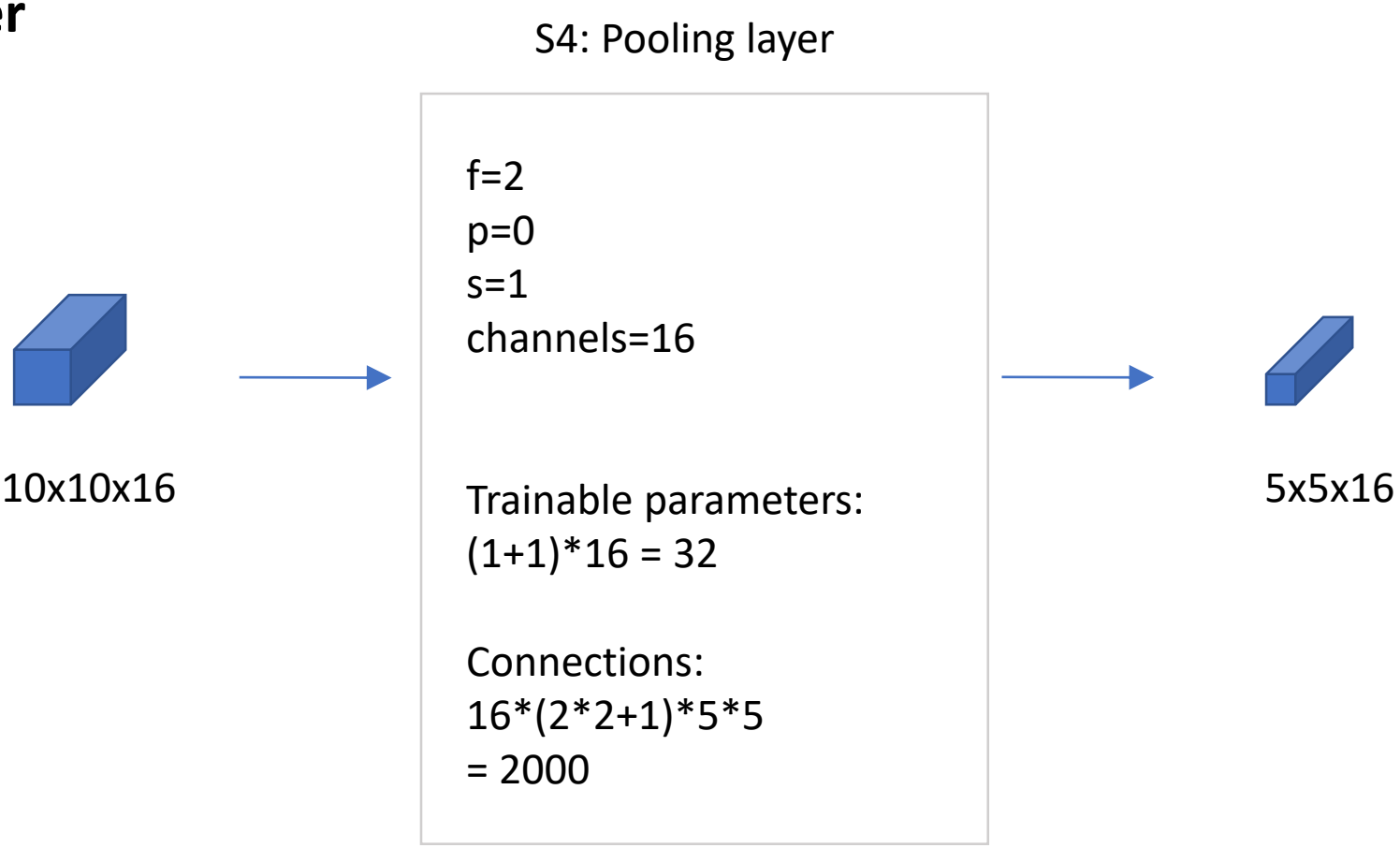
Connections:
(1516)*10*10
= 151600

10x10x16

The first 6 feature maps of C3 take 3 adjacent feature map subsets in S2 as input. The next 6 feature maps take 4 subsets of neighboring feature maps in S2 as input. The next 3 take the non-adjacent 4 feature map subsets as input. The last one takes all the feature maps in S2 as input.

