Artificial Neural Networks

Johanni Brea & W. Gerstner

EPFL, Lausanne, Switzerland

Inductive Bias and Convolutional Neural Networks

Part 1: No free lunch, Inductive Bias, and Images

Objectives for today:

- No free Lunch
- Inductive bias in machine learning
- Convolution filters as inductive bias for images
- Max Pooling for local translation invariance
- ImageNet competition and modern ConvNets
- Training ConvNets with AutoDiff

Previous slide.

This lecture has been prepared and taught several times by Dr. Johanni Brea (EPFL) for the class ‘Artificial Neural Networks’ at EPFL.

Reading for this lecture:

Goodfellow et al.,2016 *Deep Learning*

- Ch **9 + 7.4**

Further (optional) reading/watching for this lecture:

K. Fukushima, S. Miyake, **Neocognitron: a new algorithm for pattern recognition tolerant of deformations and shifts in position**
Pattern Recognition, 15 (6) (1982), pp. 455-469

Geoff Hinton, Q&A https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton/clyj4jv/

Springenberg et al. , Striving for simplicity: the all convolutional net https://arxiv.org/abs/1412.6806

Patel, Nguyen, Baraniuk, A probabilistic framework for deep learning, 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain. http://papers.nips.cc/paper/6231-a-probabilistic-framework-for-deep-learning

Gatys, Ecker, Bethge "A Neural Algorithm of Artistic Style" (2015).

Robert Geirhos et al, ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness (2019(

See also references in the slides as well as
Lectures by Andrew Ng

Previous slide

Reading: as usual the textbook on Deep Learning.

The viewpoint taken in this lecture puts more emphasis on the inductive bias.

Some of the points are controversial and reflect the bias of the teachers.

Artificial Neural Networks

Inductive Bias and Convolutional Neural Networks

Wulfram Gerstner

EPFL, Lausanne, Switzerland

Part 1: No Free Lunch Theorem

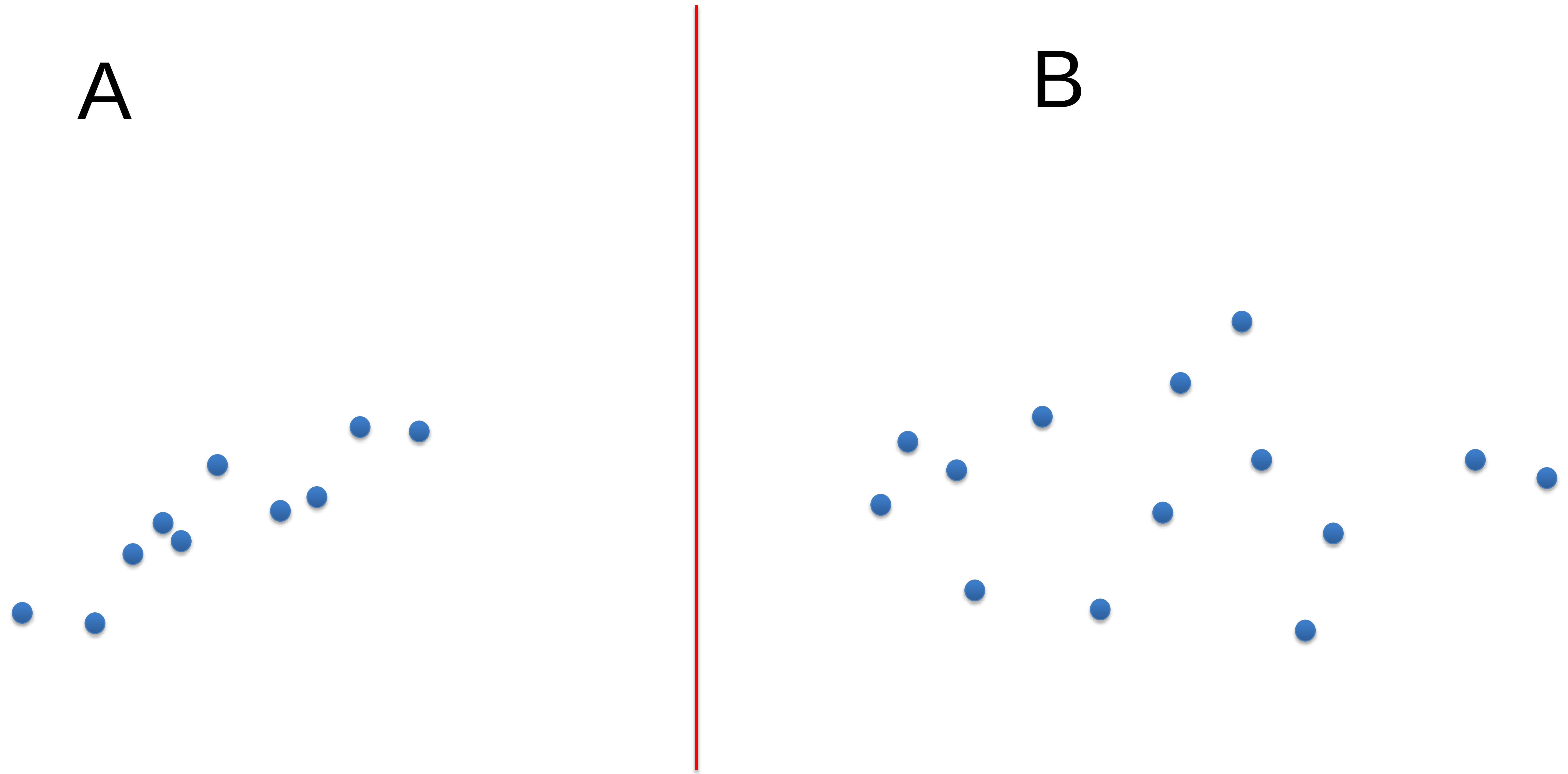
Previous slide.
No Free Lunch theorems (there are several variants) are foundational and philosophically important to answer the question: why do deep neural networks work so well?

