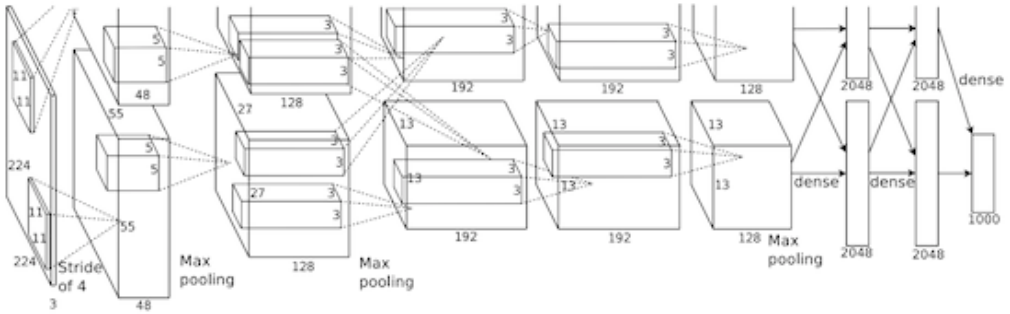
Deep Learning Week 3

Convolutional Neural Networks 1: Deep Learning with Images

Today's topics

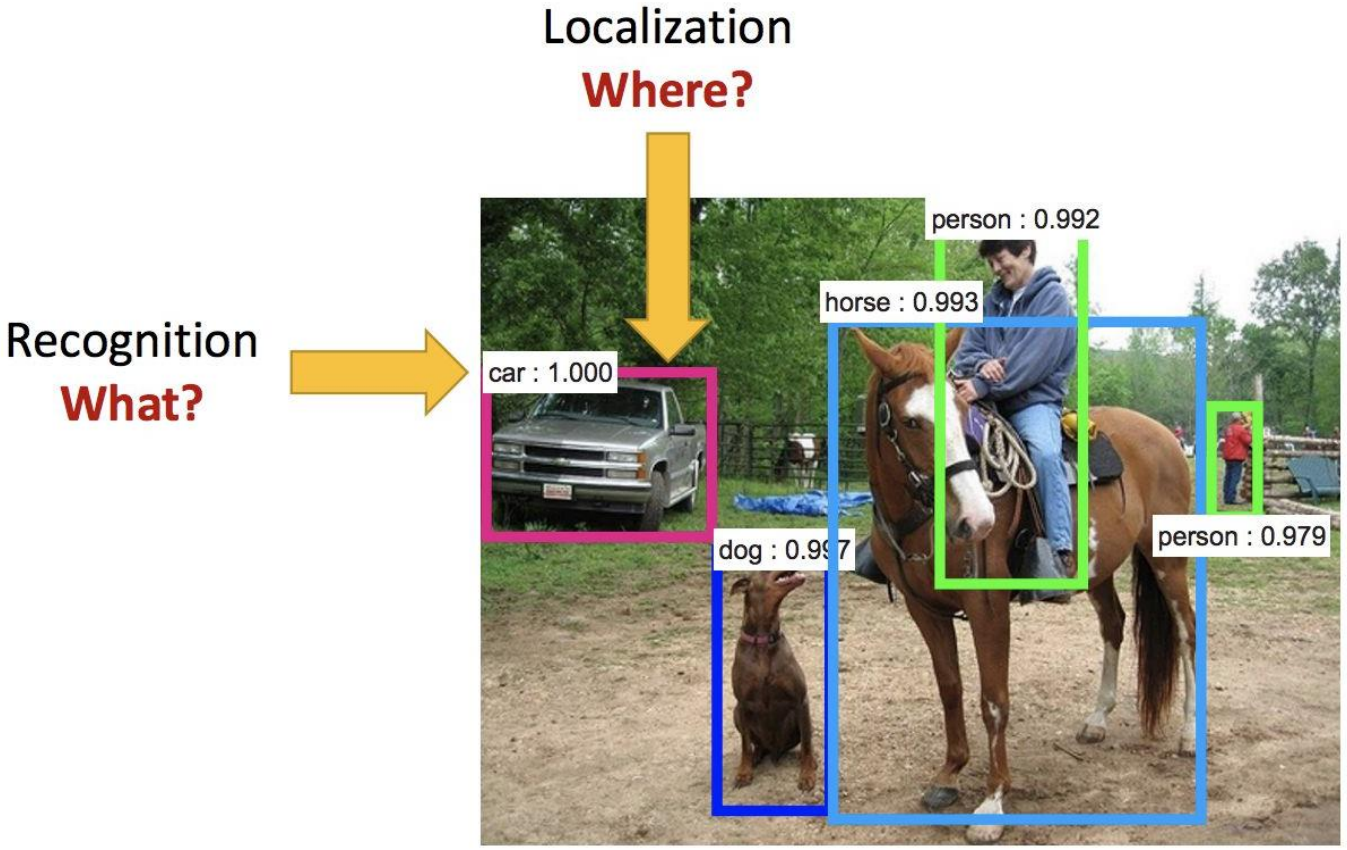
- Digital images
- Why CNN?
- The convolution operator
- Pooling operator
- Fully-connected layers

Case study: Image classification



- AlexNet:
- Developed in 2012
  - Introduced at ImageNet Large Scale Visual Recognition Challenge
  - Built on an architecture like LeNet but uses multiple convolutional layers.
  - The network achieved a top-5 error of 15.3%, more than 10.8 percentage points ahead of the runner up.
  - "Just" a modification of Yann LeCuns' network from 1989, that was implemented on a GPUs

Case study: Object detection
What and where



Slide credit: Kaiming He, ICCV 2015

Case study: Object segmentation

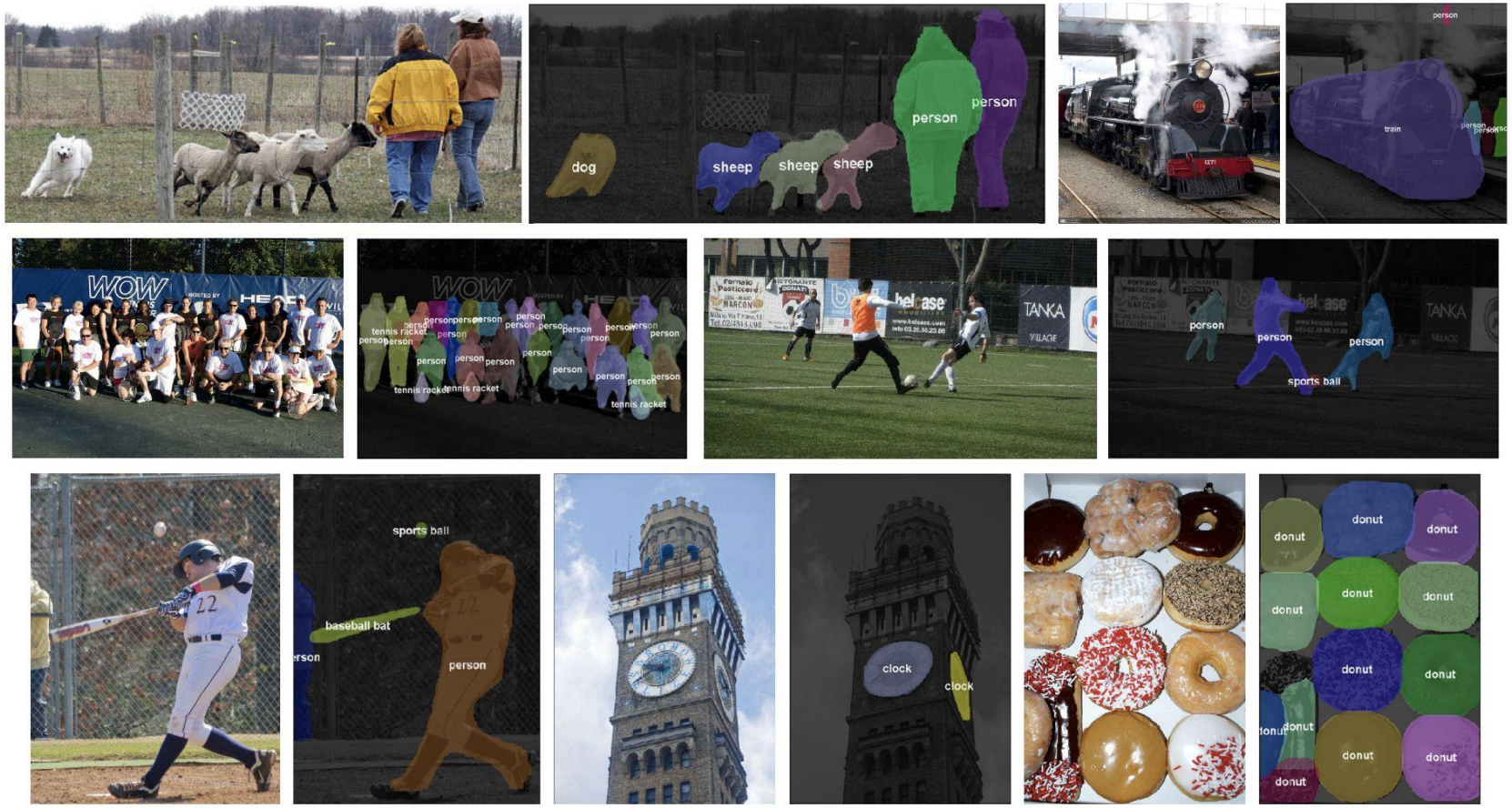


Figure credit: Dai, He, and Sun, “Instance-aware Semantic Segmentation via Multi-task Network Cascades”, CVPR 2016

Case study: Pose Estimation

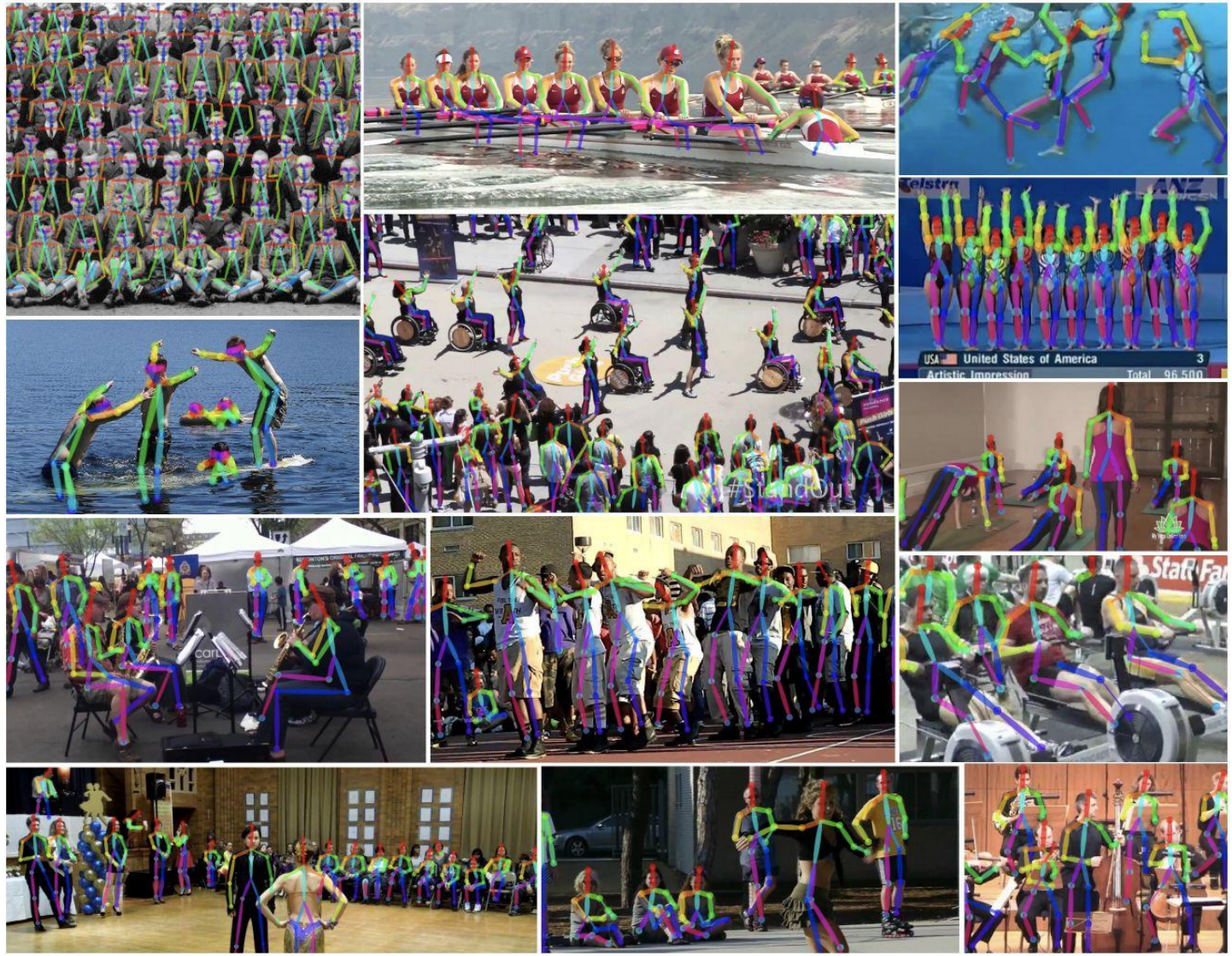


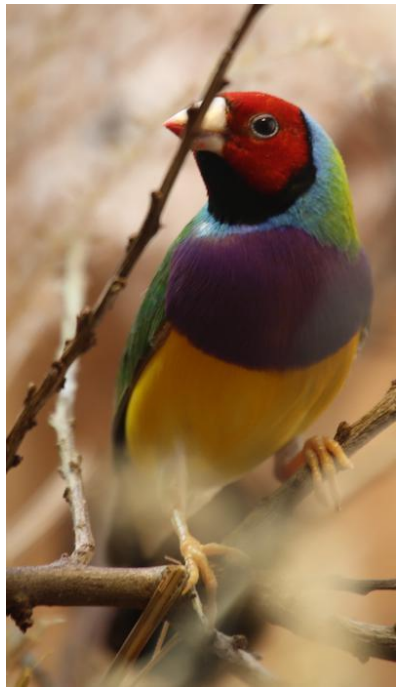
Figure credit: Cao et al., “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields", ArXiv 2016

Case study: Generating Art

Figure credit: Gatys, Ecker, and Bethge, “Image Style Transfer using Convolutional Neural Networks”, CVPR 2016

Image Classification: A core task in Computer Vision

Tell the class when given an image and a set of discrete labels



bird

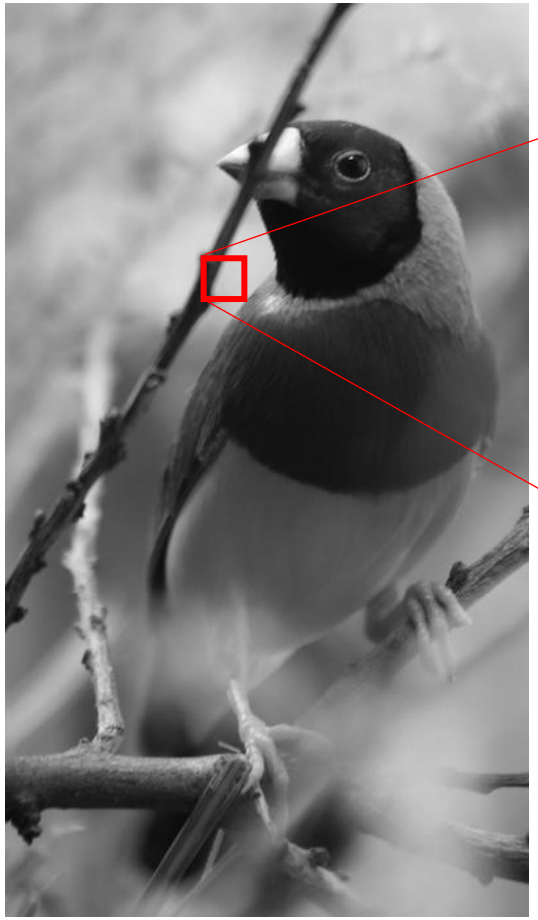
{cat, bird, dog, horse, pig, rabbit}

What is a digital image?

