

Convolutional Neural Networks



CIFAR: Canadian Institute for Advanced Research. CIFAR encourages basic research without direct application

- motivated Hinton to move to Canada in 1987 and funded his work
- the funding was ended in the mid 90s just as sentiment towards neural nets was becoming negative again
- rather than relenting and switching his focus, Hinton fought to continue work on neural nets, and managed to secure more funding from CIFAR

But in 2004, **Hinton** asked to lead a new program on neural computation. The mainstream machine learning community could not have been less interested in neural nets.



“It was **the worst possible time**,” says **Bengio**, a professor at the Université de Montréal and co-director of the CIFAR program since it was renewed last year. “Everyone else was doing something different. Somehow, Geoff convinced them. **We should give (CIFAR) a lot of credit for making that gamble.**”



CIFAR “had a huge impact in forming a community around deep learning,” adds **LeCun**, the CIFAR program’s other co-director.



2019 A.M. Turing Award: Bengio, Hinton, LeCun (&

CIFAR)
CIFAR

CIFAR convenes extraordinary minds to address science and humanity's most important questions.



ARTIFICIAL INTELLIGENCE

Turing Award honours CIFAR's 'pioneers of AI'

CIFAR Fellows Yoshua Bengio, Geoffrey Hinton and Yann LeCun were jointly awarded the prestigious A.M. Turing Award



The CIFAR-10 dataset

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:

airplane



automobile



bird



cat



deer



dog



frog



horse



ship

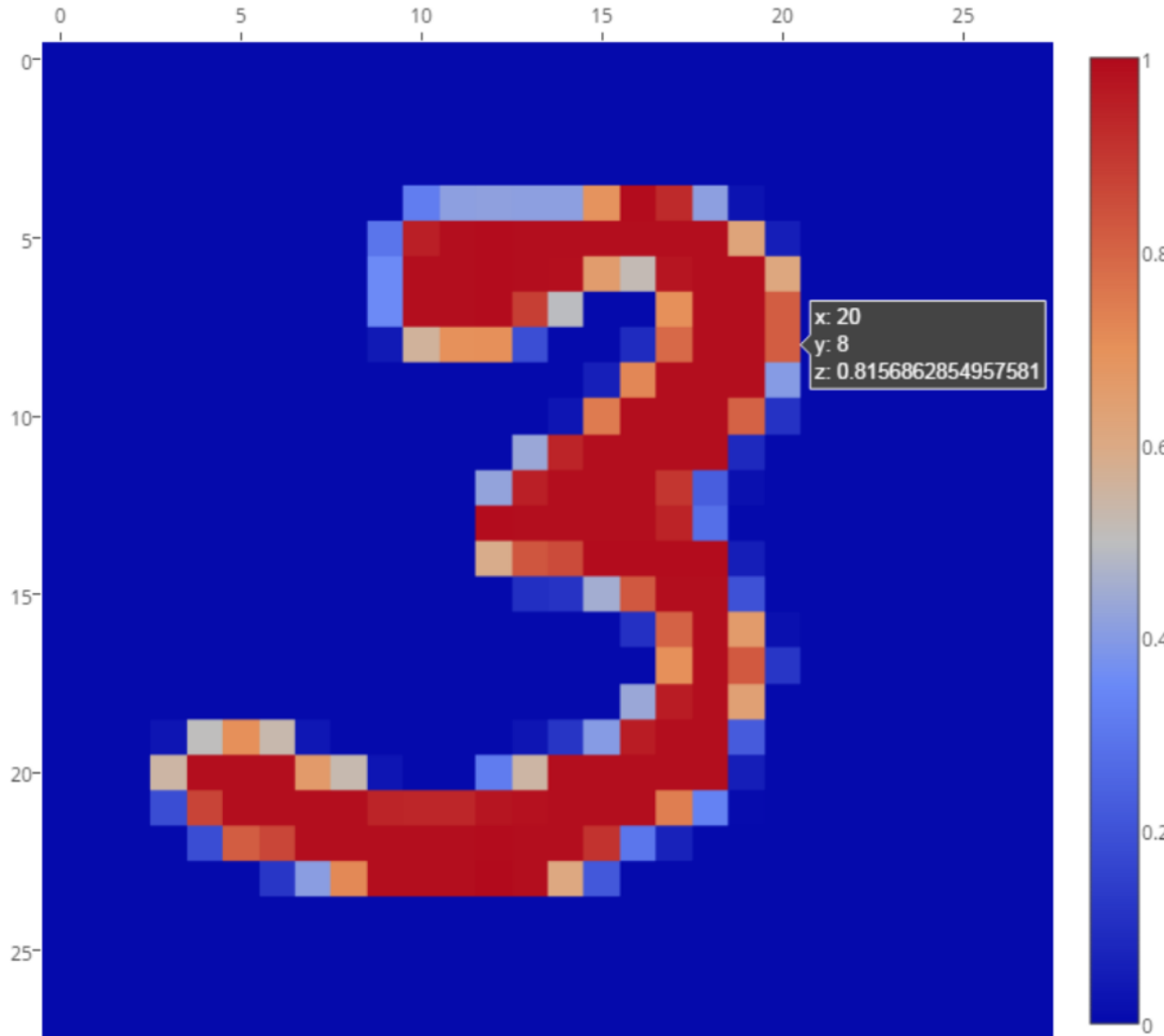


truck



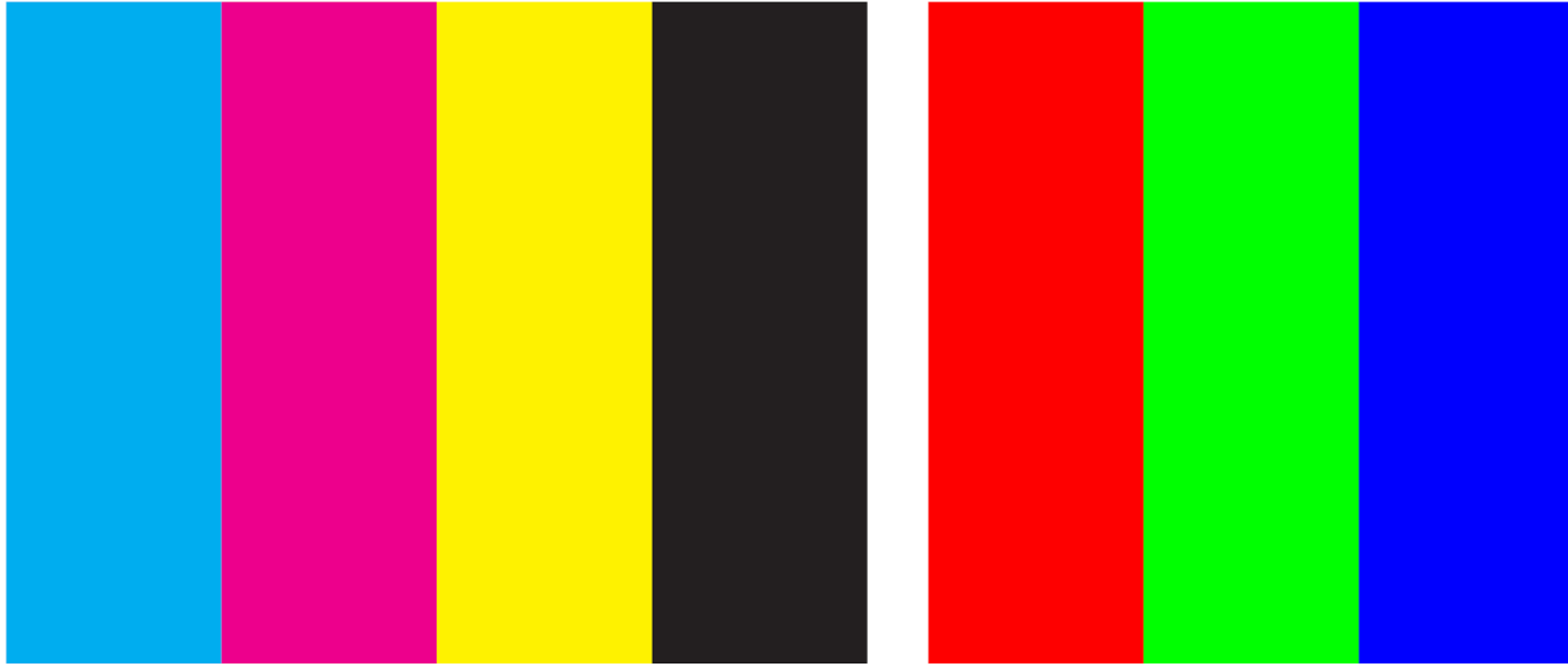
The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million [tiny images](#) dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Image representation



- An image (28 x 28 pixels, grayscale) is represented by a 28 x 28 matrix.
- The original dataset represents a brightness in an 8-bit integer ([0, 255]).
- In this lecture, a brightness is normalized within the range of [0, 1].

CMYK VS RGB



WIDTH x HEIGHT x CHANNELS -> *SHAPES & COLOUR* -> *semantic meaning for humans*

CIFAR10: each picture has 32 x 32 x 3 (in RGB) = 3072 features

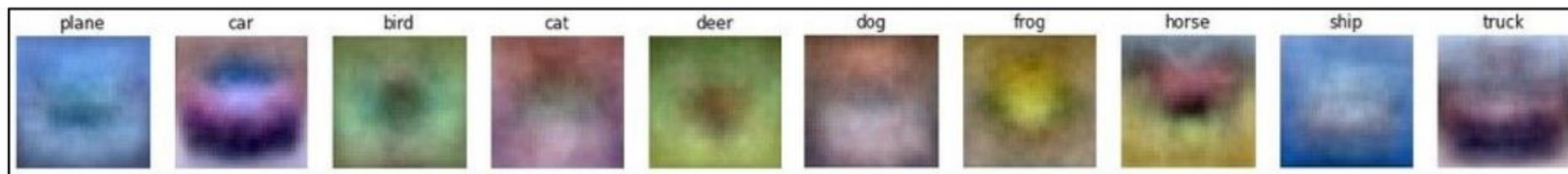
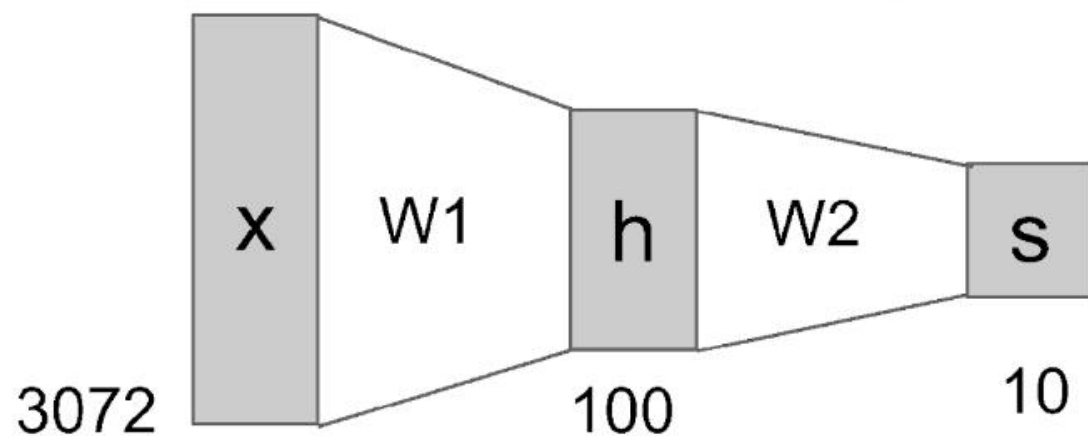
Last time: Neural Networks

Linear score function:

$$f = Wx$$

2-layer Neural Network

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



Traditional Computer Vision



- Explicit feature extraction (e.g. SIFT, SURF, ORB)
- From images we create feature vectors

```
>>> img = cv2.imread('fly.png',0)

# Create SURF object. You can specify params here or later.
# Here I set Hessian Threshold to 400
>>> surf = cv2.SURF(400)

# Find keypoints and descriptors directly
>>> kp, des = surf.detectAndCompute(img,None)

>>> len(kp)
699
```

Deep Learning (CNN)

Implicit feature extraction: The output layers of each convolutional layer are the features of the next Convolutional layer

Learning Feature Hierarchy with DL

- Deep architectures can be more efficient in feature representation;

- Natural derivation/abstraction from low level structures to high level structures;

- Share the lower-level representations for multiple tasks (such as detection, recognition, segmentation).

Feature representation



3rd layer
"Objects"



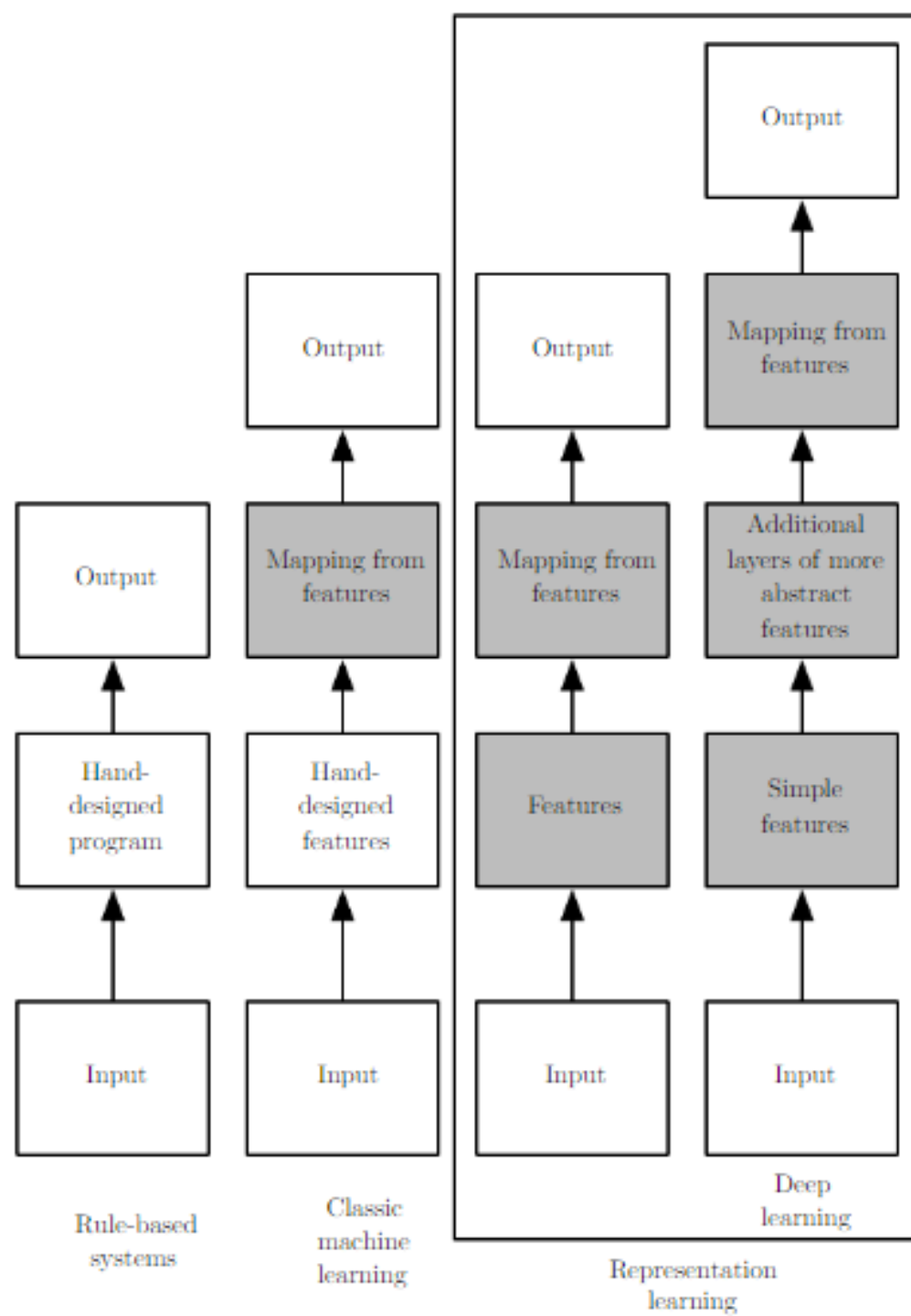
2nd layer
"Object parts"



1st layer
"Edges"



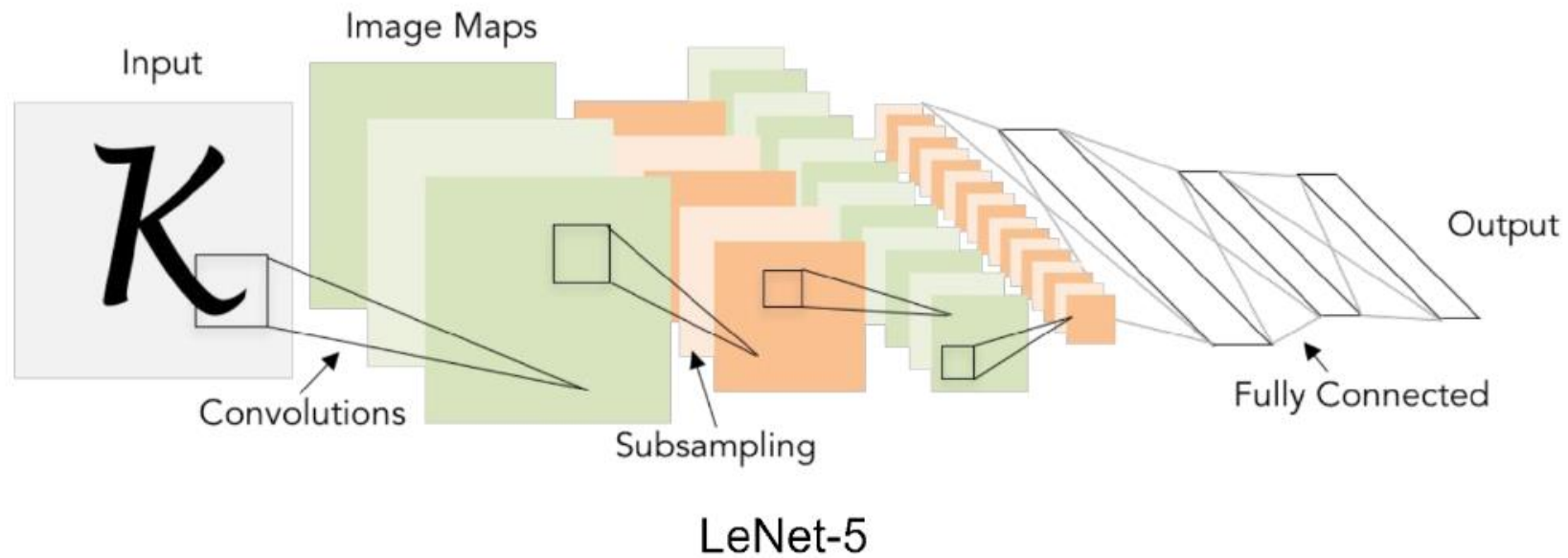
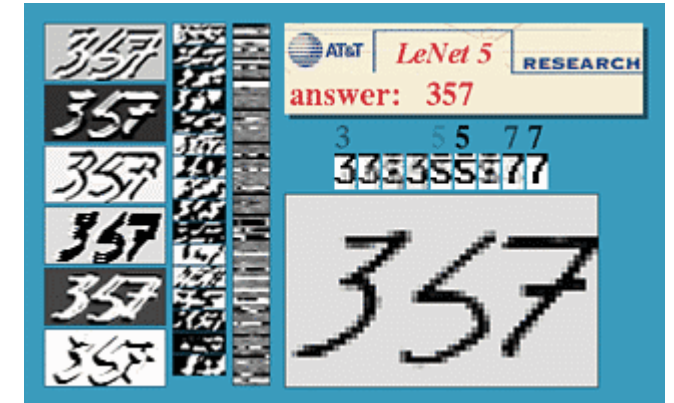
Pixels