The video for this part can be found on
https://lcnwww.epfl.ch/gerstner/VideoLecturesANN-Gerstner.html

Under
‘Deep Learning Lecture 3, part 6 (No free lunch theorems)

No Free Lunch Theorem

Which data set looks more noisy?



Commitment:
Thumbs up

Which data set is easier to fit?

Commitment:
Thumbs down

Previous slide.

Let us start with two data sets.

No Free Lunch Theorem

Previous slide.

And here a possible explanation (hidden behind the blue boxes).

No Free Lunch Theorem

Your notes

No Free Lunch Theorem

The NO FREE LUNCH THEOREM states
“ *that any two optimization algorithms are equivalent when their performance is averaged across* ***all possible problems***"

See Wikipedia/wiki/No_free_lunch_theorem

- Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation* **1**, 67.
- Wolpert, David (1996), "The Lack of *A Priori* Distinctions between Learning Algorithms", *Neural Computation*, pp. 1341-1390.

Previous slide.

The conclusion is: there is no reason to believe that an algorithm that works well on one data set will also work well on an arbitrarily chosen other data set.

No Free Lunch (NFL) Theorems

The mathematical statements are called

“NFL theorems because they demonstrate that if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems”

See Wikipedia/wiki/No_free_lunch_theorem

- •Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation* **1**, 67.
- •Wolpert, David (1996), "The Lack of *A Priori* Distinctions between Learning Algorithms", *Neural Computation*, pp. 1341-1390.

Previous slide.

Even worse, if the algo works well on some problem, there must exist another problem on which the algorithm works badly.

Quiz: No Free Lunch (NFL) Theorems

Take a neural networks with many layers, and many neurons, optimized by Backprop (with momentum/ADAM) as an example of deep learning

[ ] Deep learning performs better than most other algorithms on real world problems.

[ ] Deep learning can approximate to arbitrary degree any non-ambiguous (noise-free) data set (universal approximator theor)

[ ] Deep learning performs better than other algorithms on all problems.

Your notes.