What the computer sees

A digital image is just a bunch of numbers!
Low values = dark
High values = light

The meaning comes from the relationship between pixels



201	202	201	199	196	195	194	192	189	188	187	185	185	186	186
201	200	199	198	195	194	192	190	190	189	188	185	185	186	186
200	198	197	195	193	193	192	190	187	186	186	187	186	183	199
198	196	195	194	192	192	191	189	190	187	187	183	185	210	241
195	195	195	194	192	191	191	190	188	186	182	194	223	248	254
196	195	194	193	191	191	192	192	187	186	206	237	249	248	250
192	194	193	192	192	192	190	186	195	222	244	247	239	238	243
193	193	192	193	194	189	192	206	234	240	238	230	228	232	236
195	191	193	193	191	198	218	235	233	226	219	219	222	222	219
191	195	193	192	207	225	226	218	211	212	215	214	213	208	199
194	190	196	213	222	214	200	194	194	194	200	196	190	191	186
194	199	212	216	200	182	178	178	170	162	164	164	163	168	171
197	206	203	184	168	157	148	143	126	123	126	127	130	138	140
204	191	178	162	144	129	116	102	97	97	99	100	99	95	89
186	171	165	145	114	95	90	84	78	83	83	84	81	83	74

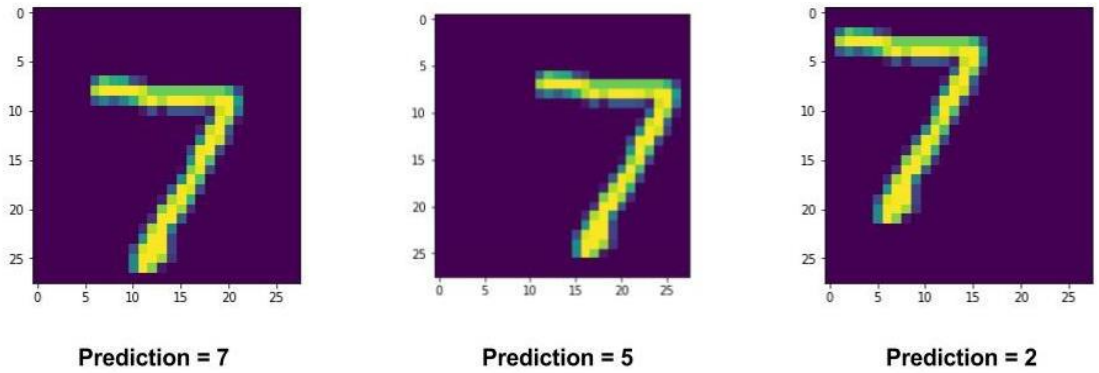
What the computer sees

Color images are made up from three layers; red, green, and blue

The dimension of this color-image is 440x320x3
I.e., 440 rows, 320 columns, 3 channels

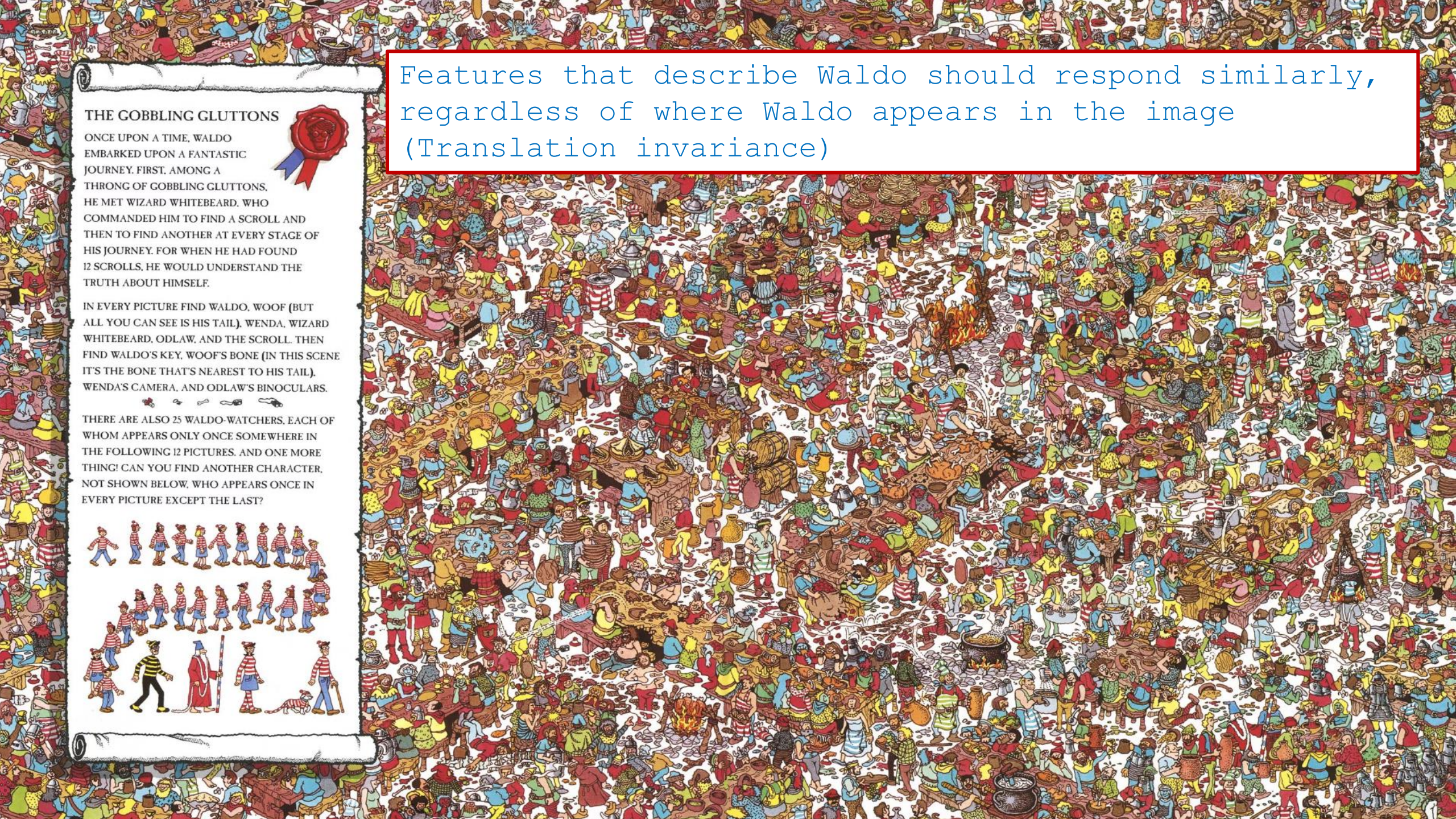
CNN motivation

- MLPs can perform pretty good. (MNIST classification accuracy>98%)
- BUT! MLPs require fixed input sizes.
- MLPs scales poorly with increased input size
- In MNIST, all samples are centered
- MLPs does not consider the spatial relationship of pixels
- MLPs are <u>NOT</u> translation invariant. Result ->



- We can dramatically reduce the number of parameters by making one reasonable assumption: That if one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2)

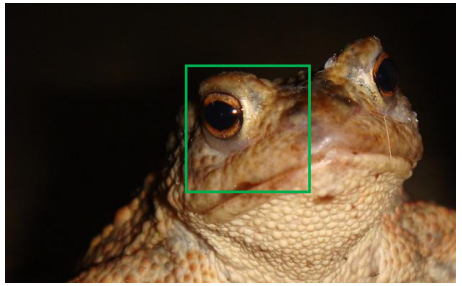
Features that describe Waldo should respond similarly, regardless of where Waldo appears in the image (Translation invariance)

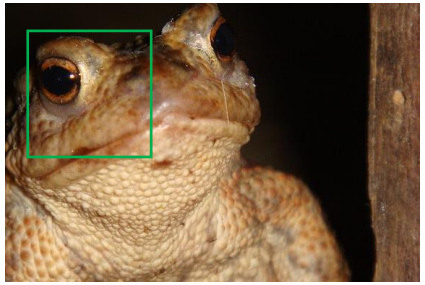


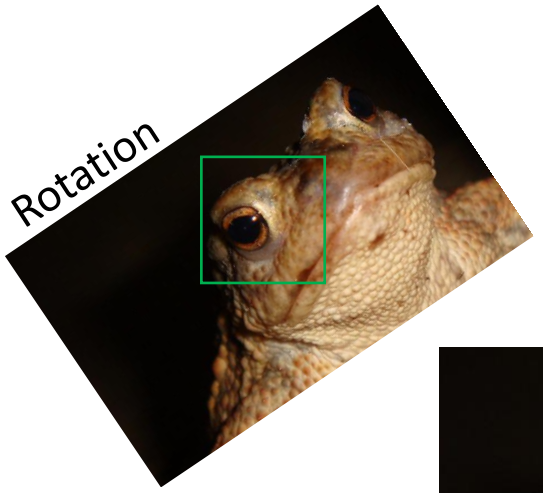
Convolutions are

- Invariant to translation
  (filter would produce same output but at another location)
- NOT invariant to rotation
- NOT invariant to scale

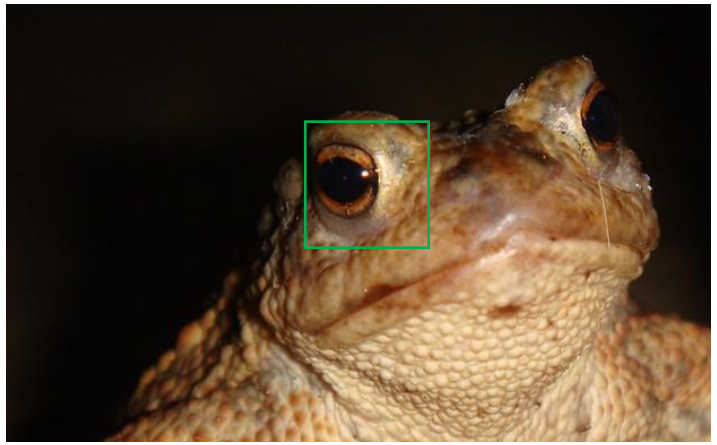
Translation







Scale

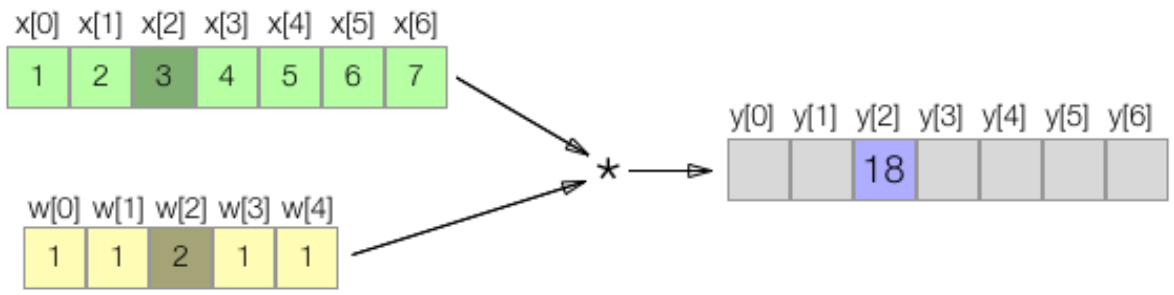


How does the convolution operator work?

1D Convolution

- A moving filter that produces a local response *y*(x) based on the overlapping parts of the signal *f* and the filter-kernel *w*

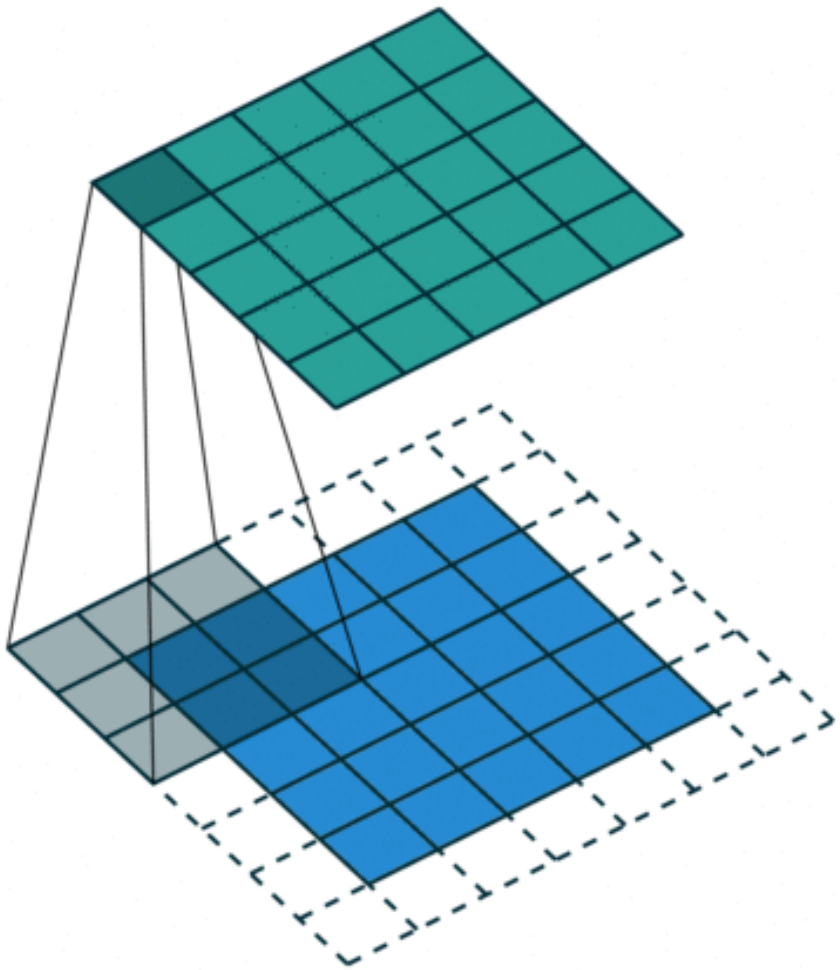
$$y(x) = (f * w)(x) = \sum_{s=-a}^{a} f(x-s) \cdot w(s)$$



y[2] = x[0]w[4] + x[1]w[3] + x[2]w[2] + x[3]w[1] + x[4]w[0] =

1*1 + 2*1 + 3*2 + 4*1 + 5*1 = 18

2d convolution



$$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t) \cdot w(s,t)$$

Example: 2D convolution with 3x3 kernel

Output is applied at center of kernel

Input image

1	2	-3	-1	-5	4	-1
-1	-4	4	-3	2	-5	-2
0	3	5	-5	3	3	1
-3	-3	-1	4	-5	-5	2
3	1	4	0	-4	1	4
-4	-4	-1	4	-2	-1	-1
-3	1	1	-1	-4	-4	-5

