CSCE 633: Machine Learning

Lecture 29: Convolutional Neural Networks

Texas A&M University
Bobak Mortazavi
Zhale Nowroozi

Last Time: Neural Networks – Original Motivation

- Inspiration from the brain
  - Brain is a powerful information processing device
  - Composed of a large number of processing units (neurons)
  - Neurons operating in parallel → large connectivity
  - Neural networks as a paradigm for parallel processing

$$f^{(1)}(\mathbf{x}) = a\left(w_0^{(1)} + \sum_{n=1}^{N} w_n^{(1)} x_n\right)$$

Last Time: Single Layer Neural Networks

Recursive rescipe for single layer perceptron units

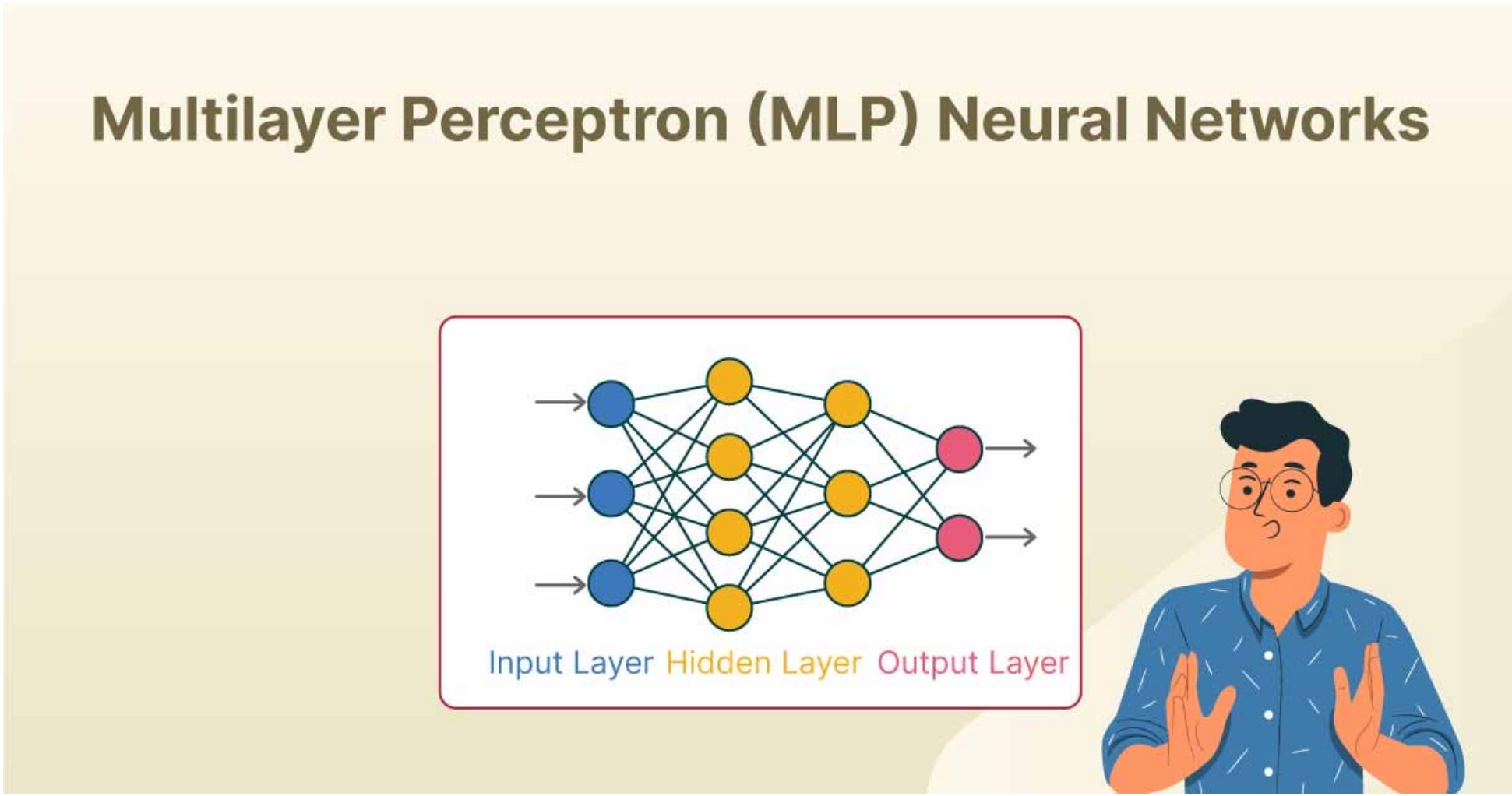
1: **input:** Activation function $a(\cdot)$

2: Compute linear combination: $v = w_0^{(1)} + \sum_{n=1}^{N} w_n^{(1)} x_n$

3: Pass result through activation: $a(v)$

4: **output:** Single layer unit $a(v)$

Last Time: Multilayer Perceptron



A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

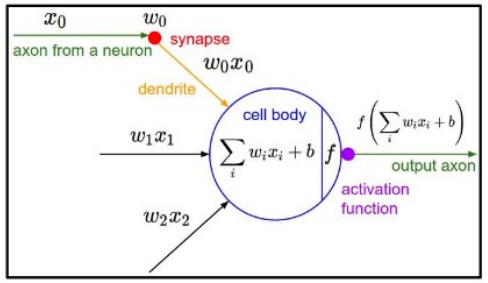
The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

recognized
letters of the alphabet

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

update rule:

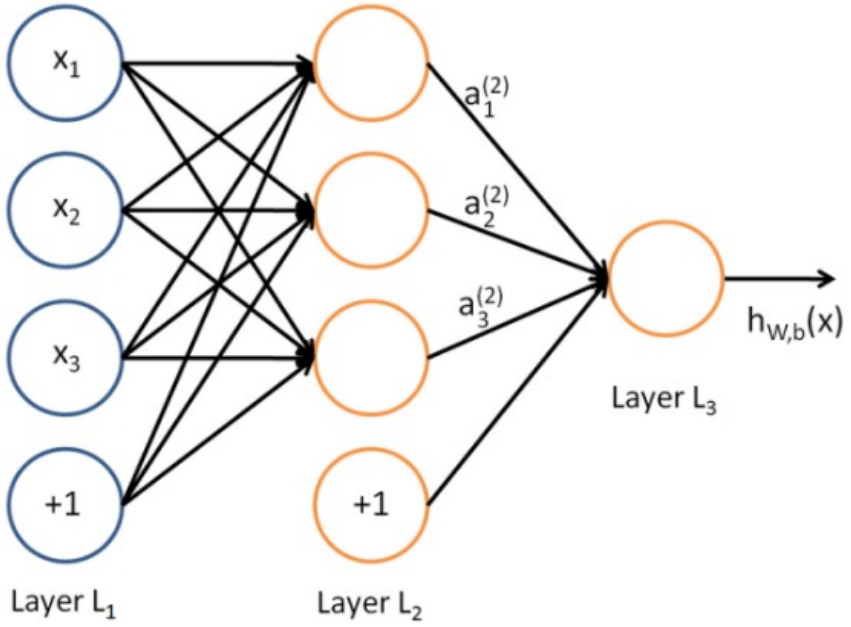
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$



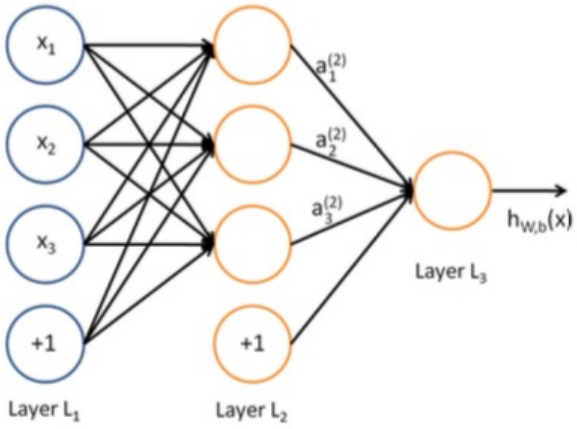
Frank Rosenblatt, ~1957: Perceptron

Last Time: Multilayer Perceptron – Forward Propagation

- Type of feedforward neural network
- Can model non-linear associations
- "Multi-level combination" of many perceptrons



Last Time: Multilayer Perceptron – Forward Propagation



$$\alpha_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$\alpha_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$\alpha_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \alpha_1^{(3)} = f(W_{11}^{(2)}\alpha_1^{(2)} + W_{12}^{(2)}\alpha_2^{(2)} + W_{13}^{(2)}\alpha_3^{(2)} + b_1^{(2)})$$

Terminology

$W_{ij}^{(l)}$: connection between unit j in layer l to unit i in layer $l+1$

$\alpha_i^{(l)}$: activation of unit i in layer l

$b_i^{(l)}$: bias connected with unit i in layer $l+1$

Forward propagation: The process of propagating the input to the output through the activation of inputs and hidden units to each node

Last Time: Multilayer Perceptron – Backpropagation

Gradient descent for regression

$$J(\mathbf{W},\mathbf{b}) = \frac{1}{N}\sum_{n=1}^{M}\frac{1}{2}\|h_{\mathbf{W},\mathbf{b}}(\mathbf{x_n}) - y_n\|_2^2 + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2$$

$$W_{ij}^{(l)} := W_{ij}^{(l)} - \alpha\frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta W_{ij}^{(l)}}$$
$$b_i^{(l)} := b_i^{(l)} - \alpha\frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta b_i^{(l)}}$$

Note: Initialize the parameters randomly $\rightarrow$ symmetry breaking

Use backpropagation to compute partial derivatives $\frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta W_{ij}^{(l)}}$ and $\frac{\vartheta J(\mathbf{W},\mathbf{b})}{\vartheta b_i^{(l)}}$

Last Time: Multilayer Perceptron – Optimization

SGD: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

Momentum: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \mathbf{v}$

Nesterov: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}), \mathbf{y}^{(i)}) \right)$ then $\theta \leftarrow \theta + \mathbf{v}$

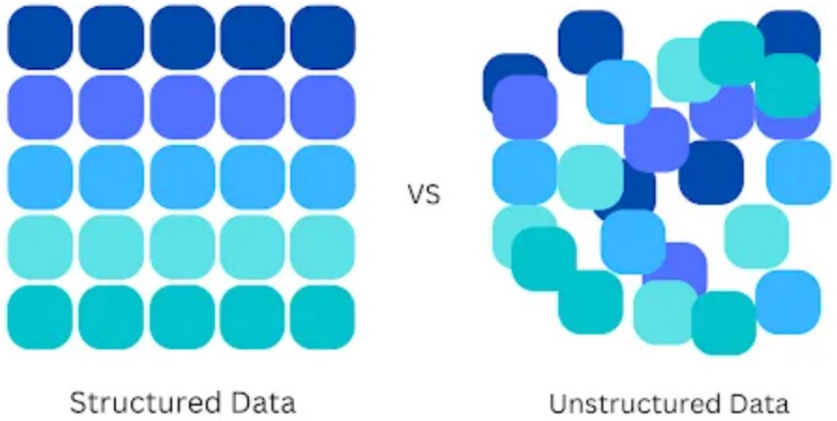
AdaGrad: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$ then $\Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ then $\theta \leftarrow \theta + \Delta\theta$

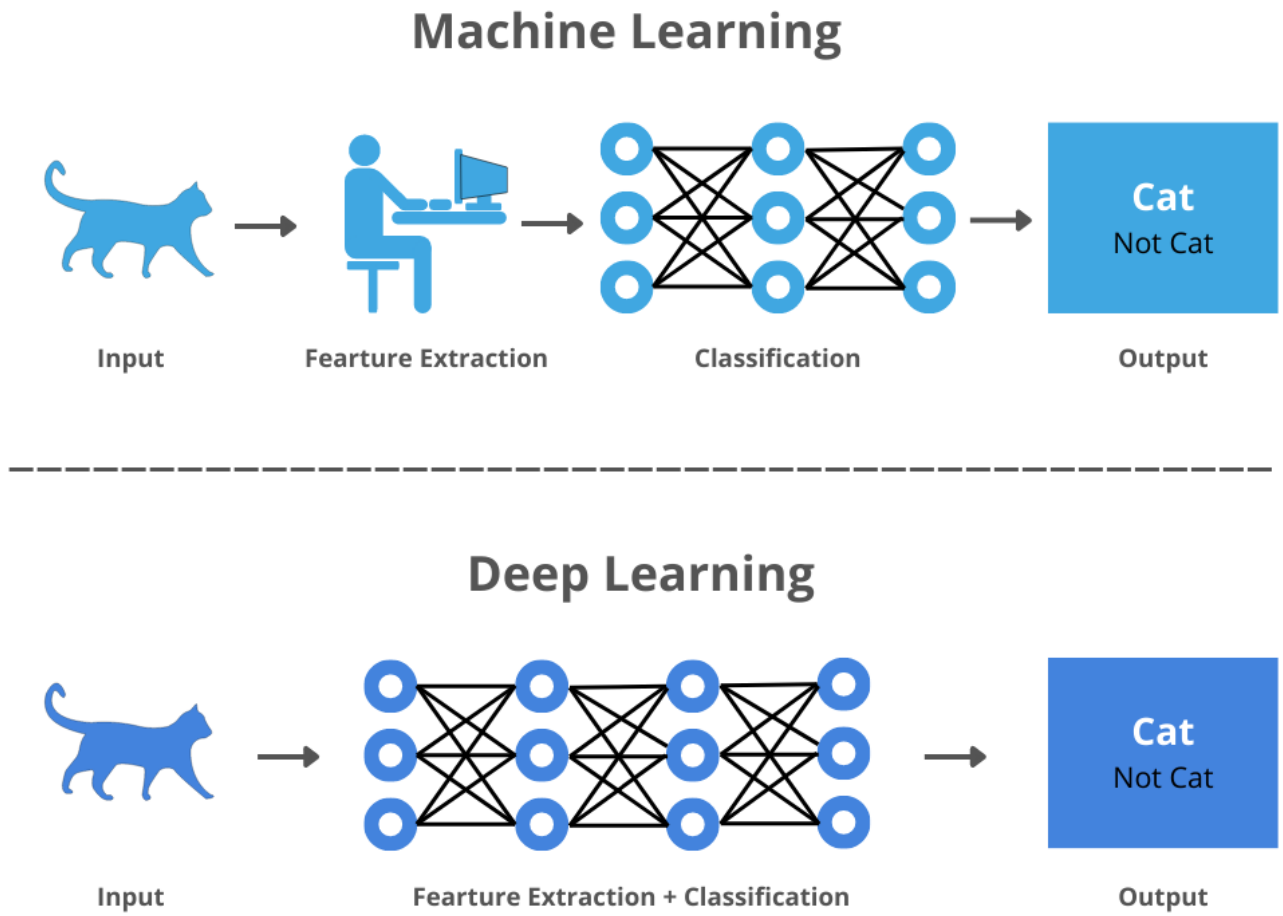
RMSProp: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

Adam: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$ then $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$ then $\theta \leftarrow \theta + \Delta\theta$

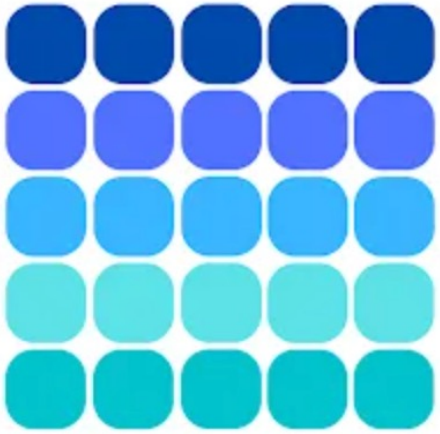
Challenge

Challenge


VS
Structured Data
Unstructured Data


Machine Learning
Cat
Not Cat
Input
Fearture Extraction
Classification
Output
Deep Learning
Cat
Not Cat
Input
Fearture Extraction + Classification
Output

Challenge



Structured Data

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10

Challenge: unstructured data

Unstructured Data

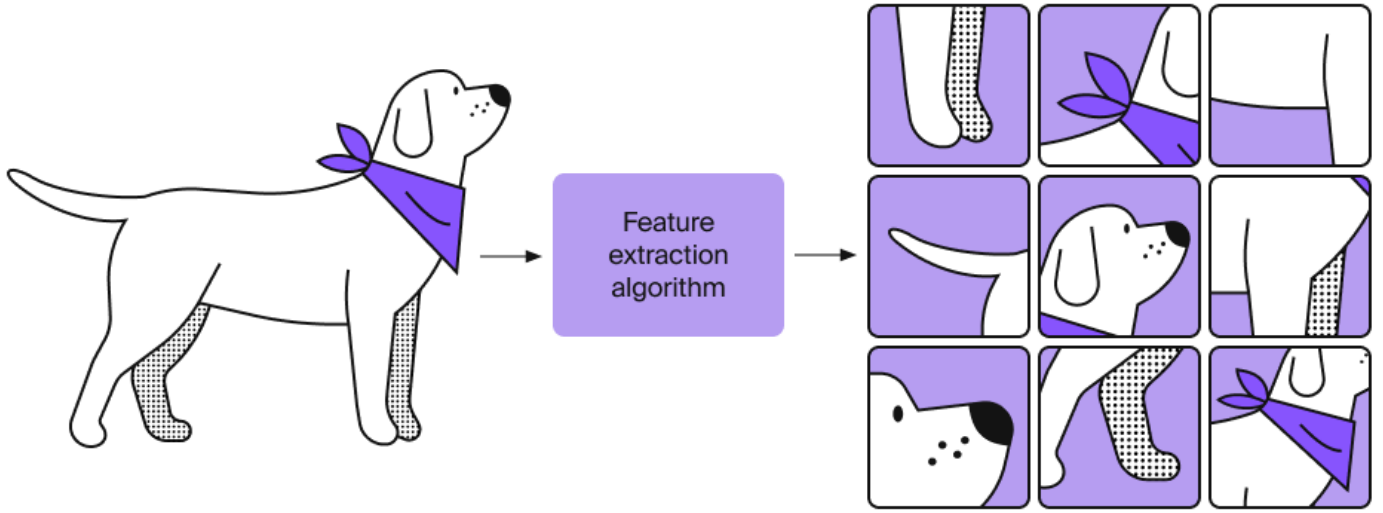
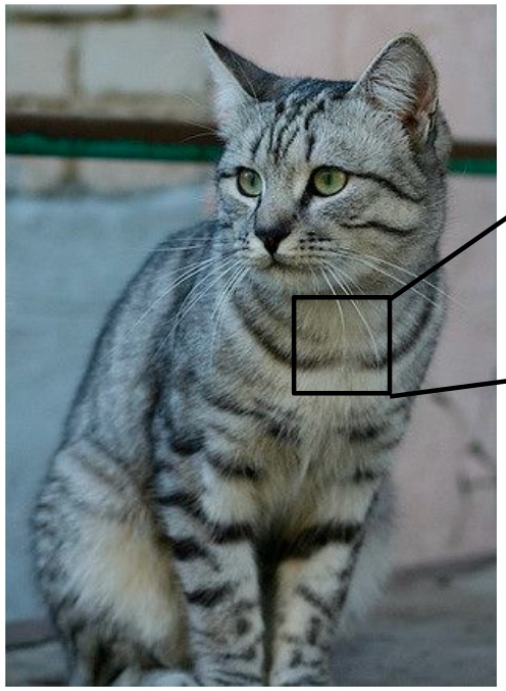