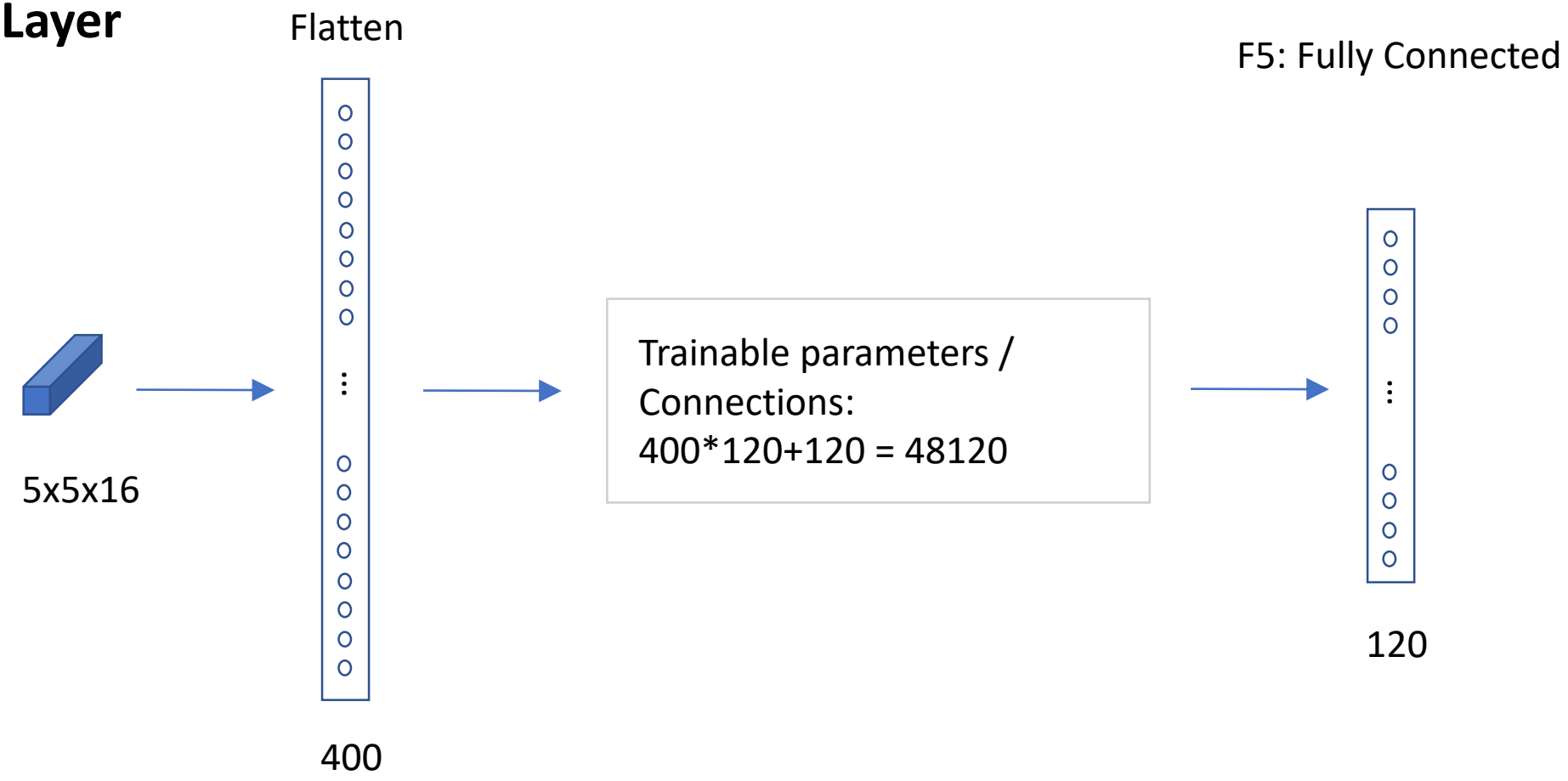
CNN Example

Fourth Layer



CNN Example

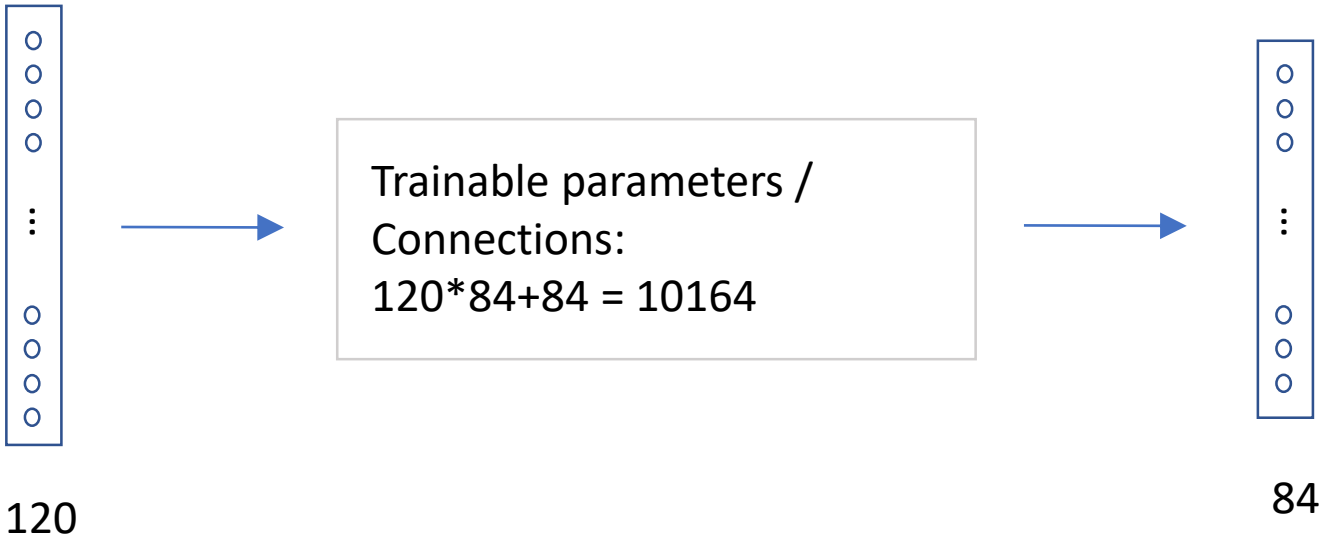
Fifth Layer



CNN Example

Sixth Layer

F6: Fully Connected



CNN Example

Output Layer

LeNet (1998): Background

- Developed by Yann LeCun
  - Worked as a postdoc at Geoffrey Hinton's lab
  - Chief AI scientist at Facebook AI Research
  - Wrote a whitepaper discovering backprop
  - Co-founded ICLR
- Problem: classify 7x12 bit images of 80 classes of handwritten characters.

Fig. 3. Initial parameters of the output RBFs for recognizing the full ASCII set.

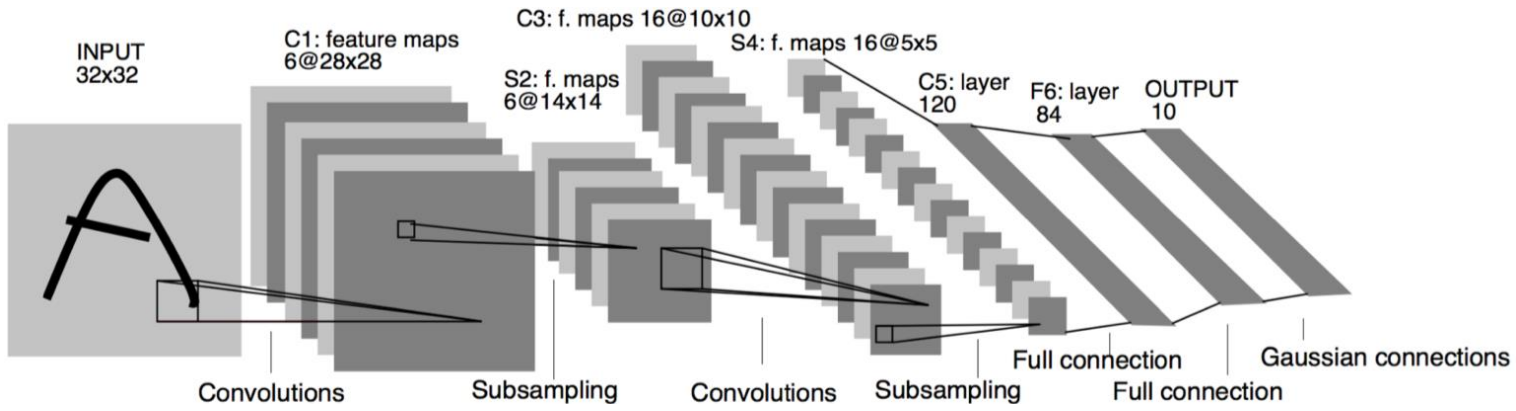
LeNet (1998): Architecture

- Convolution filter size: 5x5
- Subsampling (pooling) kernel size: 2x2
- Semi-sparse connections
- tanh activations
- 60K parameters

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.



LeNet (1998): Results

- Successfully trained a 60K parameter neural network without GPU acceleration!
- Solved handwriting for banks and pioneered automated check-reading
- 0.8% error on MNIST; near state-of-the-art at the time
  - Virtual SVM, kernelized by degree 9 polynomials, also achieves 0.8% error
- LeNet: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- SVM: http://yann.lecun.com/exdb/publis/index.html#lecun-98