*

3	1	-3
-2	4	2
1	-5	-5

kernel

=

Activation map

$$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t) \cdot w(s,t)$$

Example: 2D convolution with 3x3 kernel

Output is applied at center of kernel

1	2	-3	-1	-5	4	-1
-1	-4	4	-3	2	-5	-2
0	3	5	-5	3	3	1
-3	-3	-1	4	-5	-5	2
3	1	4	0	-4	1	4
-4	-4	-1	4	-2	-1	-1
-3	1	1	-1	-4	-4	-5

*

3	1	-3
-2	4	2
1	-5	-5

kernel

=

-26	-1	-4	73	-11
34	53	-27	-36	45
-22	-62	3	10	-38
36	65	-3	-35	42
-25	-50	-22	10	6

$1\cdot(-5) + 2\cdot(-5) + (-3)\cdot 1 + (-1)\cdot 2 + (-4)\cdot 4 + 4\cdot(-2) + 0\cdot(-3) + 3\cdot 1 + 5\cdot 3 = -26$

$$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t) \cdot w(s,t)$$

Example: 2D convolution with 3x3 kernel

Output is applied at center of kernel

1	2	-3	-1	-5	4	-1
-1	-4	4	-3	2	-5	-2
0	3	5	-5	3	3	1
-3	-3	-1	4	-5	-5	2
3	1	4	0	-4	1	4
-4	-4	-1	4	-2	-1	-1
-3	1	1	-1	-4	-4	-5

*

3	1	-3
-2	4	2
1	-5	-5

kernel

=

-26	-1	-4	73	-11
34	53	-27	-36	45
-22	-62	3	10	-38
36	65	-3	-35	42
-25	-50	-22	10	6

$1\cdot(-5) + 2\cdot(-5) + (-3)\cdot 1 + (-1)\cdot 2 + (-4)\cdot 4 + 4\cdot(-2) + 0\cdot(-3) + 3\cdot 1 + 5\cdot 3 = -26$

$$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t) \cdot w(s,t)$$

Example: 2D convolution with 3x3 kernel

Output is applied at center of kernel

1	2	-3	-1	-5	4	-1
-1	-4	4	-3	2	-5	-2
0	3	5	-5	3	3	1
-3	-3	-1	4	-5	-5	2
3	1	4	0	-4	1	4
-4	-4	-1	4	-2	-1	-1
-3	1	1	-1	-4	-4	-5

*

3	1	-3
-2	4	2
1	-5	-5

kernel

=

-26	-1	-4	73	-11
34	53	-27	-36	45
-22	-62	3	10	-38
36	65	-3	-35	42
-25	-50	-22	10	6

$1\cdot(-5) + 2\cdot(-5) + (-3)\cdot 1 + (-1)\cdot 2 + (-4)\cdot 4 + 4\cdot(-2) + 0\cdot(-3) + 3\cdot 1 + 5\cdot 3 = -26$

$$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t) \cdot w(s,t)$$

Example: 2D convolution with 3x3 kernel

Output is applied at center of kernel

1	2	-3	-1	-5	4	-1
-1	-4	4	-3	2	-5	-2
0	3	5	-5	3	3	1
-3	-3	-1	4	-5	-5	2
3	1	4	0	-4	1	4
-4	-4	-1	4	-2	-1	-1
-3	1	1	-1	-4	-4	-5

*

3	1	-3
-2	4	2
1	-5	-5

kernel

=

-26	-1	-4	73	-11
34	53	-27	-36	45
-22	-62	3	10	-38
36	65	-3	-35	42
-25	-50	-22	10	6

$1\cdot(-5) + 2\cdot(-5) + (-3)\cdot 1 + (-1)\cdot 2 + (-4)\cdot 4 + 4\cdot(-2) + 0\cdot(-3) + 3\cdot 1 + 5\cdot 3 = -26$

$$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t) \cdot w(s,t)$$

? What would the output value at the red square be?

4	6	-2	-4	2
-2	0	-1	1	0
4	2	5	-5	3
1	-5	3	-1	1
4	1	2	-1	-3

$*$

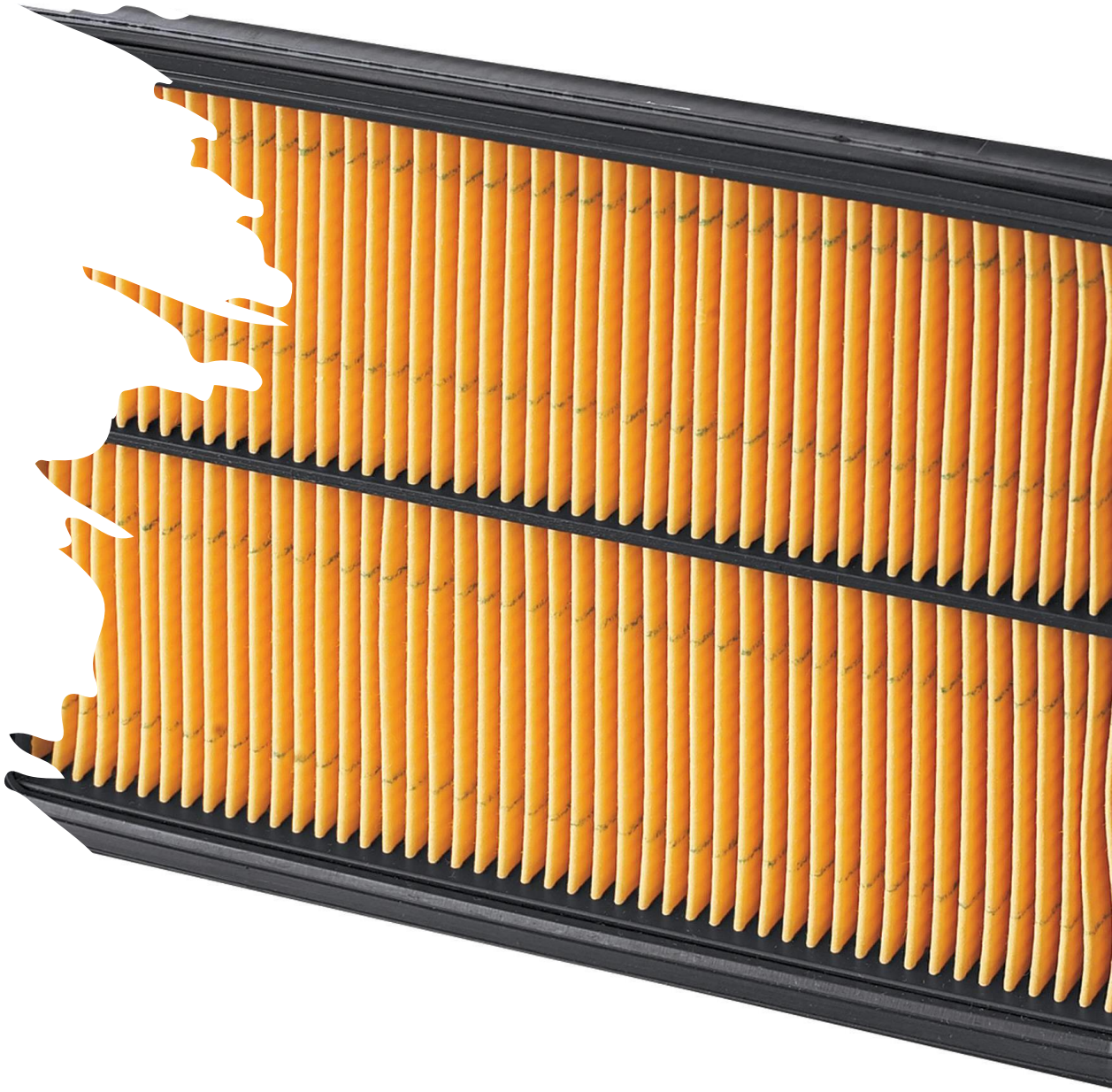
0	-3	-4
2	0	-3
1	2	0

$=$

- Spend three minutes on calculating the value
- Compare with the person next to you. Do you agree?

$$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t) \cdot w(s,t)$$

Let's handcraft
some filters to see
what they can do



mean-filter

$$* \; 1/9 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$

Vertical Prewitt-filter

$*$

1	0	-1
1	0	-1
1	0	-1

$=$

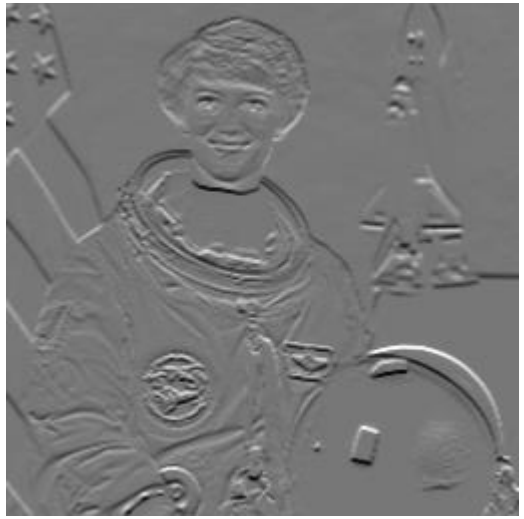
* An offset is added to the visualization since the output has zero mean

Horizontal Prewitt-filter

$*$

1	1	1
0	0	0
-1	-1	-1

$=$



* An offset is added to the visualization since the output has zero mean

NB.
In CNNs we don't handcraft filters
– the computer will learn the filters by itself!

Convolution and correlation (dot product)

- The difference between correlation and convolution is that the kernel is rotated 180 degrees
- In image processing convolution is often replaced by correlation (I.e., the dot product of the common area)
- In literature – especially related to deep learning – the word "convolution" is used, when the notation is a correlation
- In PyTorch the torch.nn.Conv2d operator does NOT perform a convolution but at correlation (!)
  - it doesn't matter, because the weights of filters are learned during training

Convolution: $$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x-s, y-t) \cdot w(s,t)$$

Correlation: $$g(x,y) = (f * w)(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x+s, y+t) \cdot w(s,t)$$

x, y are pixel coordinates, f is the input

What about the bias???

- We can (and should) add a learnable bias to the output of the convolution – one value for each kernel

$$g(x, y) = (f * w)(x, y) + b$$

- In PyTorch we don't have to do this manually, but we can specify that the convolution should be followed by a bias (default setting)

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,
groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

What about the bias???

- Importance of bias

$$g(x,y) = ReLU\big((f * w)(x,y) + b\big)$$

1	2	-3	-1	-5	4	-1
-1	-4	4	-3	2	-5	-2
0	3	5	-5	3	3	1
-3	-3	-1	4	-5	-5	2
3	1	4	0	-4	1	4
-4	-4	-1	4	-2	-1	-1
-3	1	1	-1	-4	-4	-5

$*$

3	1	-3
-2	4	2
1	-5	-5

$=$

Before ReLU

-26	-1	-4	73	-11
34	53	-27	-36	45
-22	-62	3	10	-38
36	65	-3	-35	42
-25	-50	-22	10	6

$\rightarrow$

After ReLU

0	0	0	73	0
34	53	0	0	45
0	0	3	10	0
36	65	0	0	42
0	0	0	10	6

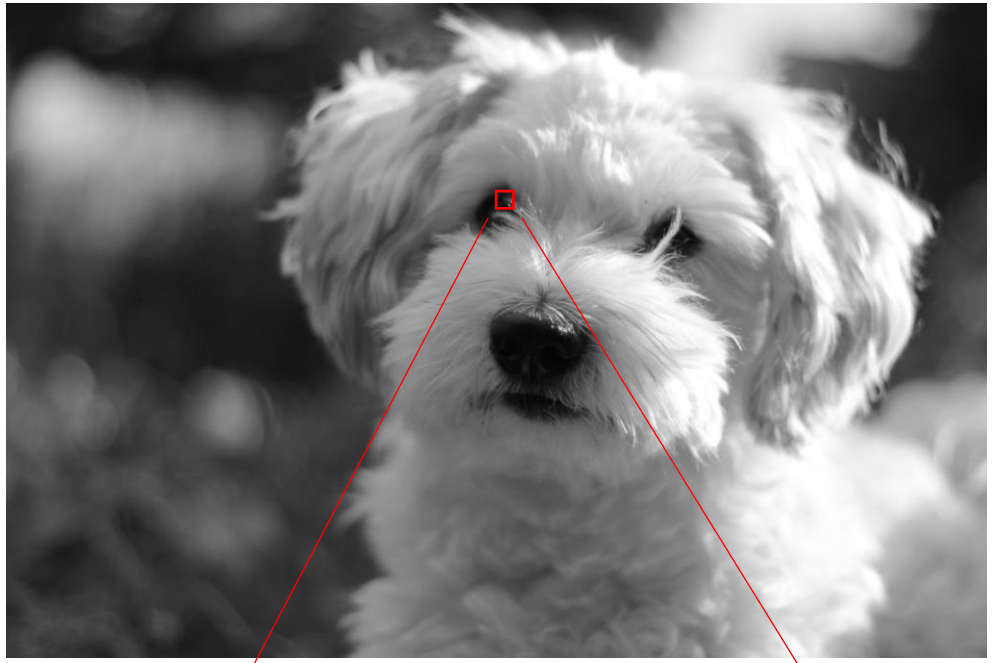
Hyperparameters related to convolution layers

Kernel size	How many weights should the kernel have?
Number of kernels	How many filters should we apply to each input?
Padding (next slides)	Should we add padding to the input to alter the output-size?
Stride (next slides)	Should the movement of the kernel be larger than one?
Dilation (next slides)	Should we add spacing between the kernel's weights?