Image Classification: A core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

cat

Image Classification: A core task in Computer Vision




```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

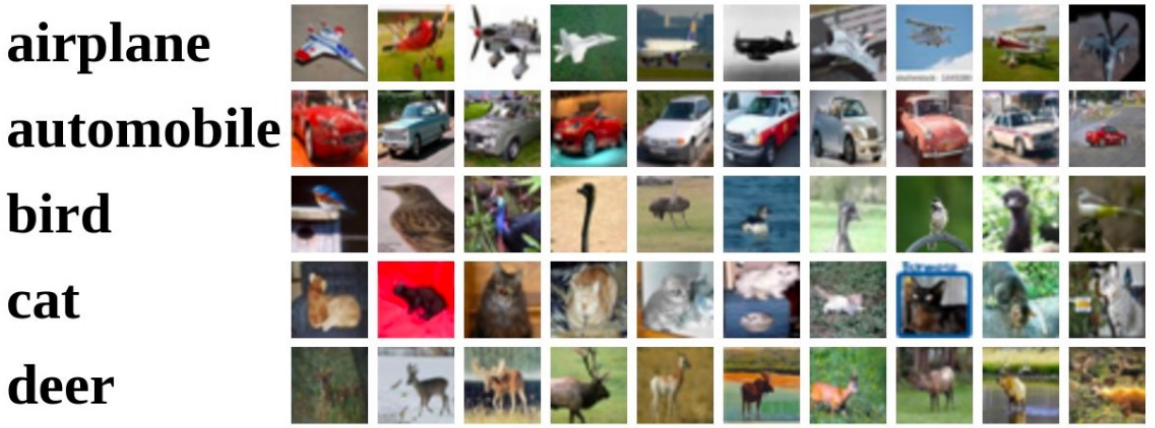
Machine Learning: Data-Driven Approach:

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new image

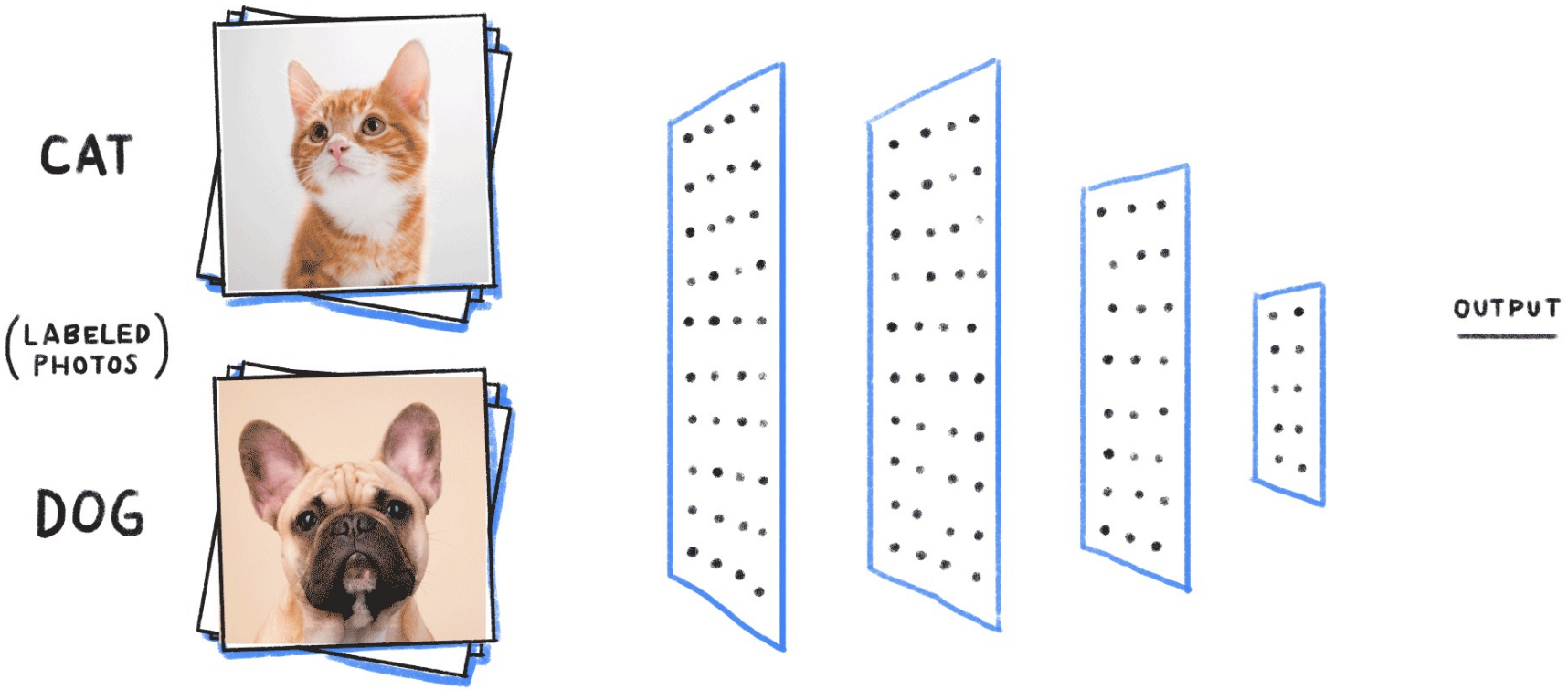
```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

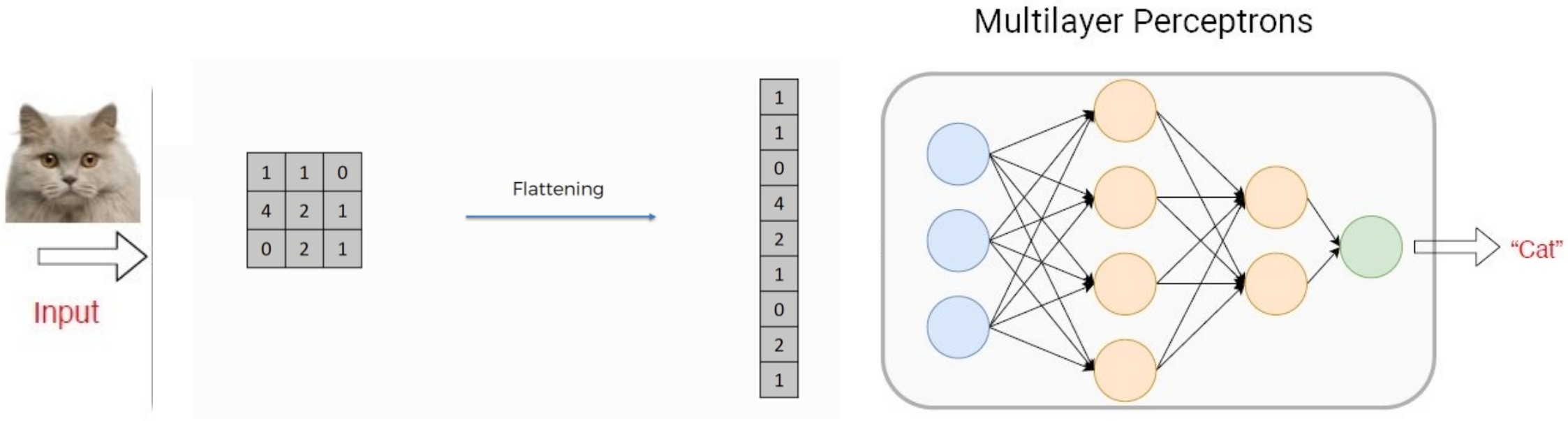
Example training set



Challenge: image



Challenge: image



Solution: Convolutional Neural Networks

Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]

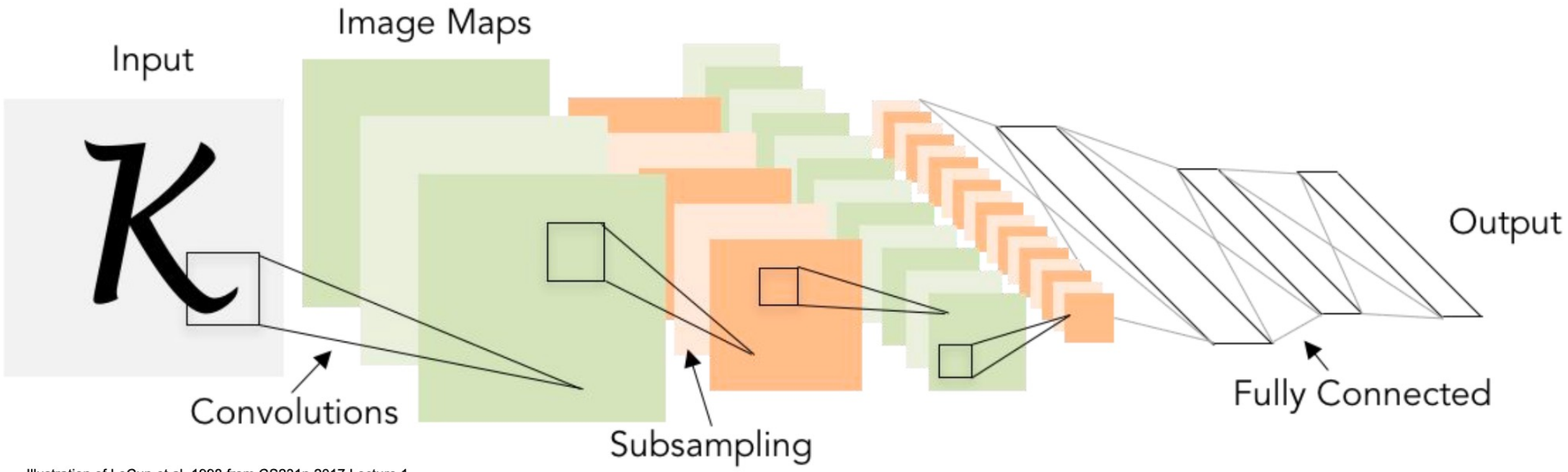
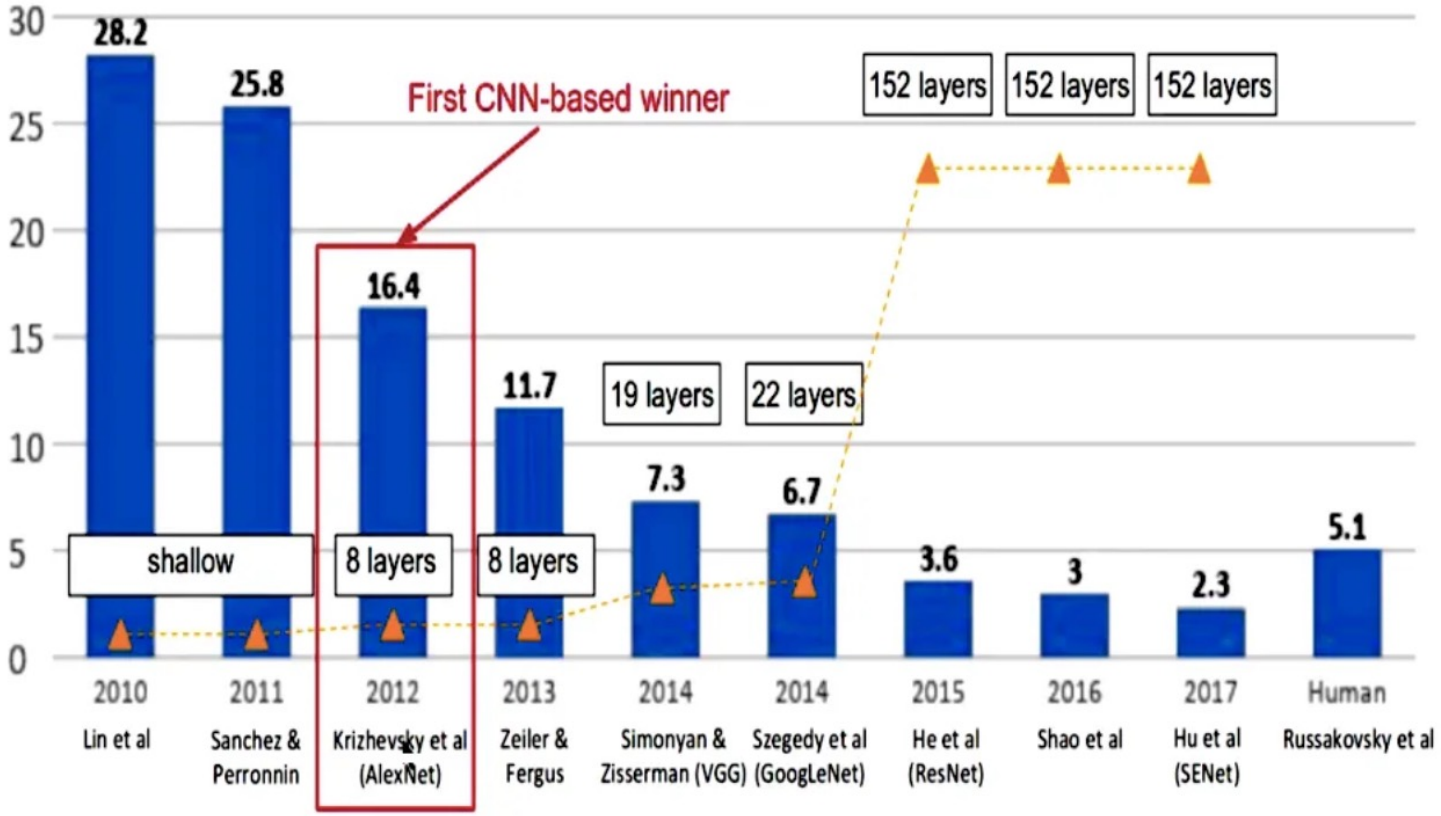


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

A bit of history…

Winners of ImageNet Classification Challenge



A bit of history…

- **ImageNet Classification with Deep Convolutional Neural Networks** *[Krizhevsky, Sutskever, Hinton, 2012]*

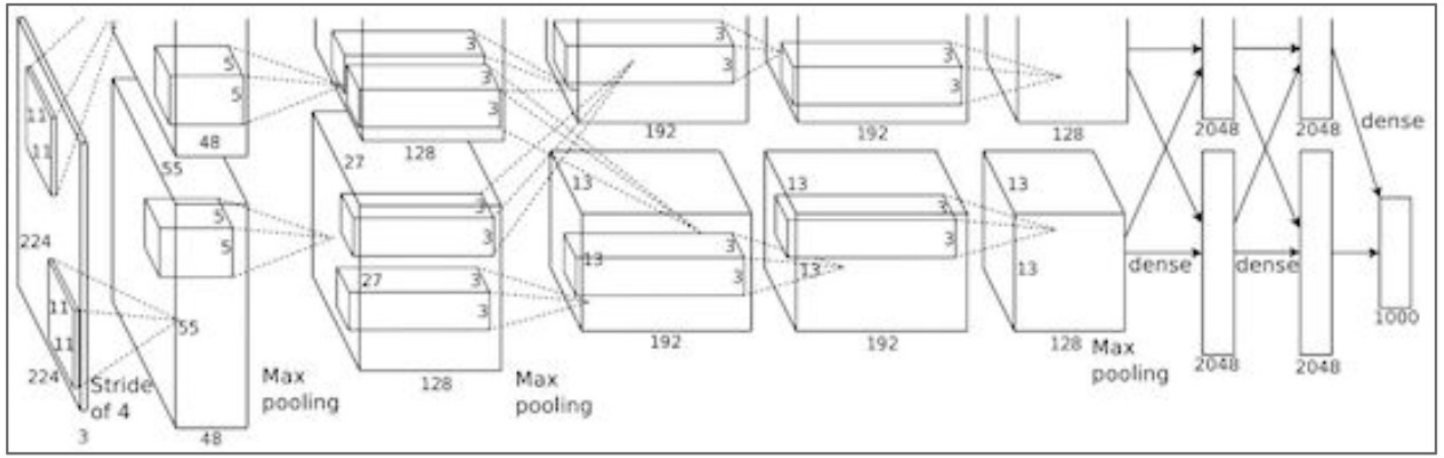
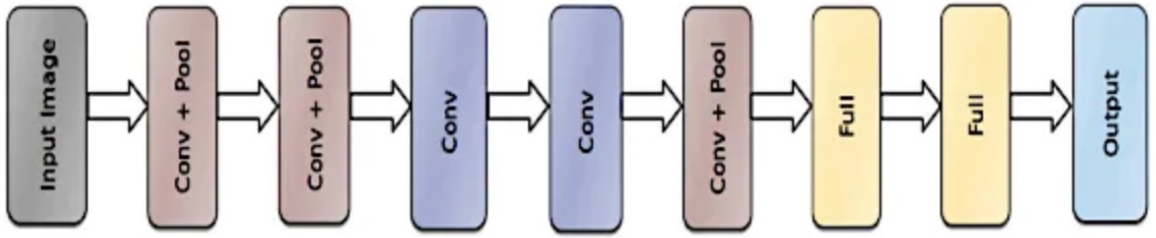
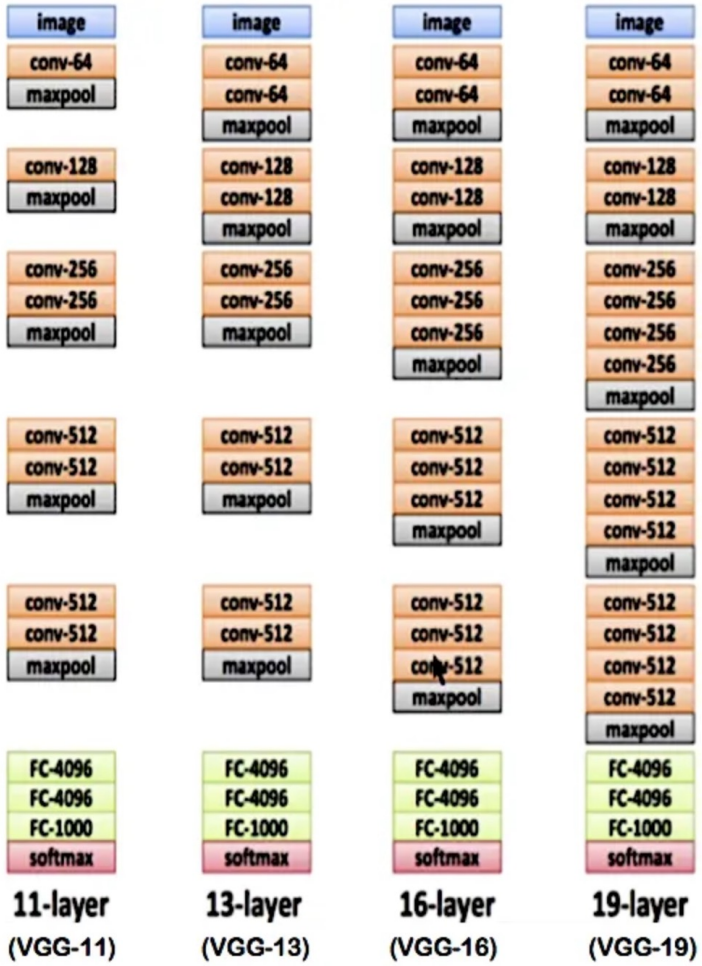


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"



A bit of history…



Moving on to 2014, one of the major contributions 2014 witnessed was introduction of a new architecture known as the VGGNet (paper).

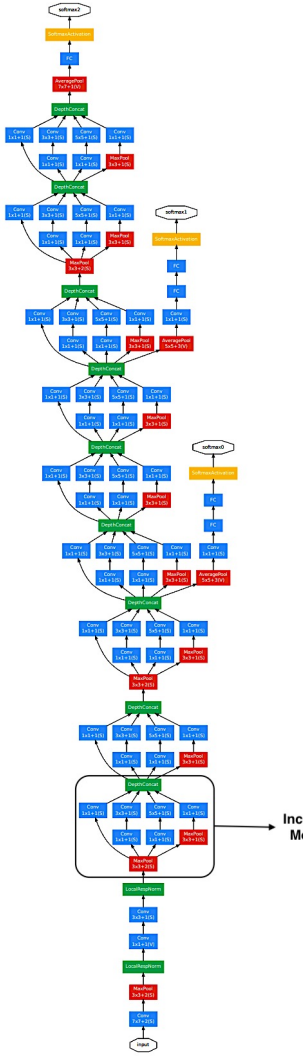
The VGGNet, stands for an (arcade) architecture, invented by Visual Geometry Group (at **Oxford University**).

It was argued that by making CNN deeper, one can solve problems better and get a lower error rate on the ImageNet classification challenge.