A bit of history:

Gradient-based learning applied to document recognition

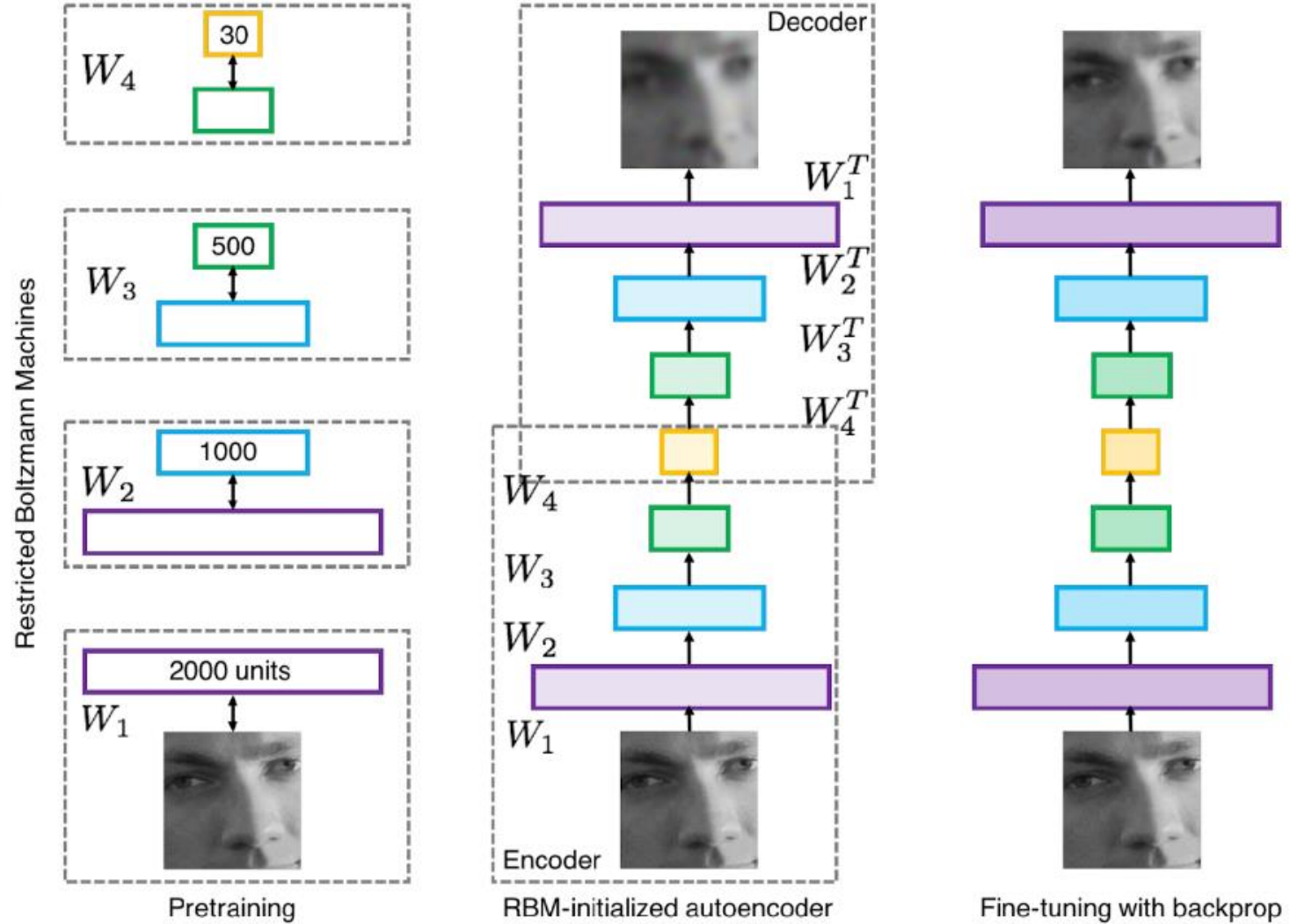
[LeCun, Bottou, Bengio, Haffner 1998]



A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning



First strong results

Acoustic Modeling using Deep Belief Networks

Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010

Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition

George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

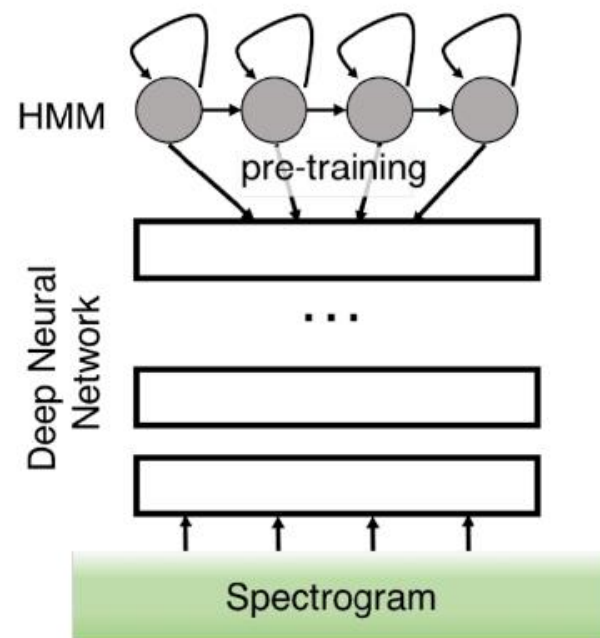
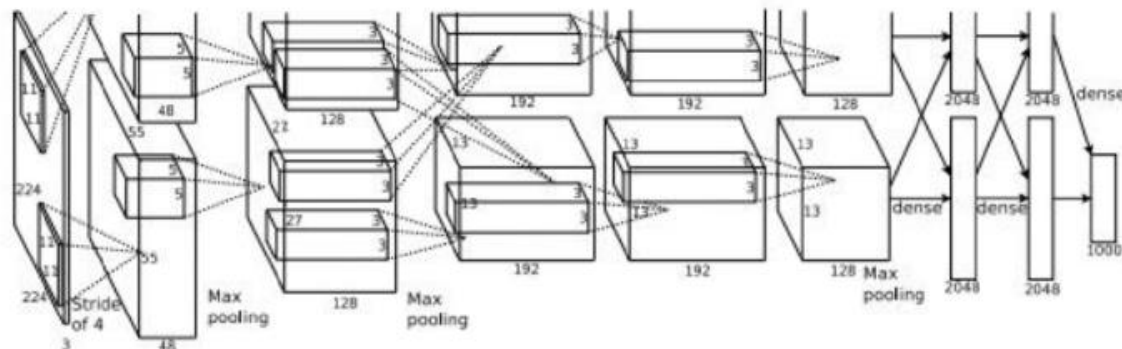


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012



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

A bit of history: ImageNet Classification with Deep Convolutional Neural Networks *[Krizhevsky, Sutskever, Hinton, 2012]*

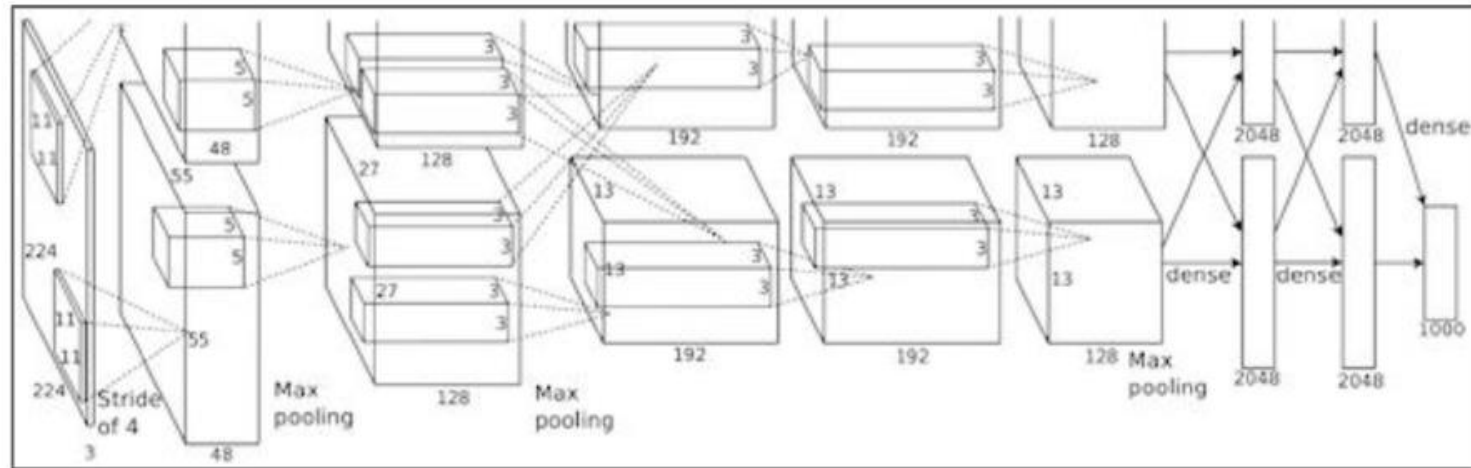


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

Fast-forward to today: ConvNets are everywhere

Classification



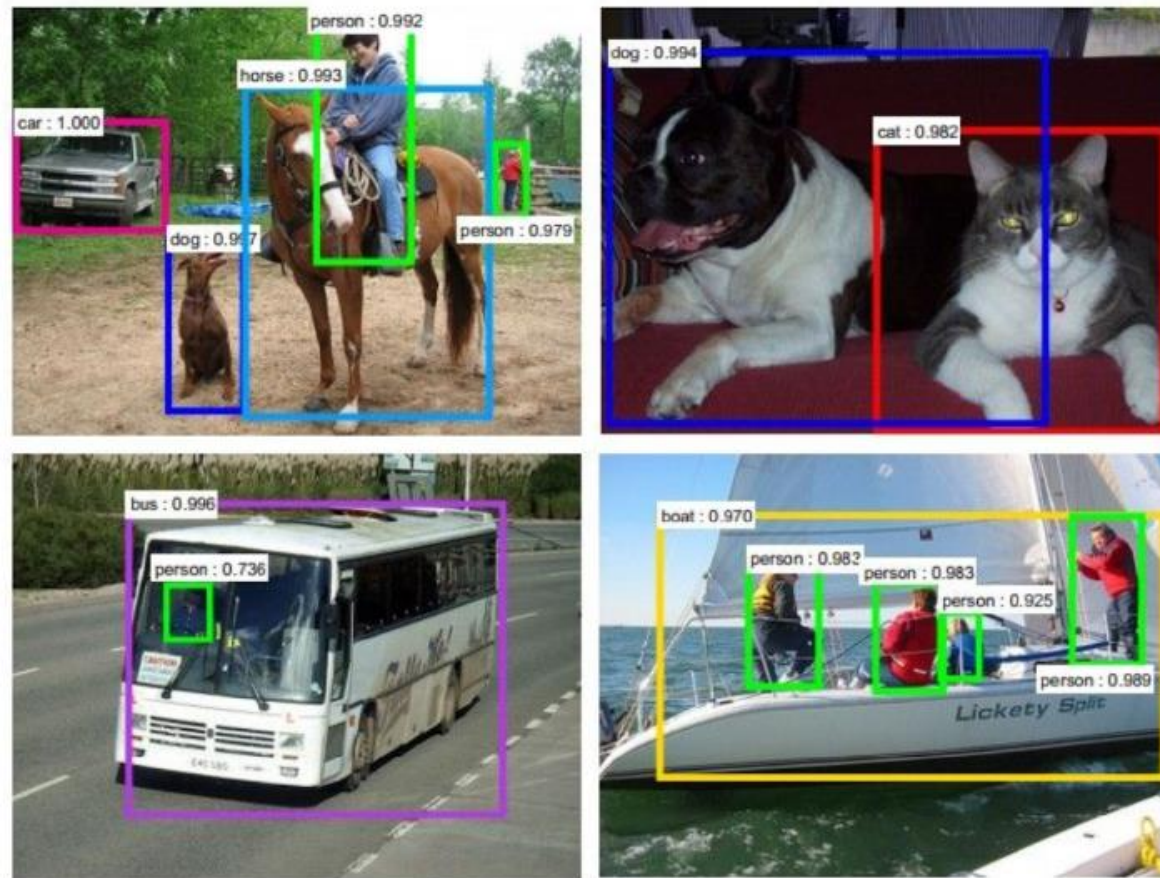
Retrieval



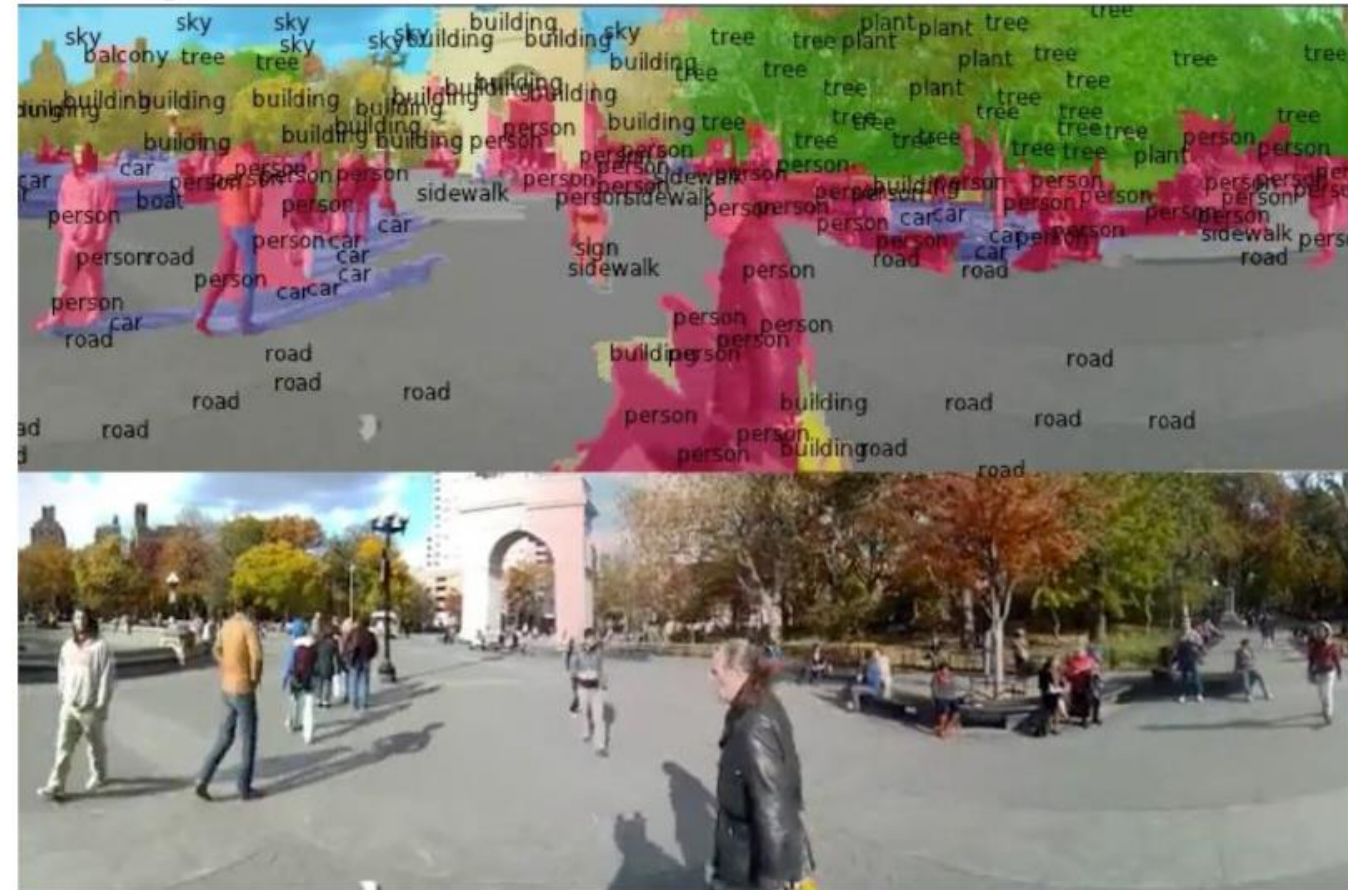
Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

Detection



Segmentation



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

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

Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere



Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars



[This image](#) by GBPublic_PR is licensed under [CC-BY 2.0](#)

NVIDIA Tesla line

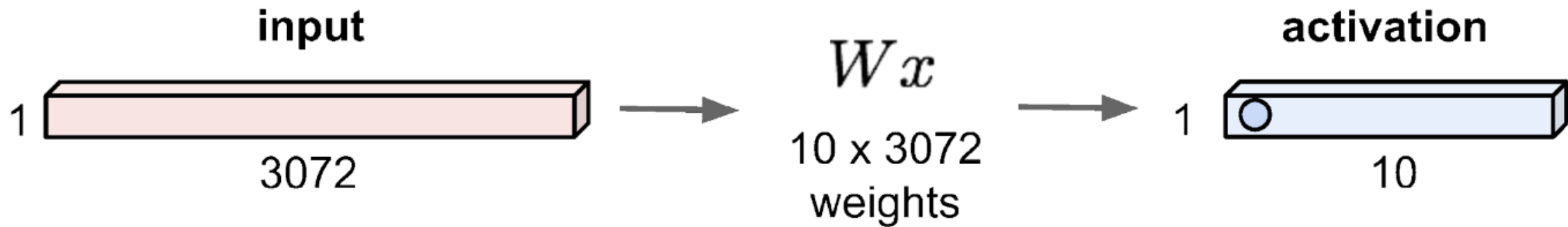
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Convolutional Neural Networks

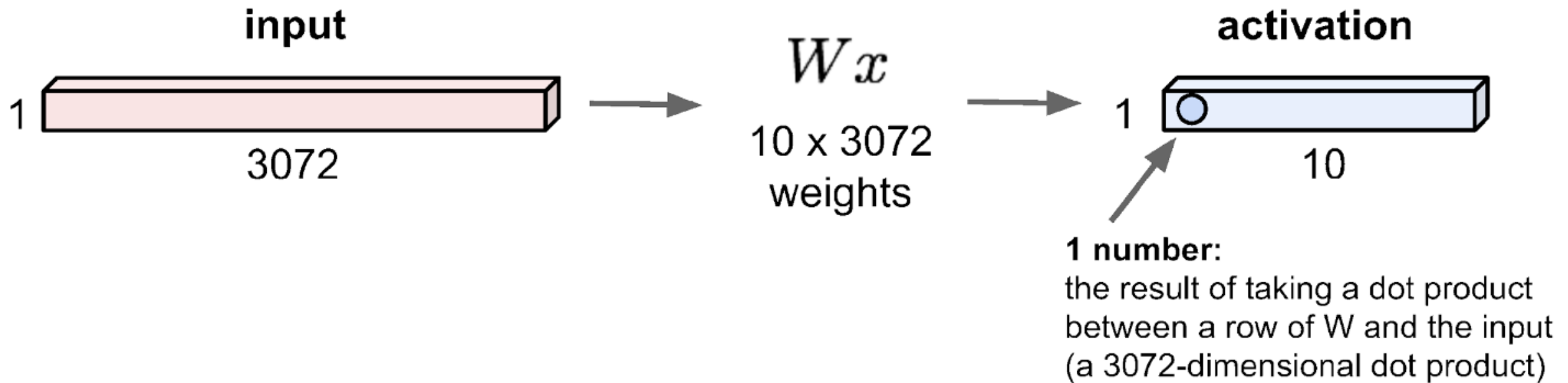
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



The Cross-Correlation Operator

In a convolutional layer, an input array and a correlation kernel array output an array through a cross-correlation operation. Let's see how this works for two dimensions. As shown below, the input is a two-dimensional array with a height of 3 and width of 3. We mark the shape of the array as 3×3 or $(3, 3)$. The height and width of the kernel array are both 2. This array is also called a kernel or filter in convolution computations. The shape of the kernel window (also known as the convolution window) depends on the height and width of the kernel, which is 2×2 .