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,
groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Kernel size

- Kernels often have an uneven height and width (3x3, 5x5, 7x7 …), so that there is a well-defined center.
- Bigger kernels requires more computations
- The bigger the kernel, the bigger the *receptive field* --> better catching semantics
  - The same field of view can be acheived by stacking multiple smaller kernels

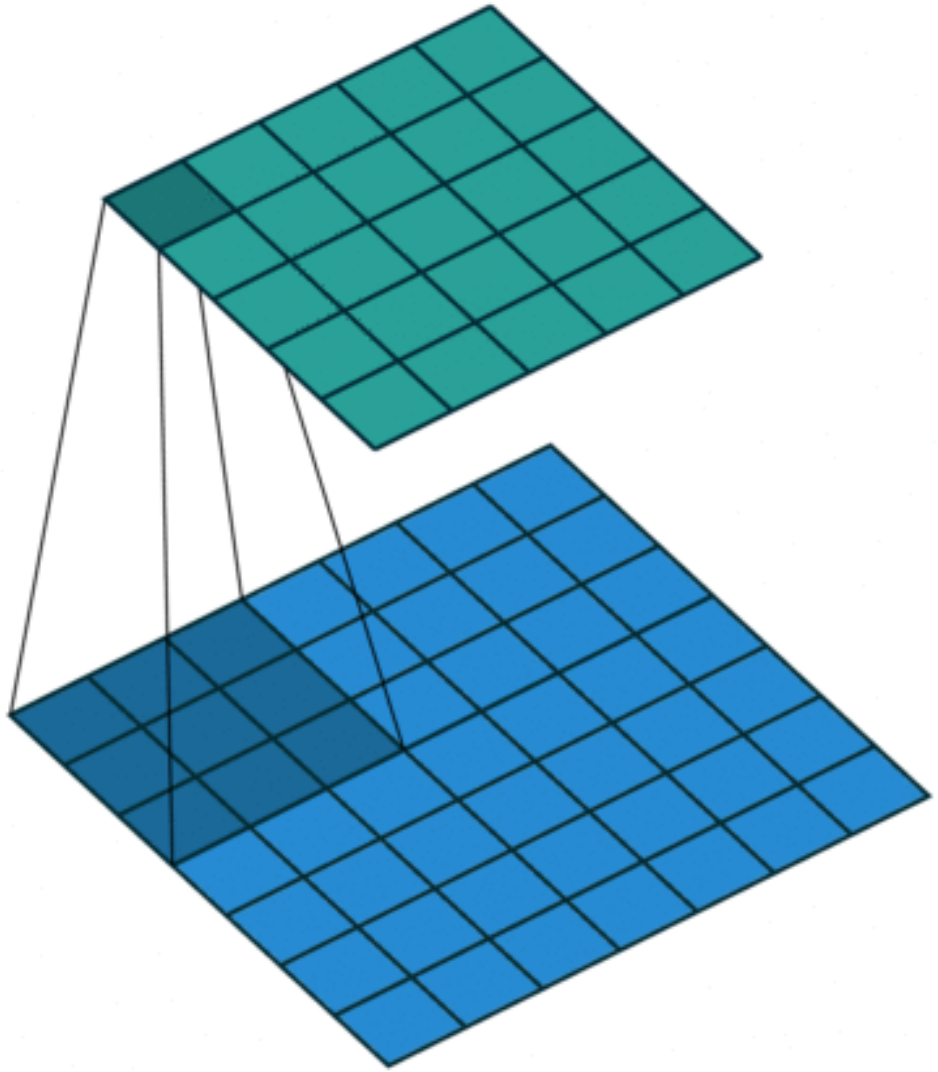


	0	1	2	3	4	5	6	7
0	42	44	51	59	70	76	83	86
1	43	39	41	50	61	70	77	83
2	39	43	48	50	51	59	68	76
3	37	41	65	64	56	55	61	65
4	27	29	45	81	87	65	53	57
5	22	30	29	78	186	107	57	52
6	20	21	23	38	165	168	77	49
7	22	16	24	25	64	169	118	49

General rule: More small kernels is better than few big kernels

No padding

- What we have looked at so far is convolution with ***valid*** outputs
  - I.e., we only consider pixels fully covered by the kernel
  - Result: Output (green) is smaller than input (blue)
- We can add padding to make the output the same size as the input

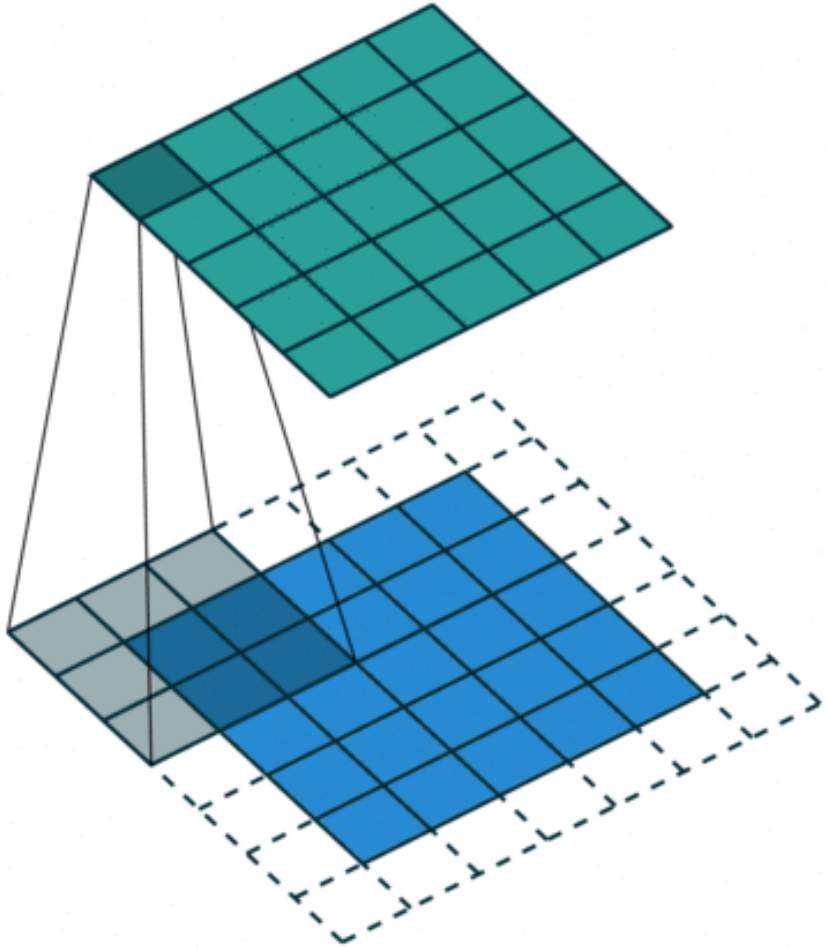


Visualization from https://github.com/vdumoulin/conv_arithmetic

"Same" Padding

- We can add padding around the image (blue) to make the output (green) the same size as the input
  - Padding is usually zeros (zero-padding) or a mirrored version of the input
  - This mode is called ***same*** in PyTorch
  - The nice feature of zero padding is that it will allow us to control (preserve) the spatial size of the output volumes when we stack multiple conv-layers
- The number of pixels to add is $P = (F - 1)/2$

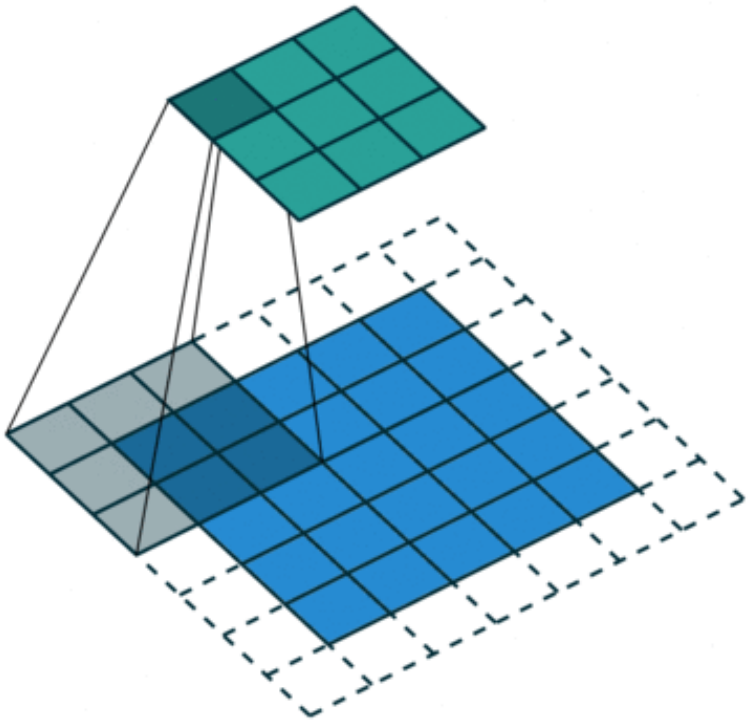
, where F is the kernel size



Visualization from https://github.com/vdumoulin/conv_arithmetic

Stride

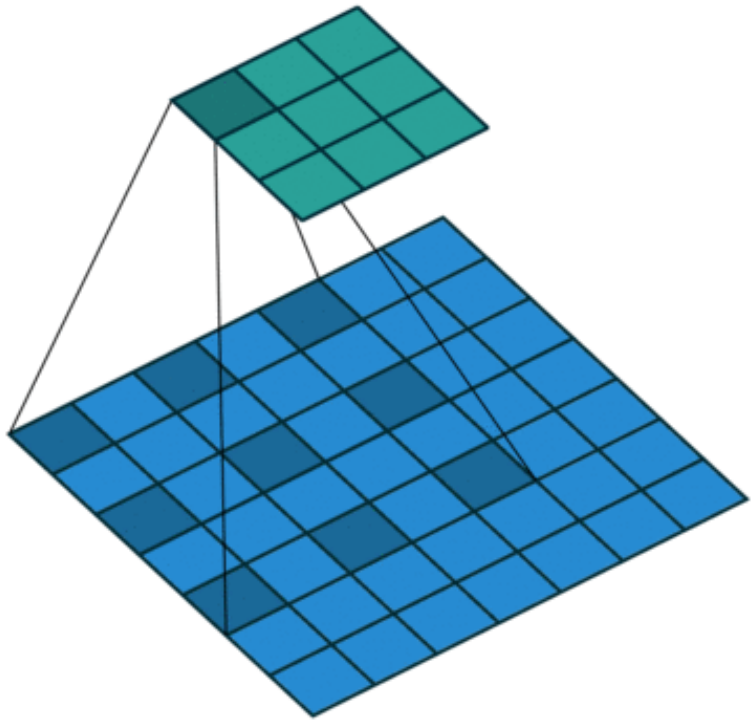
- So far, the kernel has been moved one pixel at the time However, if we move the kernel in steps greater than one pixel, the output is downsampled.
- The image to the right shows both padding and stride



Visualization from https://github.com/vdumoulin/conv_arithmetic

Dilated convolution

- Also known as atrous convolution
- The kernel is "inflated" to cover a larger area in the input.
- Useful for large images where image gradients are wide.



Visualization from https://github.com/vdumoulin/conv_arithmetic

Image dimensions in PyTorch

Usually, image dimensions are given as rows × columns × channels.

This is <u>not</u> the default case in PyTorch

In Pytorch dimensions are typically given as

Sample × channels × rows × columns

(On Intel hardware, going channel-last can increase the performance. You can find info on how to convert models and data from channel-first to channel-last on this link: https://pytorch.org/blog/accelerating-pytorch-vision-models-with-channels-last-on-cpu/)

Altering tensor dimensions

Dimensions of pytorch tensors can be altered using the permute method:

```
im2 = im.permute(2, 0, 1)
```

The order of the arguments tells the order of the dimensions.

If the dimensions of `im` are $\overset{0}{\text{rows}} \times \overset{1}{\text{columns}} \times \overset{2}{\text{channels}}$

$\Downarrow$

the dimensions of im2 are $\overset{2}{\text{channels}} \times \overset{0}{\text{rows}} \times \overset{1}{\text{columns}}$

Altering tensor dimensions

Now dimensions are channels × rows × columns

But shouldn't it be sample × channels × rows × columns?

True

We can add an extra dimension, sized 1, using the unsqueeze-method

```
Im3 = im2.unsqueeze(0)
```

The "0" indicates that the dimension should be on index 0

Now the dimensions are

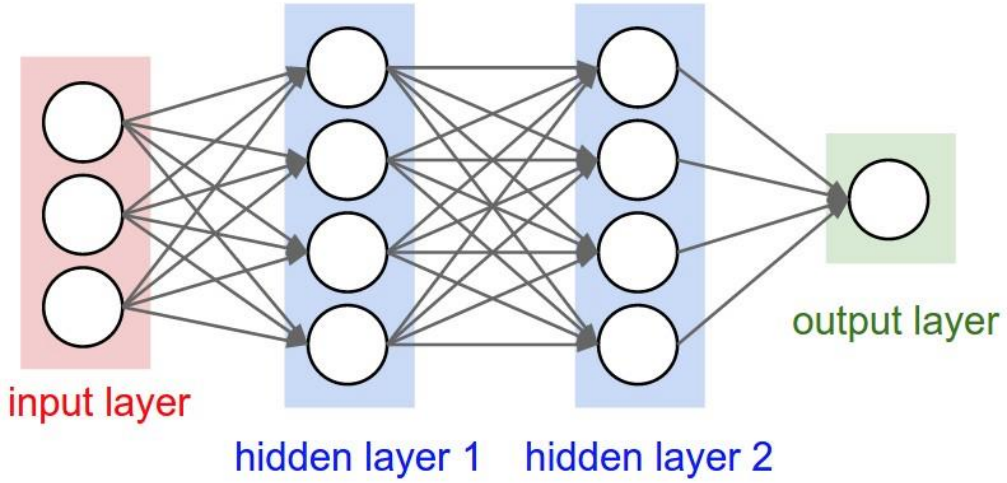
Sample × channels × rows × columns

Example: Padding & Convolution

•

MLP sum up

- Fully connected layers does not scale well
- An Image with a resolution of 200*200*3 gives 120,000 weights per neuron in the following layer
- MLPs are not invariant to translation (remember the displaced "7" digit or where's Waldo?)



Convolutional Neural Network Definition

A convolutional neural network (CNN) is a network in which at least one of the hidden layers are convolutions.

Convolution kernel weights are translation invariant. I.e., the kernel weights are the same at all locations on the image (weight sharing)

CNNs are typically used to handle image data

Architecture Overview - Layers

The following layers are by far the most popular ones in CNNs:

- **Convolutional Layer** (Conv)

  Compute output of neurons (filters) that are connected to local regions in the input.

- **Pooling Layer** (Pool)

  Downsamples images along spatial dimensions (Width, Height)

- **Fully Connected Layer** (FC)