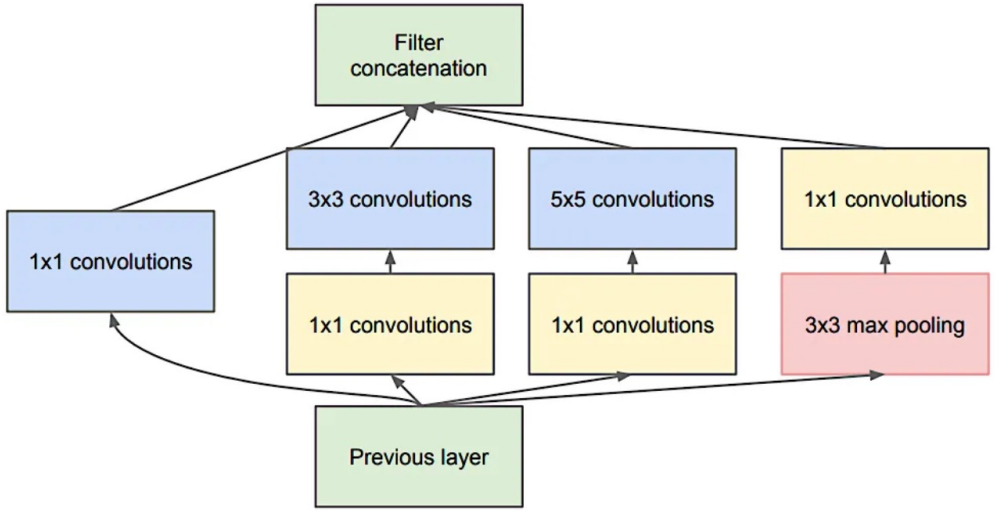
The Group worked on the philosophy that by **increasing the depth**, one can model more non-linearities in one's function and hence the key contribution was consideration of depth as a critical component in design.

The architecture **won the runner-up in the ImageNet challenge** in 2014. Notably VGG-16 (13 Conv plus 3 FC layers) consists of **138M parameters** and has a significant memory overhead of 48.6 MB (For a quick comparison AlexNet has 1.9 MB).

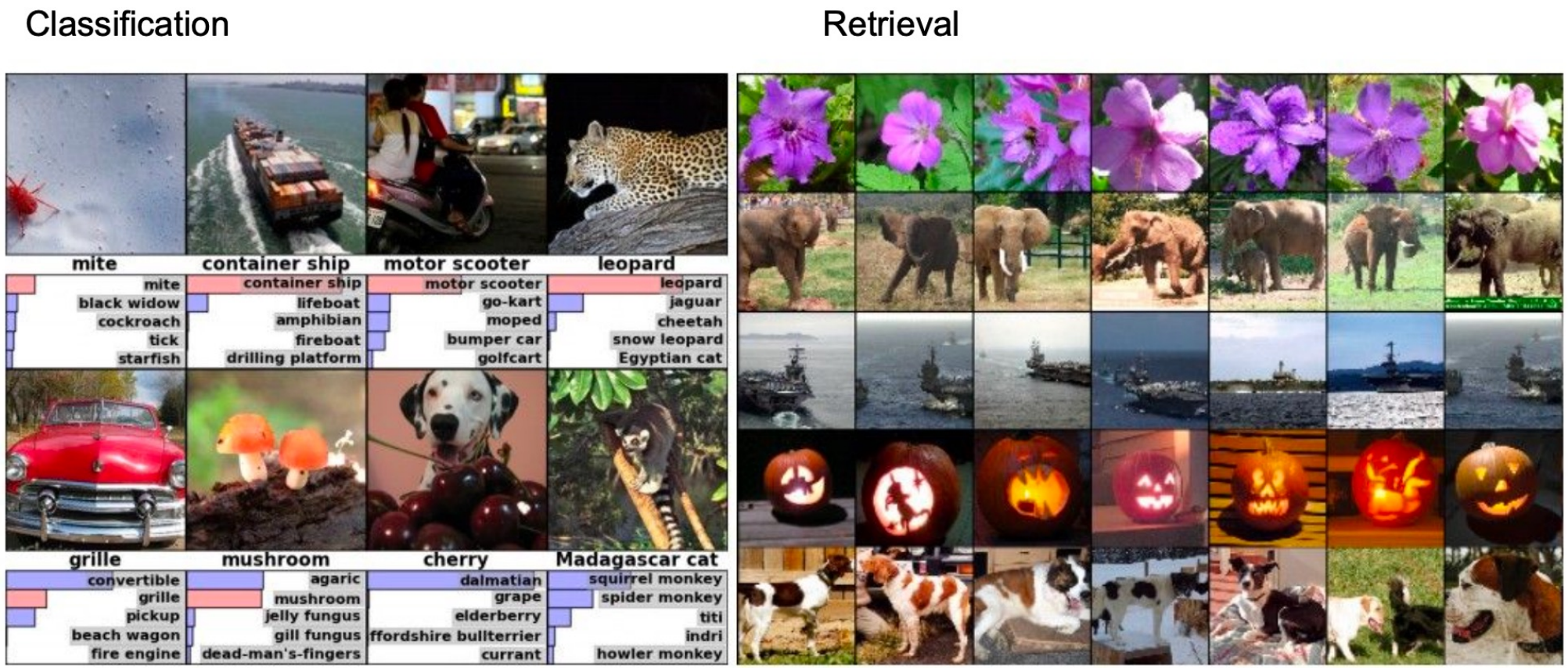
A bit of history…



In 2014, Google introduced GoogLeNet (paper). GoogLeNet again focused on deeper networks but with the objective of **greater efficiency to reduce parameter count**, **memory usage**, and **computation**.
It went deeper up to 22 layers but without any fully connected (FC) layers. By getting rid of FC layers, the total number of **parameters reduced to 5M (12x lesser than AlexNet and around 28x lesser than VGG)**.
A module named 'inception module' was introduced. GoogleNet became ILSVRC-14 classification winner (with 6.7% top-5 error).



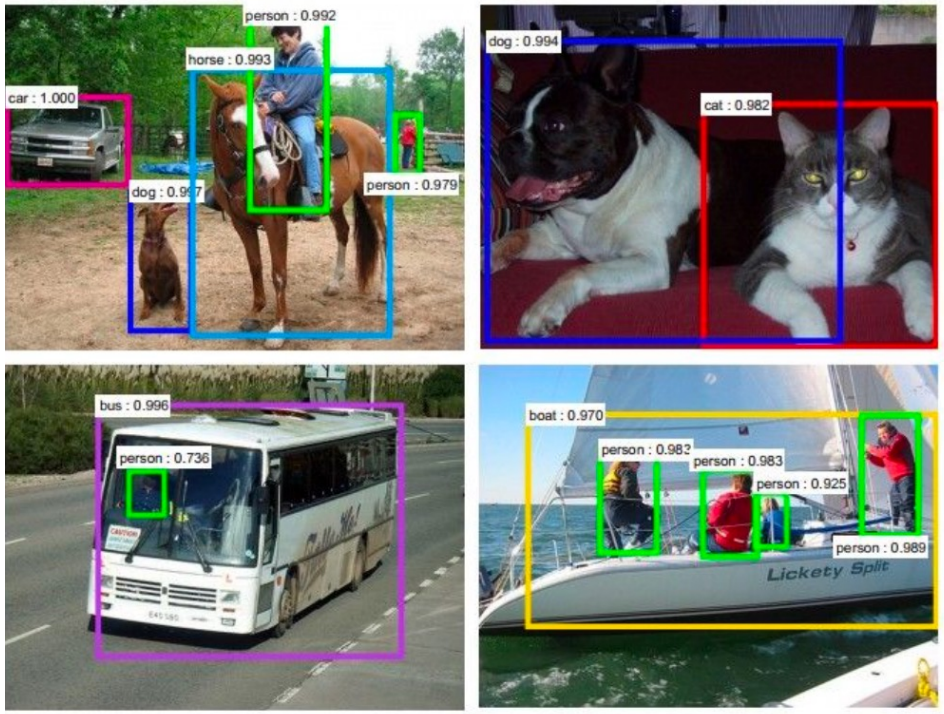
Fast-forward to today: ConvNets are everywhere



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

Fast-forward to today: ConvNets are everywhere

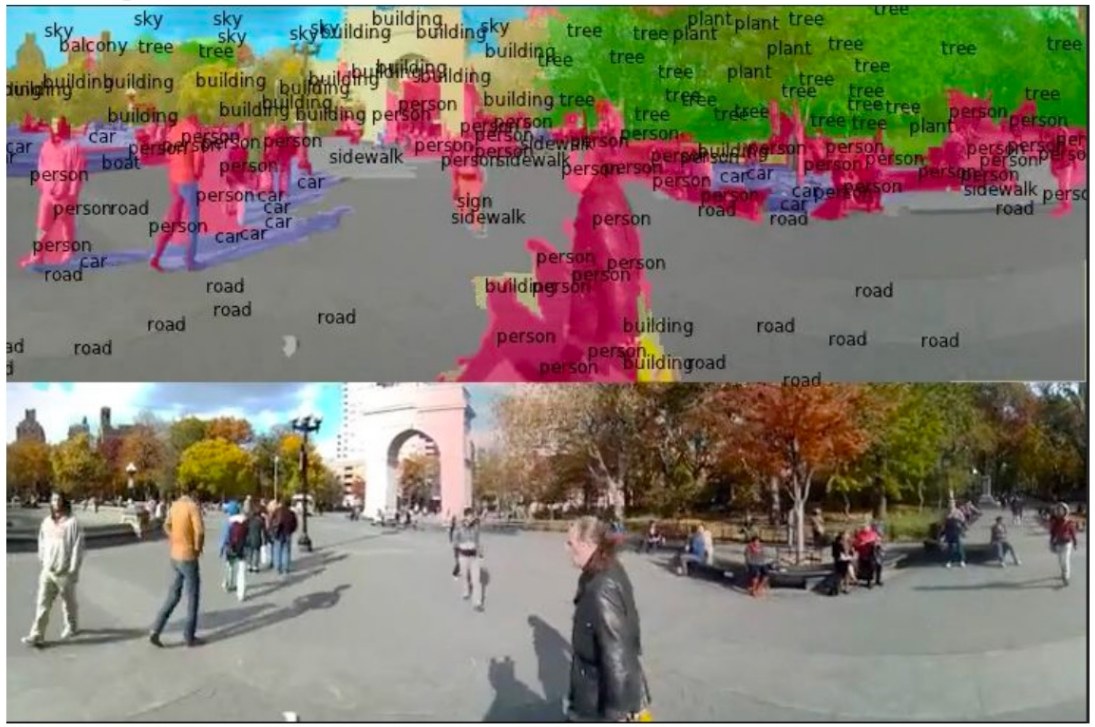
Detection



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

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

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

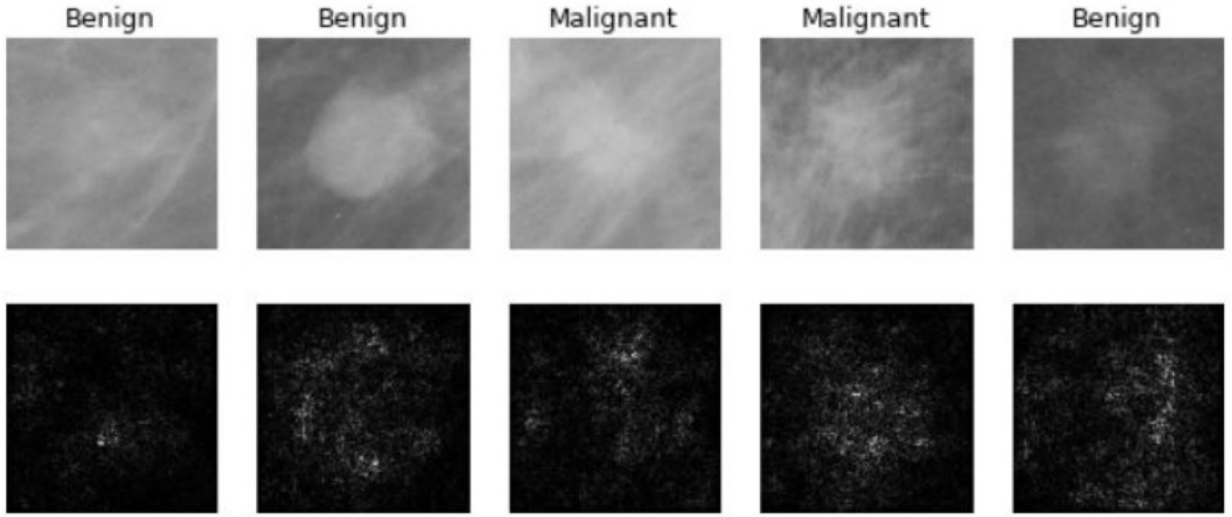
Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

Fast-forward to today: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]



[Levy et al. 2016]

Figure copyright Levy et al. 2016.
Reproduced with permission.

Fast-forward to today: Image Captioning

No errors

A white teddy bear sitting in the grass

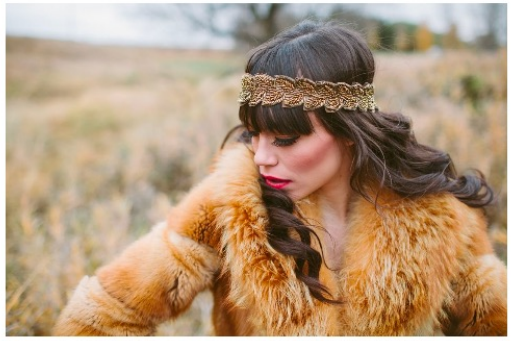
A man riding a wave on top of a surfboard

Minor errors

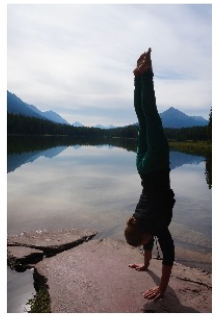
A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related



A woman is holding a cat in her hand

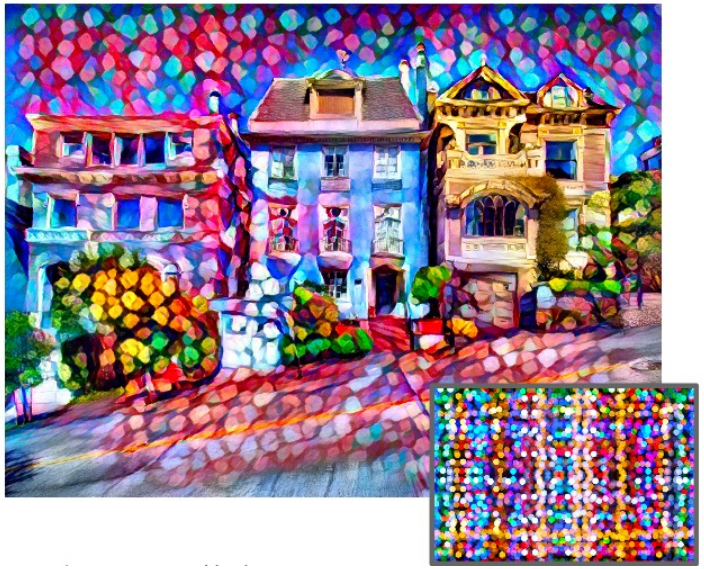


A woman standing on a beach holding a surfboard

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

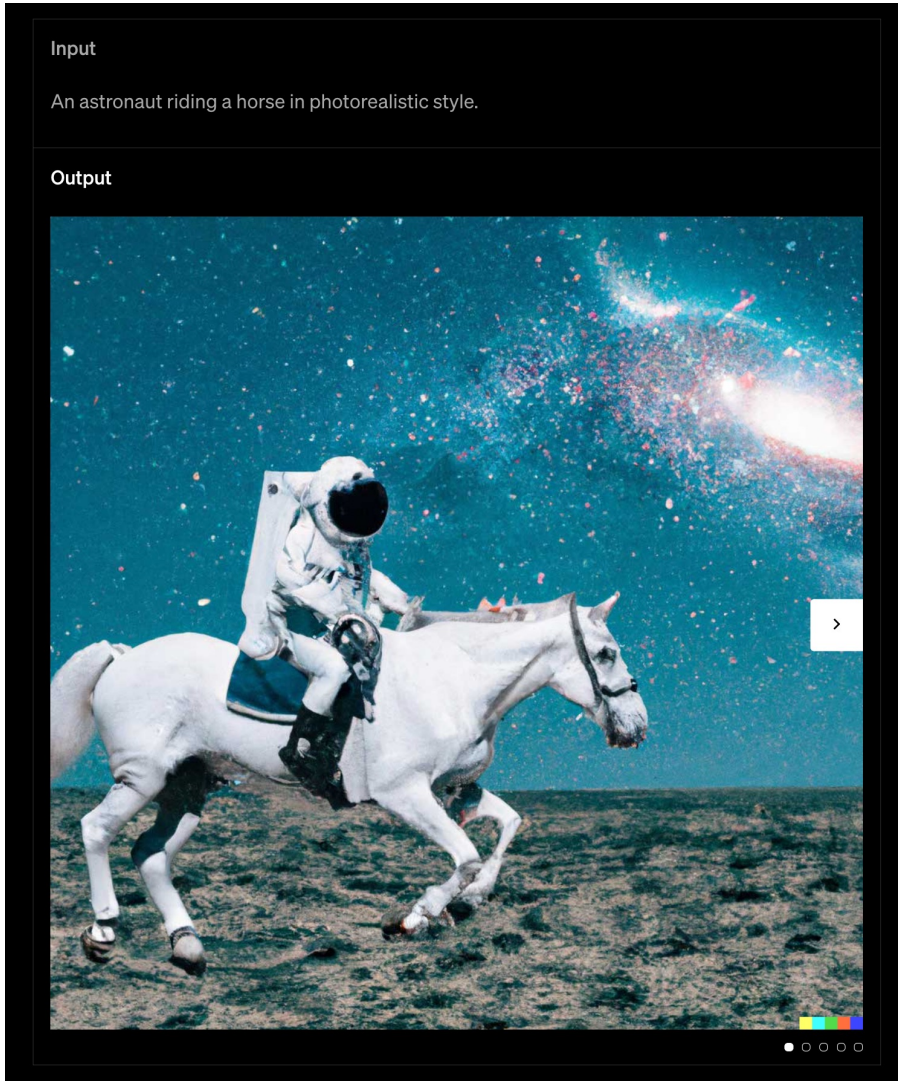
ll images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/ https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/ https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/ https://pixabay.com/en/woman-female-model-portrait-adult-983967/ https://pixabay.com/en/handstand-lake-meditation-496008/ https://pixabay.com/en/baseball-player-shortstop-infield-1045263/
Captions generated by Justin Johnson using Neuraltalk2