No Free Lunch (NFL) Theorems

- Choosing a deep network and optimizing it with gradient descent is an algorithm

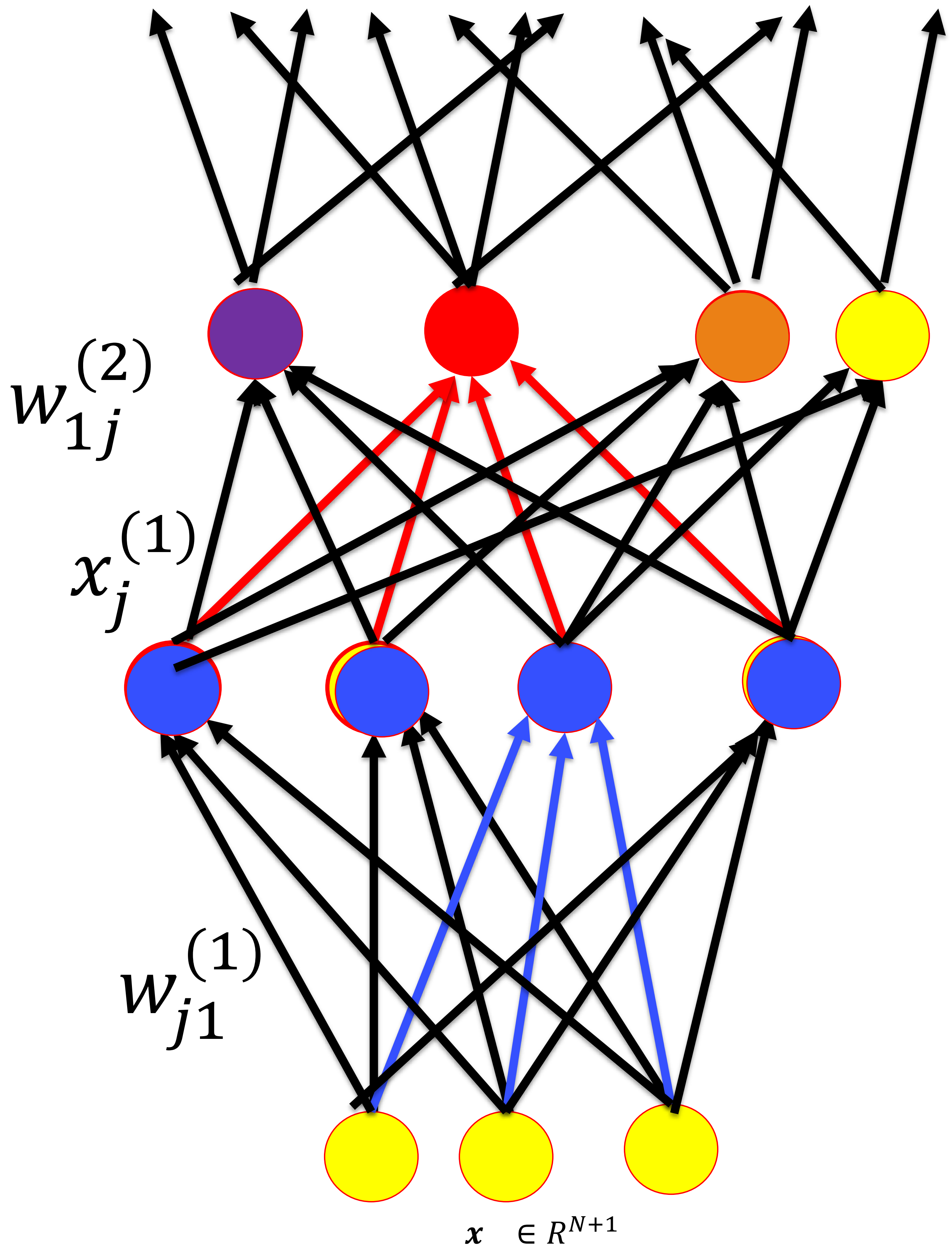
- Deep learning works well on many real-world problems

- Somehow the prior structure of the deep network matches the structure of the real-world problems we are interested in.