LeNet In PyTorch

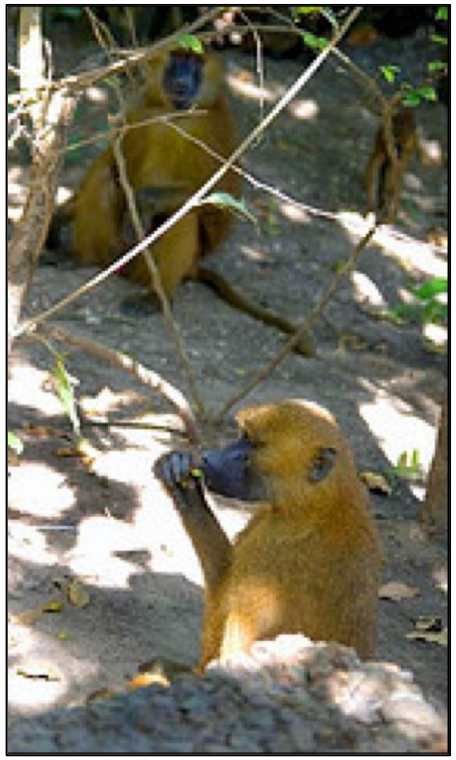
```python
'''LeNet in PyTorch.'''
import torch.nn as nn
import torch.nn.functional as F

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1   = nn.Linear(16*5*5, 120)
        self.fc2   = nn.Linear(120, 84)
        self.fc3   = nn.Linear(84, 10)

    def forward(self, x):
        out = F.relu(self.conv1(x))
        out = F.max_pool2d(out, 2)
        out = F.relu(self.conv2(out))
        out = F.max_pool2d(out, 2)
        out = out.view(out.size(0), -1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```

LeNet In TensorFlow

https://colab.research.google.com/drive/1kV3Jpxzup63GfJB1FGKxTSKd6Ek8J3sA

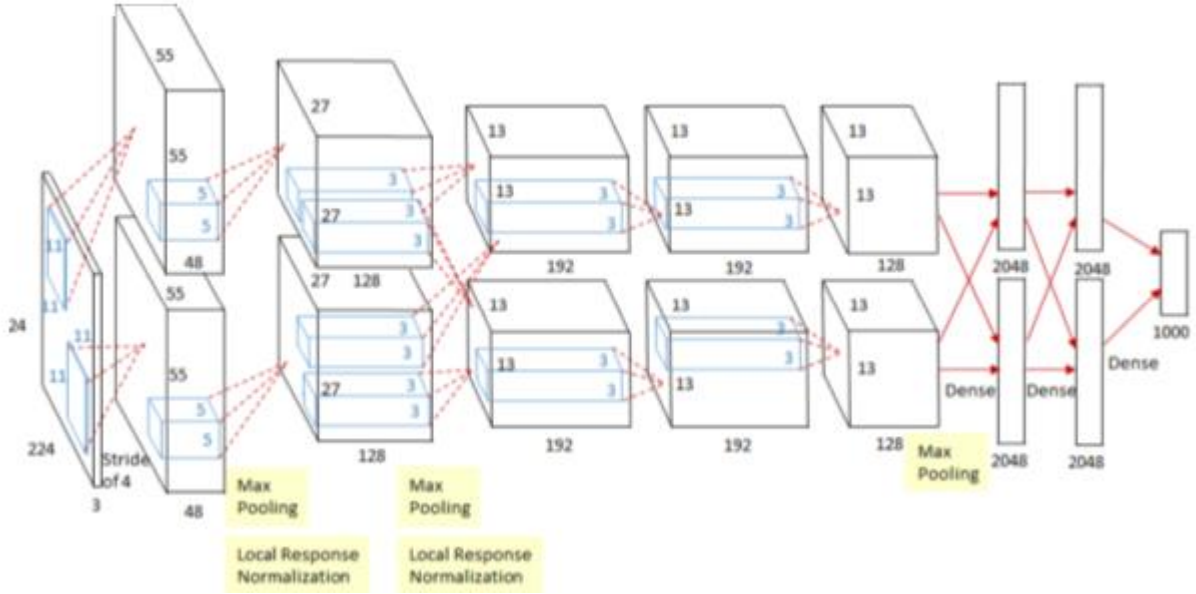
AlexNet (2012): Background

- Developed by
  - Alex Krizhevsky
  - Ilya Sutskever
    - Chief scientist, OpenAI
    - Inventor of seq2seq learning
  - Geoffrey Hinton, Alex Krizhevsky's PhD adviser