Input		Kernel		Output																	
<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	*	<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	=	<table border="1"><tr><td>19</td><td>25</td></tr><tr><td>37</td><td>43</td></tr></table>	19	25	37	43
0	1	2																			
3	4	5																			
6	7	8																			
0	1																				
2	3																				
19	25																				
37	43																				

Fig. 6.1 Two-dimensional cross-correlation operation. The shaded portions are the first output element and the input and kernel array elements used in its computation: $0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19$.

In the two-dimensional cross-correlation operation, the convolution window starts from the top-left of the input array, and slides in the input array from left to right and top to bottom. When the convolution window slides to a certain position, the input subarray in the window and kernel array are multiplied and summed by element to get the element at the corresponding location in the output array. The output array has a height of 2 and width of 2, and the four elements are derived from a two-dimensional cross-correlation operation:

$$\begin{aligned}0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 &= 19, \\1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 &= 25, \\3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 &= 37, \\4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 &= 43.\end{aligned}$$

Filters (kernels, convolutions)

Sharpen:

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0



Blur:

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



Edge Detect:

0	1	0		
1	-4	1		
0	1	0		

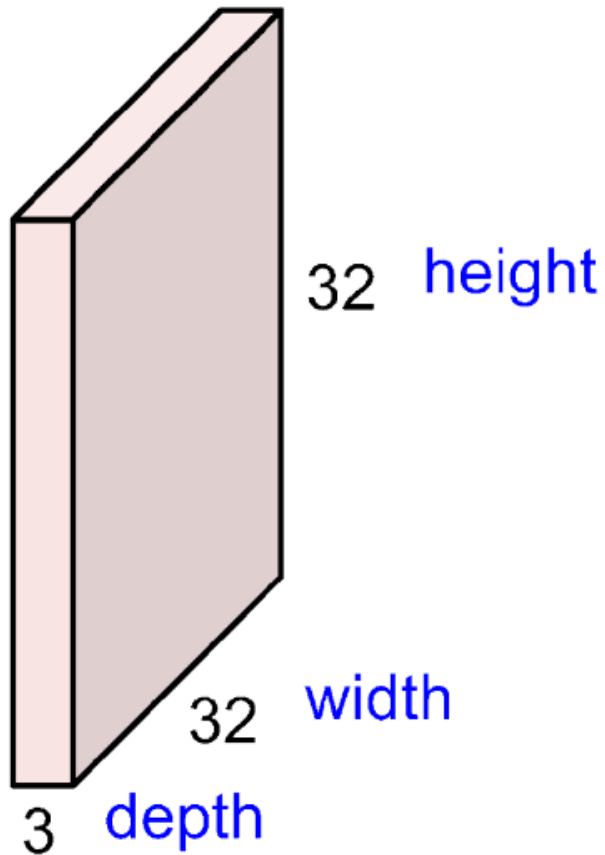


Inverse approach:

what-if we learned the filters weights?

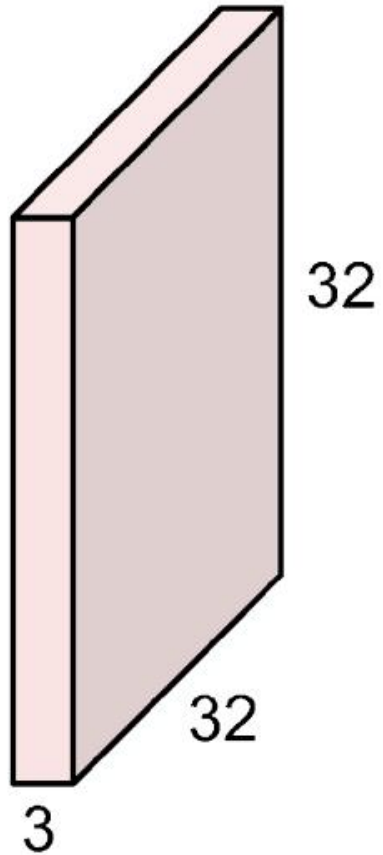
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

32x32x3 image



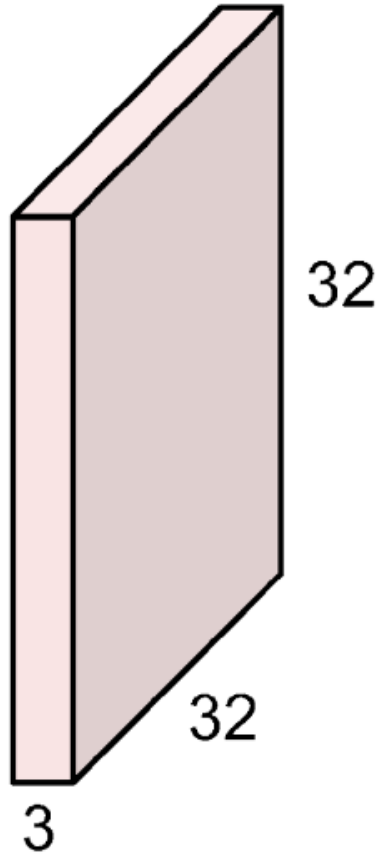
5x5x3 filter



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

Convolution Layer

32x32x3 image



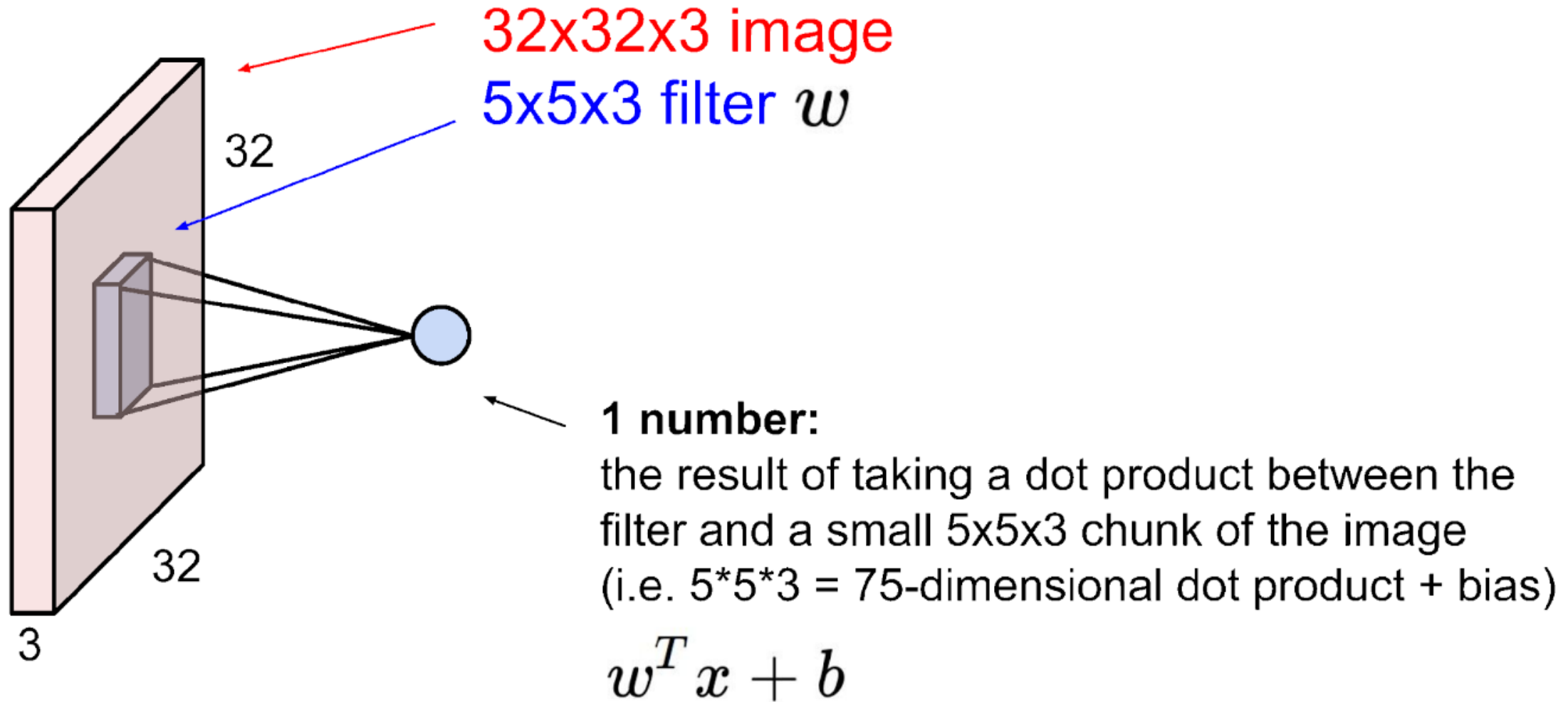
Filters always extend the full depth of the input volume

5x5x3 filter

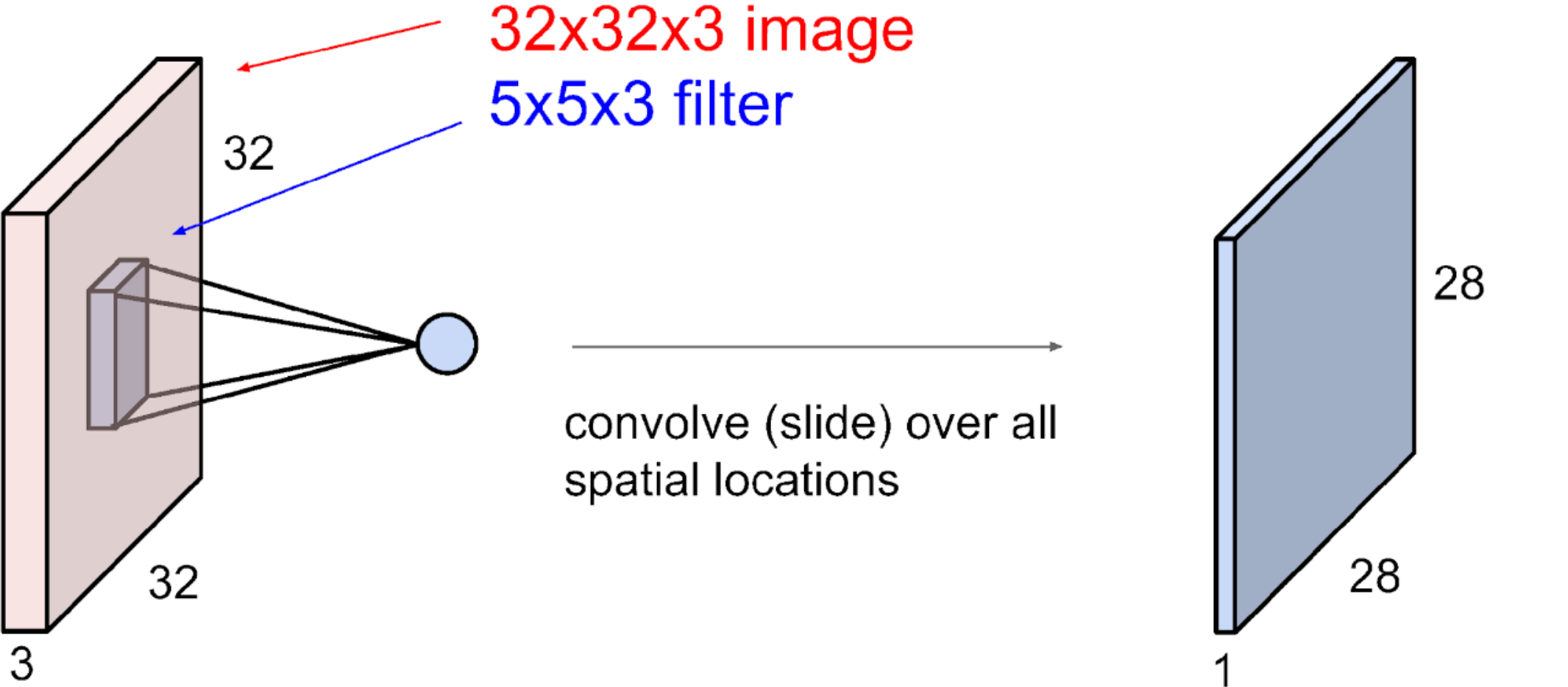


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer



Convolution Layer

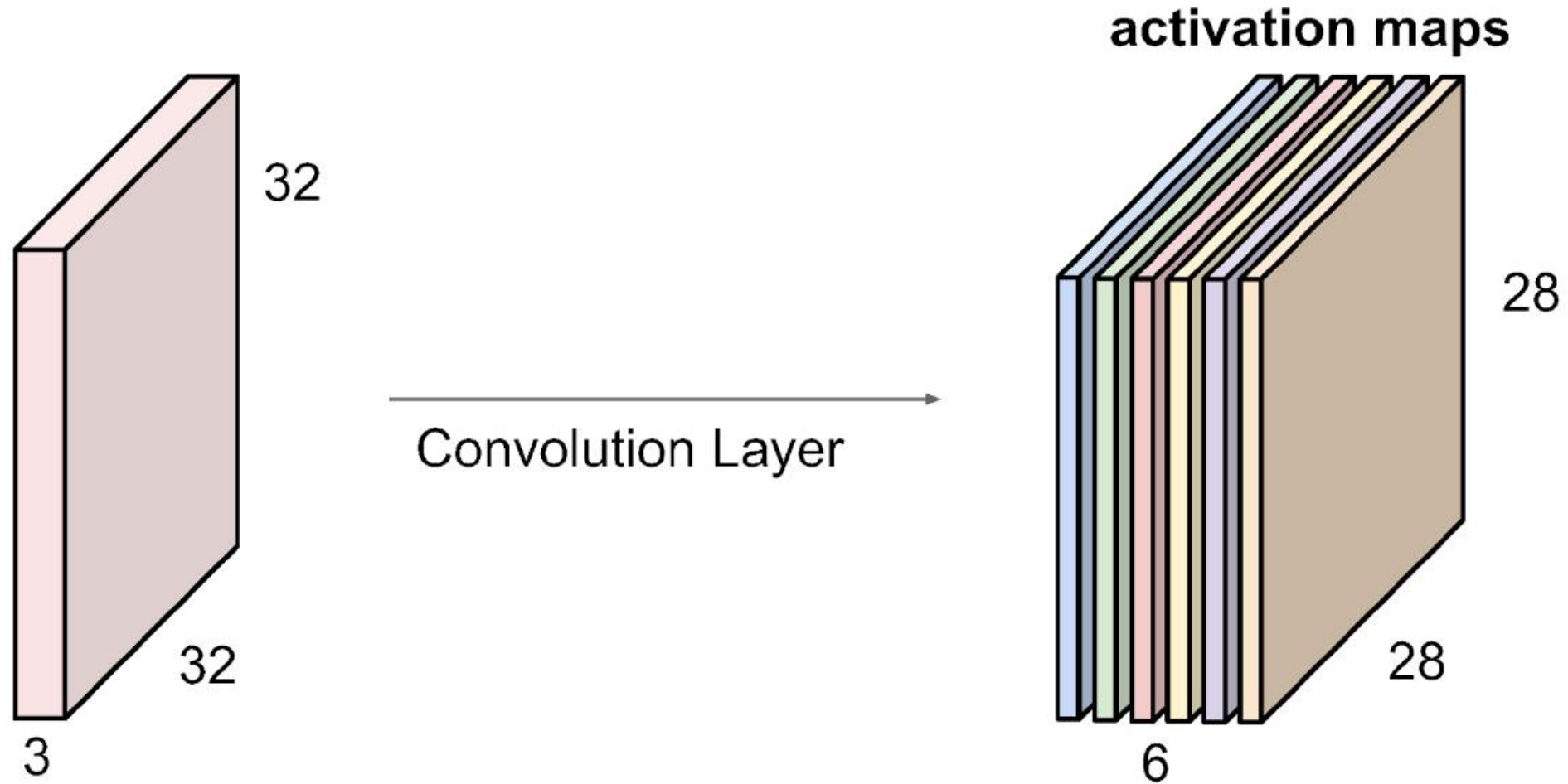


Convolution Layer

consider a second, **green** filter

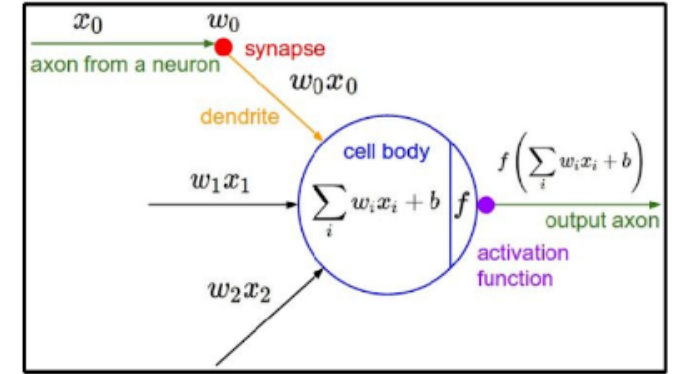
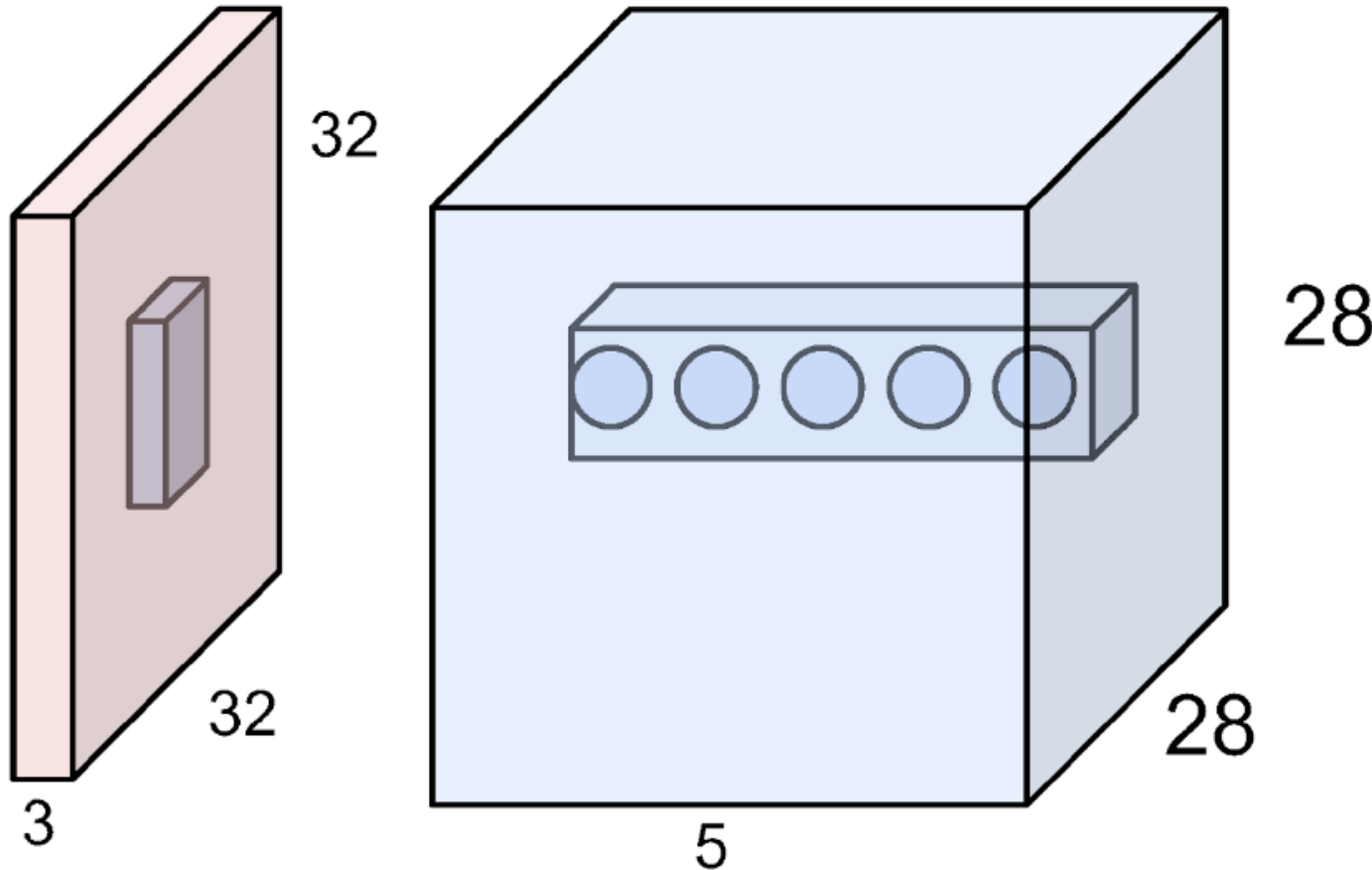