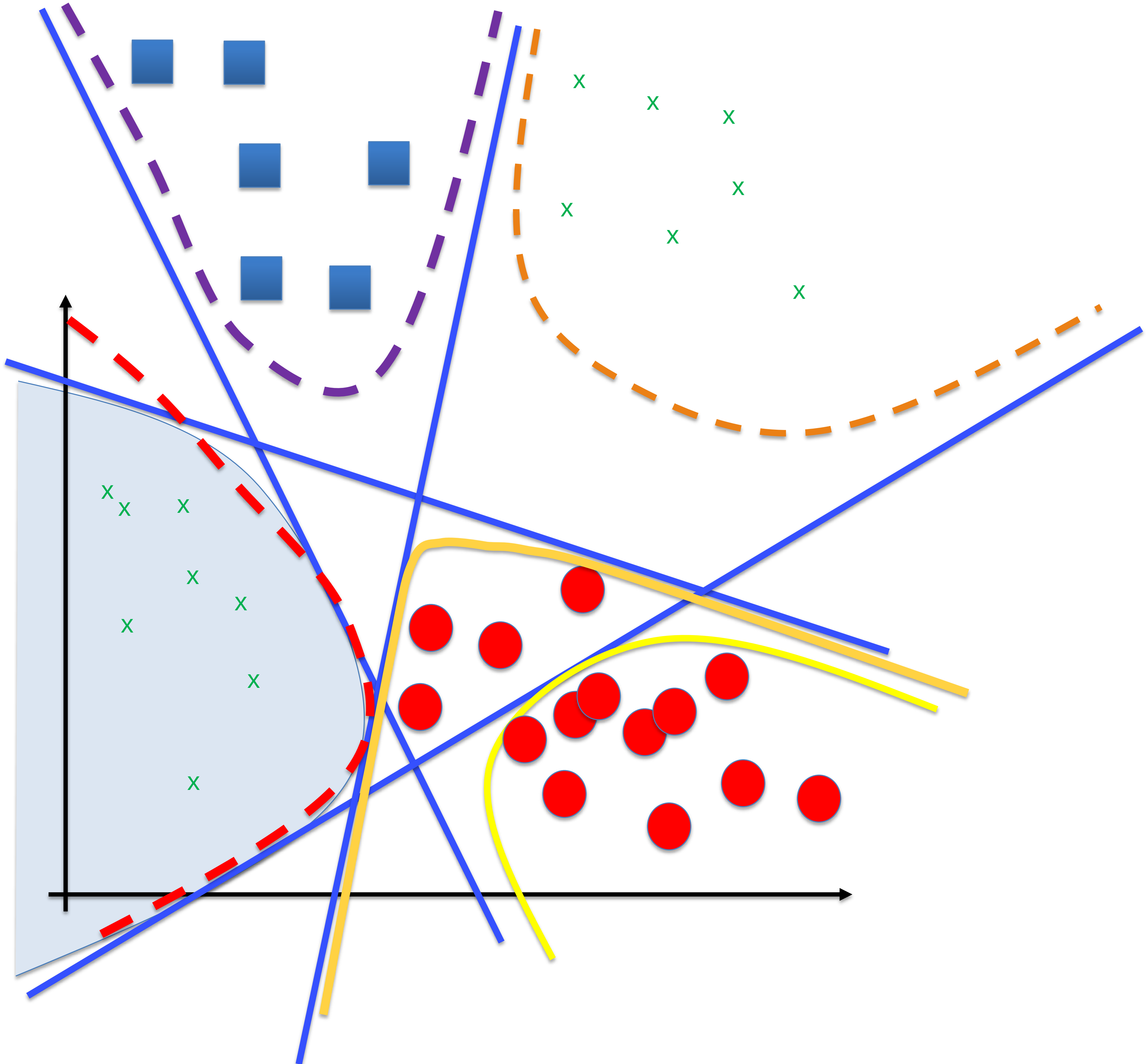
Previous slide.

The reason that deep networks work well must be linked to the type of data on which we test them.

No Free Lunch (NFL) Theorems



Geometry of the information flow in neural networks



Previous slide/next slide.

One possible explanation of why neural networks work well is the notion of hyperplanes. Even though the data is local, you make a cut through the whole space. This predefines additional ‘compartments’ that can be reused later for other data.

This argument might be applicable in the last few layers before the output. Suppose we look at layer 47 in a network of 50 layers. The previous layers have extracted high-level features (such as leg-detectors, fur-detectors etc). The last 3 layers before the output can then recombine these features in various ways to classify all sorts of animals.