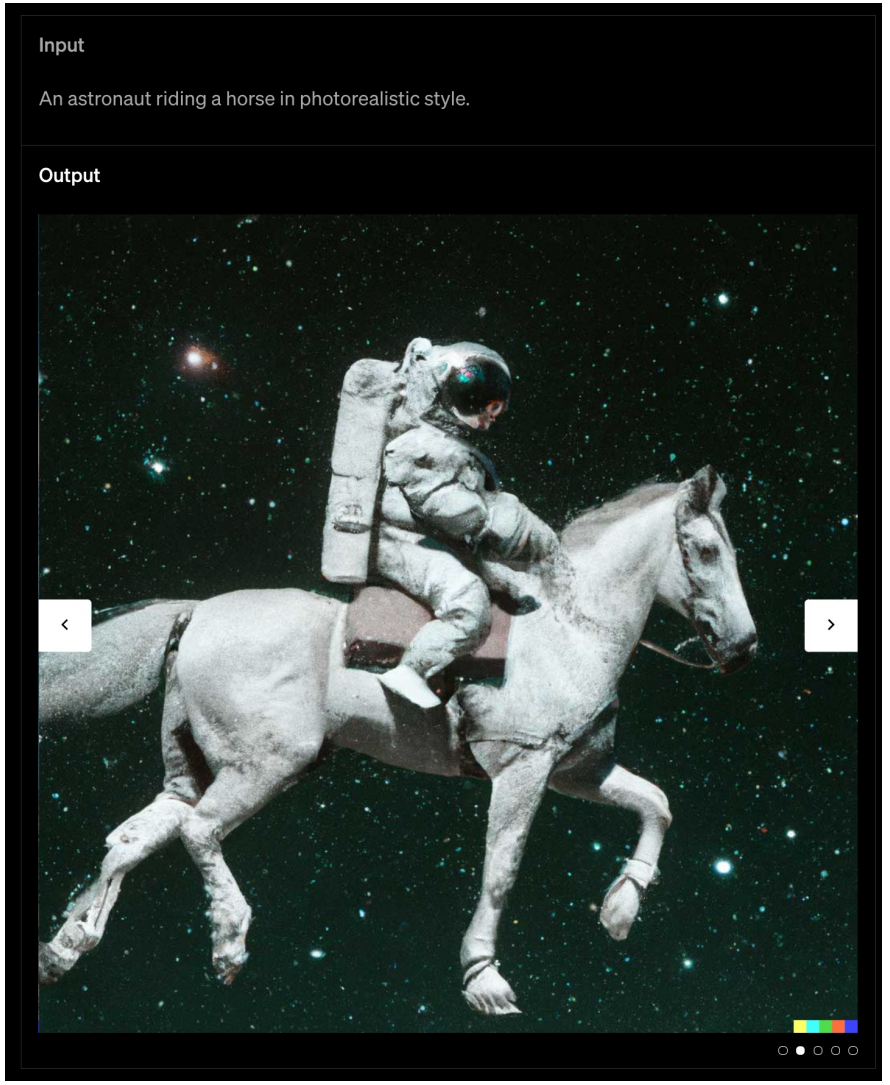
Fast-forward to today: Style Transfer



Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission
Gatys et al, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016
Gatys et al, “Controlling Perceptual Factors in Neural Style Transfer”, CVPR 2017

Fast-forward to today: Text2Image - OpenAI DALL-E





https://openai.com/dall-e-2

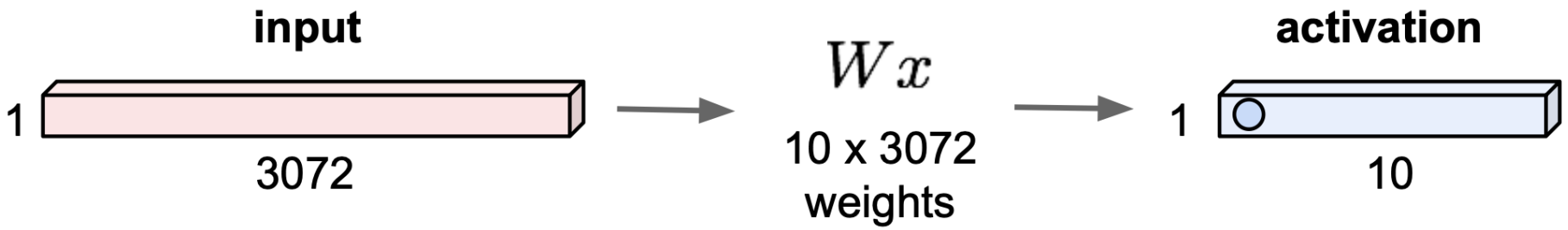
Fast-forward to today: Text2Video – OpenAI Sora

Prompt: Animated scene features a close-up of a short fluffy monster kneeling beside a melting red candle. The art style is 3D and realistic, with a focus on lighting and texture. The mood of the painting is one of wonder and curiosity, as the monster gazes at the flame with wide eyes and open mouth. Its pose and expression convey a sense of innocence and playfulness, as if it is exploring the world around it for the first time. The use of warm colors and dramatic lighting further enhances the cozy atmosphere of the image.

https://openai.com/sora

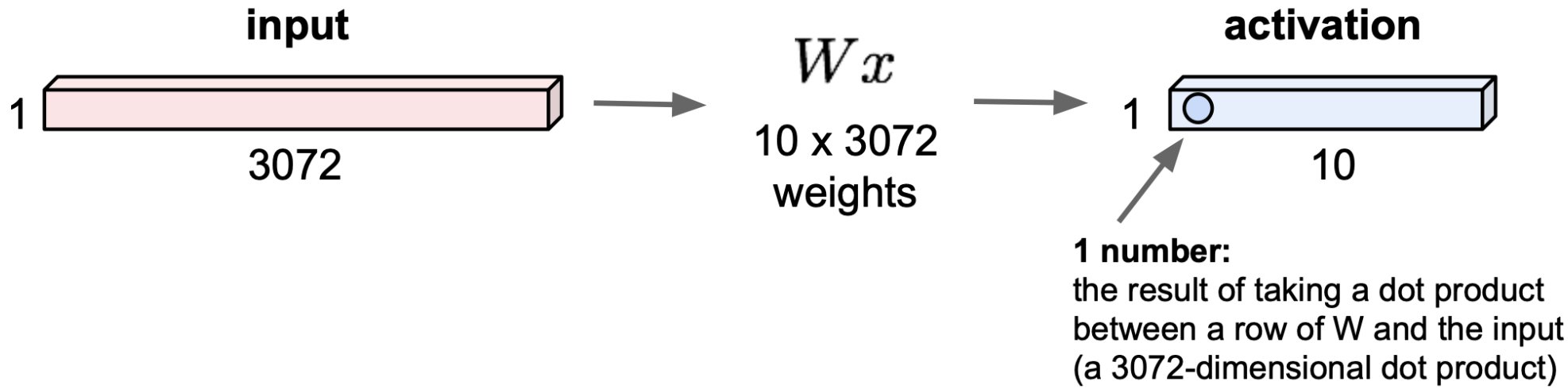
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



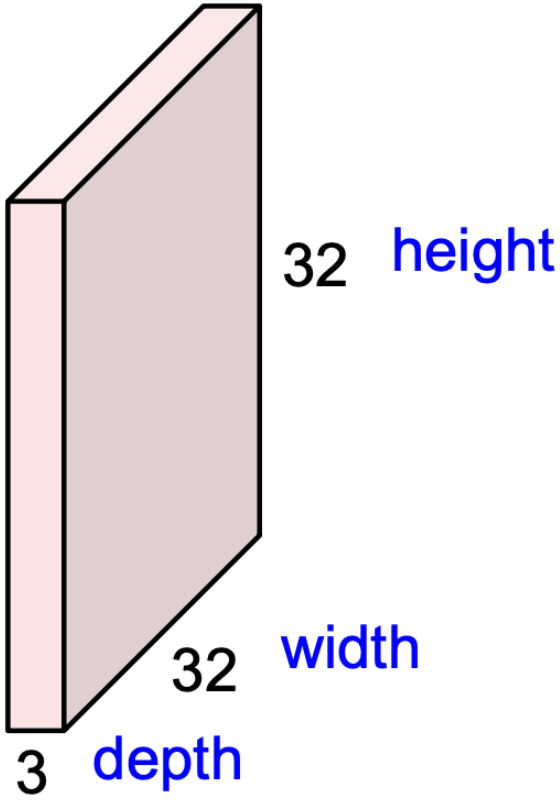
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



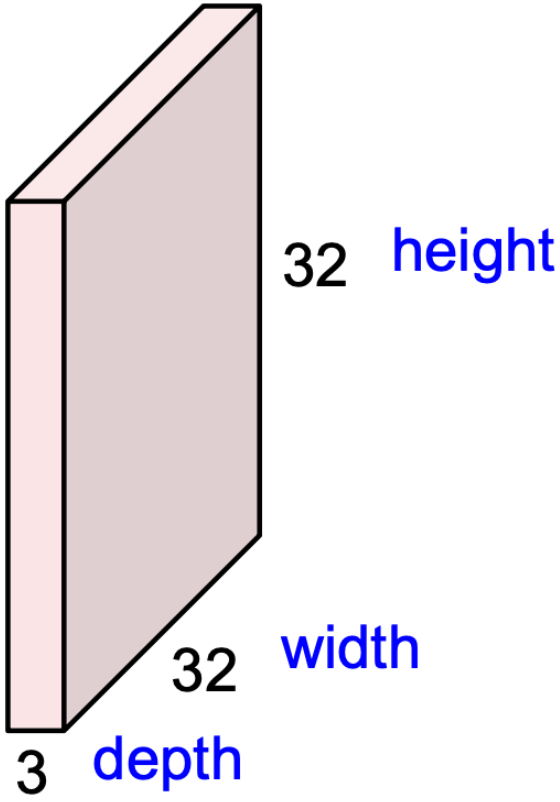
Convolution Layer

32x32x3 image -> preserve spatial structure

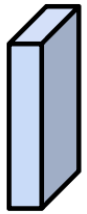


Convolution Layer

32x32x3 image -> preserve spatial structure



5x5x3 filter

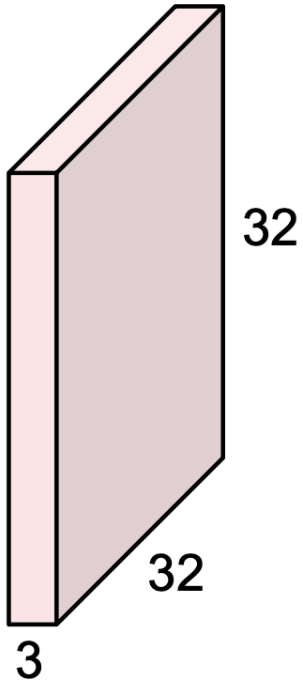


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

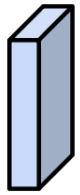
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

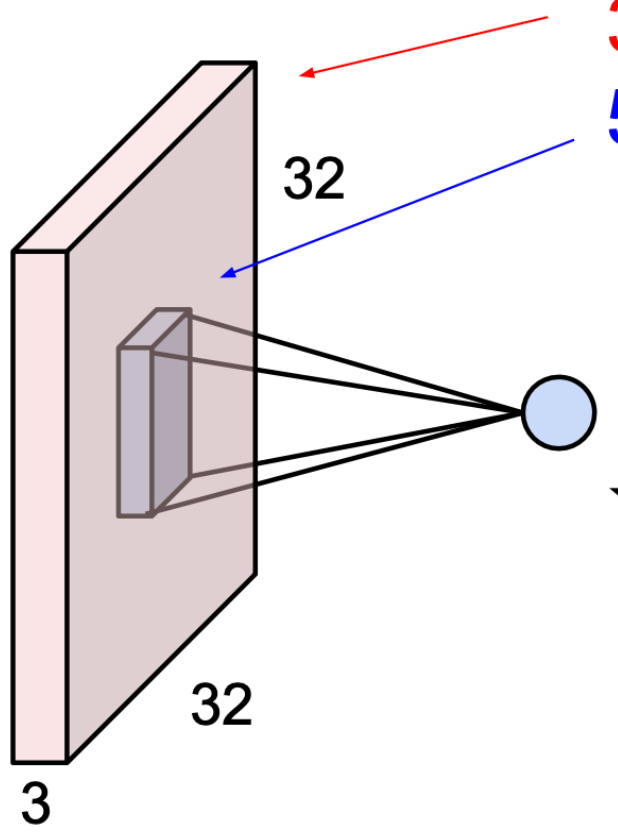


5x5x3 filter



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

Convolution Layer



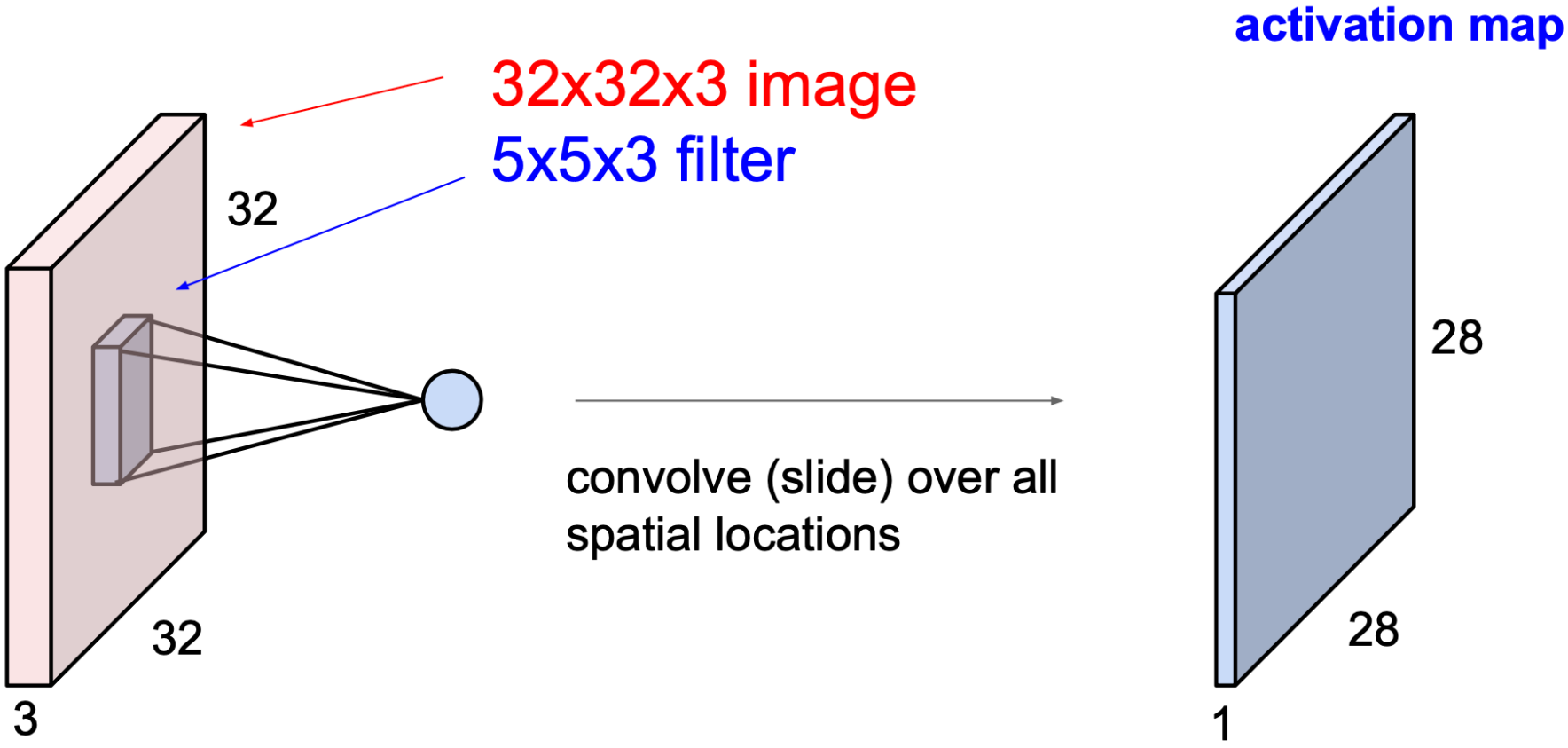
32x32x3 image

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)