  Computing class scores

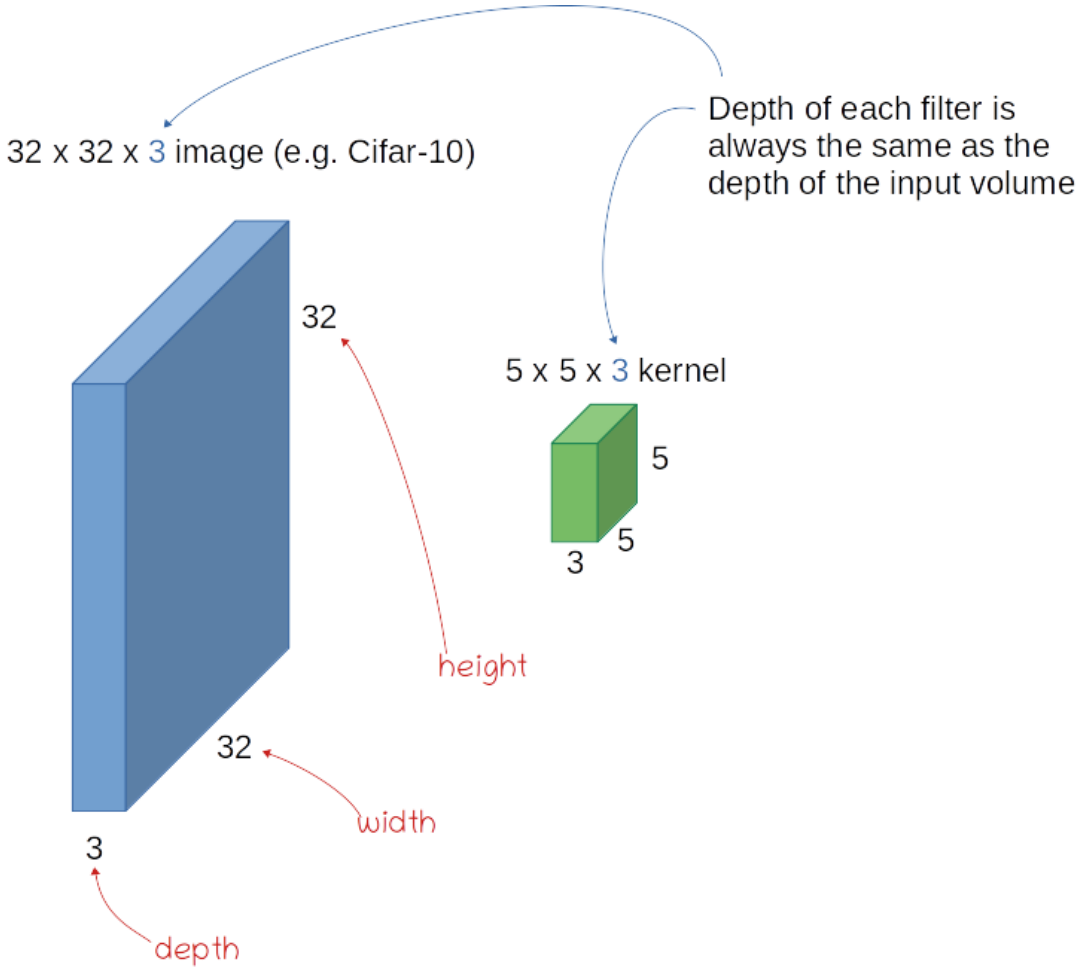
- **Activation Layer**

  E.g., ReLU

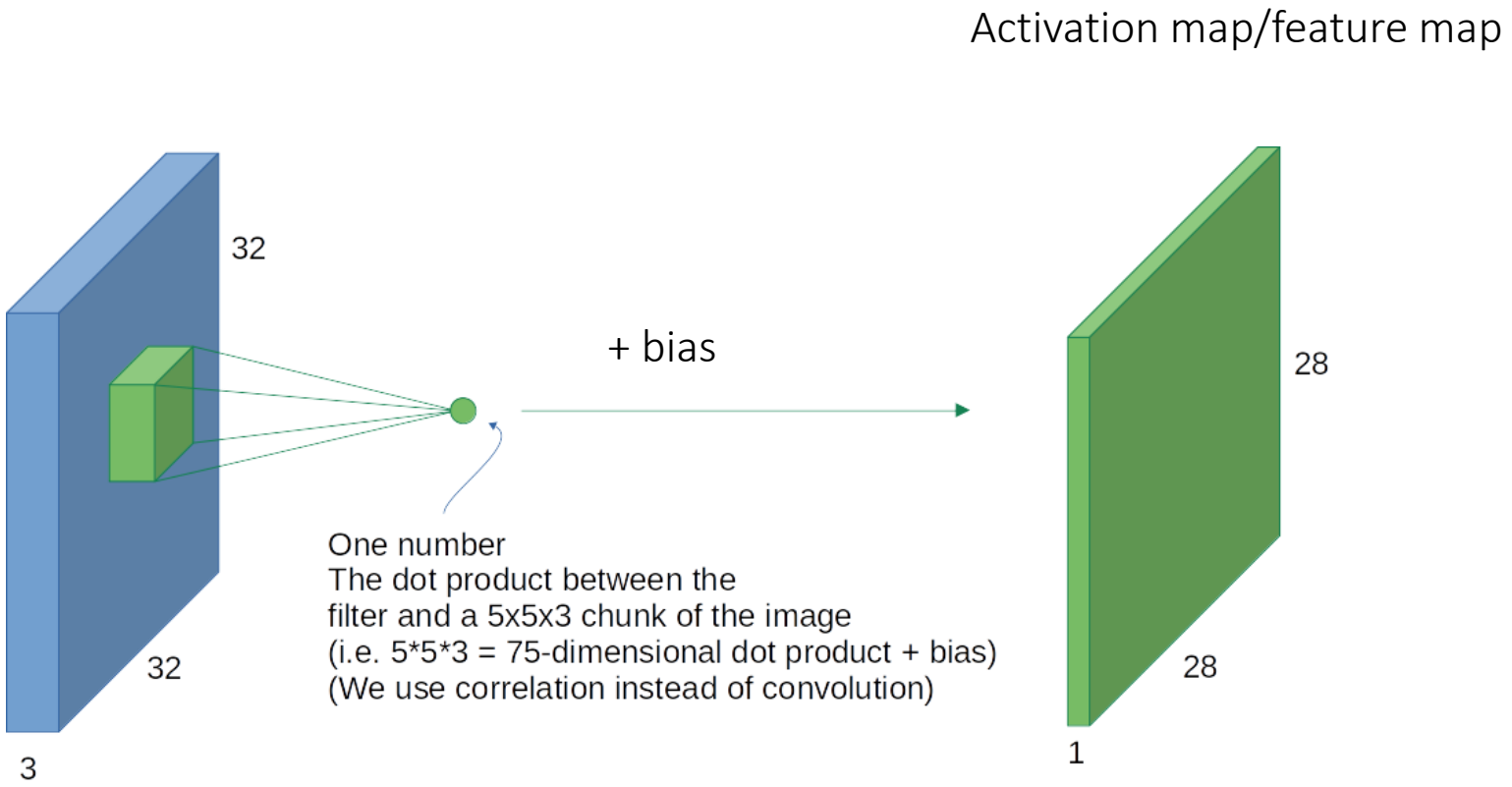
- Each Layer may or may not have parameters (e.g., CONV/FC do, RELU/POOL don't)

  Each Layer may or may not have additional hyperparameters (e.g., CONV/FC/POOL do, RELU doesn't)

Convolutional Layer with multi-layer input



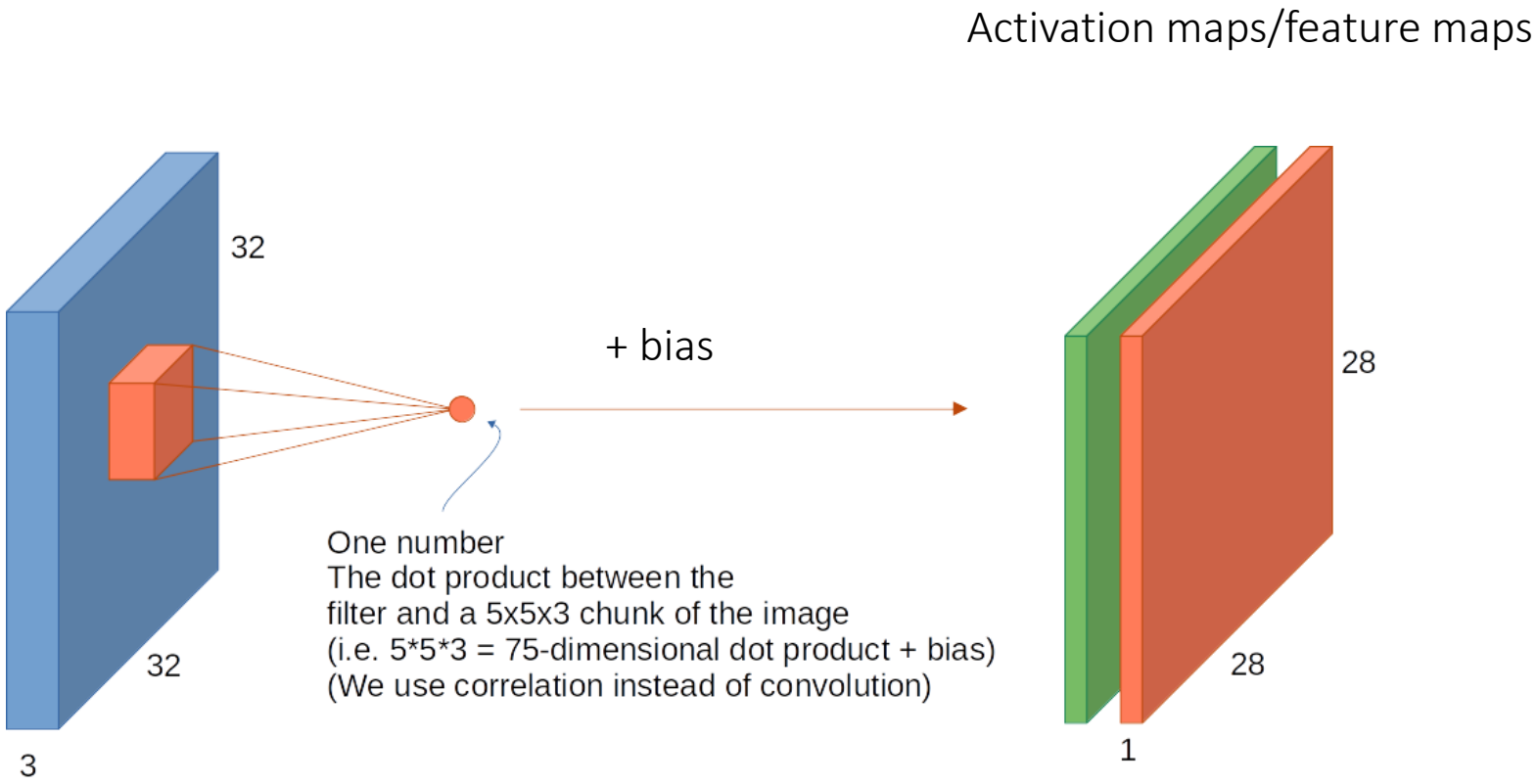
Convolutional Layer with multi-layer input



In this case, we use "valid" padding. Therefore, the two outer pixels are removed in the feature map.

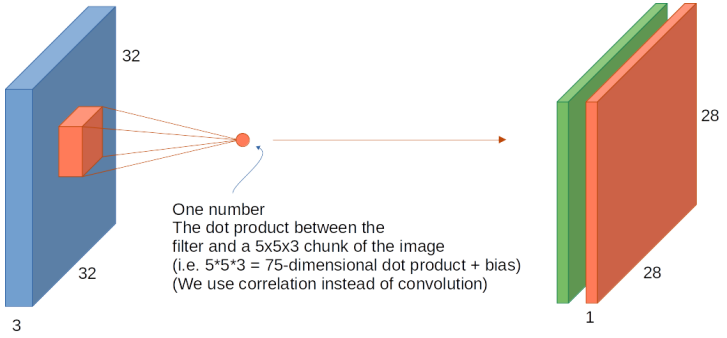
The spatial dimensions of the output is (m-f+1)x (n-f+1), where m,n are the width,height of the input volume, and f is the size of the filter

Convolutional Layer with multi-layer input



The number of kernels determines the number of layers in the activation map/feature map.
We will stack these activation maps along the depth dimension and produce the output volume.

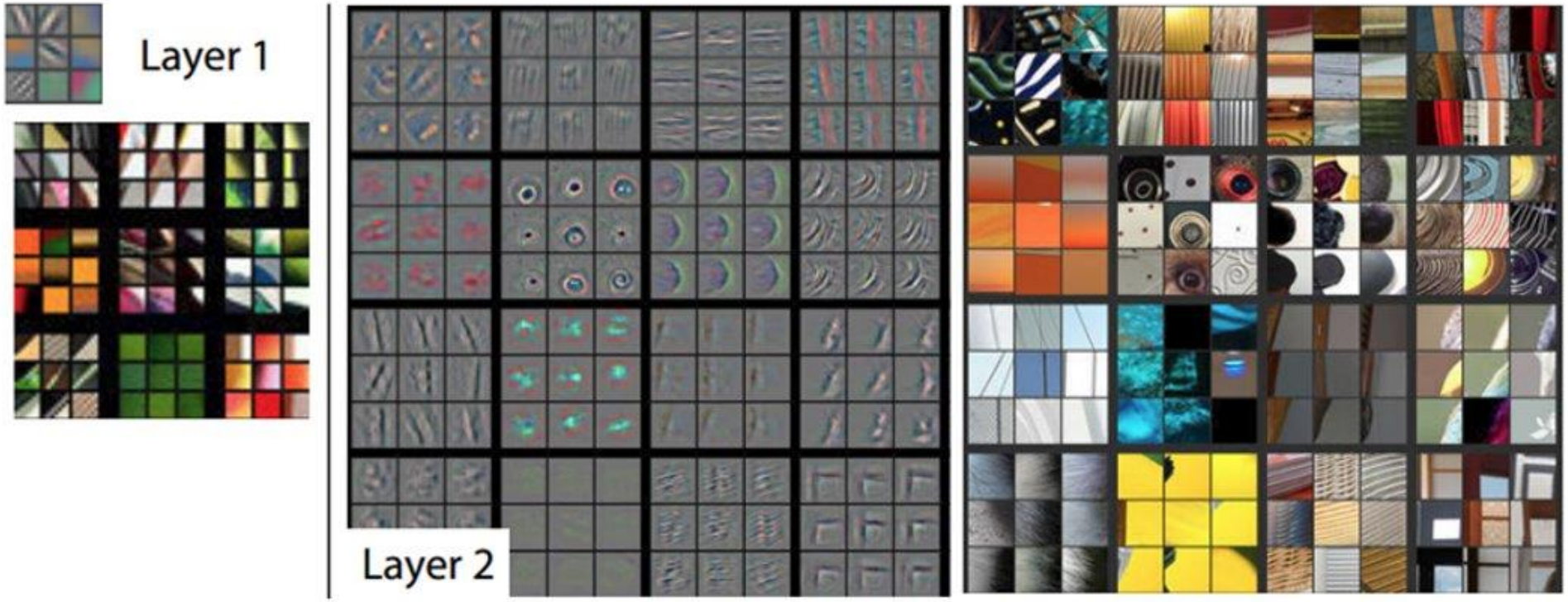
Number of parameters of a Convolutional Layer



Example 1.
An input volume has size [32x32x3], (e.g. an RGB CIFAR-10 image). If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights matching a [5x5x3] region in the input volume, for a total of 5*5*3 = 75 weights (and +1 bias parameter).

Example 2.
An input volume has size [16x16x20]. If the receptive field size of the filter is 3x3, every neuron in the Conv Layer would have a total of 3*3*20 = 180 connections to the input volume.

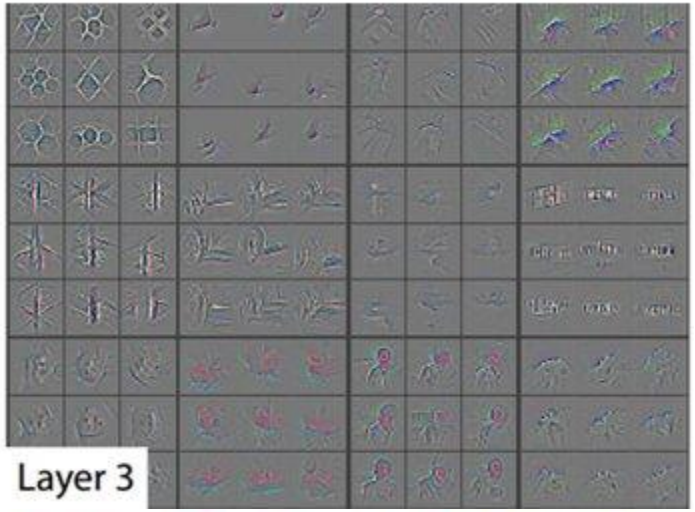
What does real filters look like?