- Compete on ImageNet, Fei-Fei Li's dataset of 14 million images with more than 20,000 categories (e.g. strawberry, balloon)



AlexNet (2012): Architecture

- Input of 224x224x3 images
- 8 weighted layers
  - 5 convolutional layers
    - Filter width 11, then 5, then 3
    - Stride 4
  - A lot of max-pooling layers
    - Kernel width 3
    - Stride 2
  - 3 fully connected layers
  - ReLU activation until last layer (which was softmax)
  - Trained with dropout and local response normalization
- 60M parameters



AlexNet (2012): Results

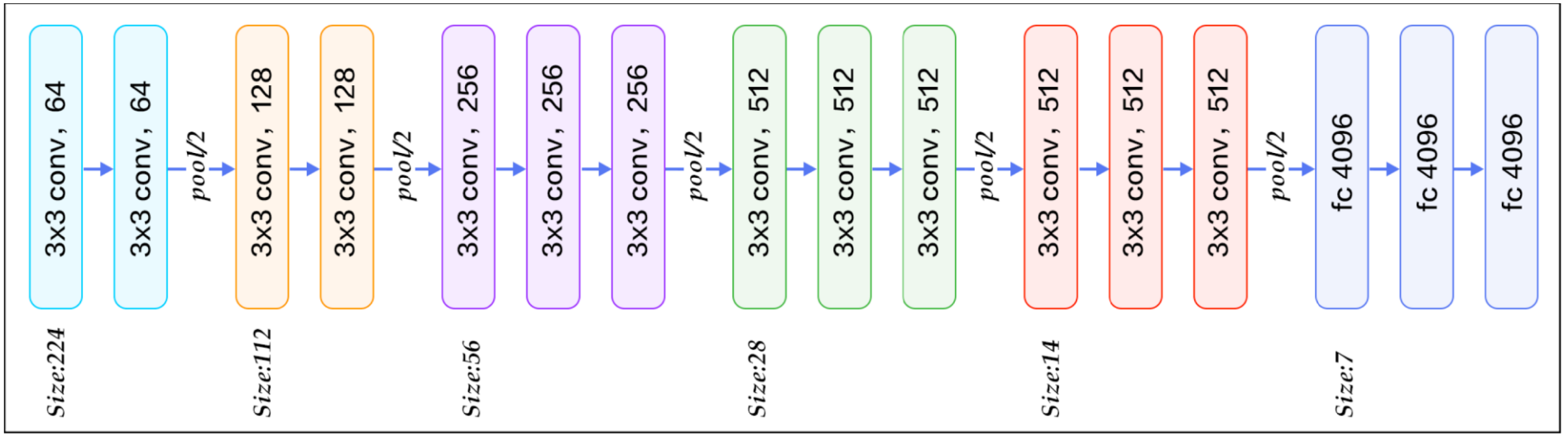
- Flew past the competition with 15.3% top-5 error (runner-up had 26.2%)
- One of the first neural nets trained on a GPU with CUDA
  - (There had been 4 previous contest-winning CNNs)
  - Trained 60 million parameters
- Cited over 130,000 times:
  - https://papers.nips.cc/paper/4824-imagenetclassification-with-deep-convolutional-neural-networks.pdf

VGG (2014): Background

- Developed by the Visual Geometry Group (Oxford)
- Beat AlexNet on ImageNet

VGG (2014): Architecture

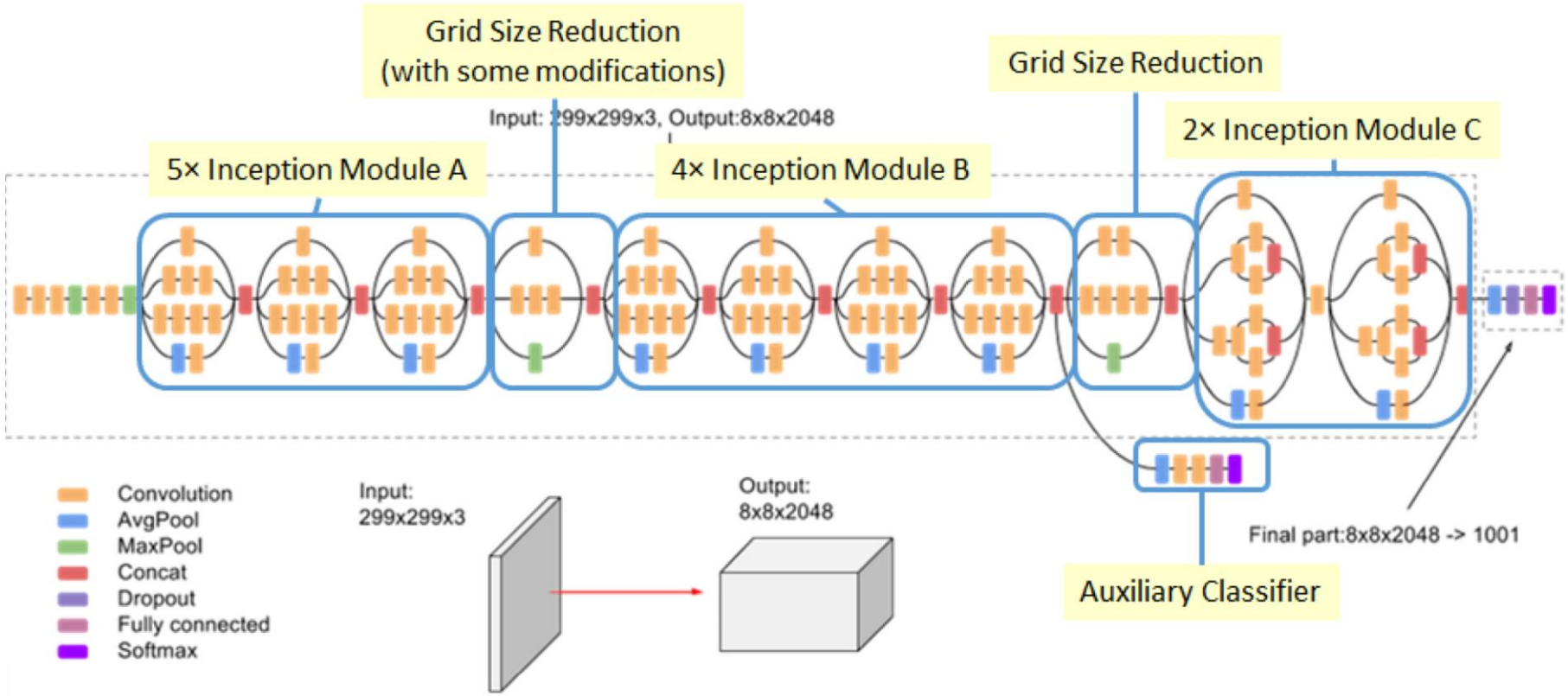
- Conv: 3x3, pooling: 2x2, ReLU w/ final softmax
- Quite simple
- At the time, the deepest architecture
- 138M parameters



VGG (2014): Results

- Configuration E (19 weighted layers) achieved 8.0% top-5 error
- https://arxiv.org/pdf/1409.1556.pdf

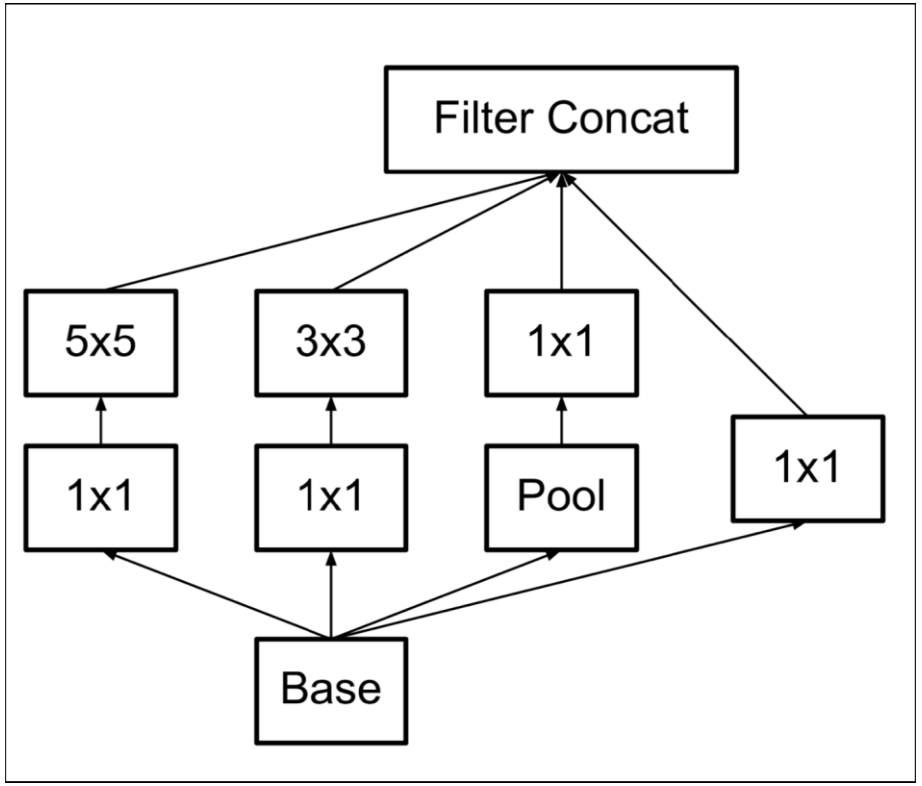
Inception(v3) (2015): Architecture



After each conv, also uses ReLU and BatchNorm

Inception v1: Tuning Kernel Size is Hard

- A computer scientist's solution: toss 'em all together



A single
"Inception Module"

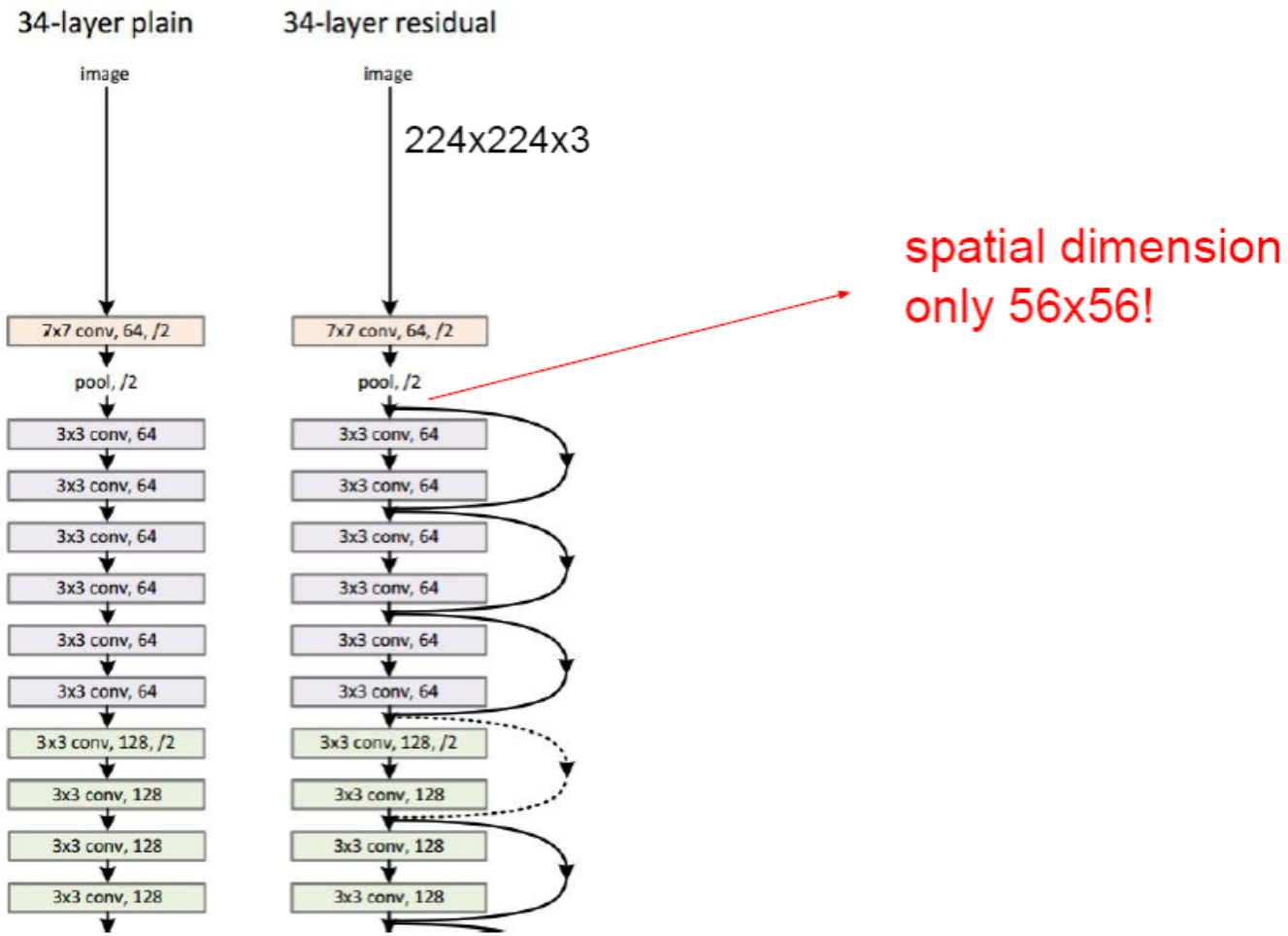
Why the Smaller Kernels?

- At first, people thought large kernels were good.