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



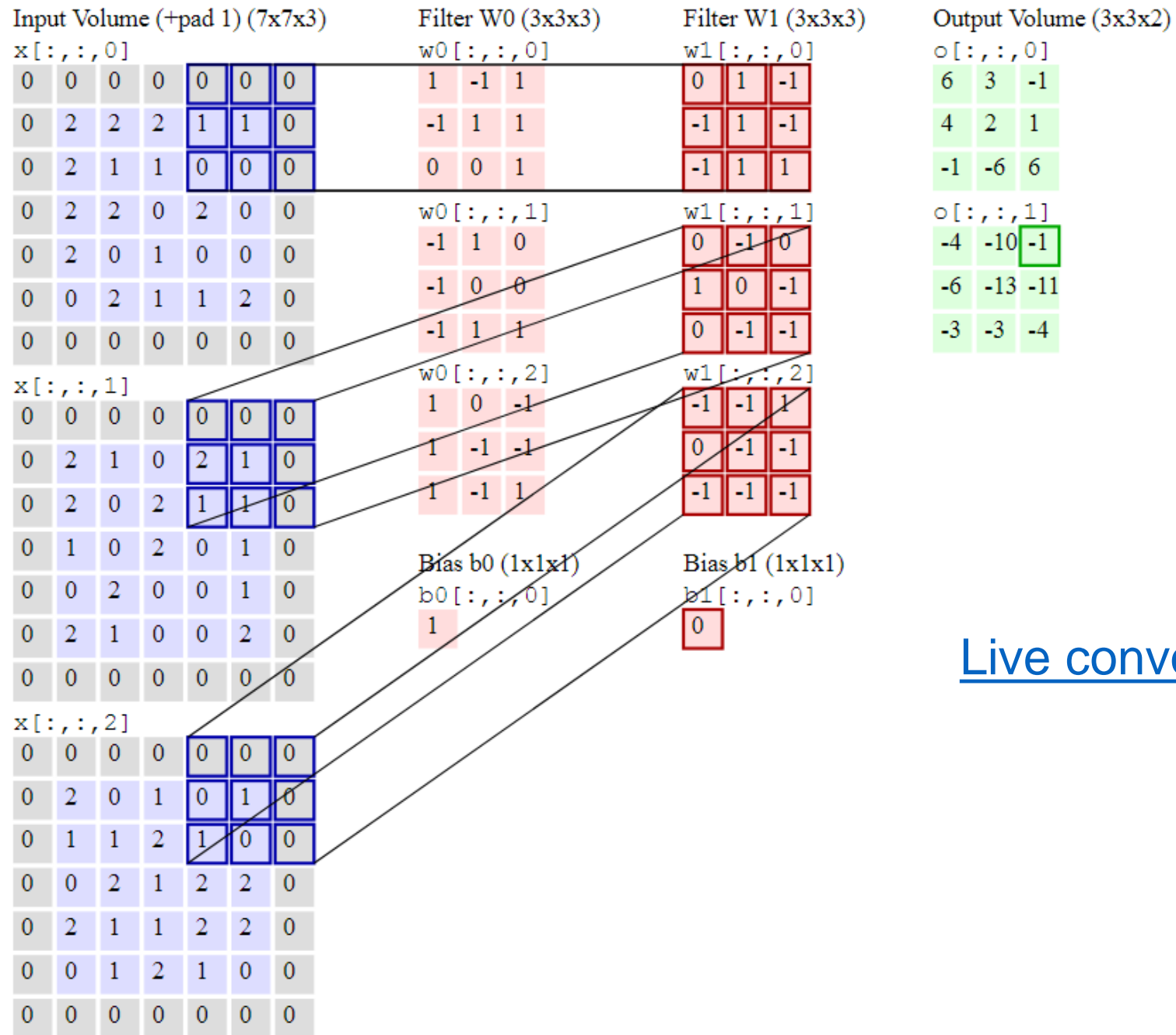
We stack these up to get a “new image” of size 28x28x6!

The brain/neuron view of CONV Layer



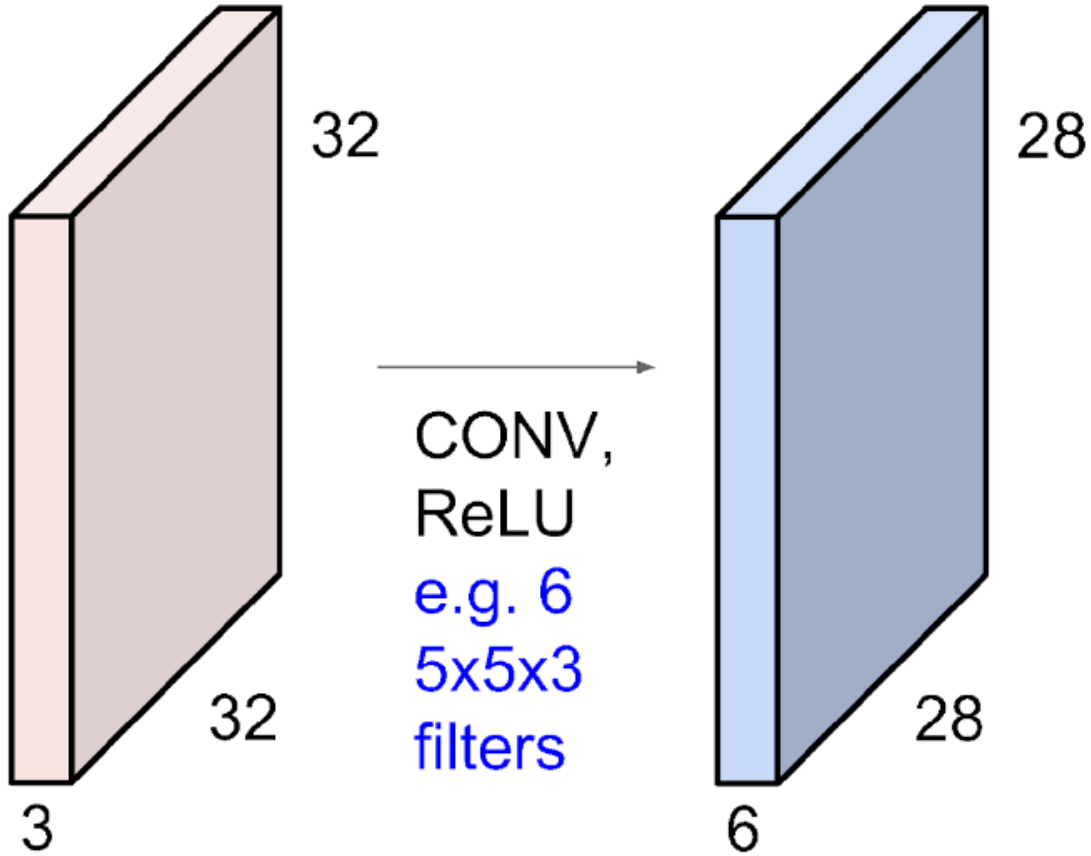
E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

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

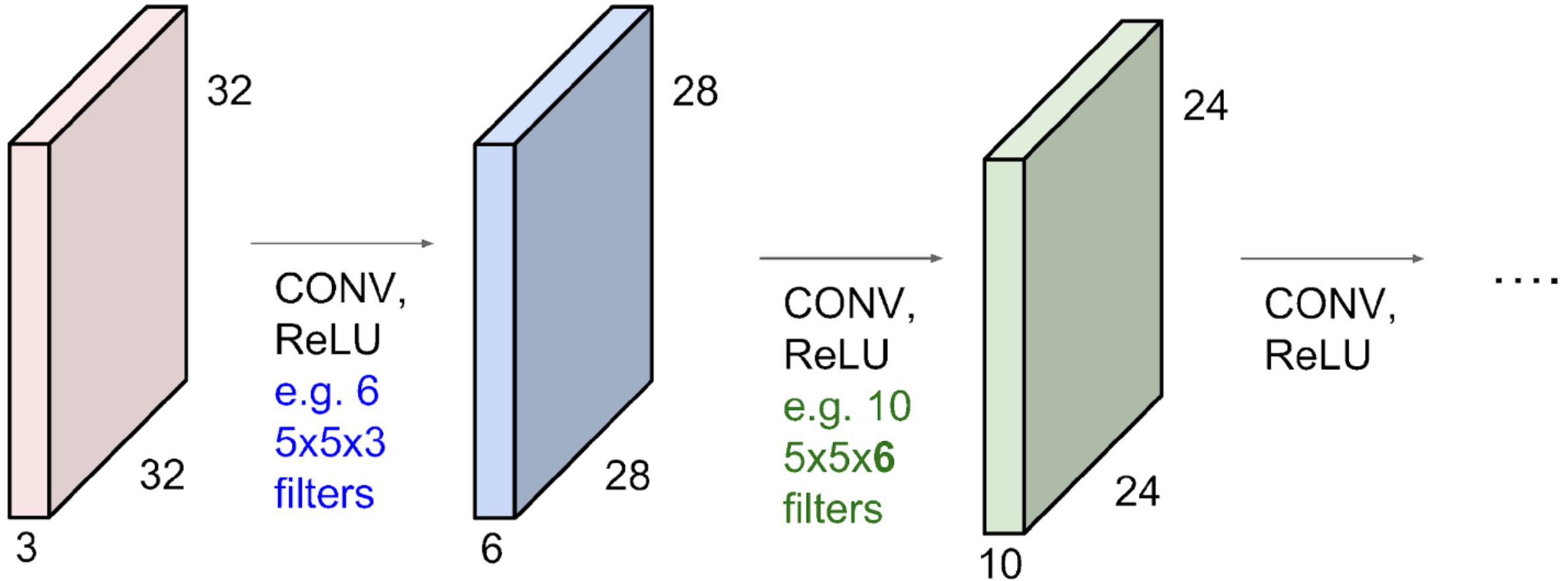


[Live convolution demo](#)

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



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

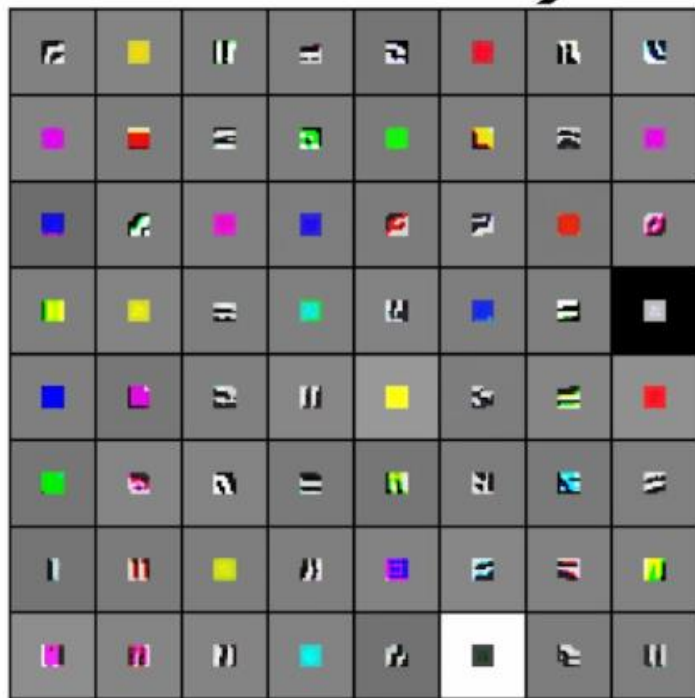
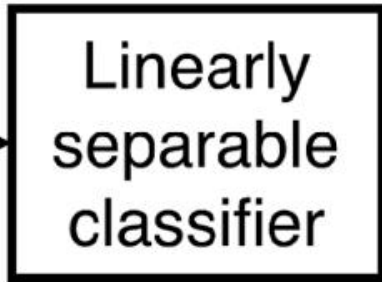
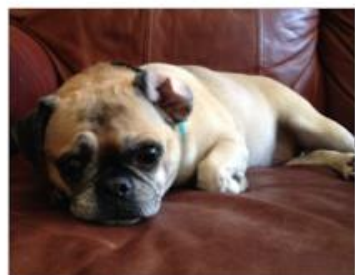




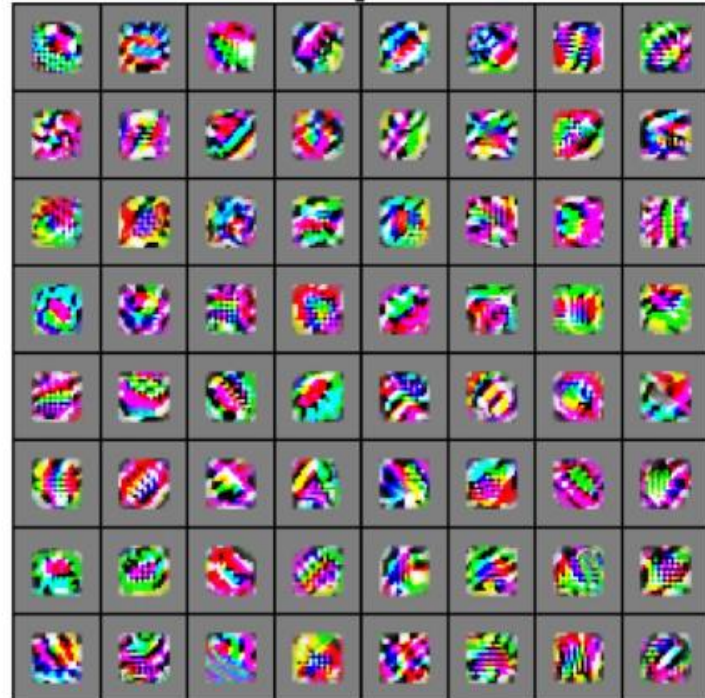
Preview

[Zeiler and Fergus 2013]

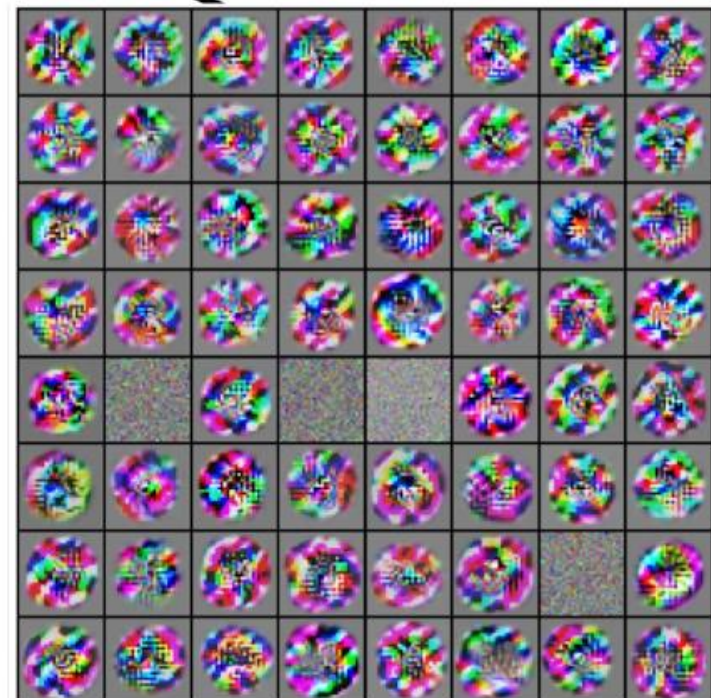
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



VGG-16 Conv1_1

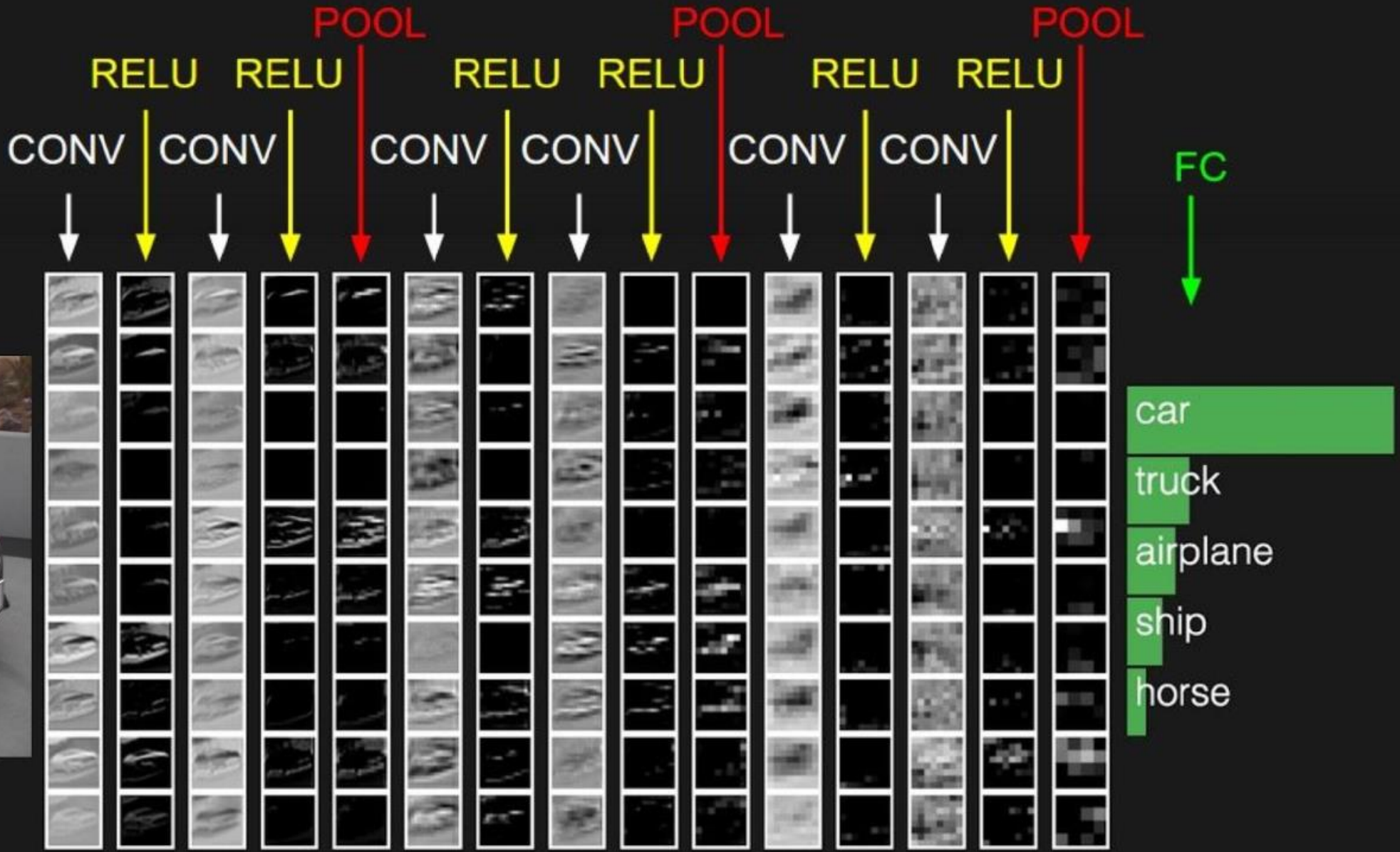


VGG-16 Conv3_2



VGG-16 Conv5_3

preview:



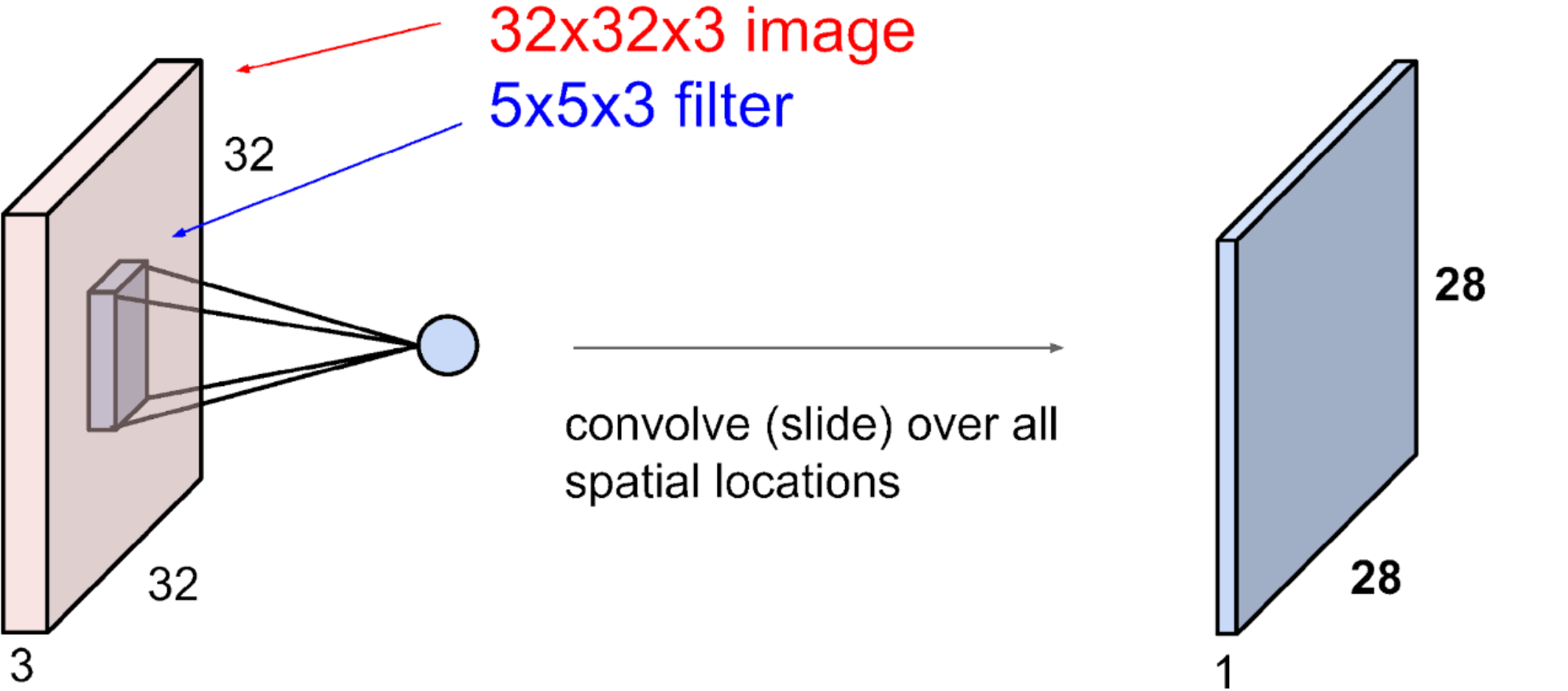
Layers used to build ConvNets

As we described above, a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet **architecture**.

Example Architecture: Overview. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail:

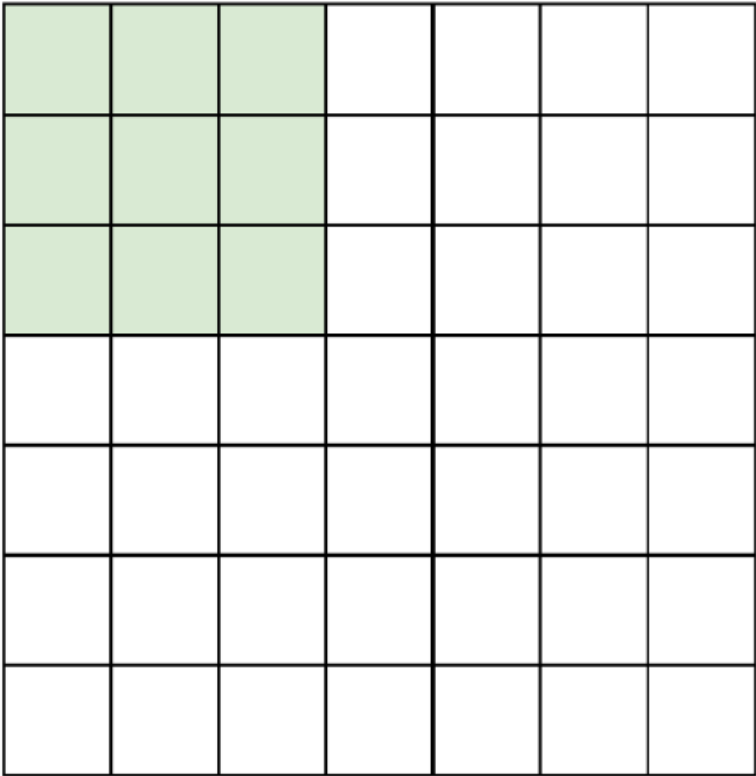
- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the $\max(\mathbf{0}, \mathbf{x})$ thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

A closer look at spatial dimensions:



A closer look at spatial dimensions:

7

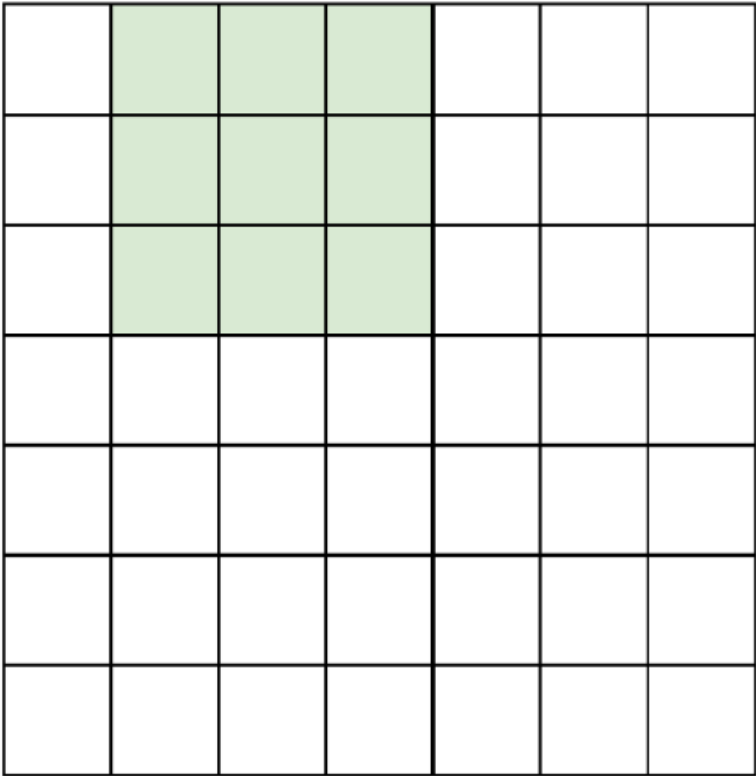


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

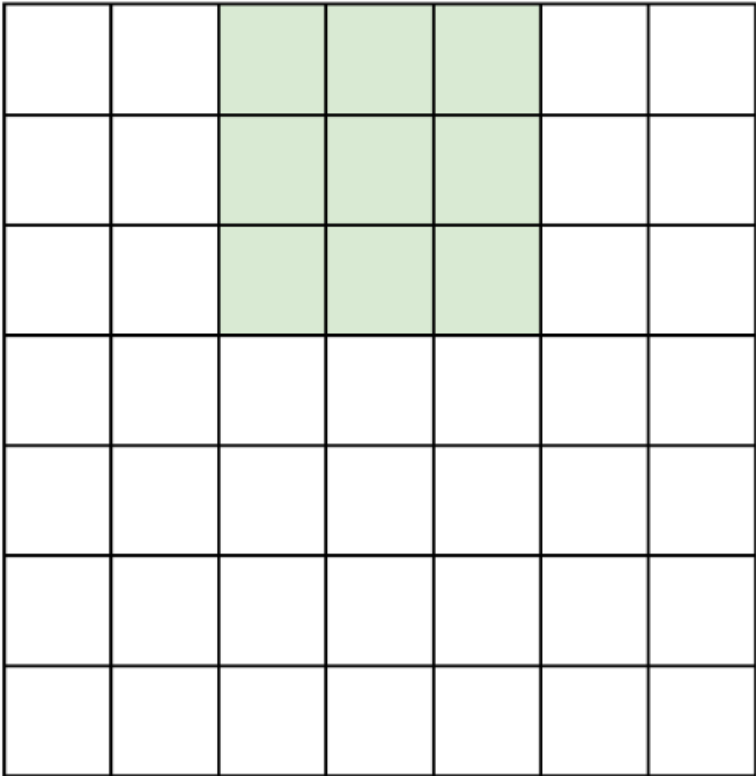


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

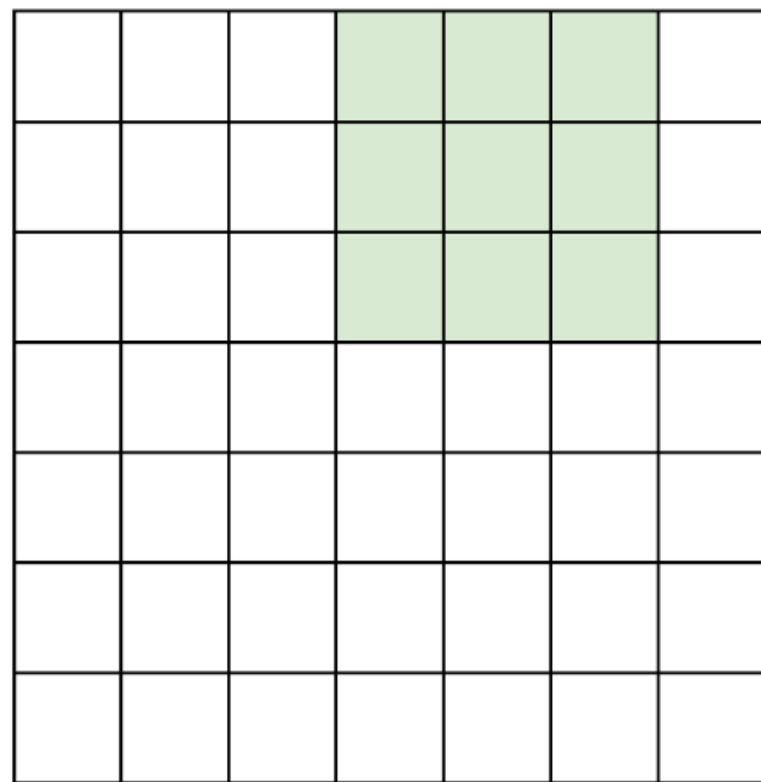


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7



7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

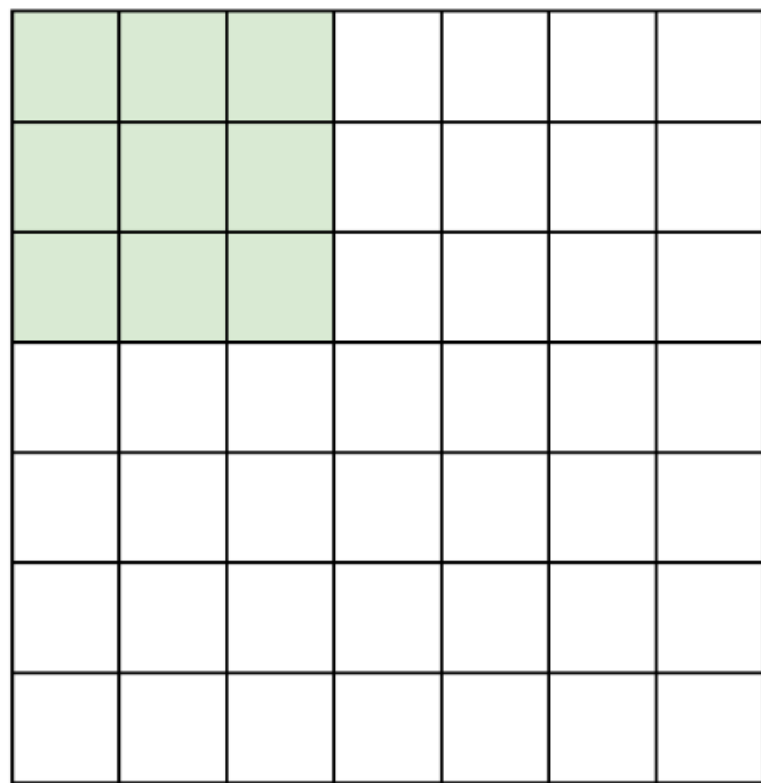
7

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

A closer look at spatial dimensions:

7

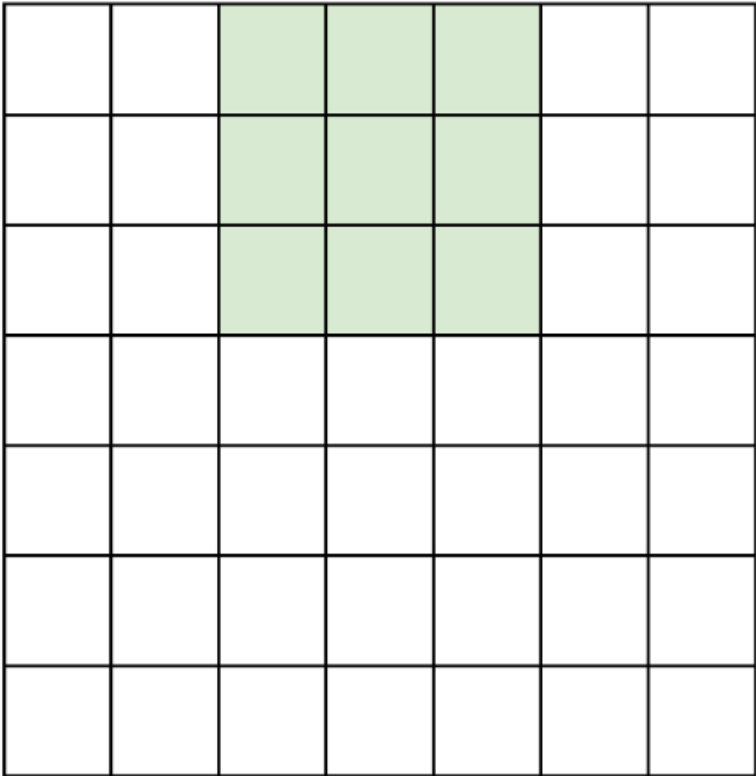


7

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

A closer look at spatial dimensions:

7

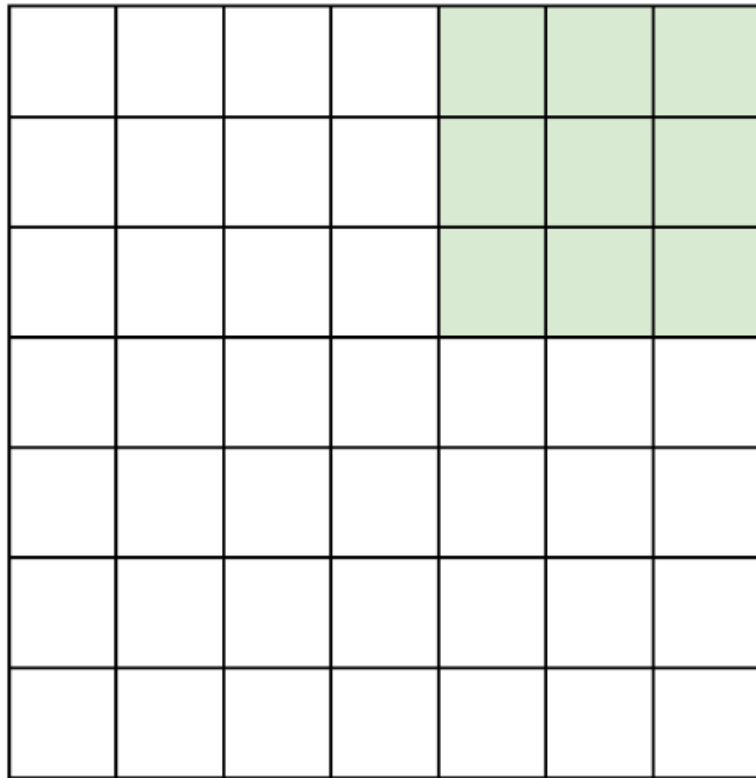


7

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

A closer look at spatial dimensions:

7

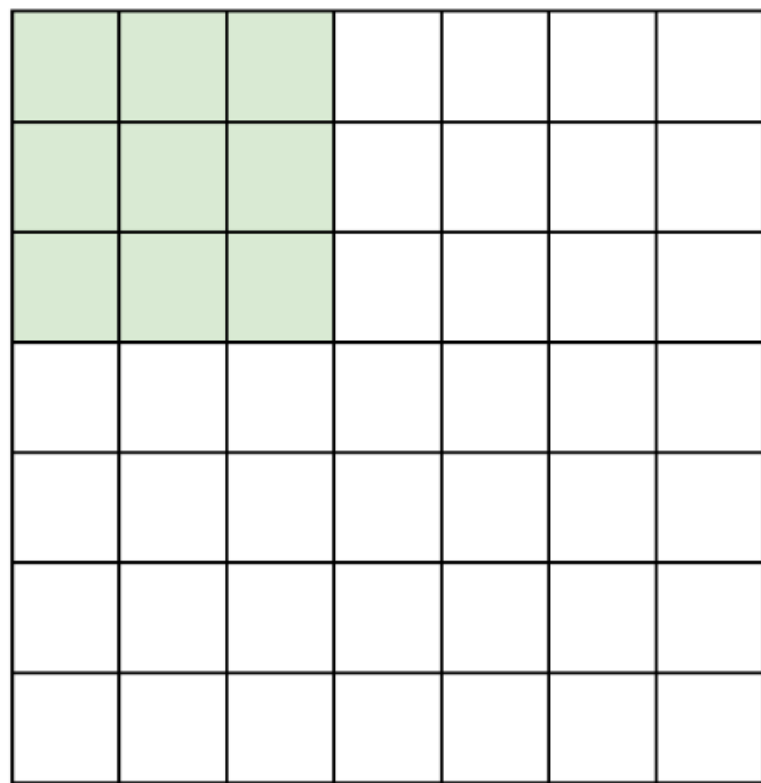


7

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

A closer look at spatial dimensions:

7

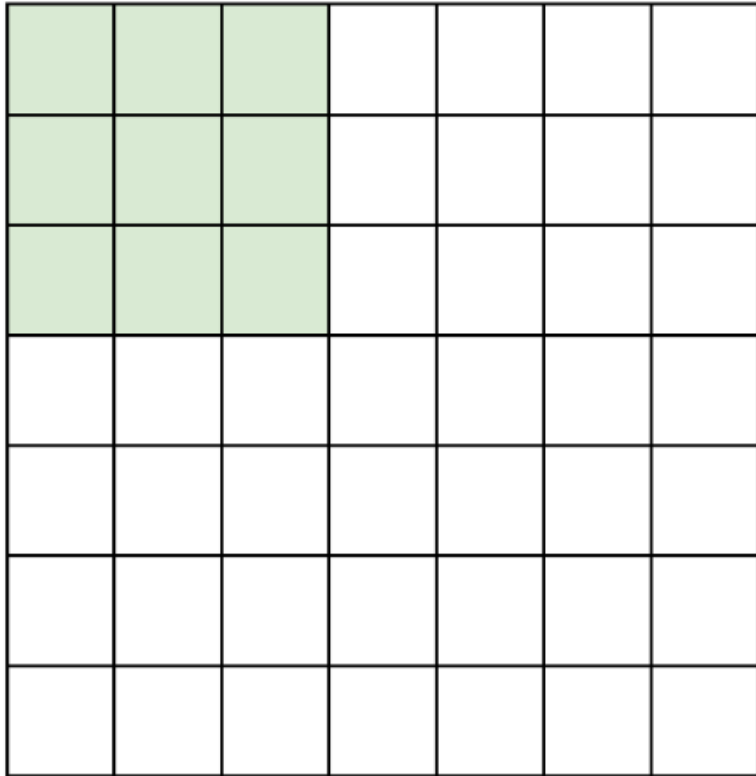


7

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

A closer look at spatial dimensions:

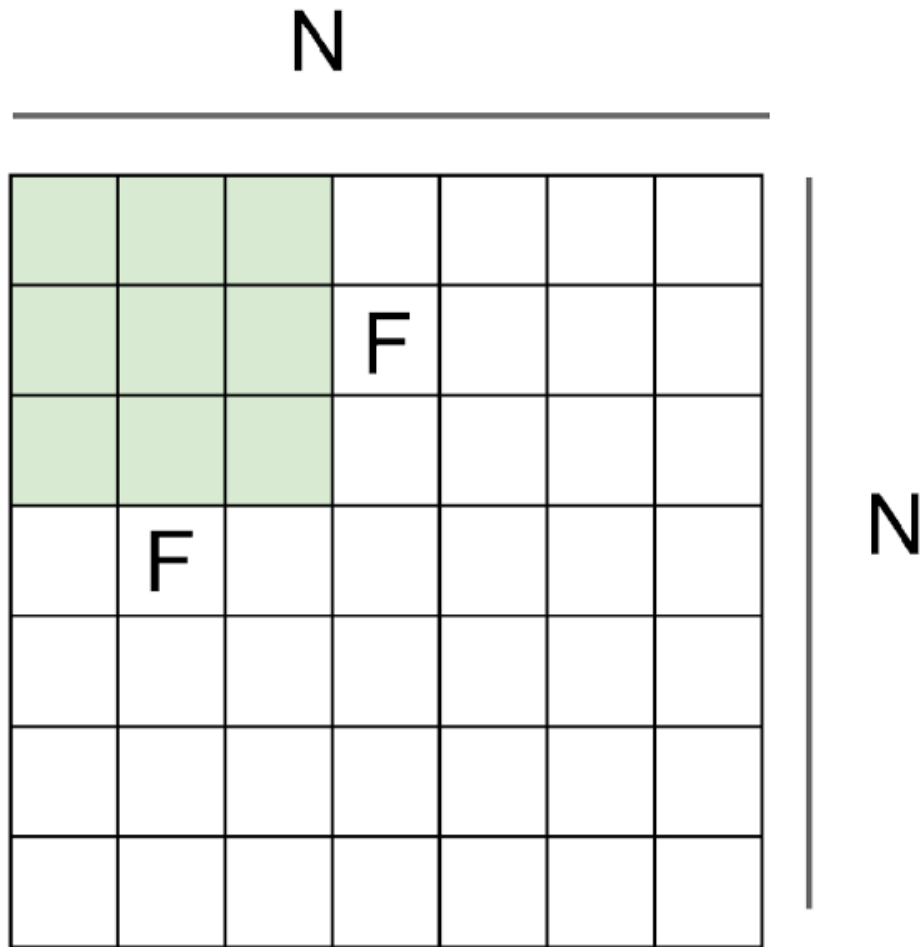
7



7

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

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



Output size:
 $(N - F) / \text{stride} + 1$

e.g. $N = 7, F = 3$:

stride 1 $\Rightarrow (7 - 3) / 1 + 1 = 5$

stride 2 $\Rightarrow (7 - 3) / 2 + 1 = 3$

stride 3 $\Rightarrow (7 - 3) / 3 + 1 = 2.33 \text{ :}\backslash$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

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

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

(recall:)

$$(N - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

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

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

7x7 output!

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

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 $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

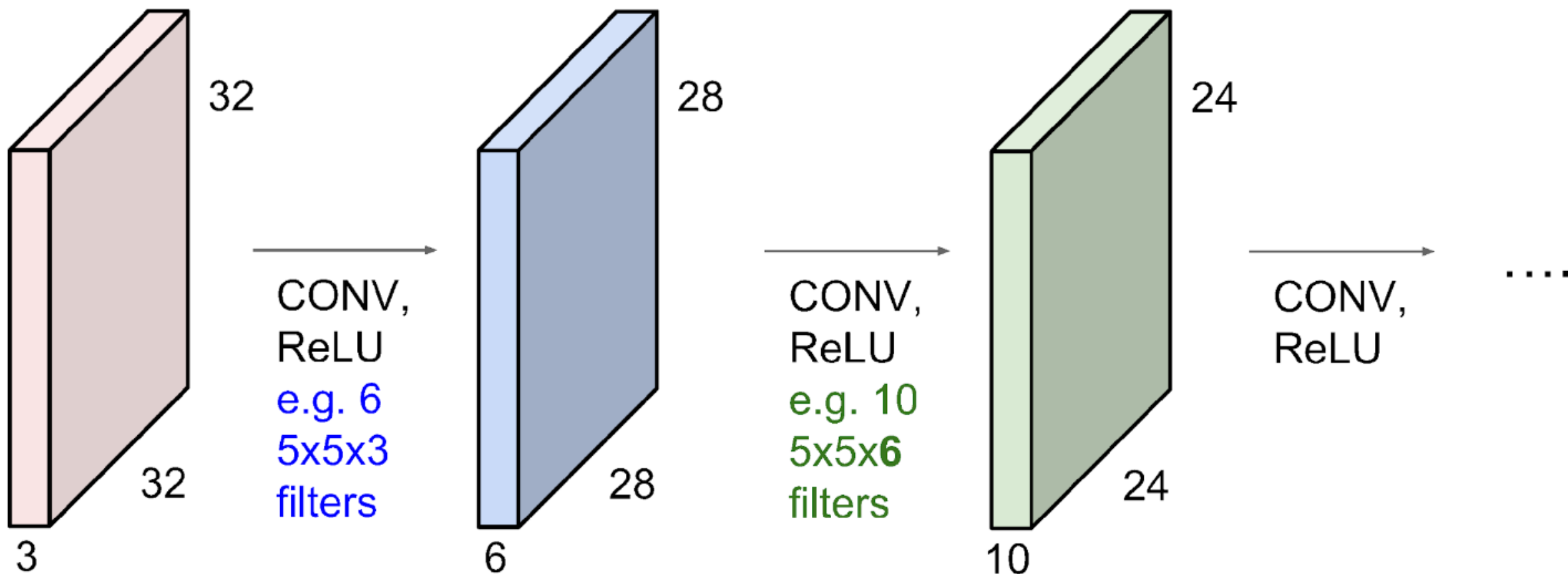
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

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.

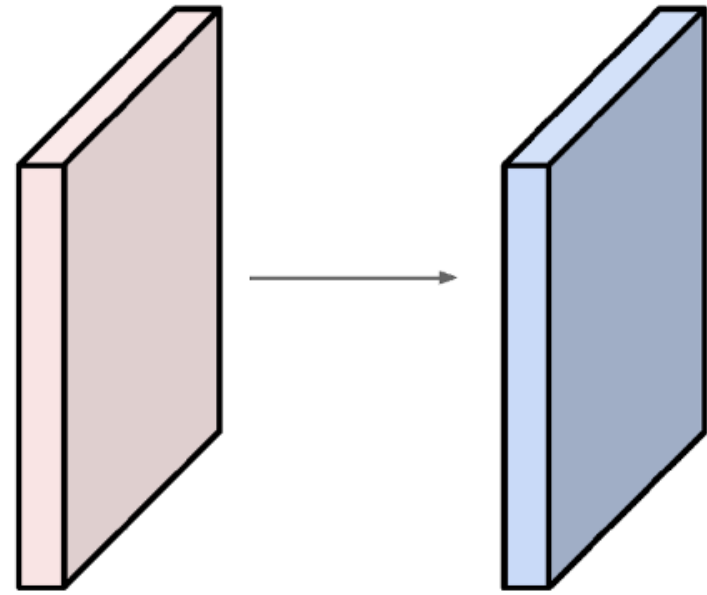


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples time:

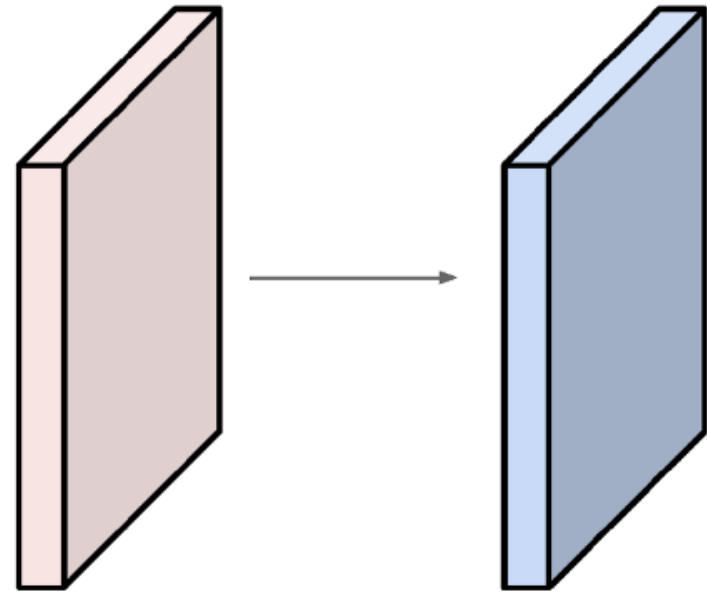
Input volume: **32x32x3**

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

Output volume size:

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

32x32x10

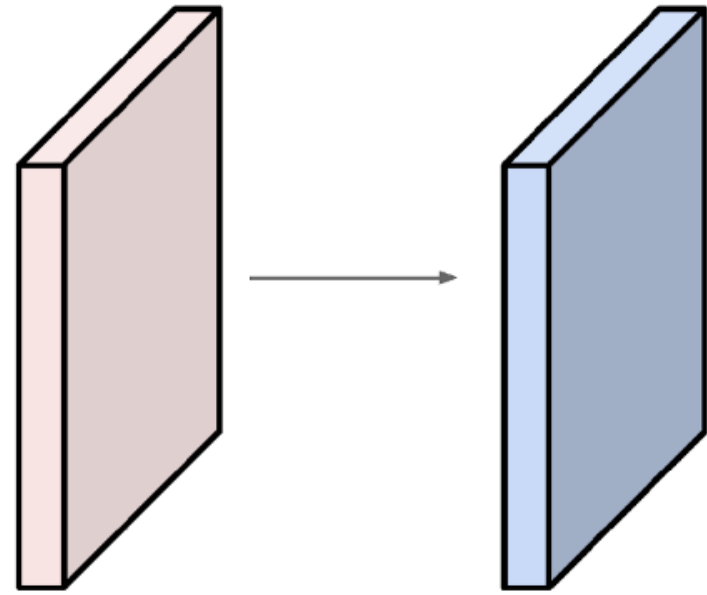


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

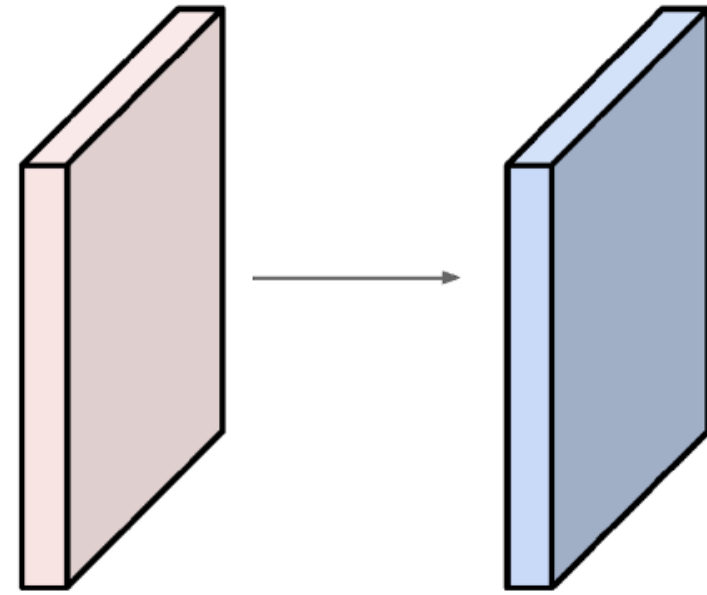
Number of parameters in this layer?



Examples time:

Input volume: **32x32x3**

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



Number of parameters in this layer?

each filter has $5*5*3 + 1 = 76$ params

(+1 for bias)

$\Rightarrow 76*10 = 760$

Summary. To summarize, the Conv Layer:

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

Summary. To summarize, the Conv Layer:

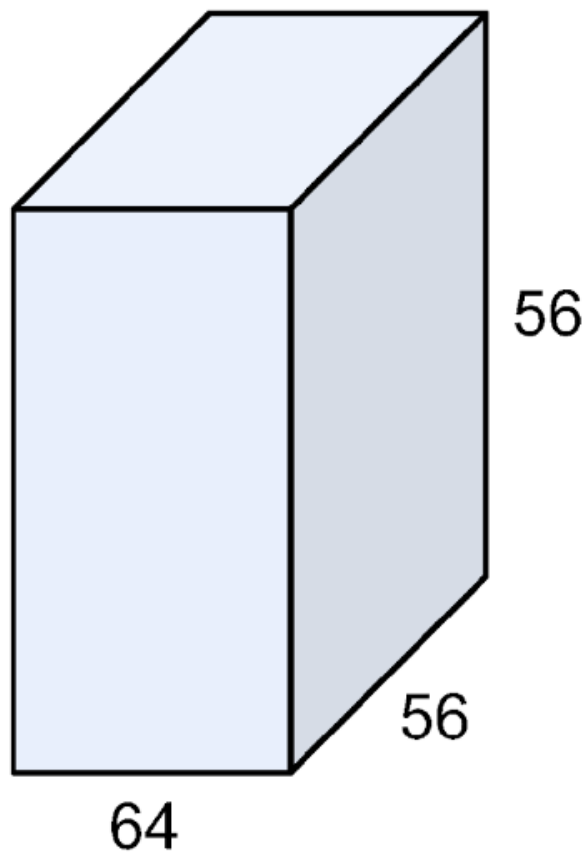
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1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

Common settings:

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

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

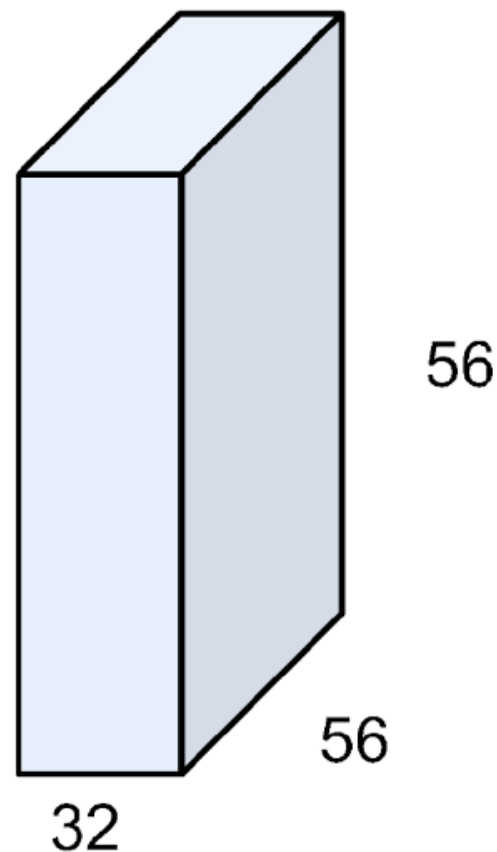
(btw, 1x1 convolution layers make perfect sense)



1x1 CONV
with 32 filters

→

(each filter has size
1x1x64, and performs a
64-dimensional dot
product)