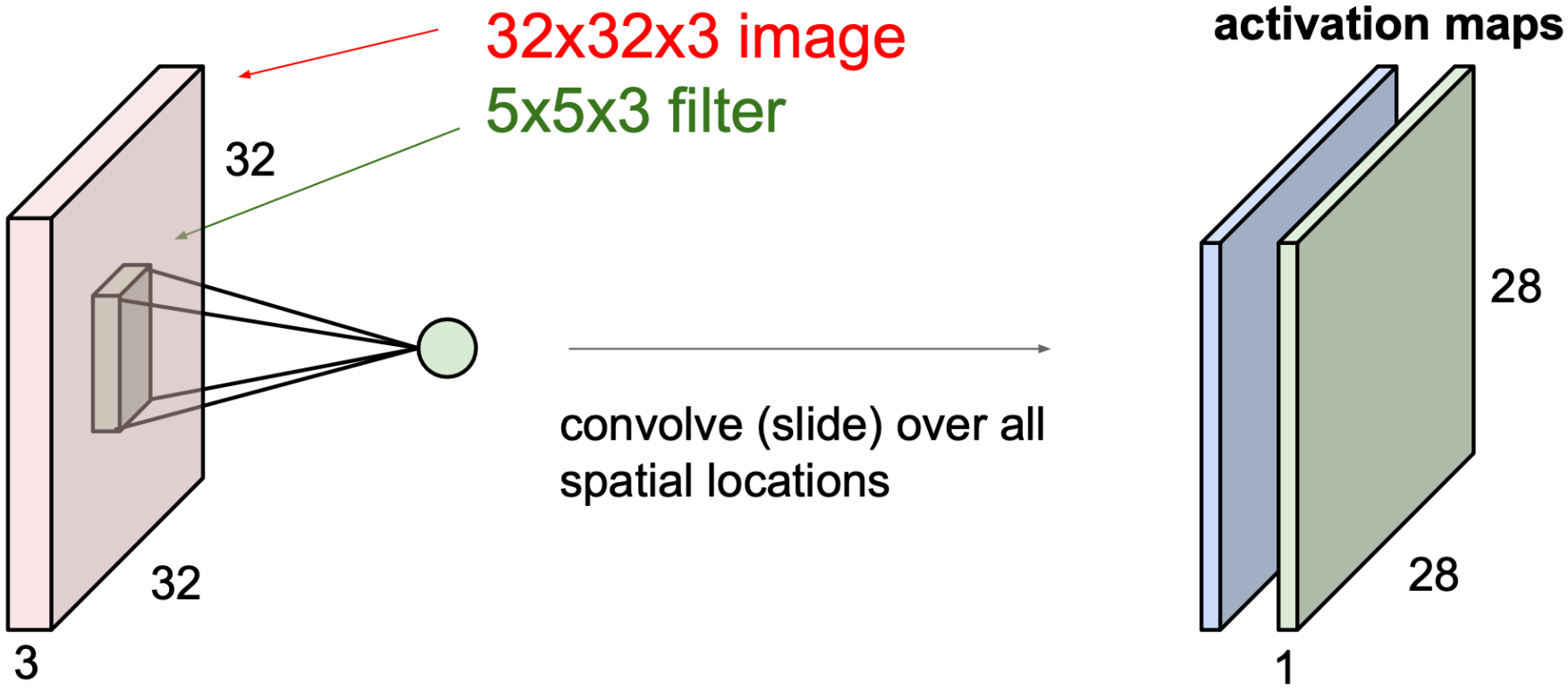
$$w^T x + b$$

Convolution Layer



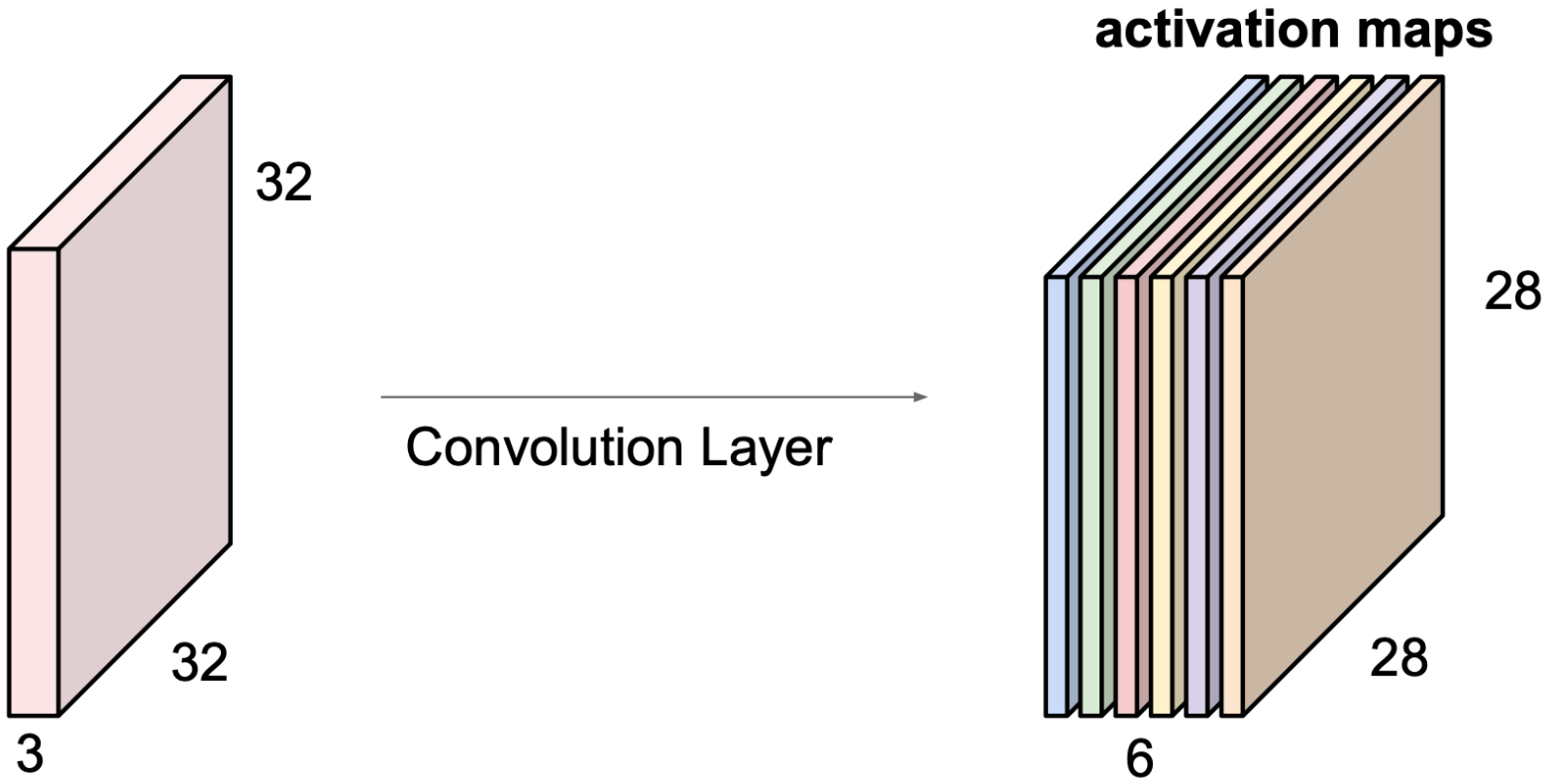
Convolution Layer

consider a second, green filter



Convolution Layer

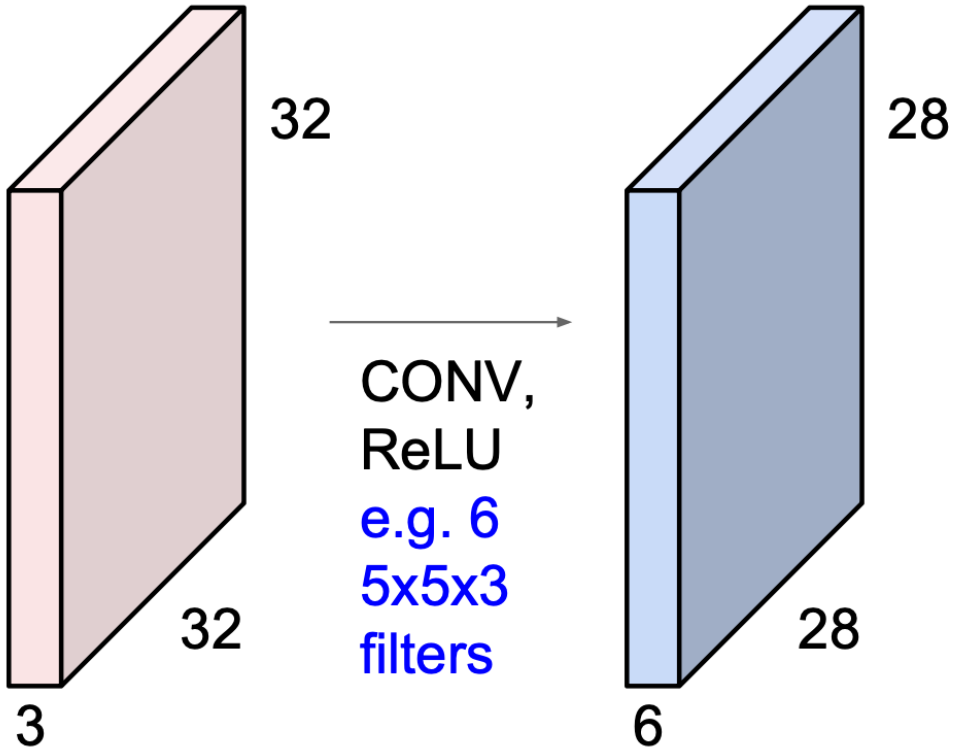
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

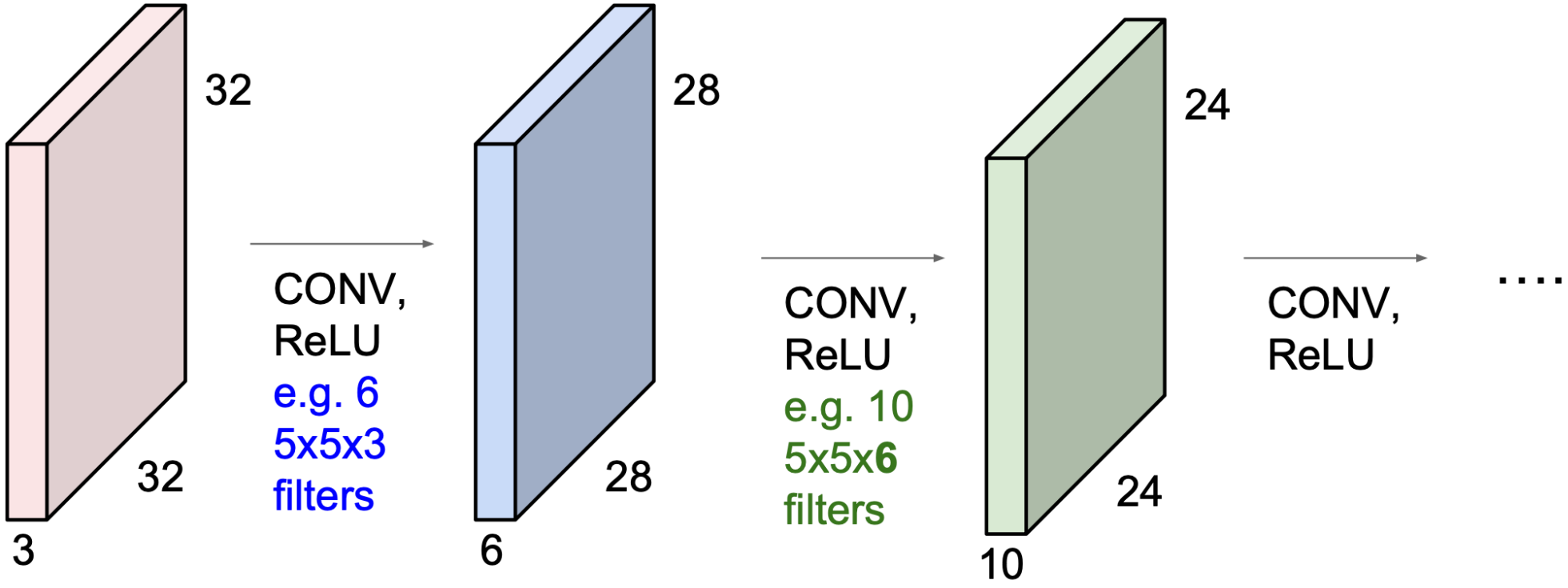
Convolution Layer

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



Convolution Layer

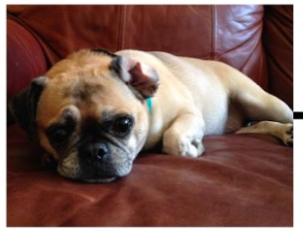
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Low-level to High-level feature extraction

[Zeiler and Fergus 2013]

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

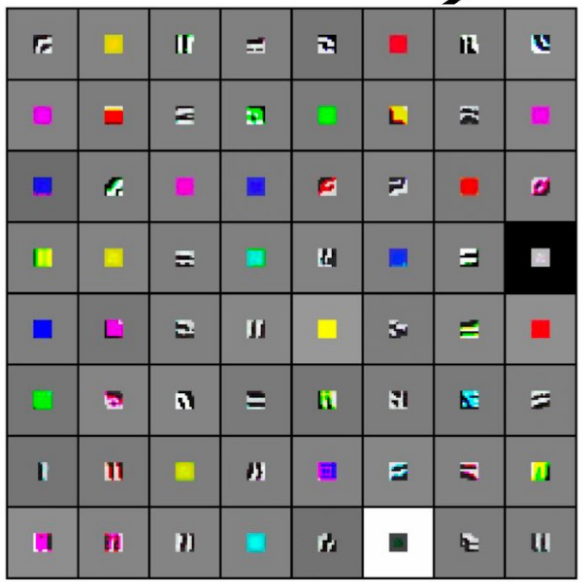


Low-level features

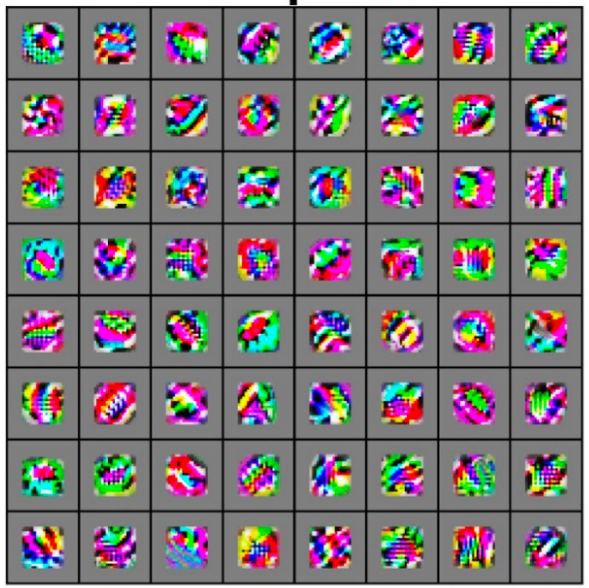
Mid-level features

High-level features

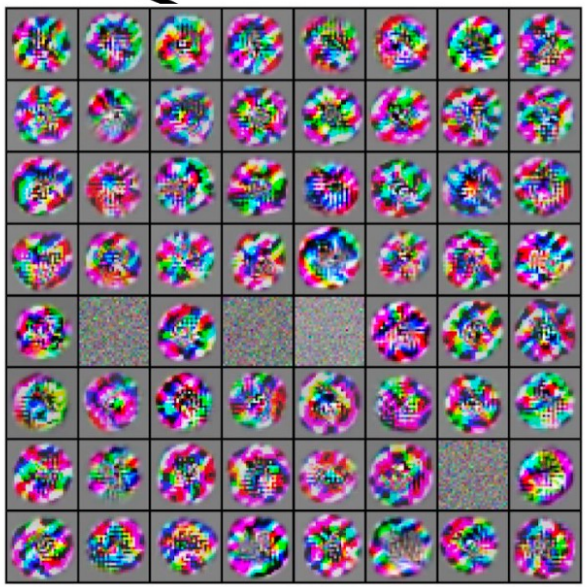
Linearly separable classifier



VGG-16 Conv1_1

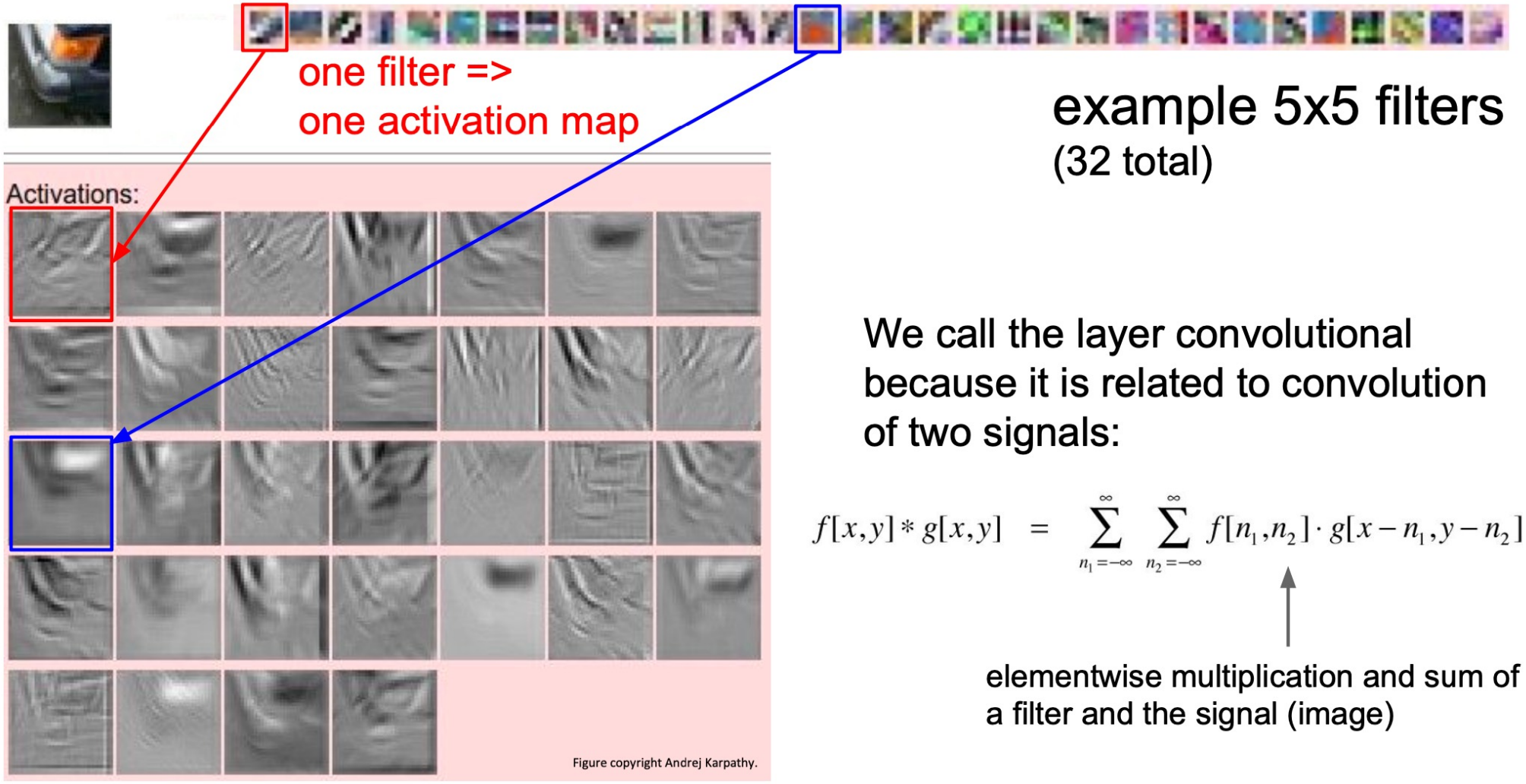


VGG-16 Conv3_2



VGG-16 Conv5_3

Why Convolution?



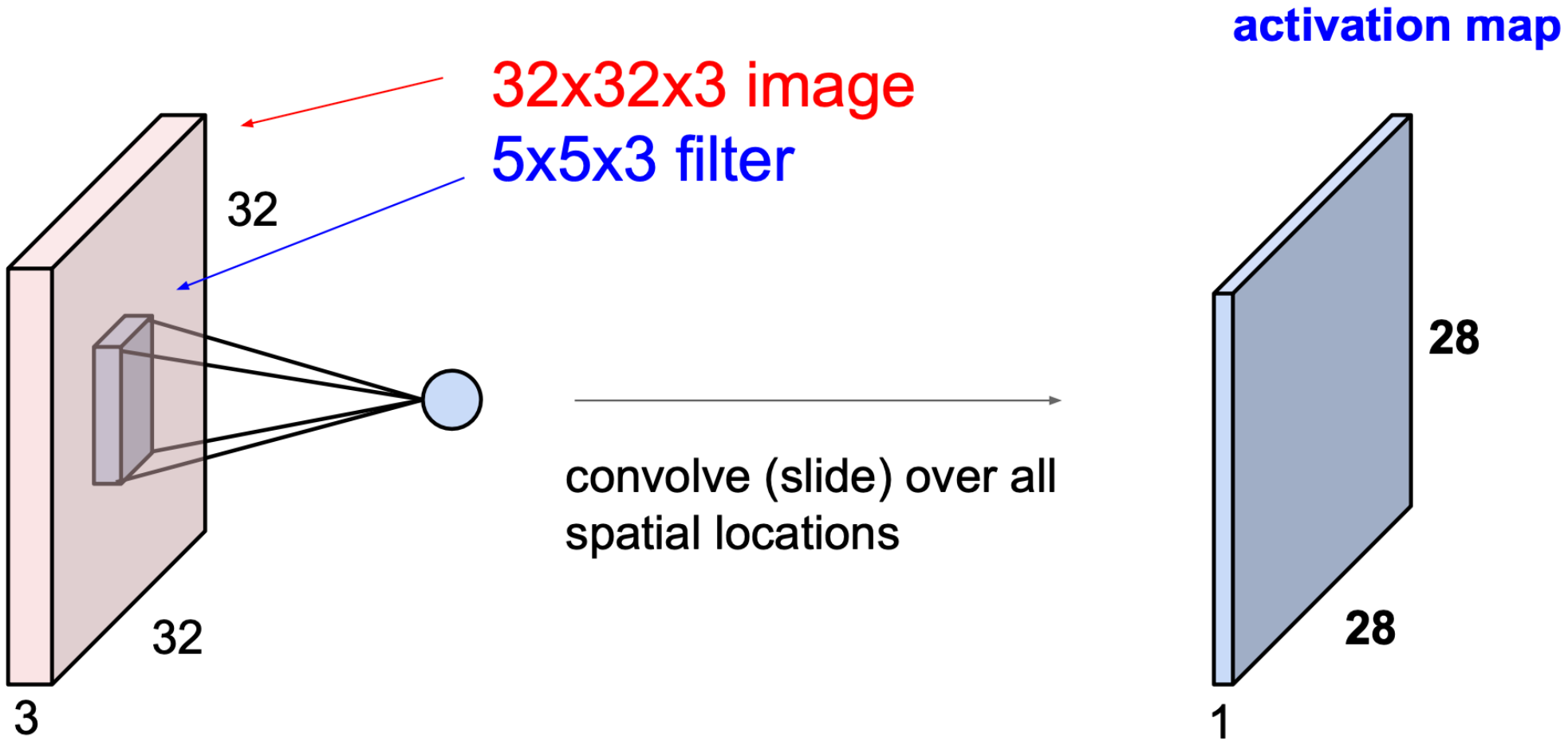
example 5x5 filters
(32 total)

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

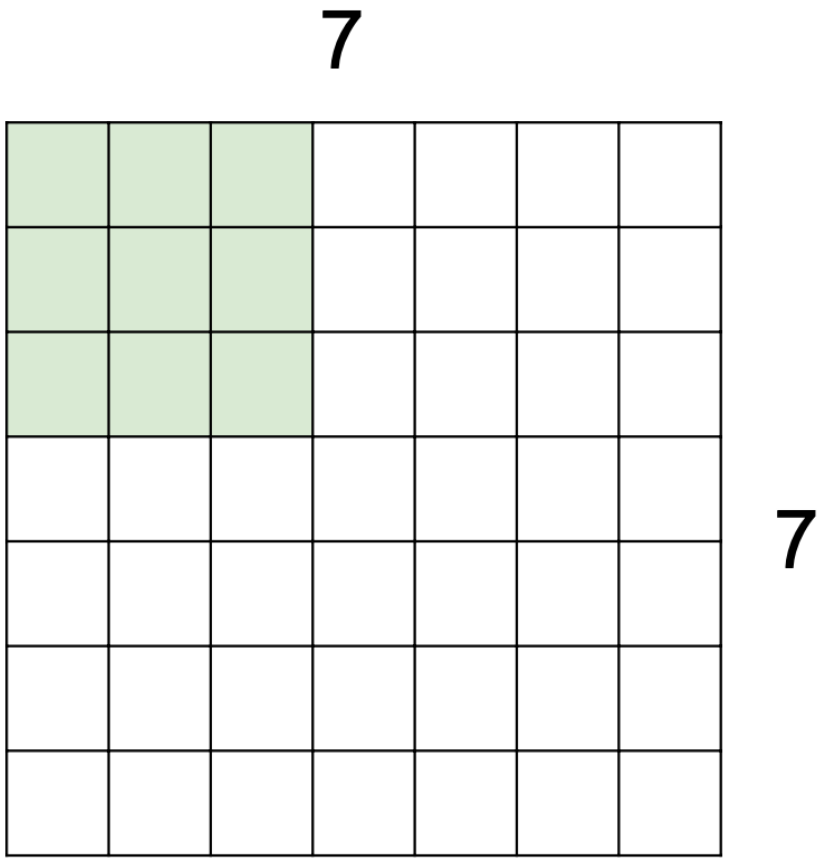
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