- After all, small ones are a special case.
- And large kernels include longer distance info.
- But, several small ones in series can have a similar impact.
- It’s cheaper - both in parameters and in computation time.
- Now, people mostly use 3x3 (or 1x1).

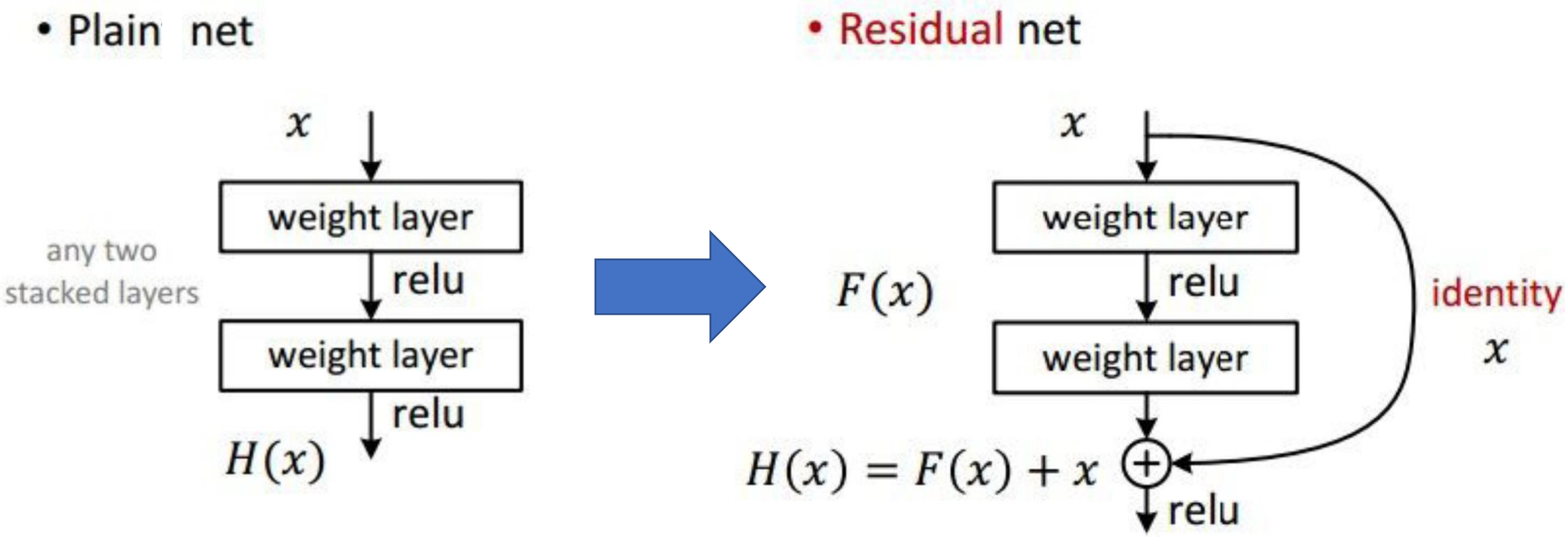
Inception (2014): Results

- 6.67% top-5 error rate!
- Andrej Karpathy achieved 5.1% top-5 error rate

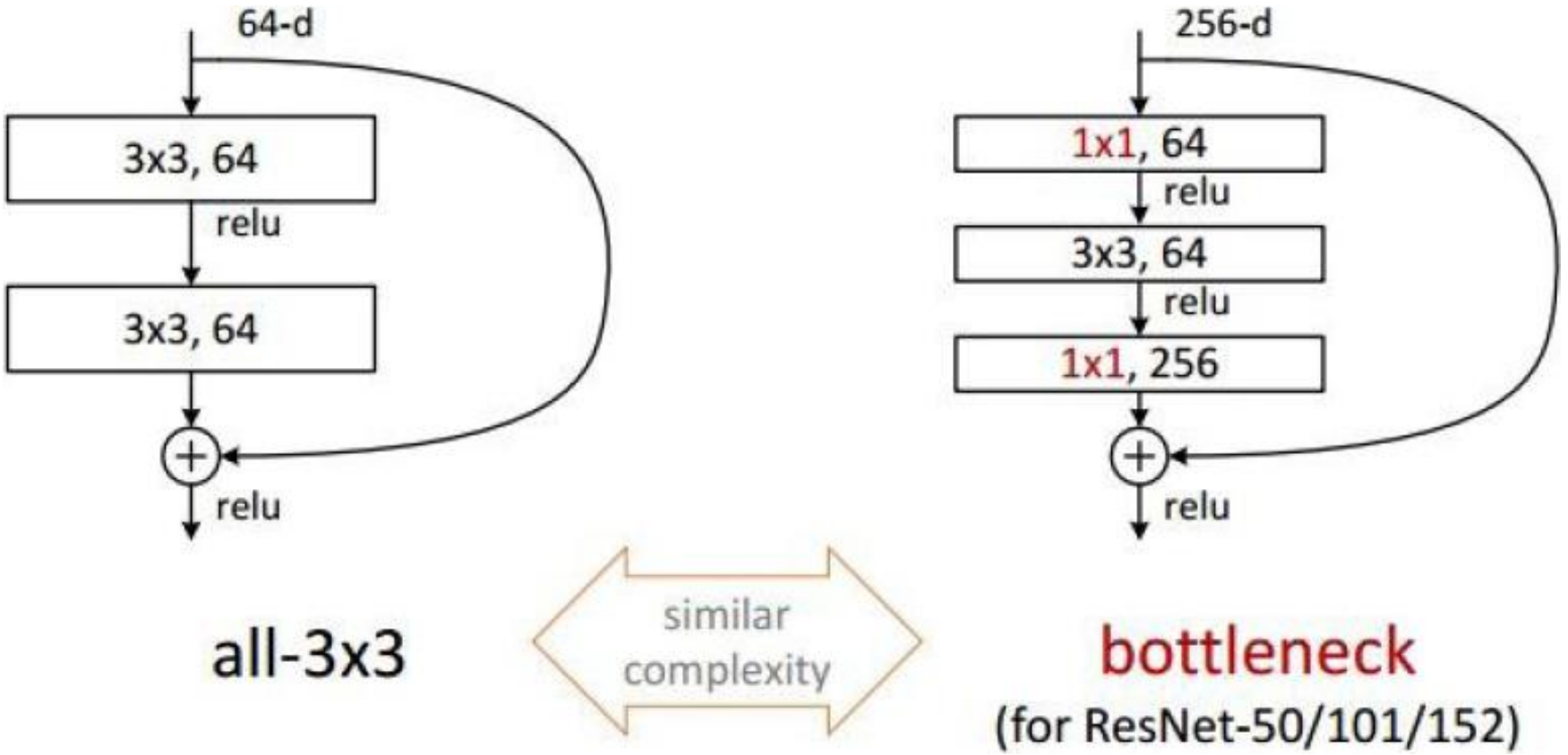
ResNet (2015): Architecture



ResNet (2015): Architecture (1/2)



ResNet (2015): Architecture (2/2)



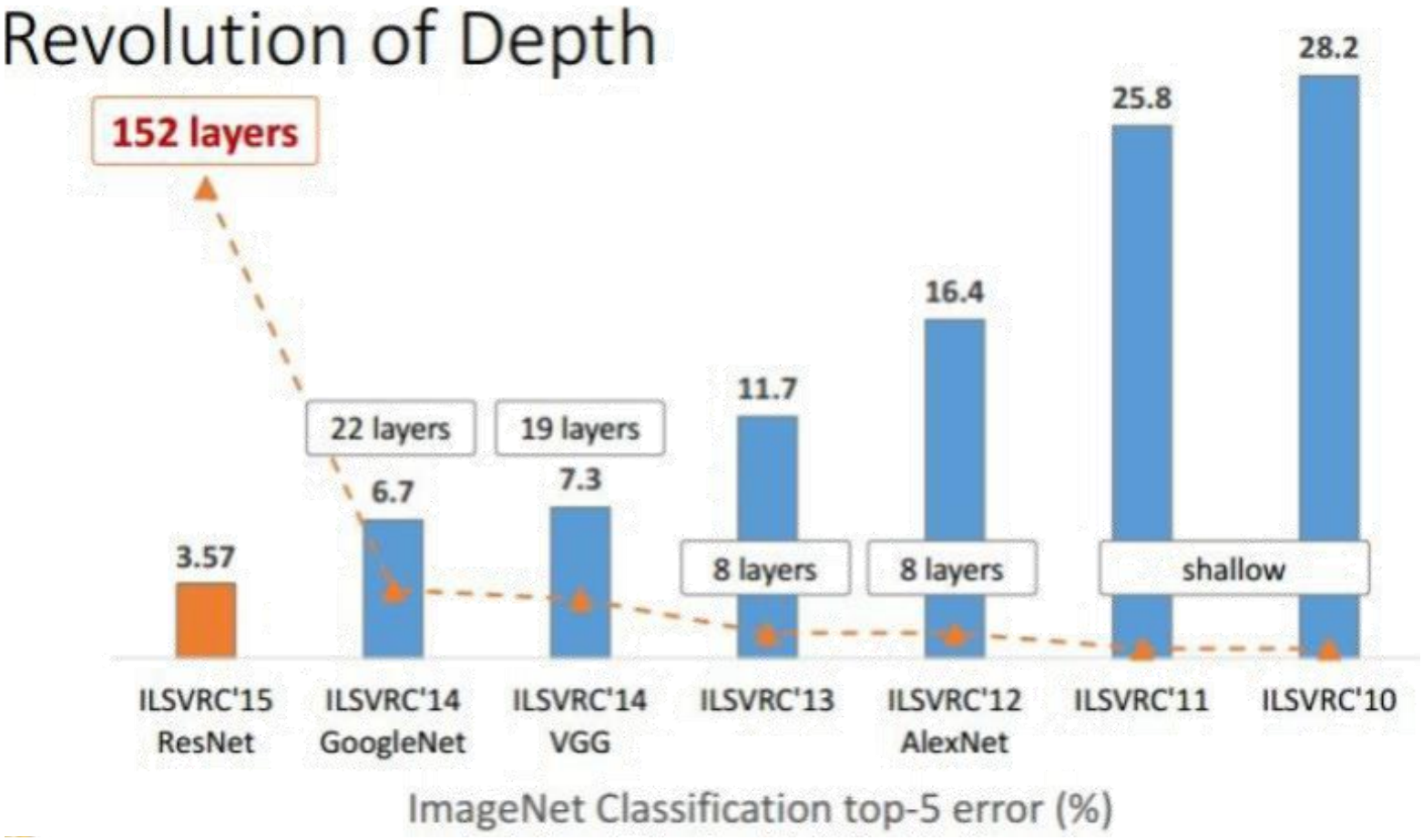
ResNet (2015): Results (1/2)

- ILSVRC 2015 winner (3.6% top 5 error)

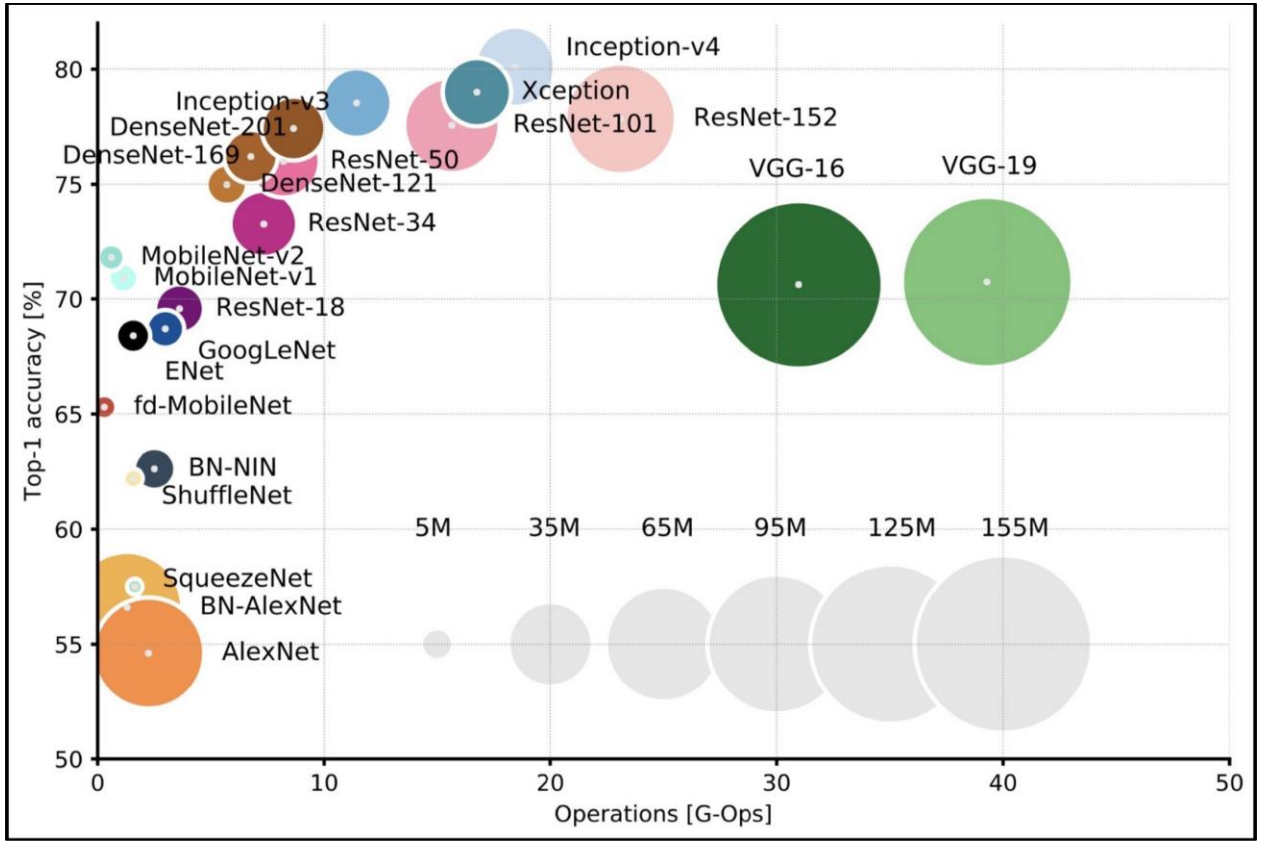
MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

ResNet (2015): Results (2/2)



Performance of Convolutional Architectures on ImageNet



Applications of CNNs

Image Recognition