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)
```

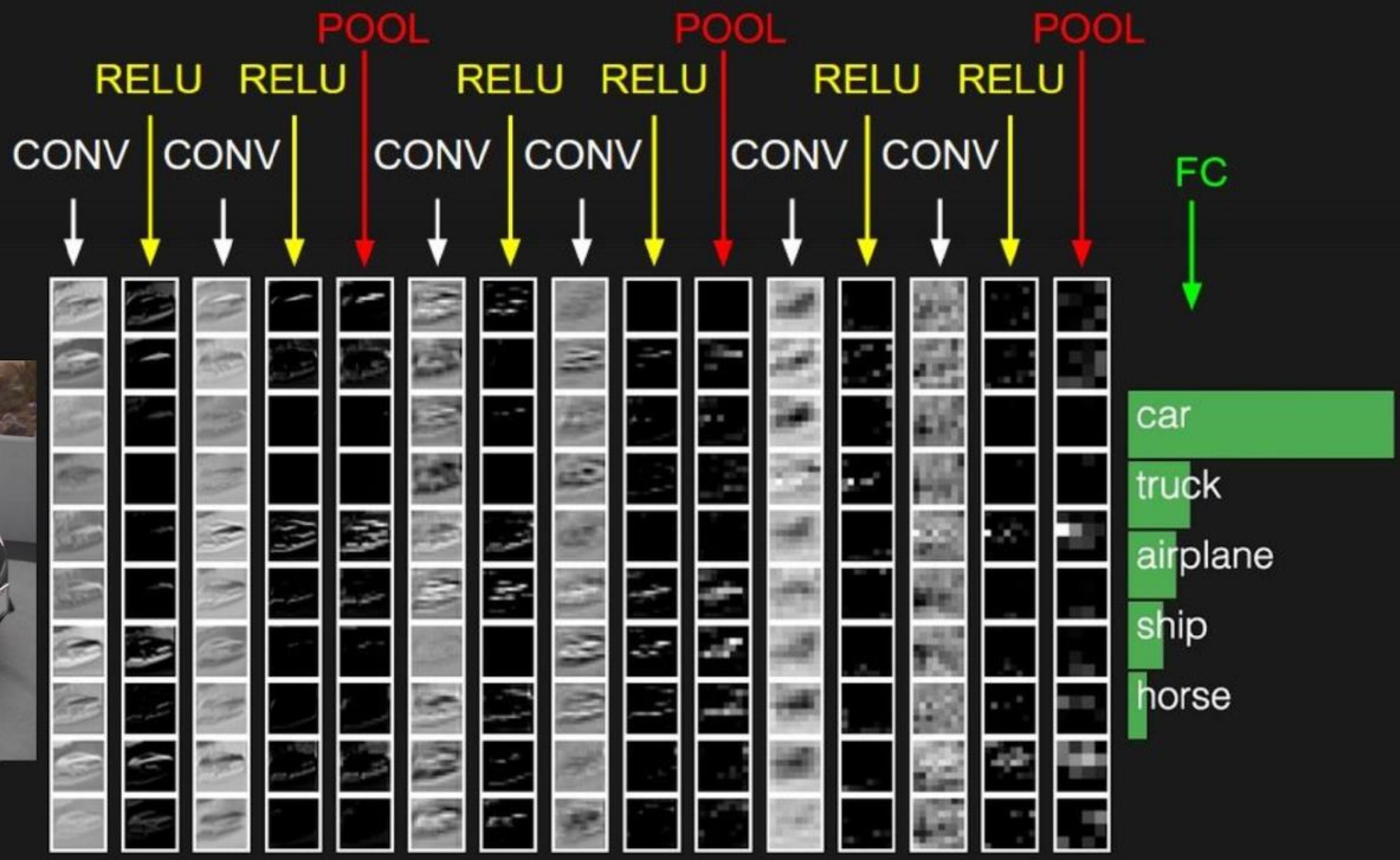
Example: CONV layer in Caffe

Summary. To summarize, the Conv Layer:

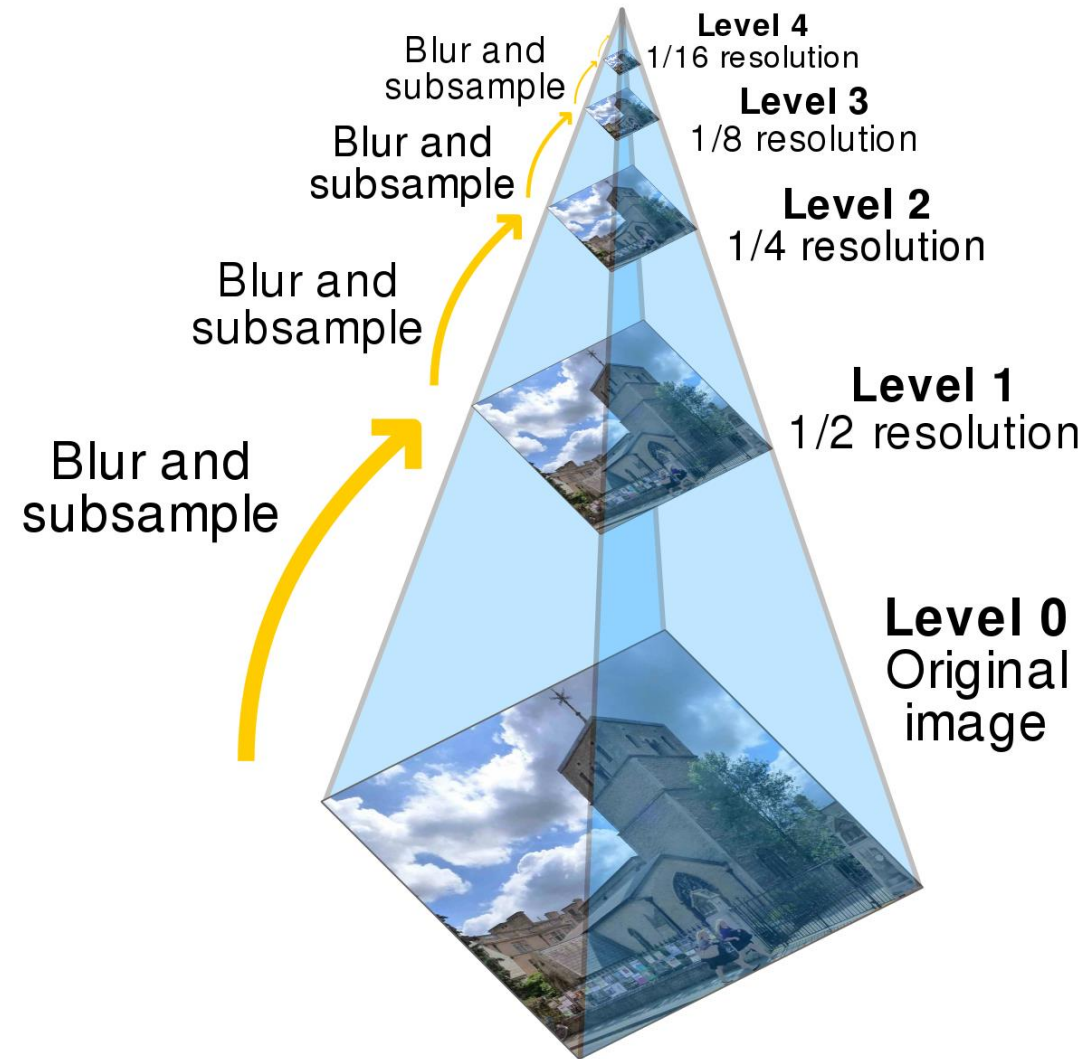
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .

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

two more layers to go: POOL/FC

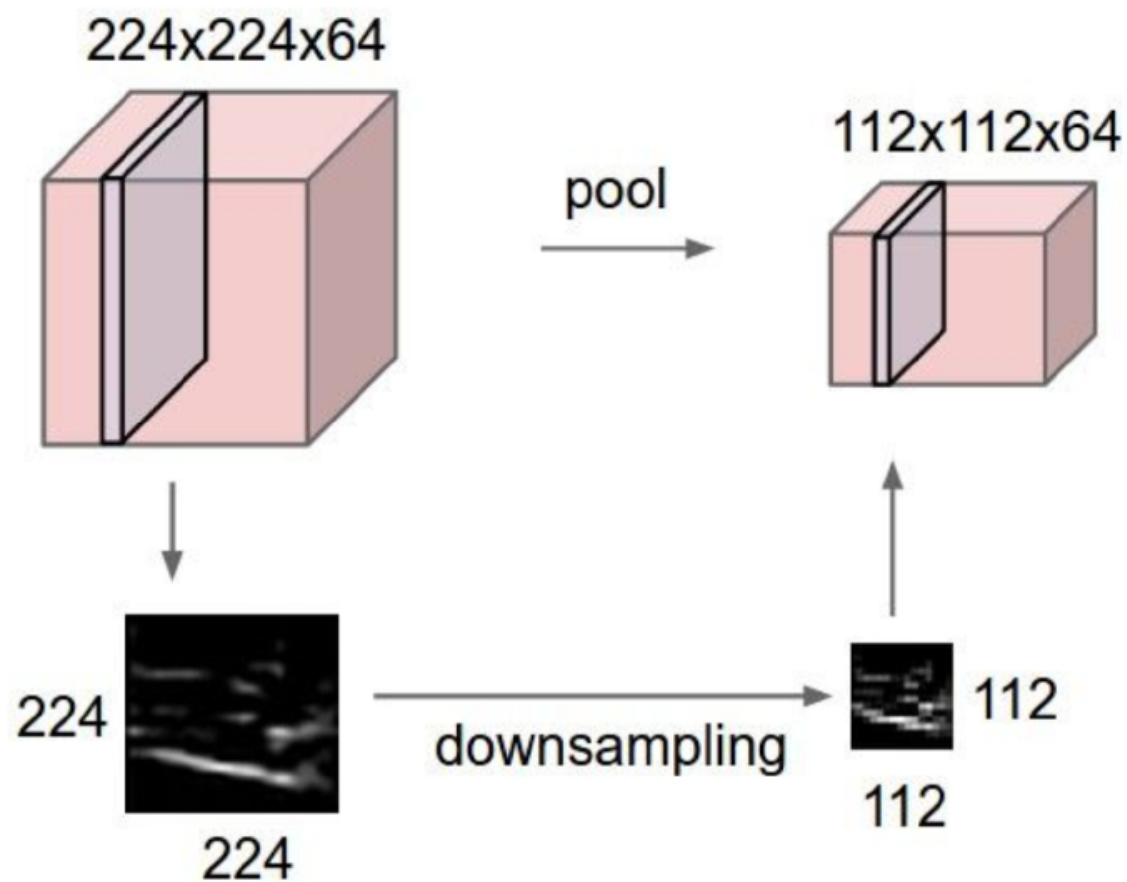


Pyramid (Image processing) - Subsampling



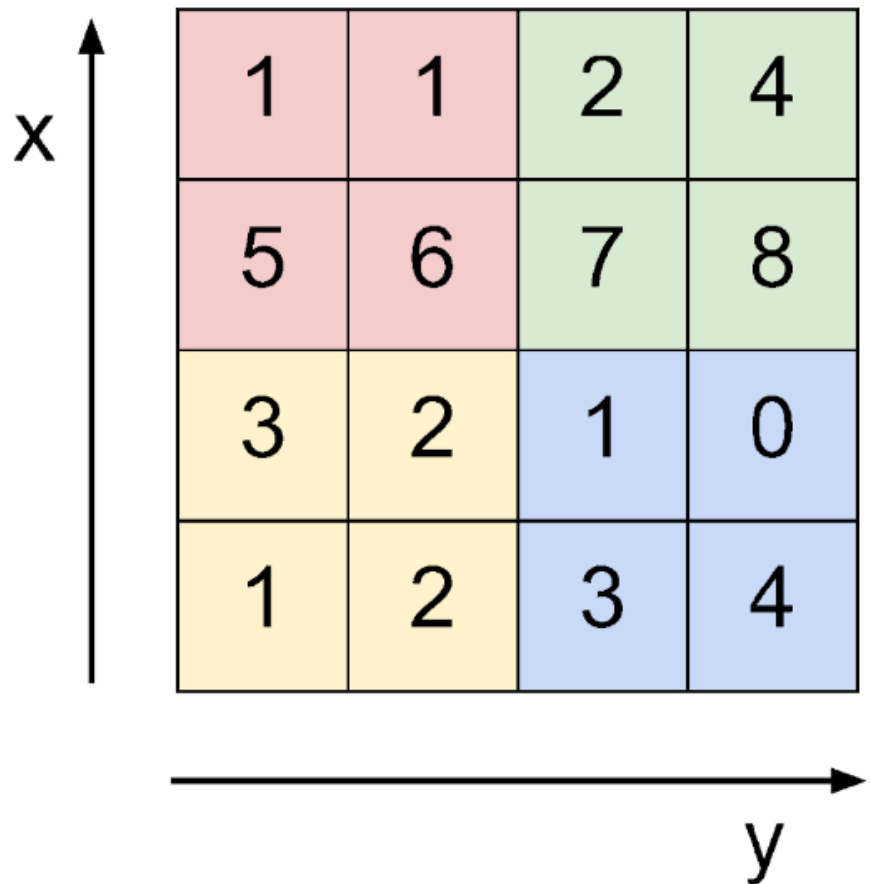
Pooling layer

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

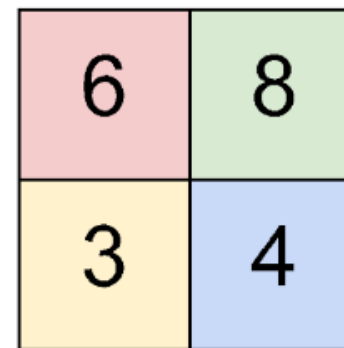


MAX POOLING

Single depth slice



max pool with 2x2 filters
and stride 2



Also used: Average
Pooling

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

Common settings:

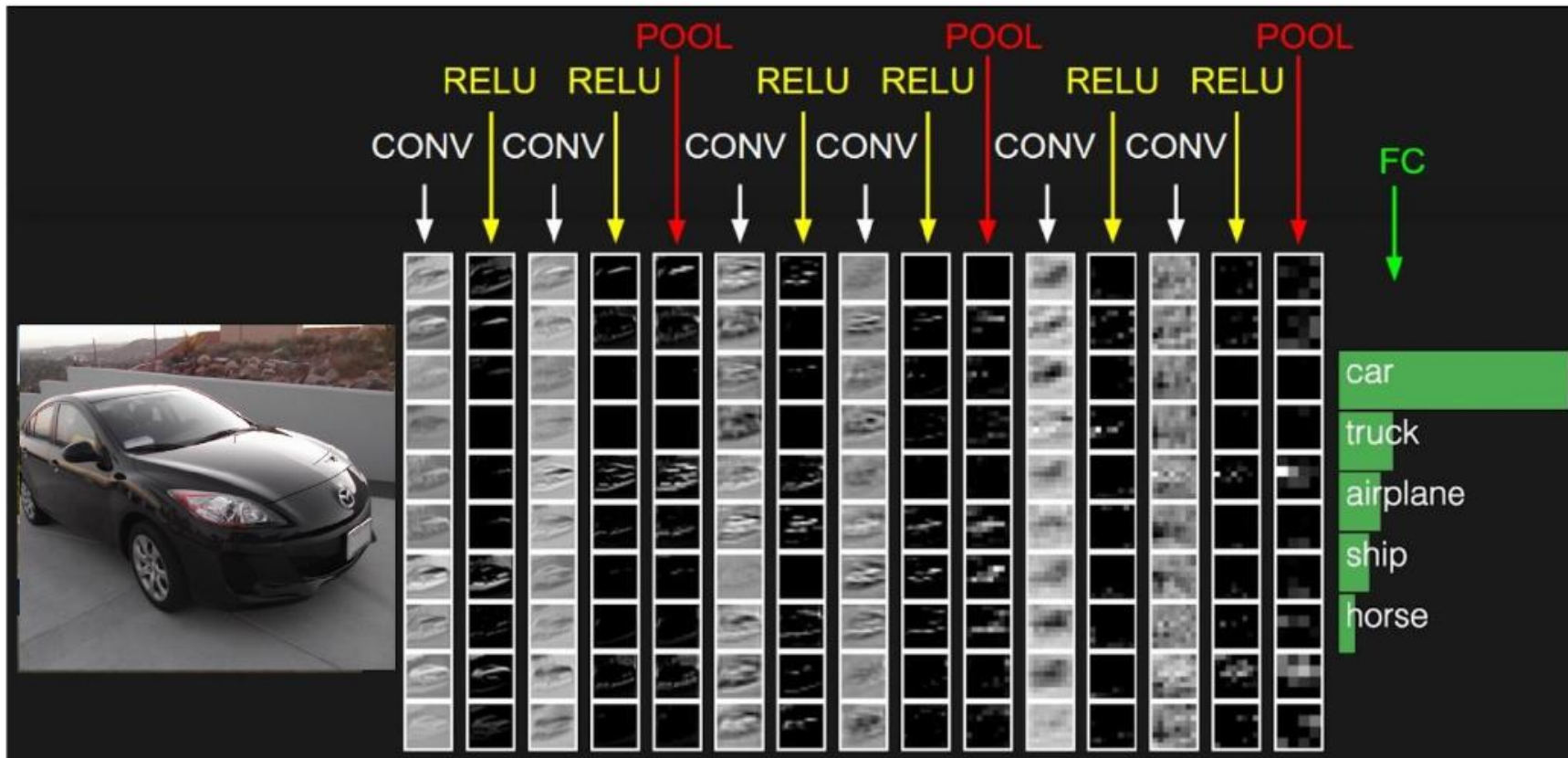
$$F = 2, S = 2$$

$$F = 3, S = 2$$

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

Fully Connected Layer (FC layer)

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



CNN Architectures (ImageNet)

ImageNet is an image database organized according to the **WordNet** hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.

[Click here](#) to learn more about ImageNet, [Click here](#) to join the ImageNet mailing list.



<http://www.image-net.org/>

A very large labeled images dataset

Currently: 14.197.122 images in 21.841 categories

IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)

<http://www.image-net.org/challenges/LSVRC/>

Competition

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale. One high level motivation is to allow researchers to compare progress in detection across a wider variety of objects -- taking advantage of the quite expensive labeling effort. Another motivation is to measure the progress of computer vision for large scale image indexing for retrieval and annotation.

For details about each challenge please refer to the corresponding page.

- [ILSVRC 2017](#)
 - [ILSVRC 2016](#)
 - [ILSVRC 2015](#)
 - [ILSVRC 2014](#)
 - [ILSVRC 2013](#)
 - [ILSVRC 2012](#)
 - [ILSVRC 2011](#)
 - [ILSVRC 2010](#)
- **LeNet 1998** (the grandfather)
 - **AlexNet 2012**. The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error). The Network had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other (previously it was common to only have a single CONV layer always immediately followed by a POOL layer).
 - **ZF Net**. ILSVRC 2013 winner
 - **GoogLeNet**. ILSVRC 2014 winner
 - **VGGNet**. The runner-up in ILSVRC 2014.
 - **ResNet**. winner of ILSVRC 2015.

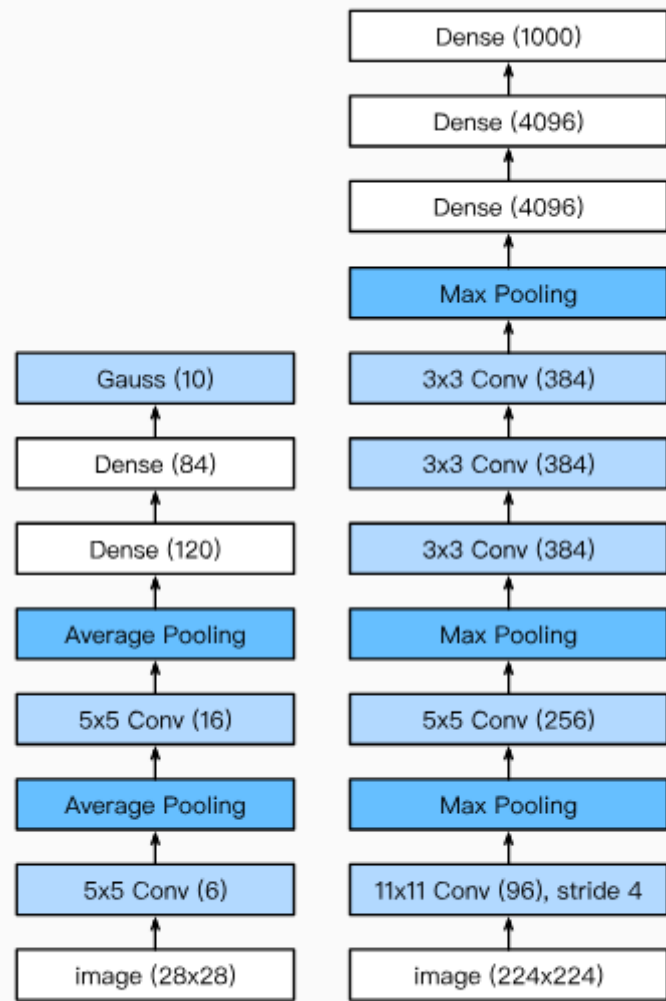


Fig. 6.10 LeNet (left) and AlexNet (right)

VGGNet in detail. Lets break down the [VGGNet](#) in more detail as a case study. The whole VGGNet is composed of CONV layers that perform 3x3 convolutions with stride 1 and pad 1, and of POOL layers that perform 2x2 max pooling with stride 2 (and no padding). We can write out the size of the representation at each step of the processing and keep track of both the representation size and the total number of weights:

```
INPUT: [224x224x3]      memory: 224*224*3=150K  weights: 0
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]    memory: 112*112*64=800K  weights: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M  weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M  weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]    memory: 56*56*128=400K  weights: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K  weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K  weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K  weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]    memory: 28*28*256=200K  weights: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K  weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K  weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K  weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]    memory: 14*14*512=100K  weights: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K  weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K  weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K  weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]     memory: 7*7*512=25K   weights: 0
FC: [1x1x4096]        memory: 4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]        memory: 4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]        memory: 1000  weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```

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 architectures look like
[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K, SOFTMAX
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
 - but recent advances such as ResNet/GoogLeNet challenge this paradigm

Transfer Learning

“You need a lot of a data if you want to train/use CNNs”