A closer look at spatial dimensions:

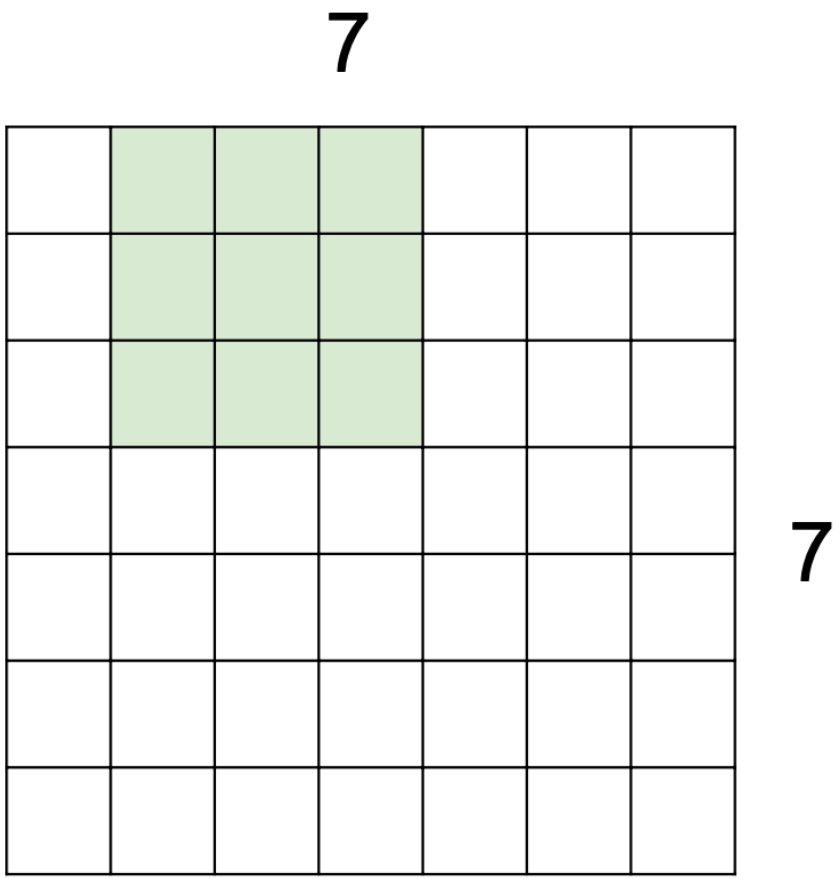


A closer look at spatial dimensions:



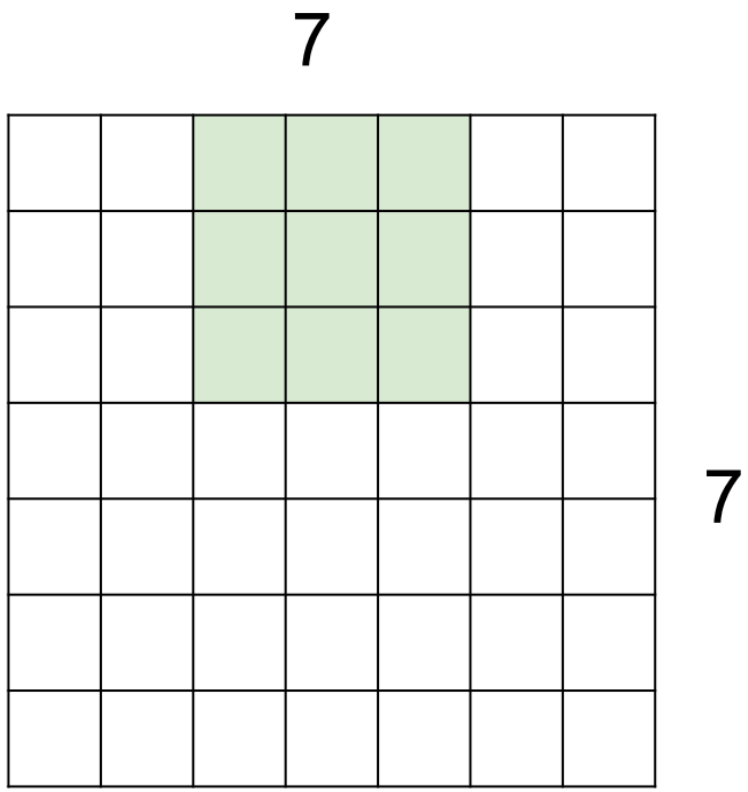
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



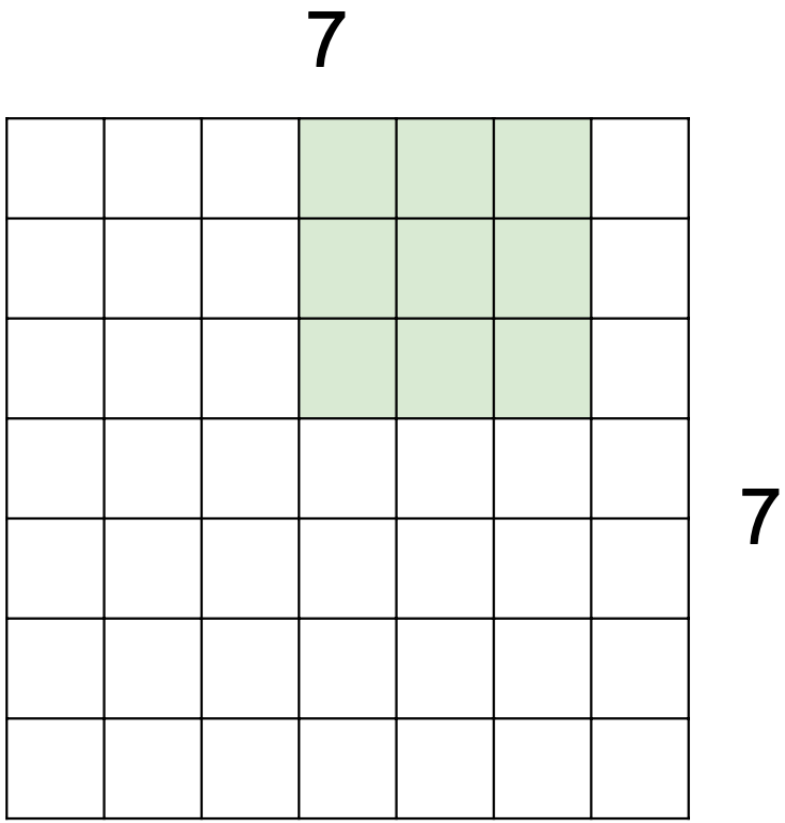
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



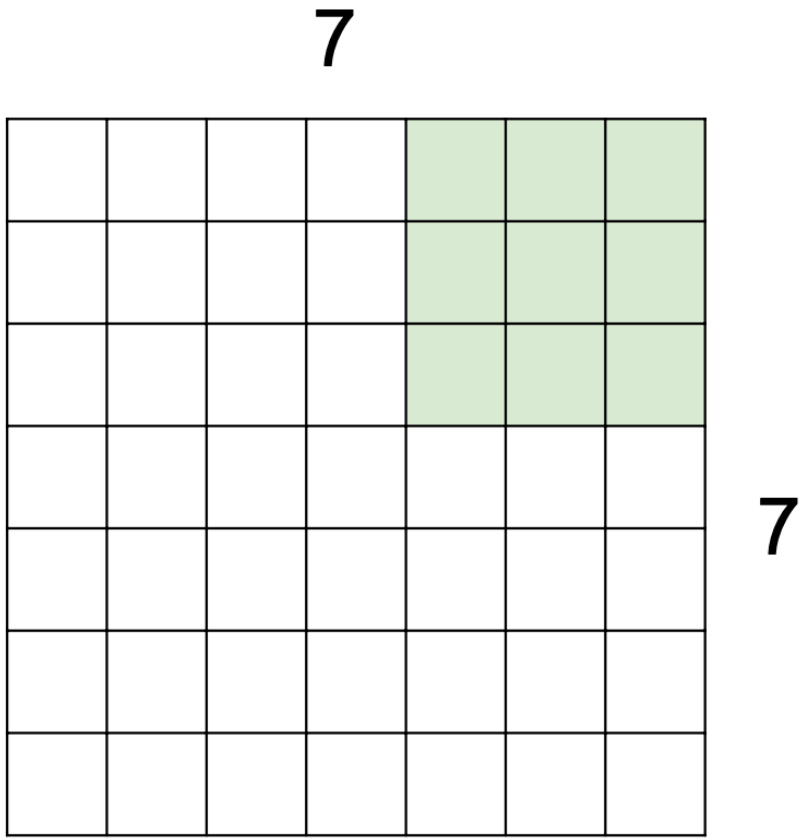
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

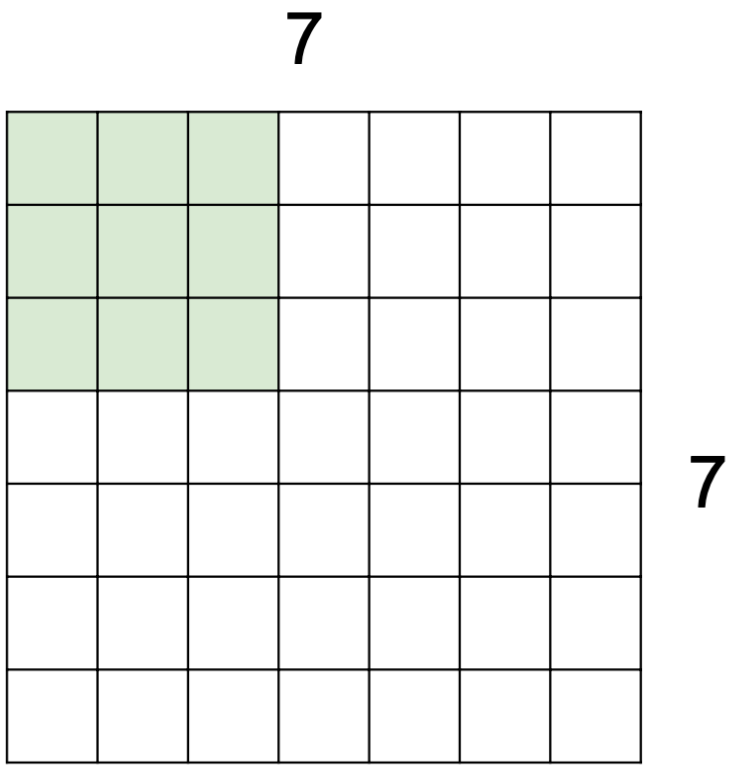
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

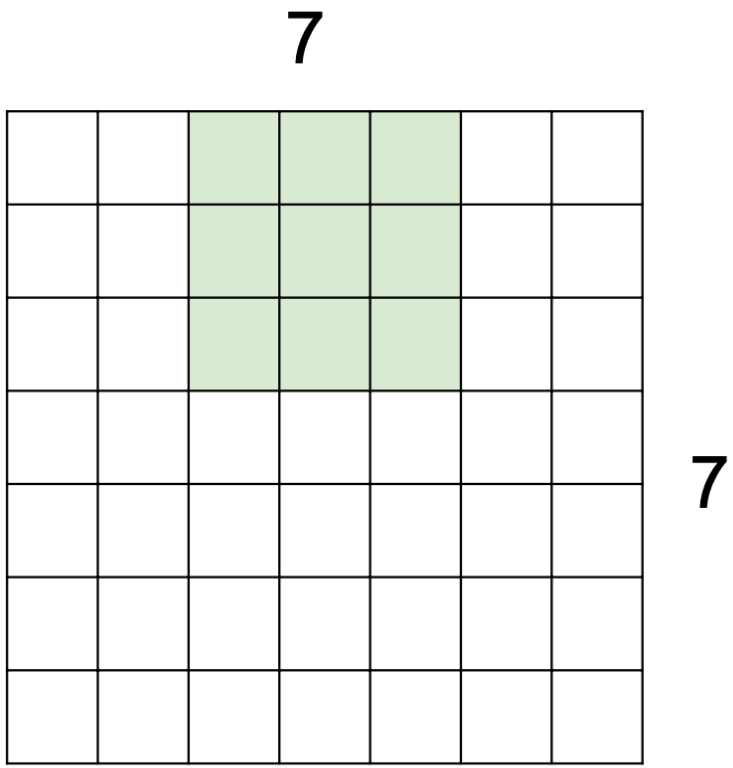
=> 5x5 output

A closer look at spatial dimensions:



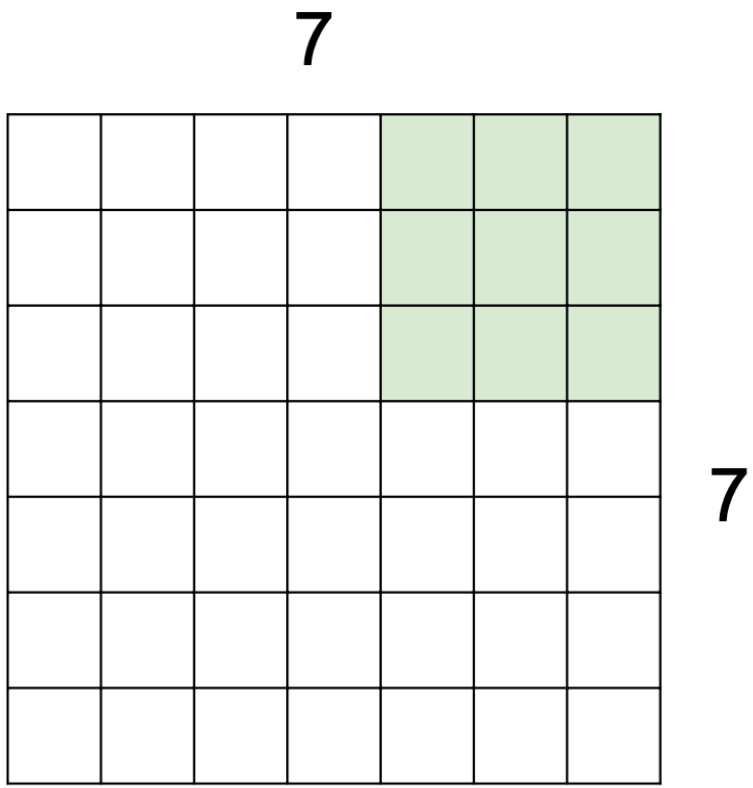
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



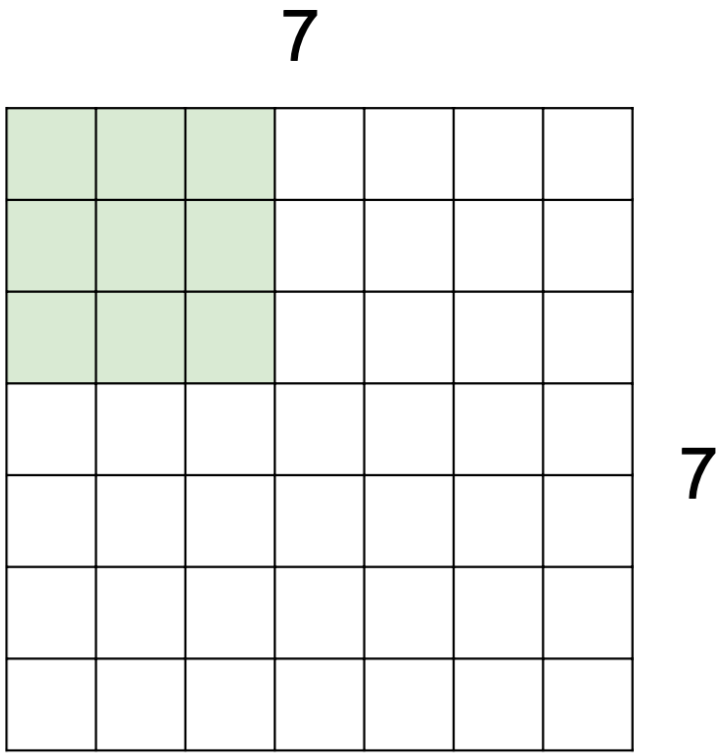
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



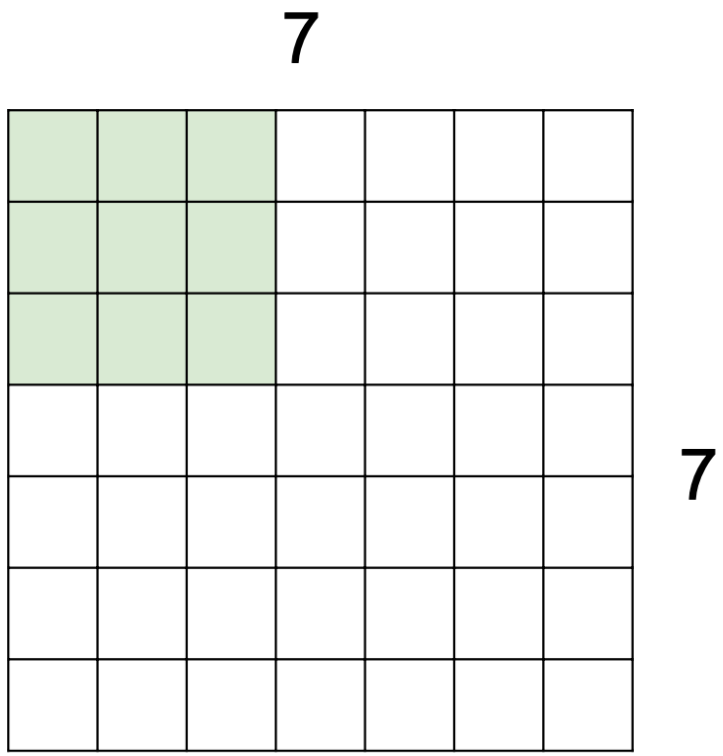
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



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

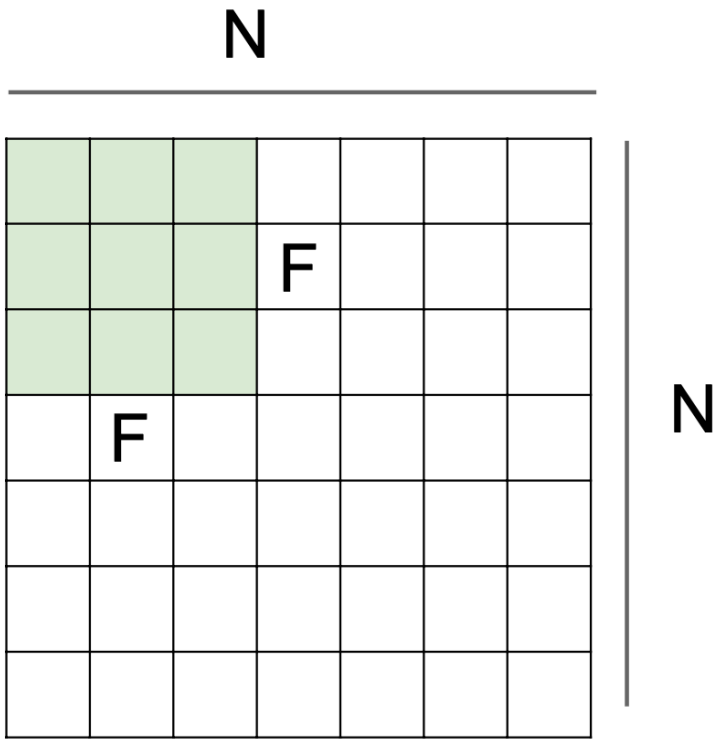
A closer look at spatial dimensions:



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

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

A closer look at spatial dimensions:

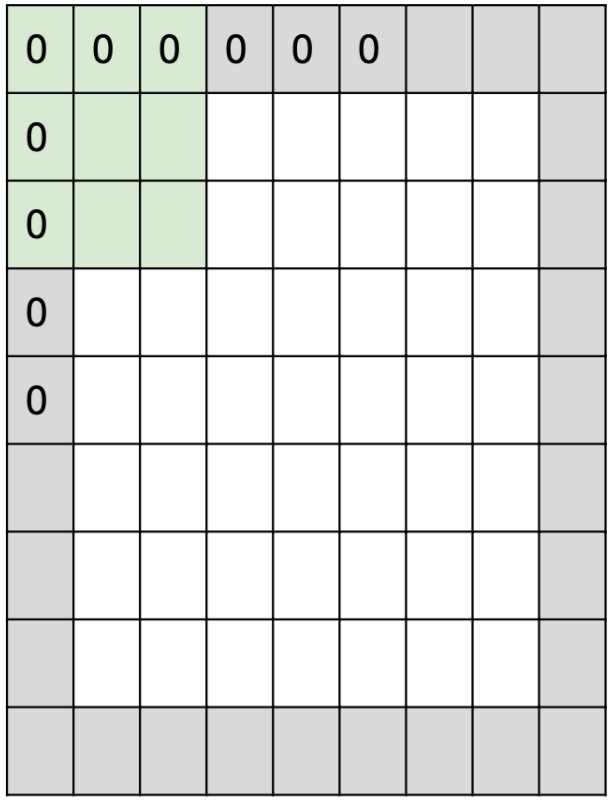


Output size:
(N - F) / stride + 1

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 :\

A closer look at spatial dimensions:

In practice: Common to zero pad the border

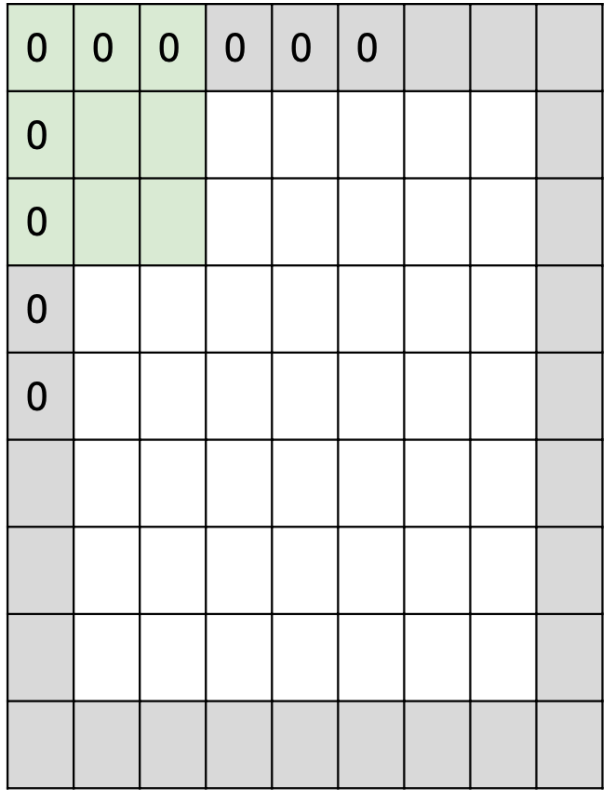


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

A closer look at spatial dimensions:

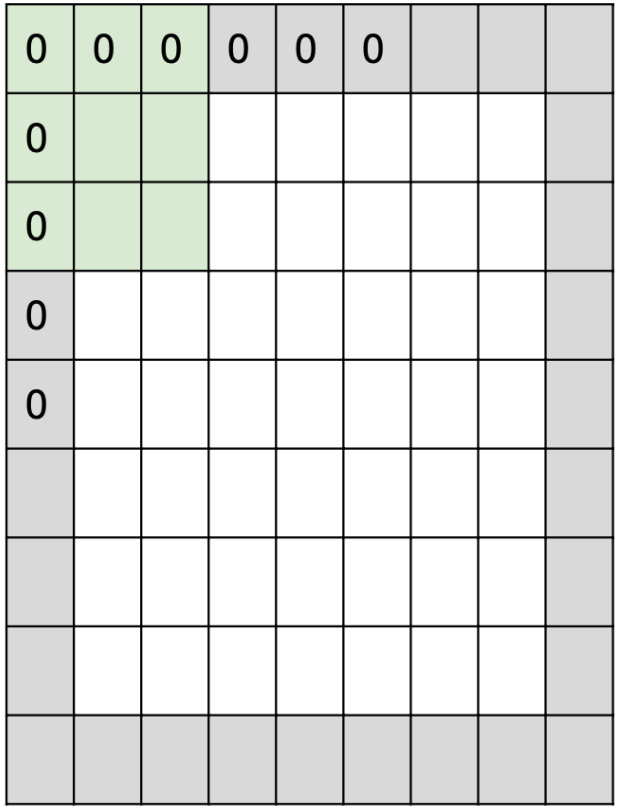
In practice: Common to zero pad the border



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

7x7 output!

In practice: Common to zero pad the border

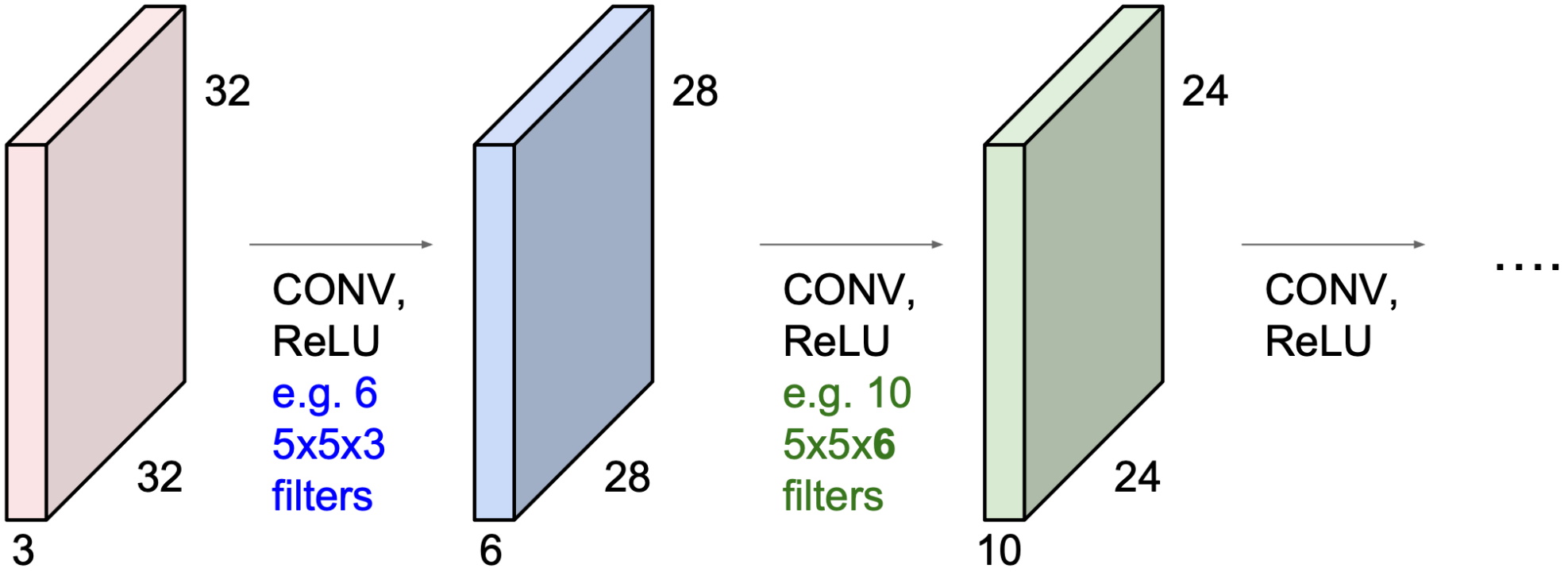


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

A closer look at spatial dimensions:

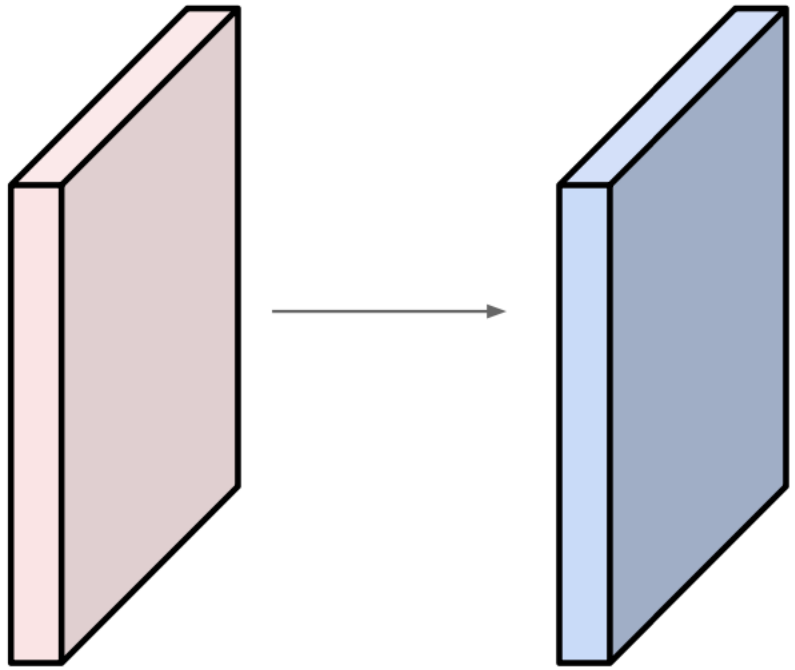
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Example

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

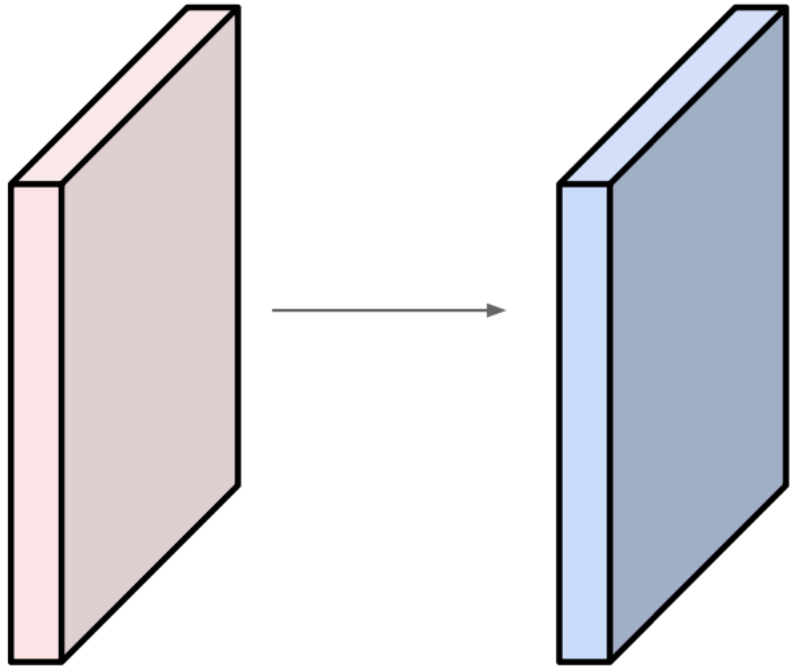
Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



Example

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

Number of parameters in this layer?

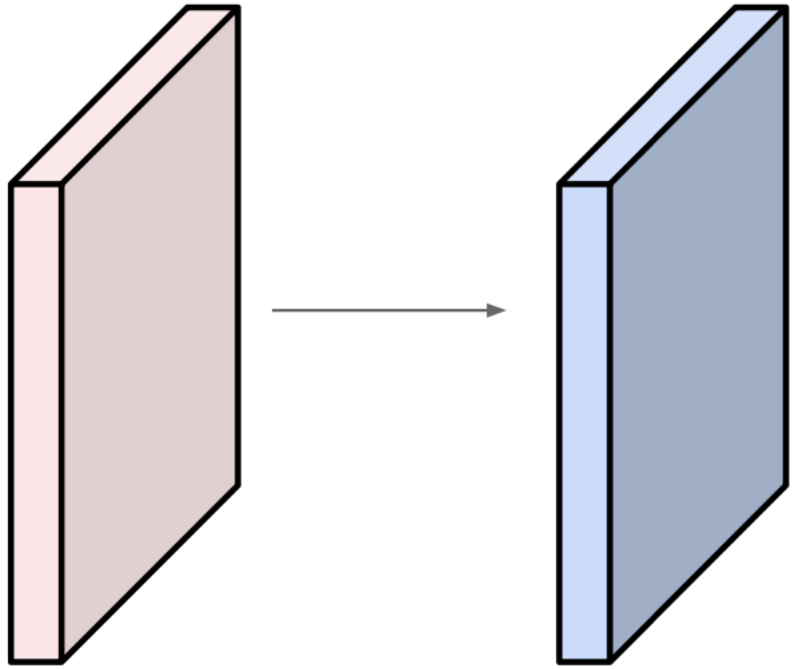


Example

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
=> 76*10 = **760**

Why +1?



Summary: 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: 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

Summary: Conv Layer in PyTorch

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At groups=1, all inputs are convolved to all outputs.
  - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At groups= `in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$.

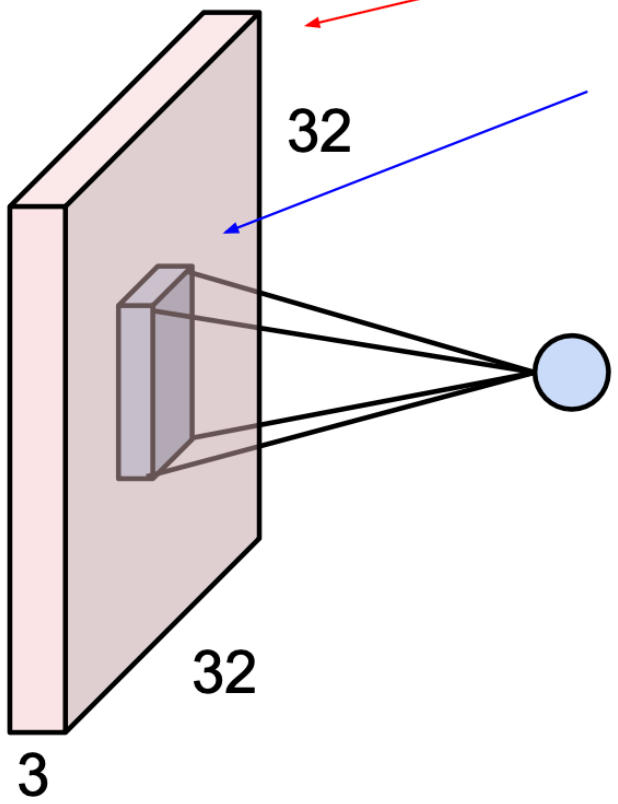
The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

Summary. To summarize, the Conv Layer:

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

The brain/neuron view of CONV Layer

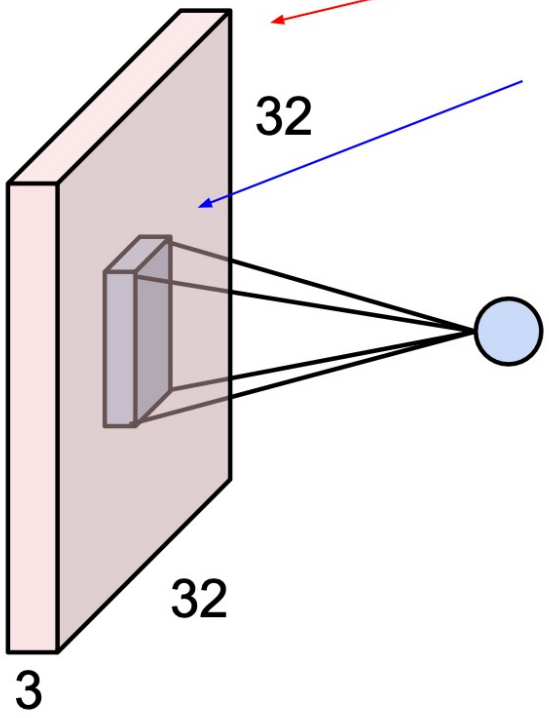


32x32x3 image

5x5x3 filter

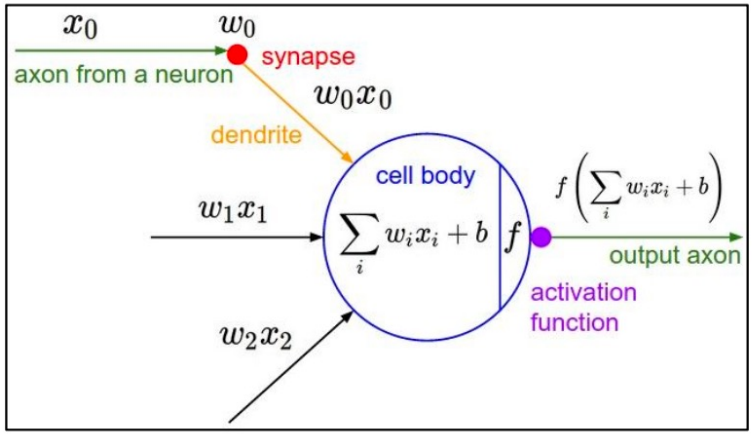
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer



32x32x3 image

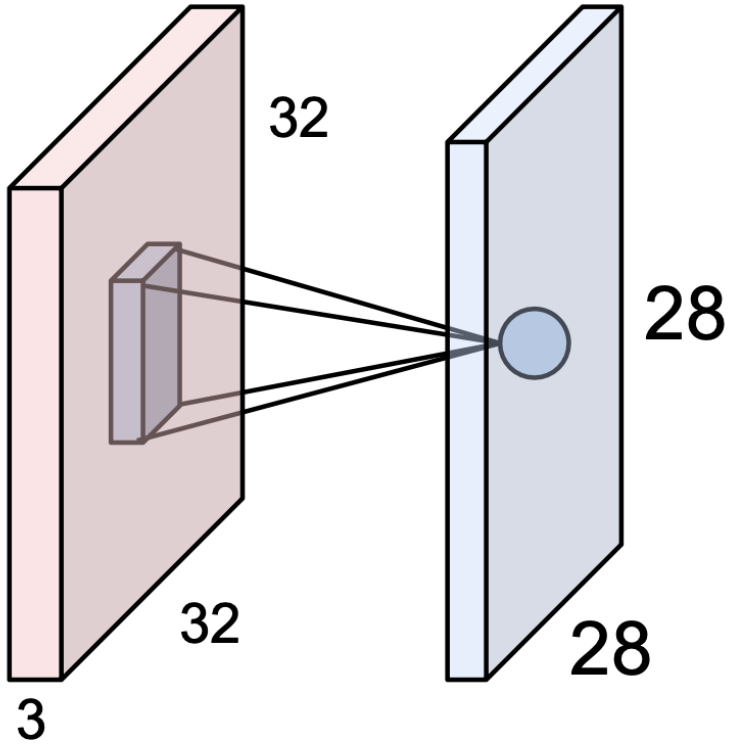
5x5x3 filter

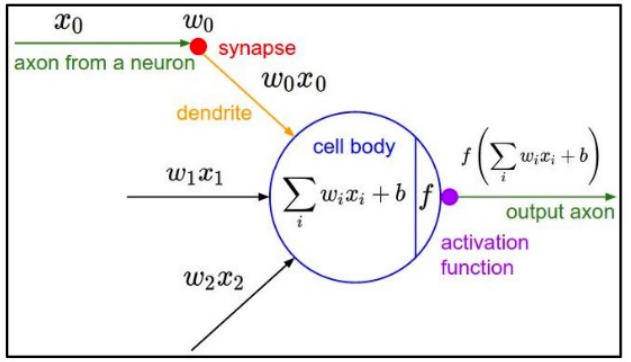


It's just a neuron with local connectivity...