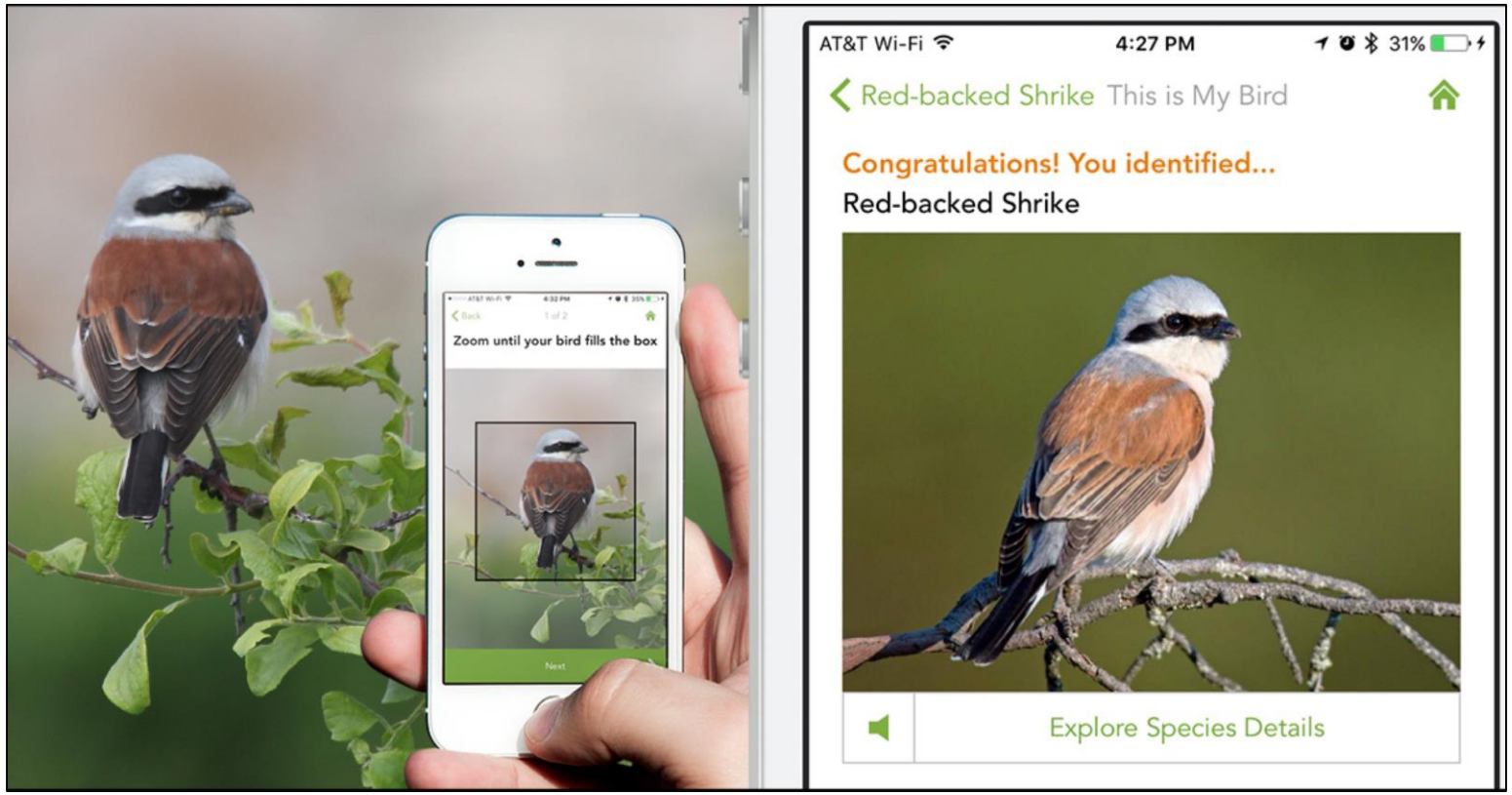


Image source: Cornell Lab of Ornithology, Merlin bird identification app (see Van Horn *et al.* 2015)

Scene Labeling Via Deep Learning

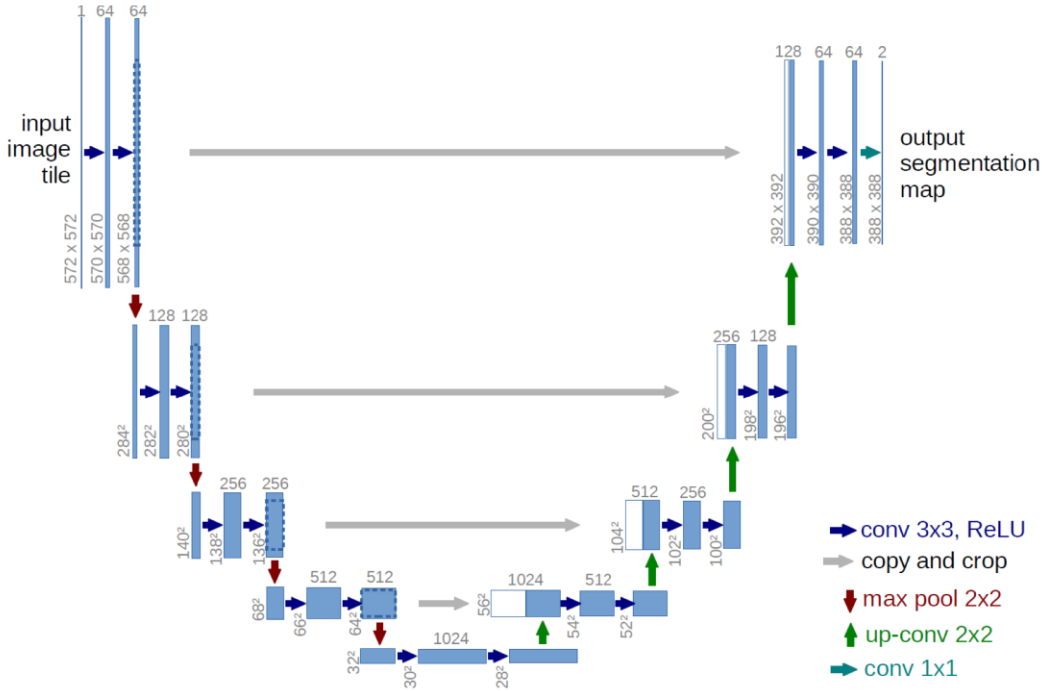
Image source: Farabet *et al.* ICML 2012, PAMI 2013

Style Transfer

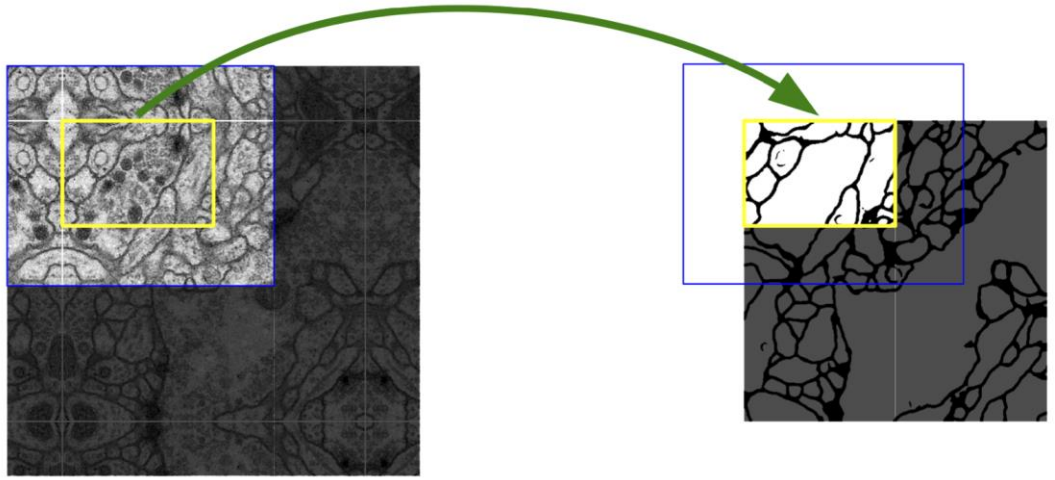
- Content is measured by deeper layers
- Style is measured by the correlations between feature vectors at lower layers
- Objective 1: Minimize content difference between new image and content template
- Objective 2: Minimize style difference between new image and style template

Image Segmentation

U-Net (Ronneberger *et al.* 2015)



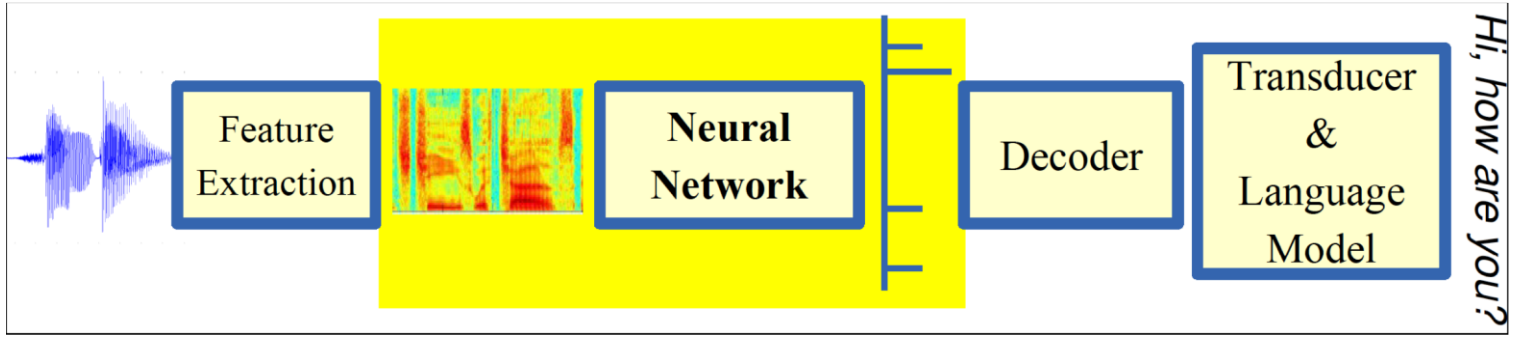
Segmenting biological cell membranes



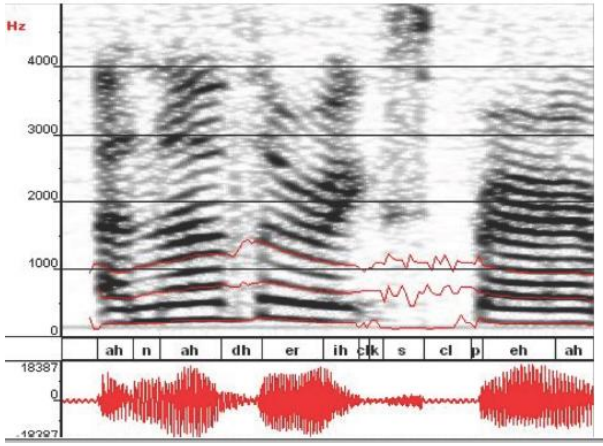
Insights of U-Net

- “Up-convolution”
  - Fixes the problem of shrinking images with CNNs