Transfer Learning

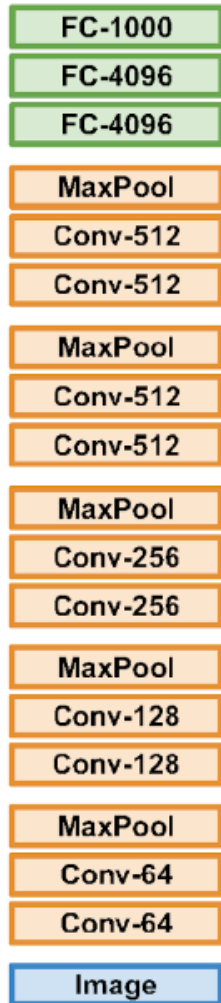
“You need a lot of data if you want to train/use CNNs”

BUSTED

Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet



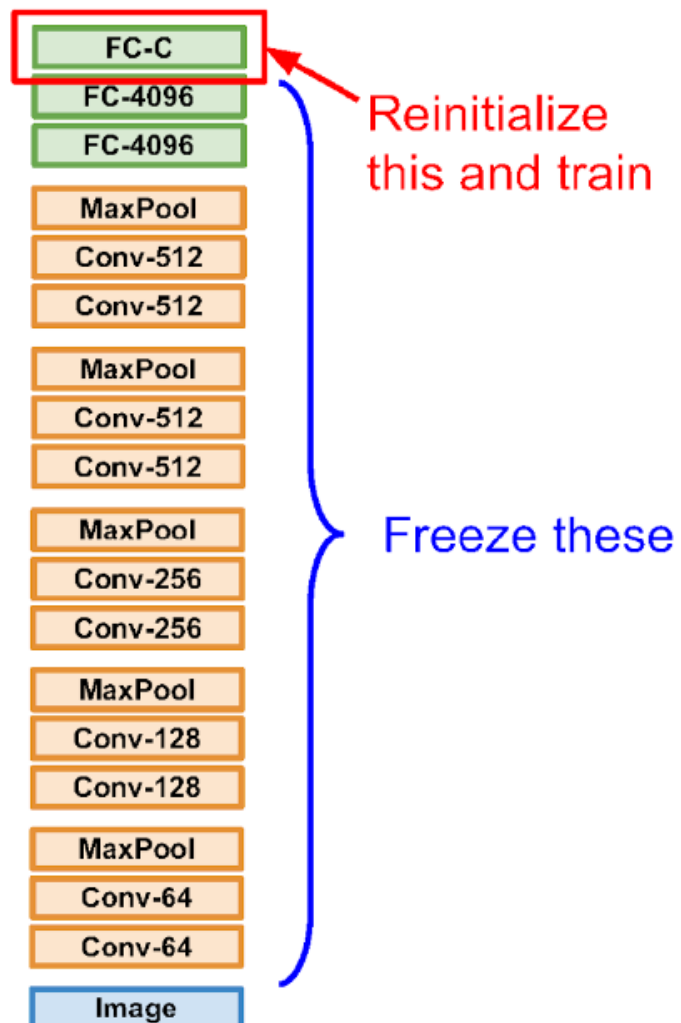
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet



2. Small Dataset (C classes)



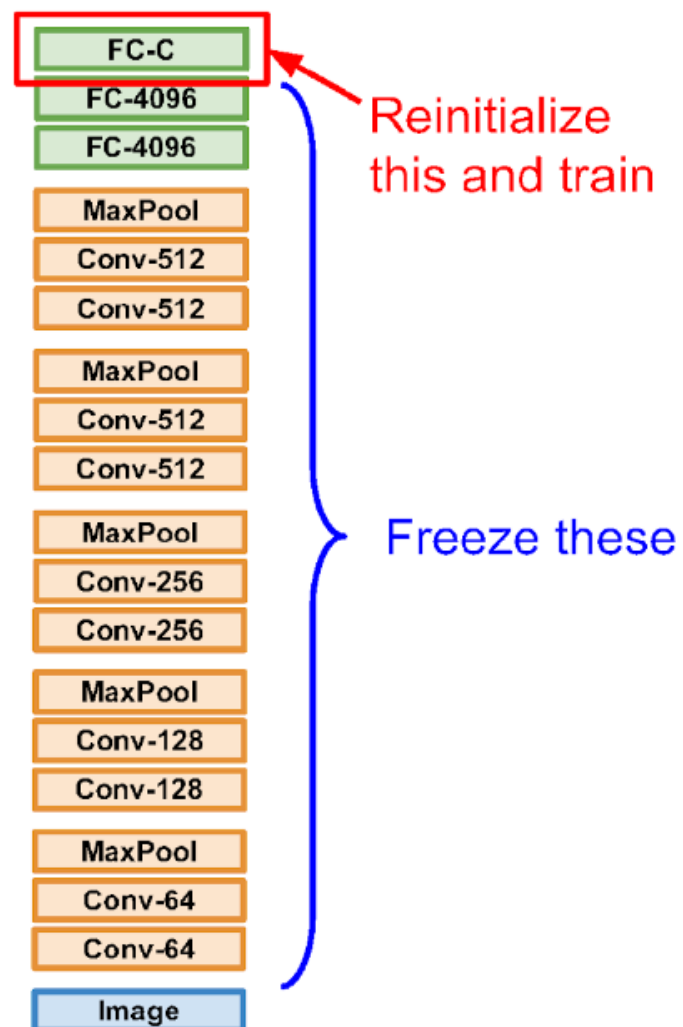
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

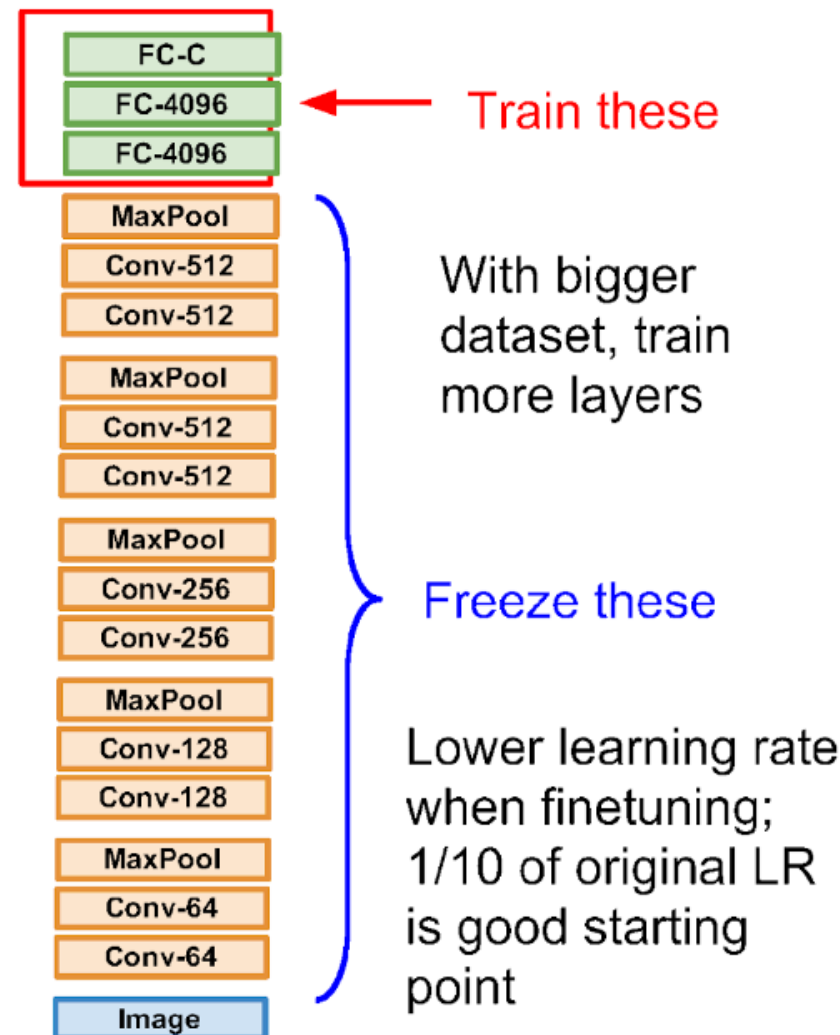
1. Train on Imagenet



2. Small Dataset (C classes)



3. Bigger dataset





More specific



More generic

	very similar dataset	very different dataset
very little data	?	?
quite a lot of data	?	?



More specific



More generic

	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	?



More specific



More generic

	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Transfer learning with CNNs is pervasive... (it's the norm, not an exception)

Object Detection (Fast R-CNN)

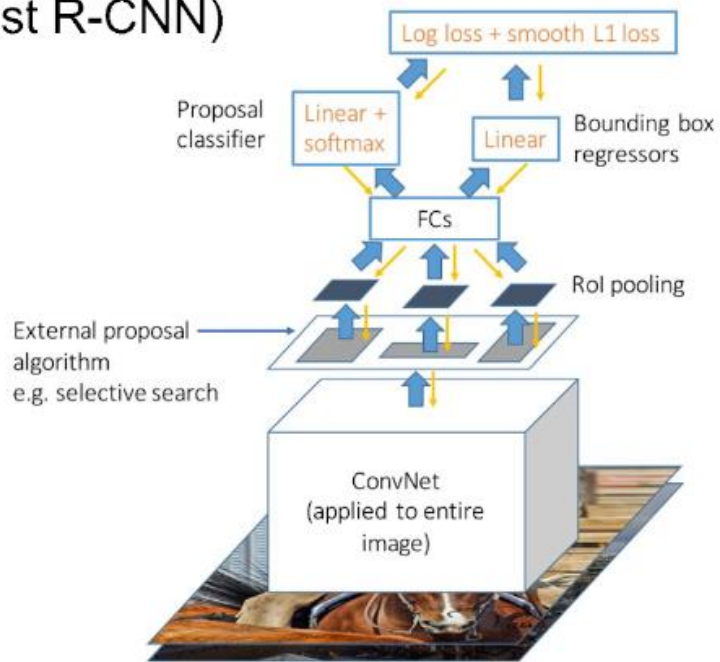
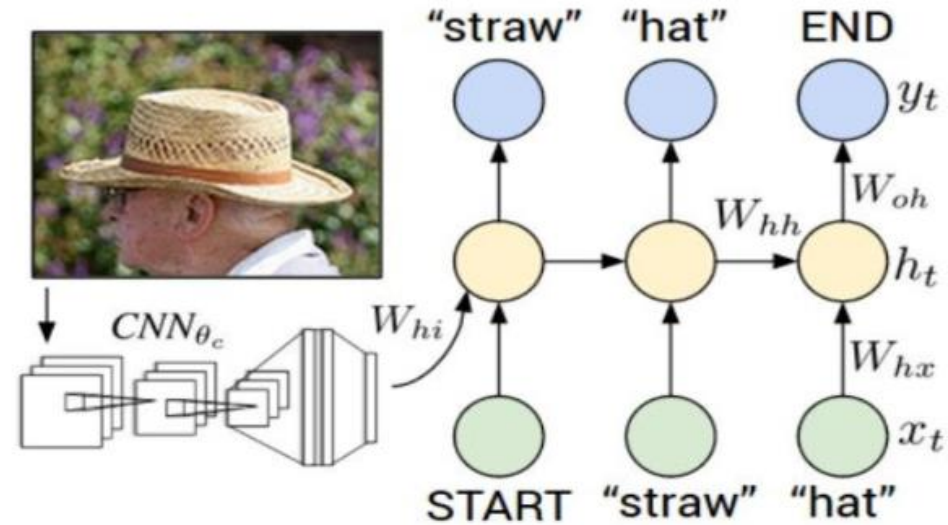
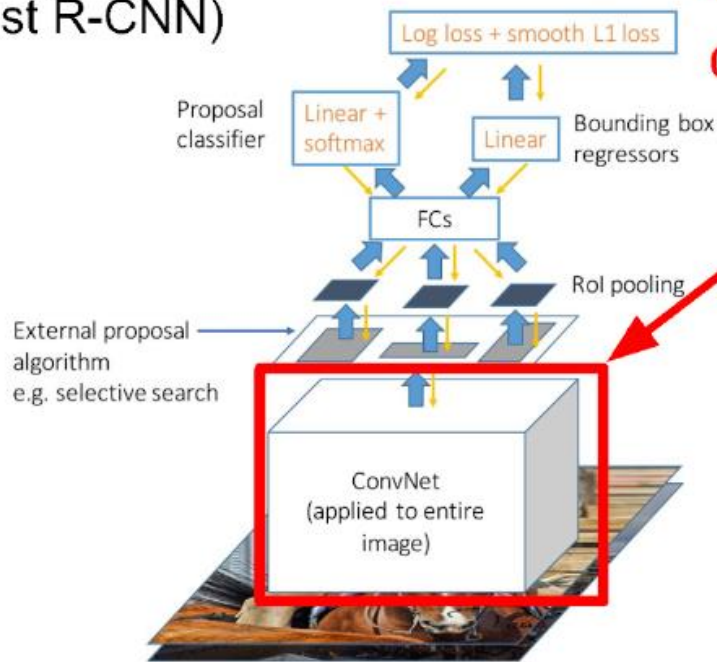


Image Captioning: CNN + RNN



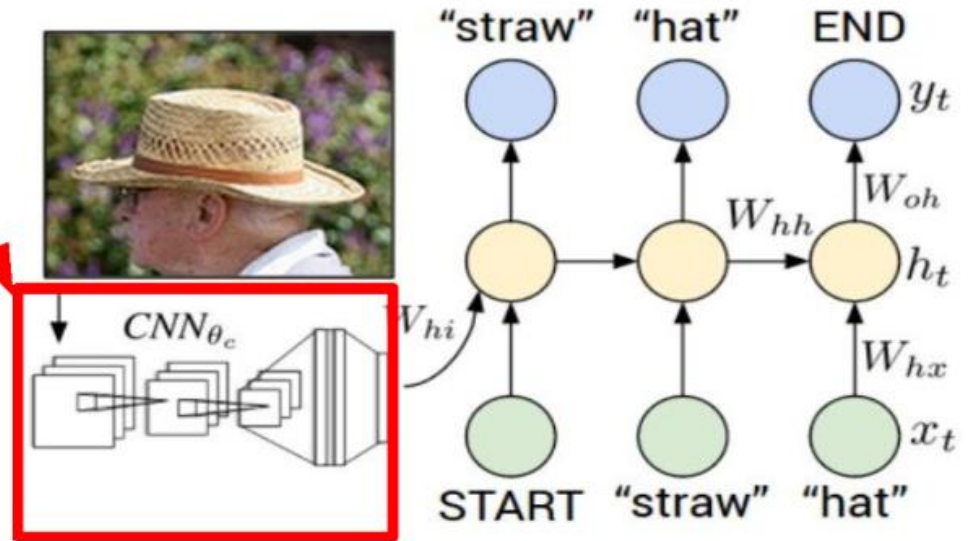
Transfer learning with CNNs is pervasive... (it's the norm, not an exception)

Object Detection
(Fast R-CNN)



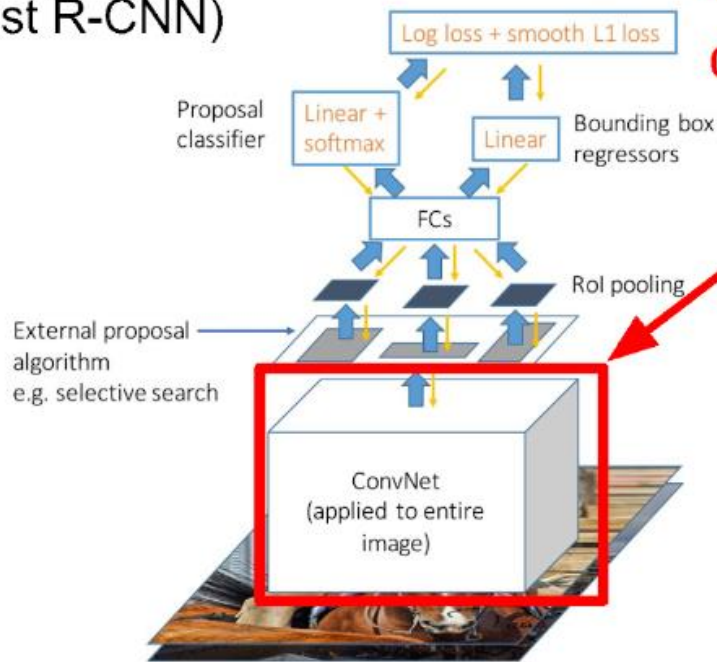
CNN pretrained
on ImageNet

Image Captioning: CNN + RNN



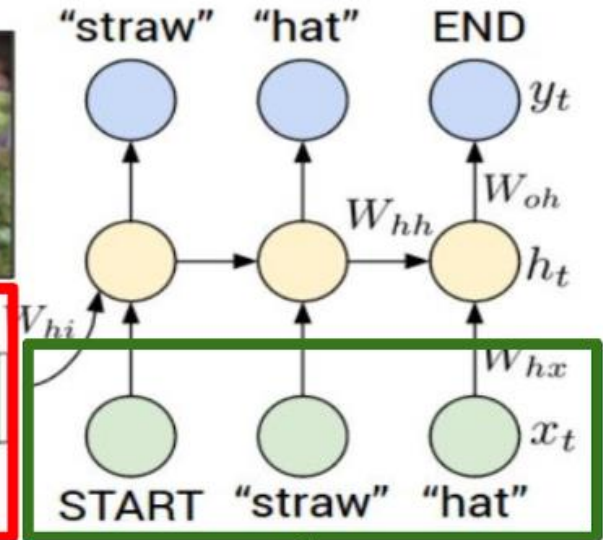
Transfer learning with CNNs is pervasive... (it's the norm, not an exception)

Object Detection
(Fast R-CNN)



CNN pretrained
on ImageNet

Image Captioning: CNN + RNN



Word vectors pretrained
with word2vec

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for
Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Takeaway for your projects and beyond:

Have some dataset of interest but it has $< \sim 1\text{M}$ images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

Keras <https://keras.io/applications/>

PyTorch <https://pytorch.org/docs/stable/torchvision/models.html>

MXNet (Gluon) https://mxnet.apache.org/api/python/gluon/model_zoo.html

TensorFlow <https://github.com/tensorflow/models>

Caffe <https://github.com/BVLC/caffe/wiki/Model-Zoo>

Επιλεγμένες πηγές και πρακτικά παραδείγματα

Convolutional Neural Networks

[Stanford Intro to CNNs](#)

[Dive into Deep Learning](#)

Περιέχουν εισαγωγή στα CNNs και παρουσίαση των διαφόρων αρχιτεκτονικών ConvNets του Imagenet

Ένα [απλό παράδειγμα CNN στο MNIST με Keras](#). Η είσοδος (εικόνα) έχει ένα μόνο επίπεδο καθώς είναι grayscale. [Keras CNN στο CIFAR-10](#) (πατήστε “Next” για να δείτε το ίδιο πρόβλημα με data augmentation, με το ResNet καθώς και μια οπτικοποίηση των συνελκτικών φίλτρων).

Transfer Learning

[Εισαγωγή του Stanford](#)

[Tutorial](#) σε Pytorch που δείχνει δύο διαφορετικές στρατηγικές training μετά το transfer learning.

Εισάγουμε το ResNet18 και κάνουμε train πρώτα σε ολόκληρο το δίκτυο και μετά μόνο στο τελικό fully connected επίπεδο

Βασική ιστορική βιβλιογραφία

- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.
- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786), 504-507.
- Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7), 1527-1554.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

στον φάκελο bibliography. Μπορείτε να βρείτε τις βιβλιογραφικές αναφορές και τα papers για τις διάφορες αρχιτεκτονικές του ImageNet στις εισαγωγές του Stanford και του Dive (προηγούμενο slide)