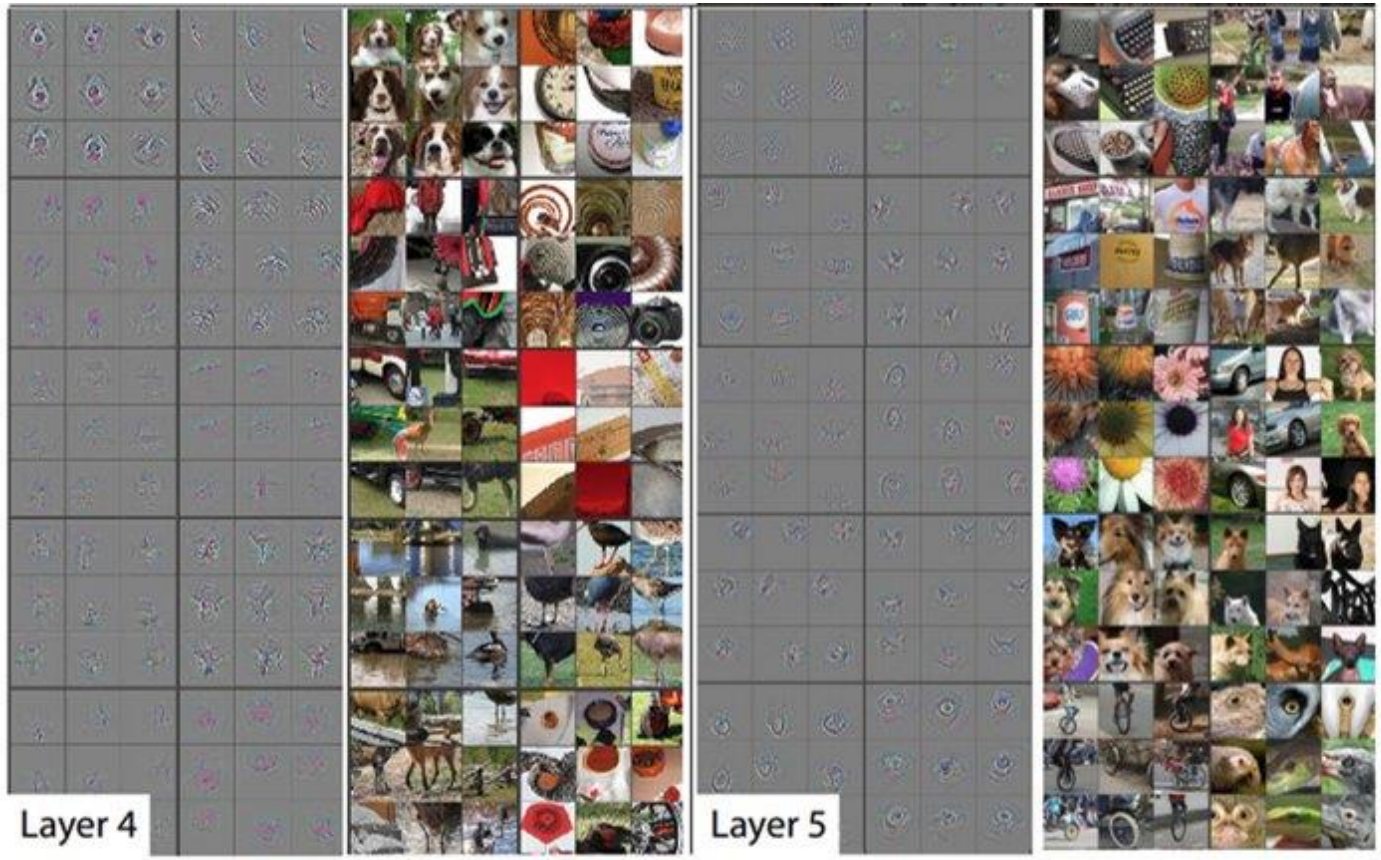


Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)

Matthew D Zeiler, Rob Fergus, Visualizing and Understanding Convolutional Networks

What does real filters look like?

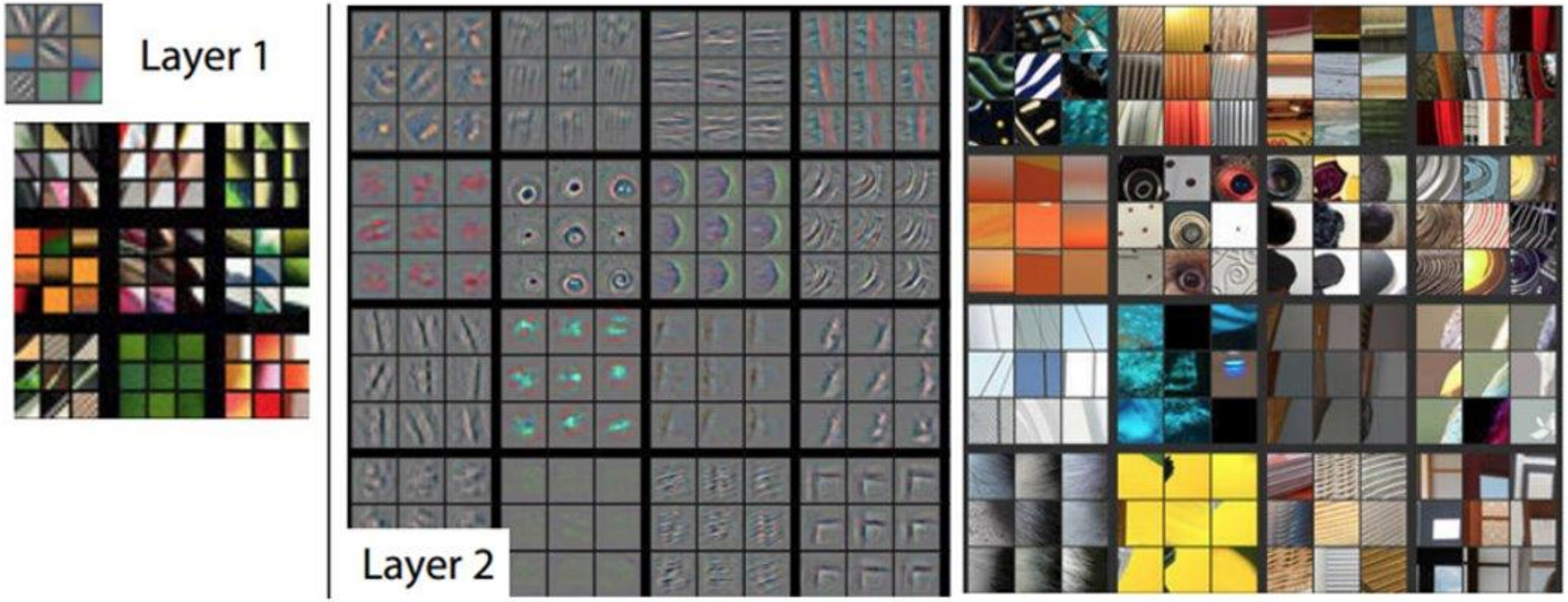




Visualizations of Layers 3, 4, and 5

Matthew D Zeiler, Rob Fergus, Visualizing and Understanding Convolutional Networks

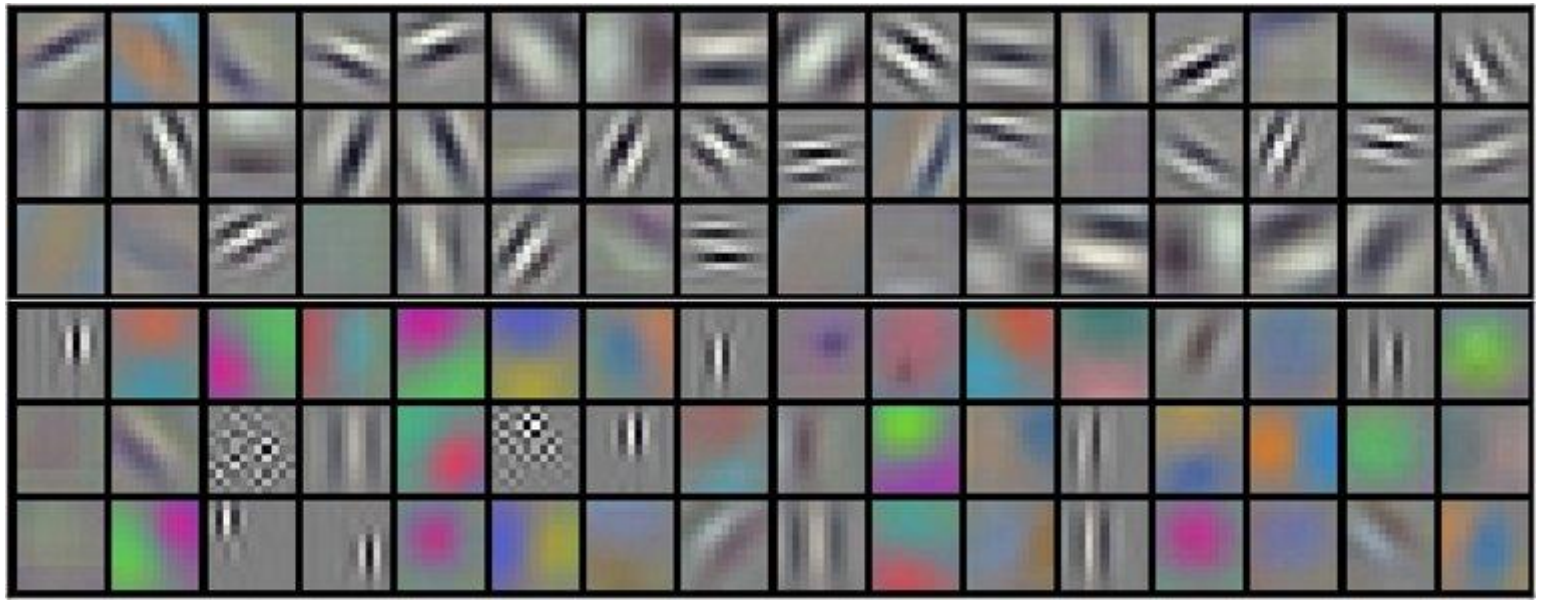
What does real filters look like?



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)

Matthew D Zeiler, Rob Fergus, Visualizing and Understanding Convolutional Networks

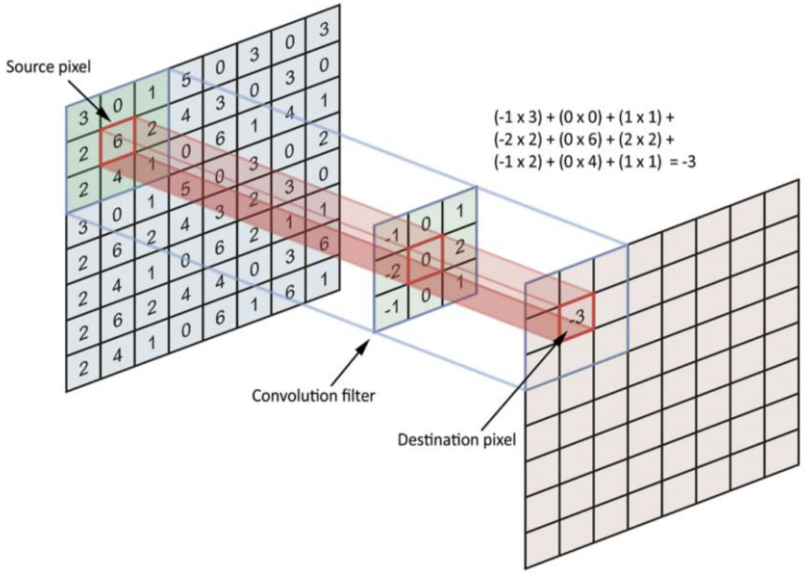
AlexNet weights (first layer)



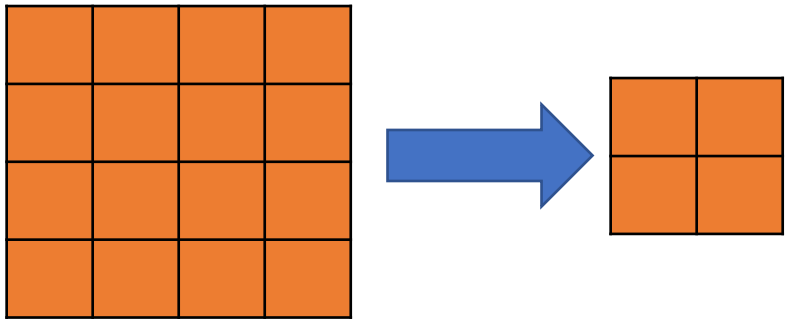
Convolution recap

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.

<u>Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a dot of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.</u>



Pooling layer



- Decreases the dimensions of feature maps
  - Makes the memory footprint smaller
  - Removes less important features
- Works independently in each channel (depth slice) of the input volume.
- Because of pooling, the convolutions of the next layer will cover a larger part of the input
- Pooling layers can be *local* and *global*:
  - Local pooling is the default. The output is based on a local input
  - Global pooling pools across the full feature map.
    - Usually only used once in a network since spatial information is lost
- There are two main types of pooling:
  - Max pooling
  - Average pooling
- Other types exist
  - Minimum pooling
  - Adaptive pooling (size depends on input size)

Max pooling

- Only the max value within an area of the input is send to the output.

1	9	8	3
4	1	7	2
8	6	4	6
3	2	2	5

9	8
8	6

Average pooling

- The average value within an area of the input is send to the output.

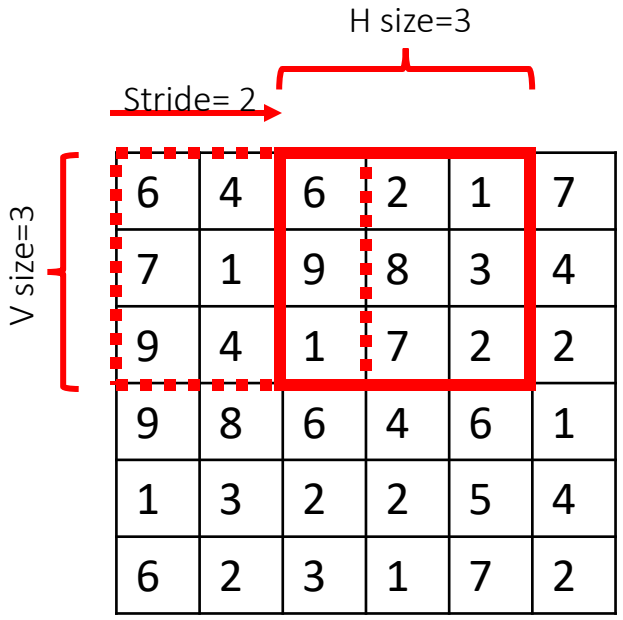
1	9	8	3
4	1	7	2
8	6	4	6
3	2	2	5

3,75	5
4,75	4,25

- Not often used, since similar results can be achieved by a convolution with all weights set to 1/n and stride

Pooling: Size and stride

- The pooling size and stride determines the size of the output.
- Pooling size: The area the pooling is applied over
- Pooling stride: The distance between areas
- The previous slides showed two cases of 2x2 pooling with a stride of 2



Global pooling (average)

- Global pooling is a special case, where the pooling kernel has the same width and height as the feature map
- Each layer outputs only one value
- (More about this concept in a few slides time)

1	9	8	3
4	1	7	2
8	6	4	6
3	2	2	5

→

4,4375

Dimension quiz

Given an input tensor with size
`(channels x height x width) = (10, 33, 33)`

Convolved with 3 filters of size
`(channels x height x width) = (10, 3, 3)` with stride `(2, 2)` and no padding

followed by a 2x2 max pooling layer with stride (2, 2)

What is the size of the output?