  - One example of how to make “fully convolutional” nets: pixels to pixels
  - “Up-convolution” is just upsampling, then convolution
  - Allows for refinement of the upsample by learned weights
  - Goes along with decreasing the number of feature channels
  - Not the same as “de-convolution”
- Connections across the “U” in the architecture

Speech Recognition



ML used to predict of phone states from the sound spectrogram



Deep learning has state-of-the-art results

# Hidden Layers	1	2	4	8	10	12
Word Error Rate %	16.0	12.8	11.4	10.9	11.0	11.1

Baseline GMM performance = 15.4%

[Zeiler et al. "On rectified linear units for speech recognition" ICASSP 2013]

Speech Recognition

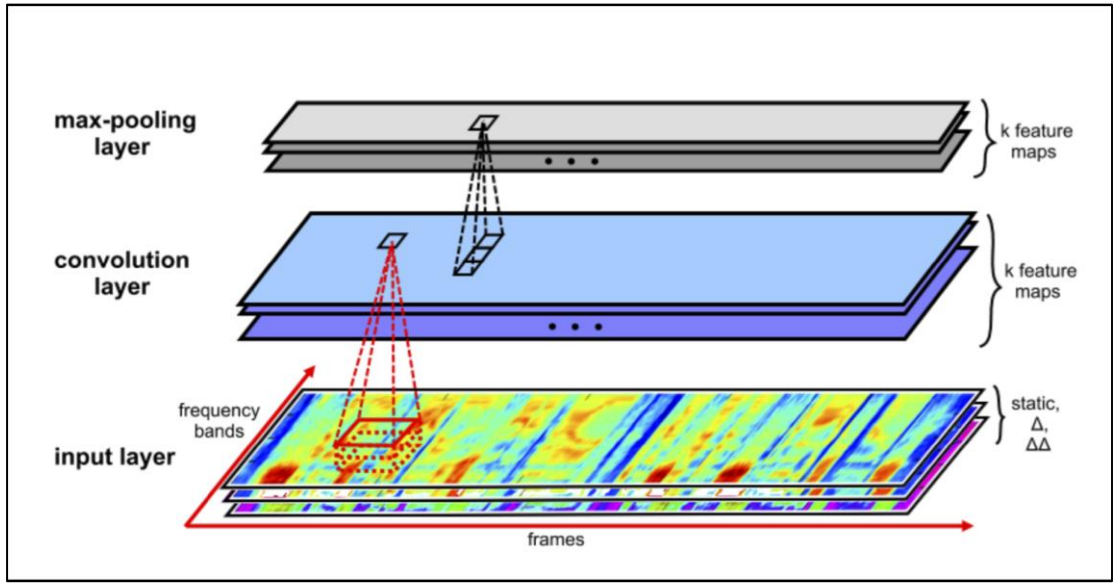


Image source: Zhang *et al.* 2017

- CNN competitive with RNNs (e.g. LSTMs)
- 10 layers, 3x5 conv, 3x1 pooling - deep enough for temporal dependencies
- TIMIT task, classifying phonemes