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

The brain/neuron view of CONV Layer



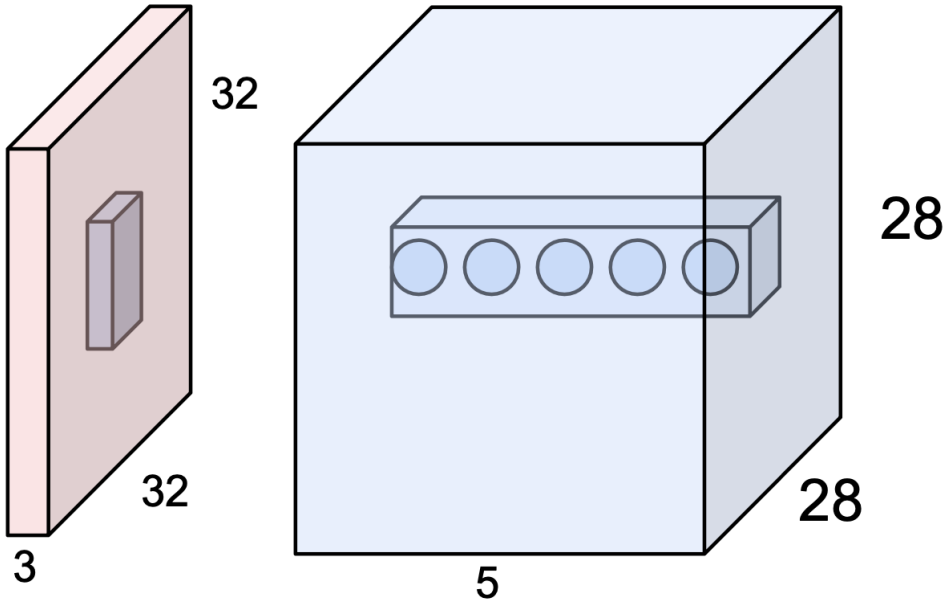


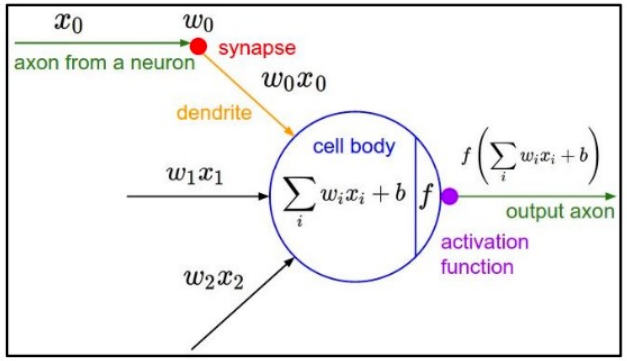
An activation map is a 28x28 sheet of neuron outputs:

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

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

The brain/neuron view of CONV Layer



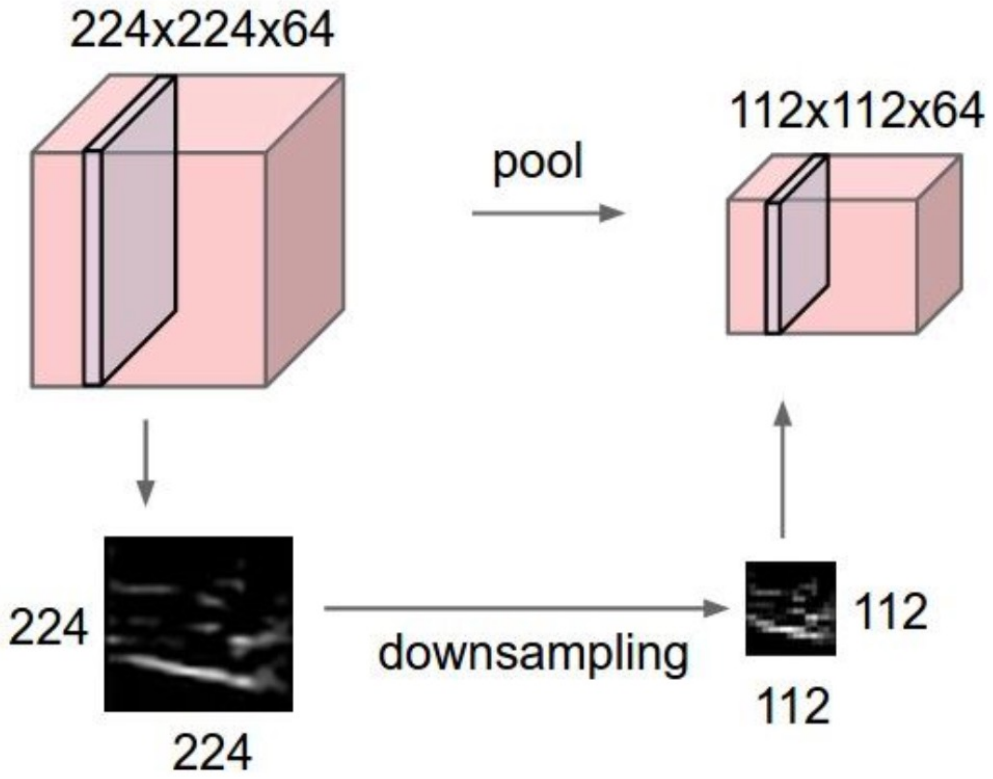


E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

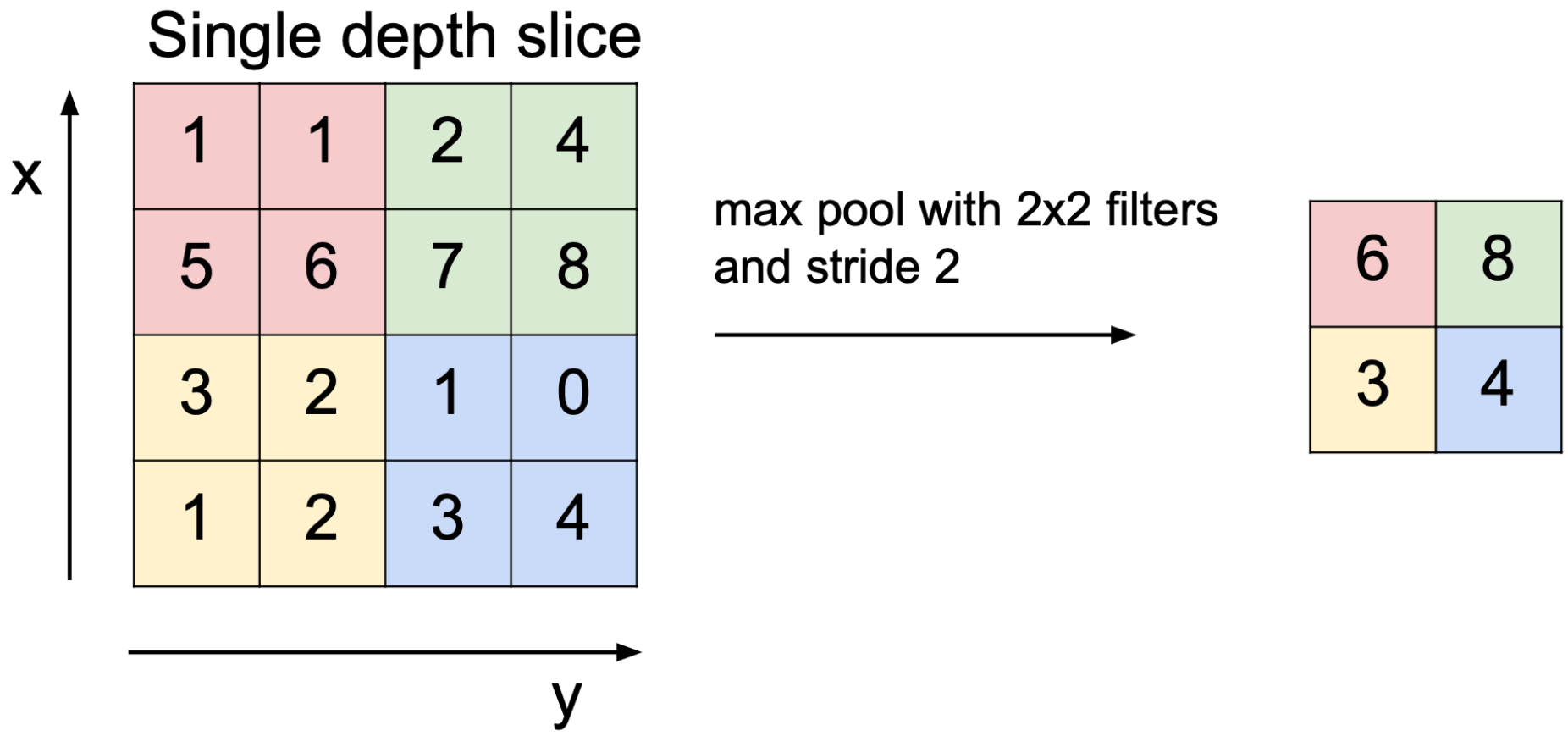
There will be 5 different neurons all looking at the same region in the input volume

Pooling Layer

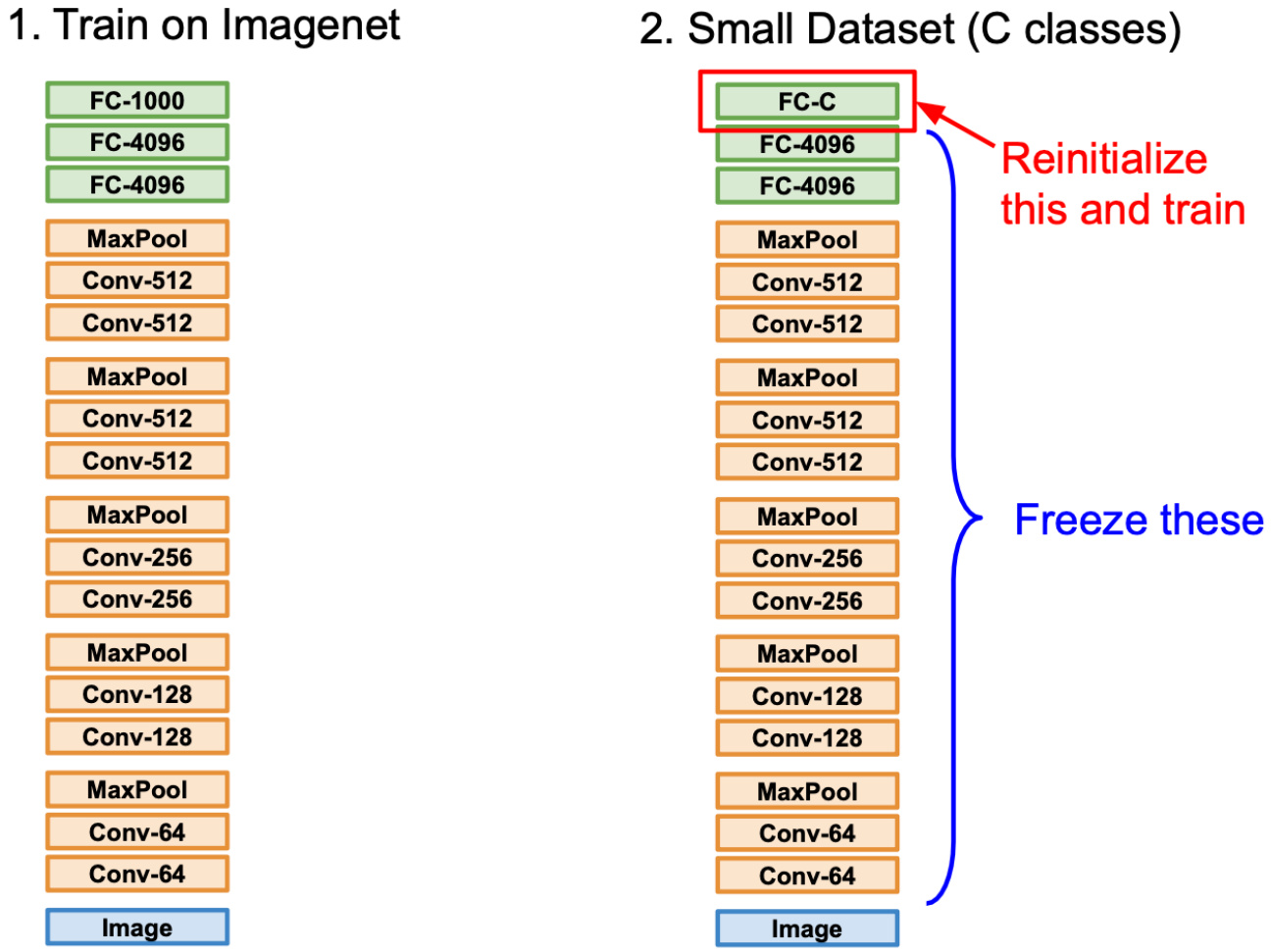
- Makes the representations smaller and more manageable
- Operates over each activation map independently:



Pooling Layer: Max Pooling



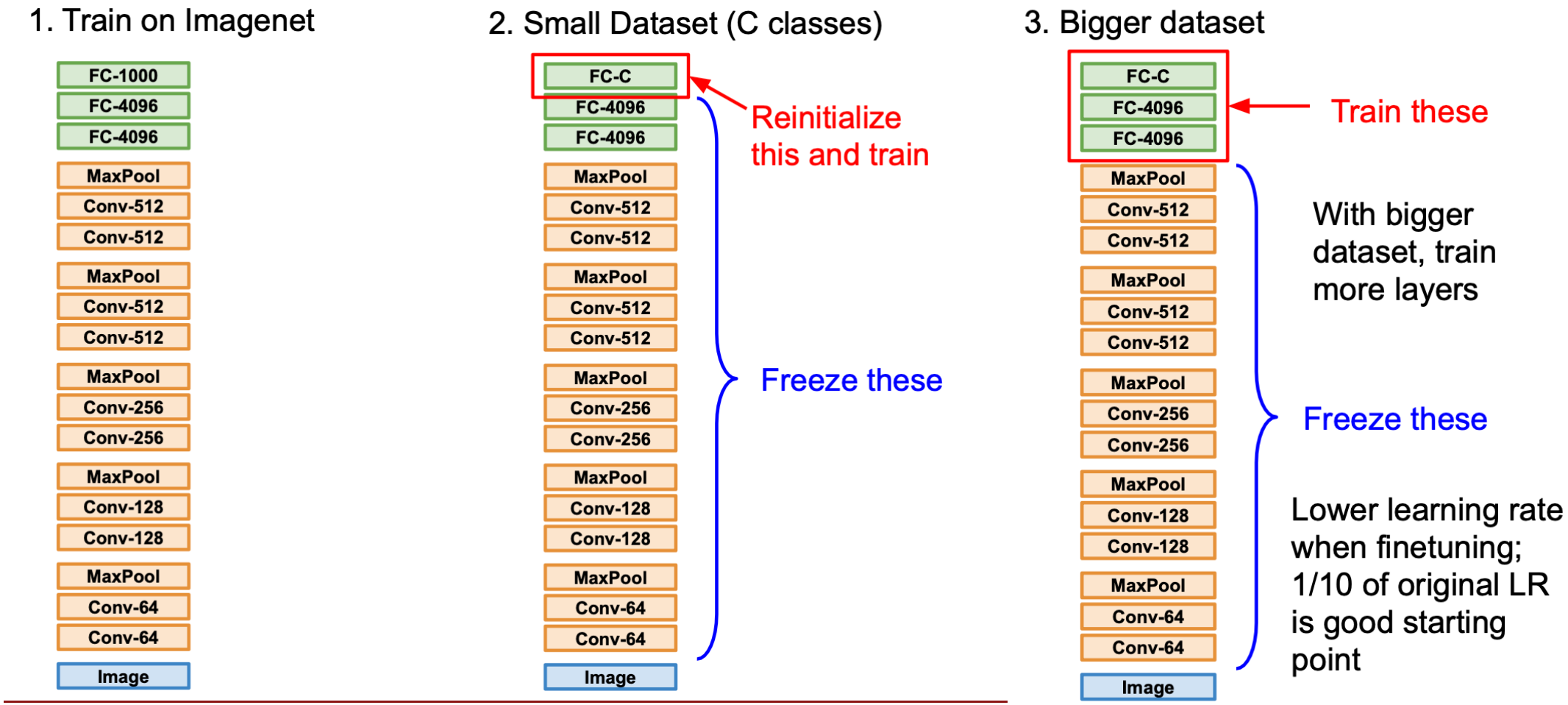
Transfer Learning with CNNs



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

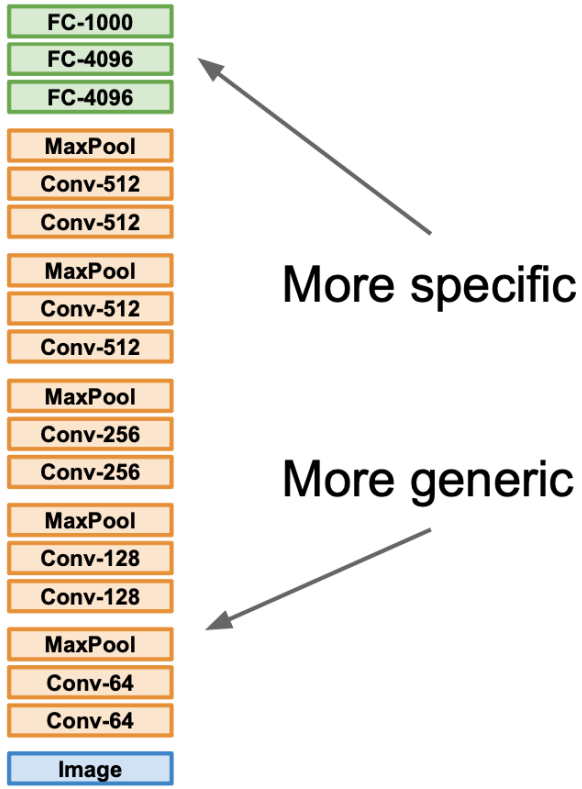
Transfer Learning with CNNs

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



Transfer Learning with CNNs

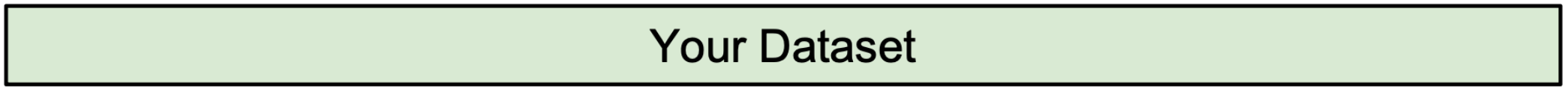
Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014 Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014



	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

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data



Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

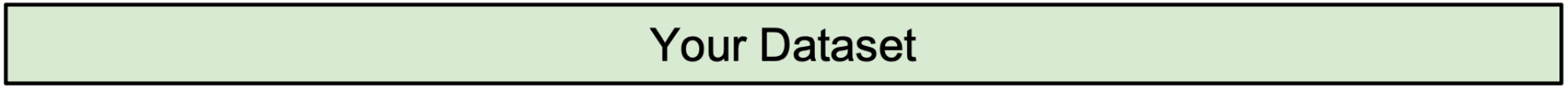
BAD: K = 1 always works perfectly on training data

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data



Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

References

1. Deep Learning Slides by Fei-Fei LI & Justin Johnson & Serena Yeung & Charu C. Aggarwal
2. https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597
3. https://www.shiksha.com/online-courses/articles/understanding-multilayer-perceptron-mlp-neural-networks/