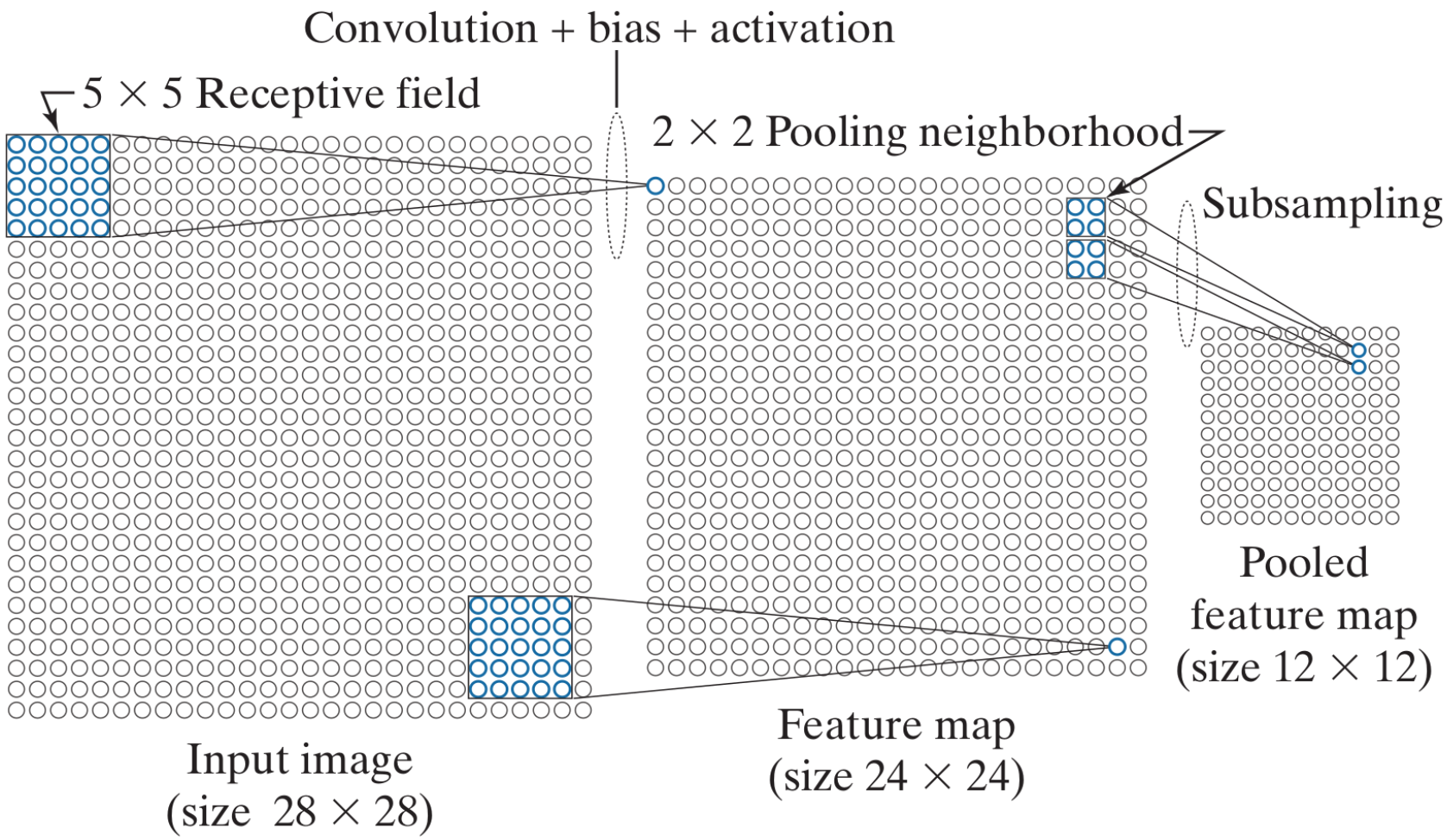
A (10x8x8)

B (3x8x8)

C (3x15x15)

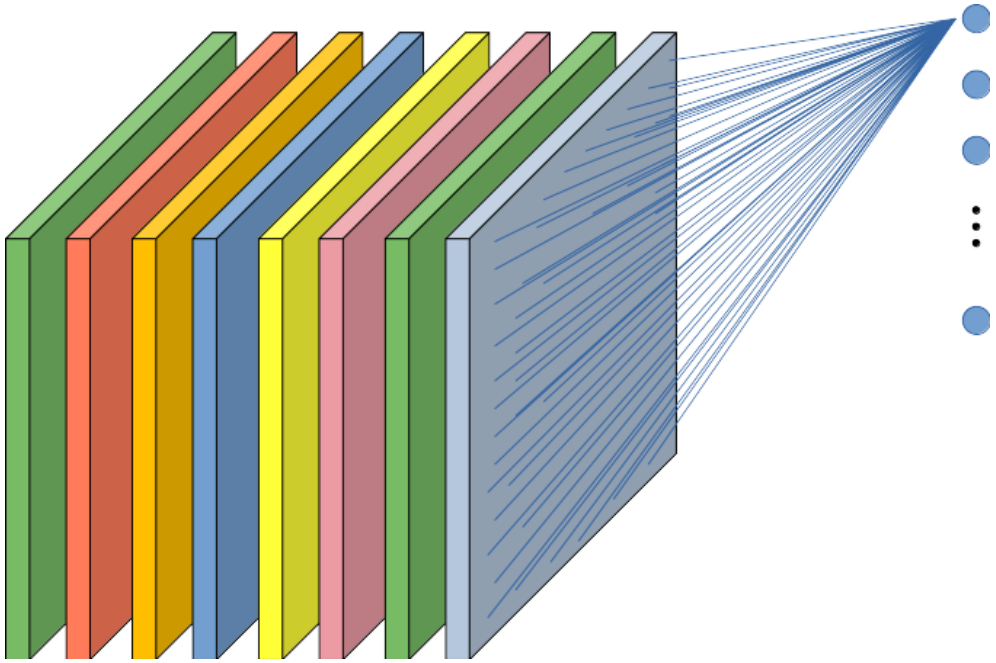
D (3x33x33)

Let's put together Convolution and pooling

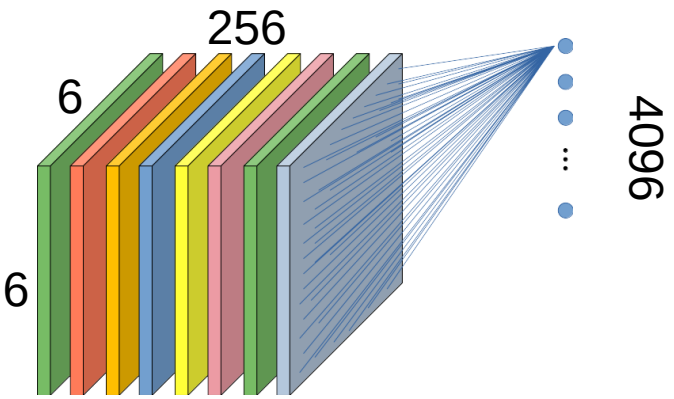


Fully connected

- A "classic" neural network.
- Sometimes called "dense"-layers
- Each neuron in the layer is connected to all values of the previous volume through a weight and bias
- (Usually) No weight sharing
- High memory footprint
- Often used to compute the class scores, resulting in volume of size [1 x 1 x N], where each of the N numbers correspond to a class score. (I.e., in MNIST, N=10)
- Locks the size of the input image (more about this later)

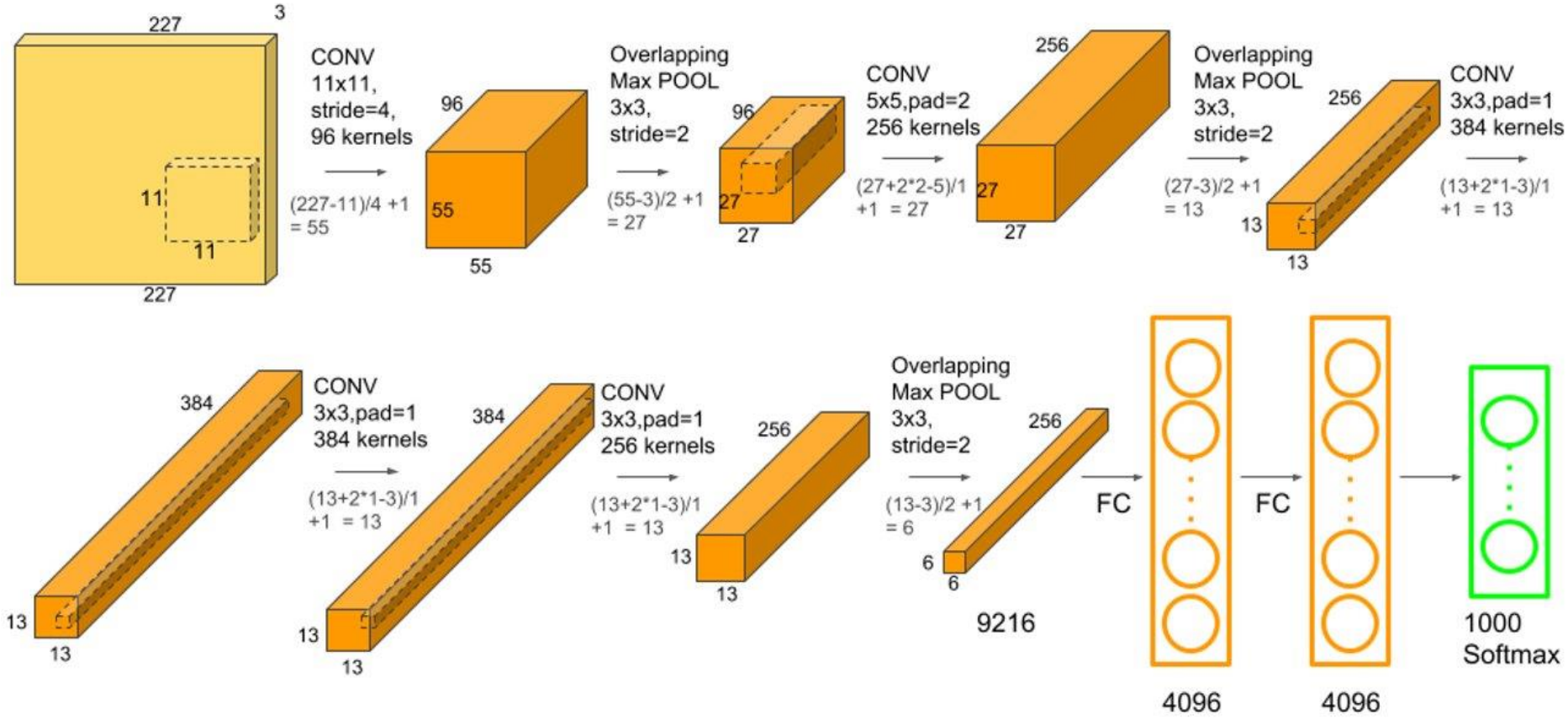


? Fully connected

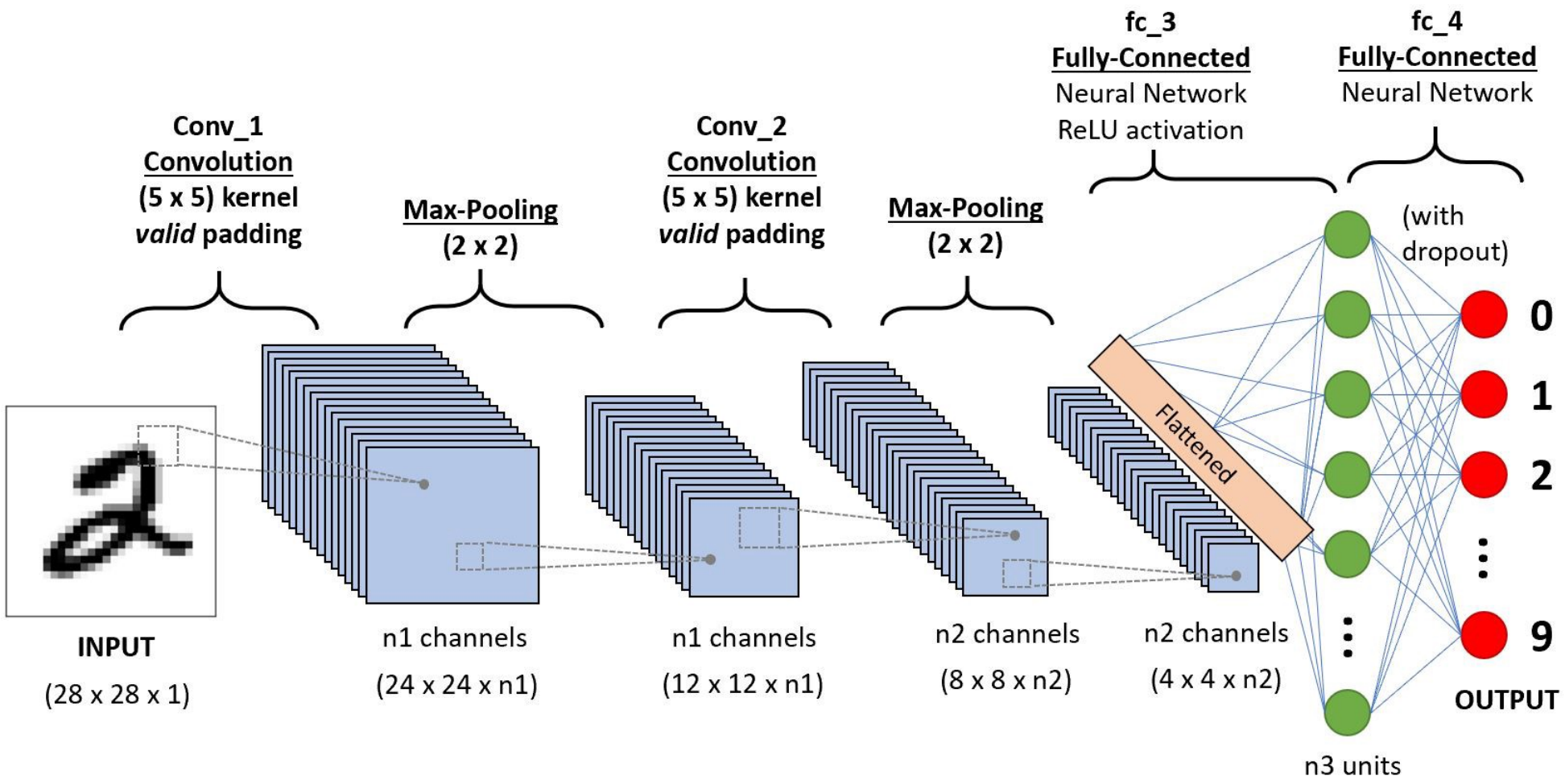


- If the input to a 4096 unit Fully connected layer is a feature map sized 6x6x256.
- How many parameters do we have?

Let's put it all together (AlexNet (2012))

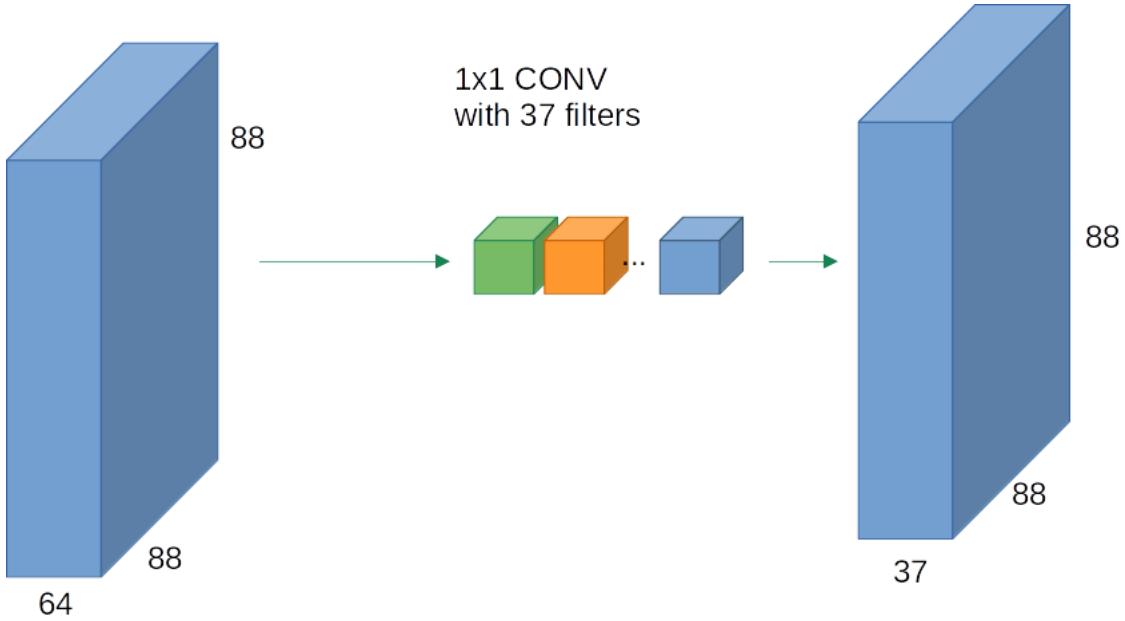


Case: Simple CNN for MNIST classification



1x1 convolution

- Several papers use 1x1 convolutions, as first investigated in the paper [Network in Network](#).
- At first, a 1x1 filter does not seem to make sense, since it is just a point-wise scaling. However, remember that the convolution operate over 3-dimensional volumes. Therefore, it can be used to change the depth of activation maps.



This allows us to get rid of fully connected layers (together with what is presented on the next slide)

Global average pooling (AdaptiveAvgPool2d)

- Alternative to the Fully Connected Layer(FC layer)
- Global average pooling returns the average value of each feature layer.
- Output is a vector with the same length as the number of layers.
- Often a 1x1 convolution can help to set the number of layers before Global average pooling
  - Useful if you just want to change the output layers without retraining other layers from scratch

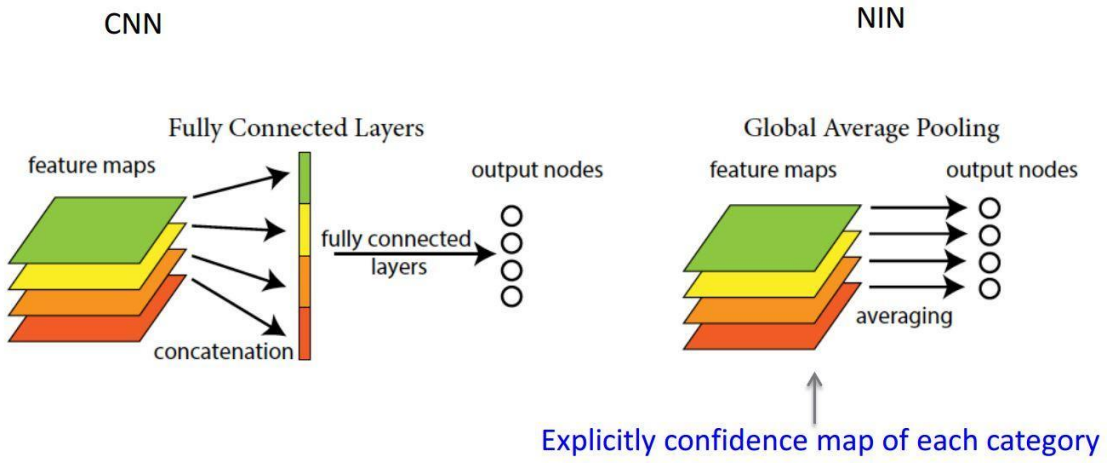


Image from: https://seungmi122.github.io/2021-02-11-global_avg_pooling_en/

Backpropagation of convolution (correlation)

The backward pass of a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). For simplicity we consider convolution and correlation to be the same.

Signal

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$*$

Filter

F_{11}	F_{12}
F_{21}	F_{22}

$=$

Output

O_{11}	O_{12}
O_{21}	O_{22}

$\longrightarrow E$

Backpropagation of convolution (correlation)

The backward pass of a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). For simplicity we consider convolution and correlation to be the same.

Signal

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$*$

Filter

F_{11}	F_{12}
F_{21}	F_{22}

$=$

Output

O_{11}	O_{12}
O_{21}	O_{22}

$\rightarrow E$

$$O_{11} = F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22}$$

Backpropagation of convolution (correlation)

The backward pass of a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). For simplicity we consider convolution and correlation to be the same.

Signal

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$*$

Filter

F_{11}	F_{12}
F_{21}	F_{22}

$=$

Output

O_{11}	O_{12}
O_{21}	O_{22}

$\longrightarrow E$

$$O_{11} = F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22}$$
$$O_{12} = F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23}$$

Backpropagation of convolution (correlation)

The backward pass of a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). For simplicity we consider convolution and correlation to be the same.

Signal

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$*$

Filter

F_{11}	F_{12}
F_{21}	F_{22}

$=$

Output

O_{11}	O_{12}
O_{21}	O_{22}

$\longrightarrow E$

$$O_{11} = F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22}$$
$$O_{12} = F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23}$$
$$O_{21} = F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{23}$$

Backpropagation of convolution (correlation)

The backward pass of a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). For simplicity we consider convolution and correlation to be the same.

Signal

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$*$

Filter

F_{11}	F_{12}
F_{21}	F_{22}

$=$

Output

O_{11}	O_{12}
O_{21}	O_{22}

$\rightarrow E$

$$O_{11} = F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22}$$
$$O_{12} = F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23}$$
$$O_{21} = F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{23}$$
$$O_{22} = F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33}$$

Backpropagation of convolution (correlation)

The backward pass of a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). For simplicity we consider convolution and correlation to be the same.

Signal

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$*$

Filter

F_{11}	F_{12}
F_{21}	F_{22}

$=$

Output

O_{11}	O_{12}
O_{21}	O_{22}

$\longrightarrow E$

$$
\begin{aligned}
O_{11} &= F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22} \\
O_{12} &= F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23} \\
O_{21} &= F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{23} \\
O_{22} &= F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33}
\end{aligned}
$$

To calculate the gradients of filter ‘F’ with respect to the error ‘E’, the following equations need to be solved.

$$
\begin{aligned}
\frac{\partial E}{\partial F_{11}} &= \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{11}} \\
\frac{\partial E}{\partial F_{12}} &= \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{12}} \\
\frac{\partial E}{\partial F_{21}} &= \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{21}} \\
\frac{\partial E}{\partial F_{22}} &= \frac{\partial E}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial E}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial E}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial E}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{22}}
\end{aligned}
$$

which evaluates to

$$
\begin{aligned}
\frac{\partial E}{\partial F_{11}} &= \frac{\partial E}{\partial O_{11}}X_{11} + \frac{\partial E}{\partial O_{12}}X_{12} + \frac{\partial E}{\partial O_{21}}X_{21} + \frac{\partial E}{\partial O_{22}}X_{22} \\
\frac{\partial E}{\partial F_{12}} &= \frac{\partial E}{\partial O_{11}}X_{12} + \frac{\partial E}{\partial O_{12}}X_{13} + \frac{\partial E}{\partial O_{21}}X_{22} + \frac{\partial E}{\partial O_{22}}X_{23} \\
\frac{\partial E}{\partial F_{21}} &= \frac{\partial E}{\partial O_{11}}X_{21} + \frac{\partial E}{\partial O_{12}}X_{22} + \frac{\partial E}{\partial O_{21}}X_{31} + \frac{\partial E}{\partial O_{22}}X_{32} \\
\frac{\partial E}{\partial F_{22}} &= \frac{\partial E}{\partial O_{11}}X_{22} + \frac{\partial E}{\partial O_{12}}X_{23} + \frac{\partial E}{\partial O_{21}}X_{32} + \frac{\partial E}{\partial O_{22}}X_{33}
\end{aligned}
$$

Backpropagation of convolution

The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters).

The gradients of filter ‘F’ with respect to the error ‘E’:

Signal

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$*$

Filter

$\frac{\partial E}{\partial O_{11}}$	$\frac{\partial E}{\partial O_{12}}$
$\frac{\partial E}{\partial O_{21}}$	$\frac{\partial E}{\partial O_{22}}$

$=$

$\frac{\partial E}{\partial F_{11}}$	$\frac{\partial E}{\partial F_{12}}$
$\frac{\partial E}{\partial F_{21}}$	$\frac{\partial E}{\partial F_{22}}$

$$\frac{\partial E}{\partial F_{11}} = \frac{\partial E}{\partial O_{11}} X_{11} + \frac{\partial E}{\partial O_{12}} X_{12} + \frac{\partial E}{\partial O_{21}} X_{21} + \frac{\partial E}{\partial O_{22}} X_{22}$$

$$\frac{\partial E}{\partial F_{12}} = \frac{\partial E}{\partial O_{11}} X_{12} + \frac{\partial E}{\partial O_{12}} X_{13} + \frac{\partial E}{\partial O_{21}} X_{22} + \frac{\partial E}{\partial O_{22}} X_{23}$$

$$\frac{\partial E}{\partial F_{21}} = \frac{\partial E}{\partial O_{11}} X_{21} + \frac{\partial E}{\partial O_{12}} X_{22} + \frac{\partial E}{\partial O_{21}} X_{31} + \frac{\partial E}{\partial O_{22}} X_{32}$$

$$\frac{\partial E}{\partial F_{22}} = \frac{\partial E}{\partial O_{11}} X_{22} + \frac{\partial E}{\partial O_{12}} X_{23} + \frac{\partial E}{\partial O_{21}} X_{32} + \frac{\partial E}{\partial O_{22}} X_{33}$$

which turns out to just be a convolution :-)

Backpropagation of convolution

Now we have the gradients of the filter.

We can also find the gradients of input ‘X’ with respect to the error ‘E’, which we use for calculating the gradients of filters present in previous layer:

$$\frac{\partial E}{\partial X_{11}} = \frac{\partial E}{\partial O_{11}} F_{11} + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{12}} = \frac{\partial E}{\partial O_{11}} F_{12} + \frac{\partial E}{\partial O_{12}} F_{11} + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{13}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} F_{12} + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{21}} = \frac{\partial E}{\partial O_{11}} F_{21} + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} F_{11} + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{22}} = \frac{\partial E}{\partial O_{11}} F_{22} + \frac{\partial E}{\partial O_{12}} F_{21} + \frac{\partial E}{\partial O_{21}} f_{12} + \frac{\partial E}{\partial O_{22}} F_{11}$$

$$\frac{\partial E}{\partial X_{23}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} F_{22} + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} F_{11}$$

$$\frac{\partial E}{\partial X_{31}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} F_{21} + \frac{\partial E}{\partial O_{22}} 0$$

$$\frac{\partial E}{\partial X_{32}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} F_{22} + \frac{\partial E}{\partial O_{22}} F_{21}$$

$$\frac{\partial E}{\partial X_{33}} = \frac{\partial E}{\partial O_{11}} 0 + \frac{\partial E}{\partial O_{12}} 0 + \frac{\partial E}{\partial O_{21}} 0 + \frac{\partial E}{\partial O_{22}} F_{22}$$

This turns out to also just be a convolution with a zero-padded filter that is rotated 180°.

$$\begin{bmatrix} \frac{\partial E}{\partial O_{11}} & \frac{\partial E}{\partial O_{12}} \\ \frac{\partial E}{\partial O_{21}} & \frac{\partial E}{\partial O_{22}} \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & F_{22} & F_{21} & 0 \\ 0 & F_{12} & F_{11} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial X_{11}} & \frac{\partial E}{\partial X_{12}} & \frac{\partial E}{\partial X_{13}} \\ \frac{\partial E}{\partial X_{21}} & \frac{\partial E}{\partial X_{22}} & \frac{\partial E}{\partial X_{23}} \\ \frac{\partial E}{\partial X_{31}} & \frac{\partial E}{\partial X_{32}} & \frac{\partial E}{\partial X_{33}} \end{bmatrix}$$

Hence both the forward and backward propagation can be performed using the convolution operation.

NB. This is already implemented in autograd in pytorch

Backpropagation of max pooling

Max-pooling is defined as

$$y = \max(x_1, x_2, \cdots, x_n)$$

Which we can also write as

$$y = \sum_{i=1}^{n} w_i x_i \qquad \text{where} \qquad w_i = \begin{cases} 1 & \text{if } x_i = \max(x_1, x_2, \cdots, x_n) \\ 0 & \text{otherwise} \end{cases}$$

Here, the gradient to each neuron is

$$\frac{\partial y}{\partial x_i} = w_i = \begin{cases} 1 & \text{if } x_i = \max(x_1, x_2, \cdots, x_n) \\ 0 & \text{otherwise} \end{cases}$$

I.e., the backward-pass of a max(x,y) -operation only routes the gradient to the input that has the highest value in the forward pass.

During the forward-pass of a pooling layer you would, therefore, keep track of the index of the max value

NB. This is already implemented in autograd in pytorch

Weight initialization in CNN

- Weights should be initialized to non-zero values. Why?

Weight initialization in CNN

- Weights should be initialized to non-zero values. Why?

If all weights are zero, their contribution to the loss is equal -> the derivative with respect to the loss is the same for every weight -> all layers will stay equal after the update

- If the weights are too small, then the signal shrinks as it passes through the network until it is too small to be useful.
- If the weights are too large, the signal grows as it passes through the network until it is too massive to be useful.
- A good starting point is random initialization with zero-mean and std. division of 1.
- Common initialization strategies are:
    - Xavier initialization
    - He initialization
    - Transfer learning (More about this later)

Convnet in PyTorch

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):   # Here, we define layers that we want to train (store their weights)
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        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):  # Here, we define our architecture
        x = self.conv1(x)
        x = F.relu(x)   # Since we don't train the ReLu, it is not defined in __init__
        x = self.pool(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.pool(x)
        x = torch.flatten(x, 1) # flatten all dimensions except batch-dimension
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        return x


net = Net()
```

We could build convolutional neural networks from scrach in PyTorch (there are tutorials at PyTorch's website), but PyTorch provide a helper class, **nn.Module**, that we can use to define our network.

The nn.Module will help us create clearer and more compact code by encapsulating the network in a class. It also helps us, if we want to save our trained weights and load them again.

Exercise

MLP architecture:
https://colab.research.google.com/drive/12rDMtQbo6akjjhczZ4l2UX-kyF0Gq113?usp=sharing

Exercise: Modify the MLP code to make a CNN:
https://colab.research.google.com/drive/1tNRU3PsPtLQVi8eumda_5KV3guxnPzT8?usp=sharing