Reuse of features in Deep Networks (schematic)

animals

birds

4 legs

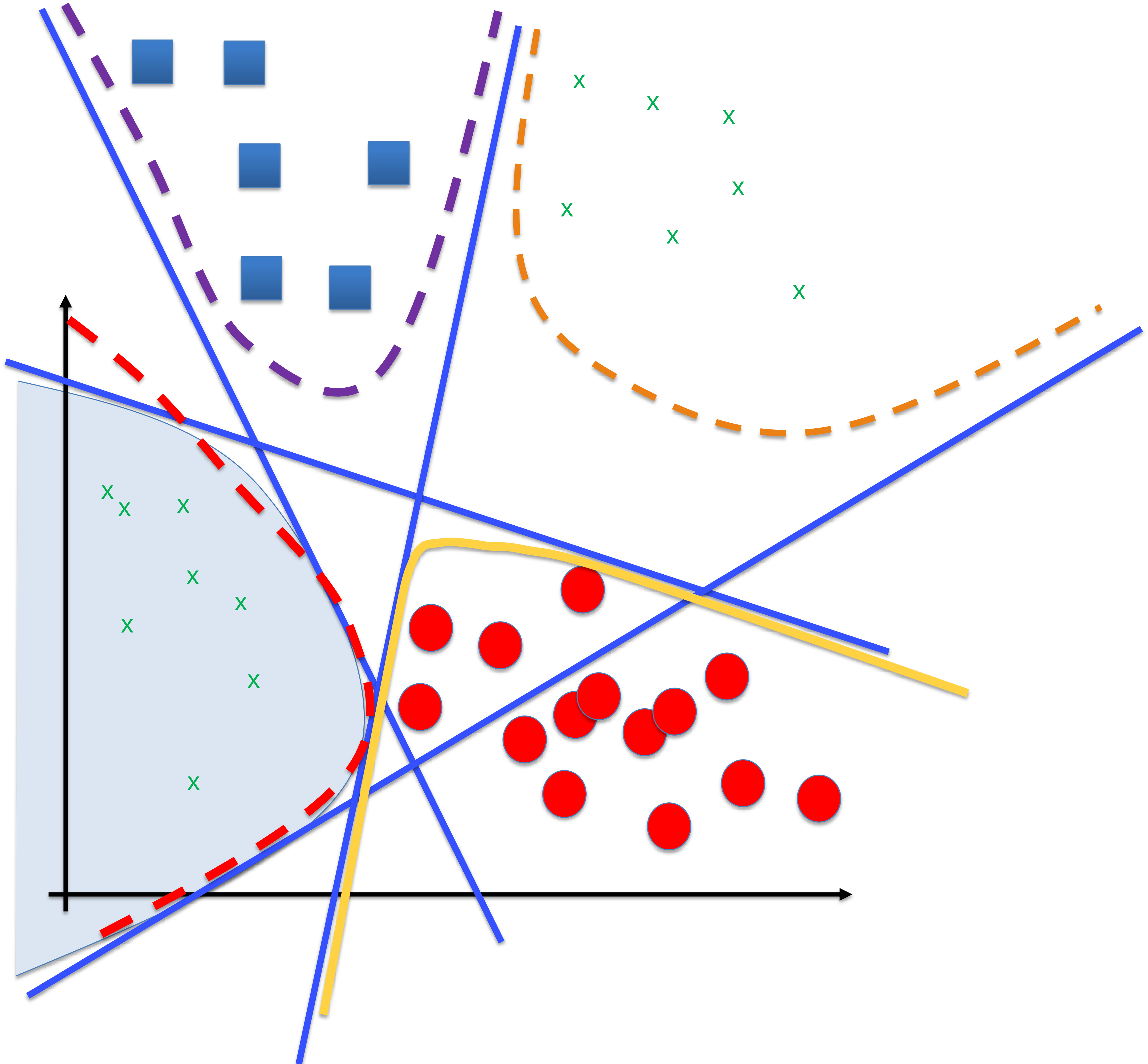
wings

snout

fur

eyes

tail



Summary: No Free Lunch (NFL) Theorems and Deep Networks

Somehow the prior structure of the deep network
matches the structure of the real-world problems
we are interested in.

The above example is applicable to layers close to the output,
- relevant features have been extracted in earlier layers
- the network can recombine these in various ways

A good representation (in layers close to the output) ensures
- similar = neighbor in high-dimensional feature space

→This is the reason why transfer learning works:
- train a deep network on one data base (e.g. imageNet)
- retrain only a few layers close to output for new task

Summary: No Free Lunch (NFL) Theorems and Deep Networks

More generally the prior structure of a deep network
should match the structure of the real-world problems
we are interested in.

→ Always use prior knowledge if you have some!

Example: - images, translation invariance
- music, tone translation invariance
- known symmetries of tasks

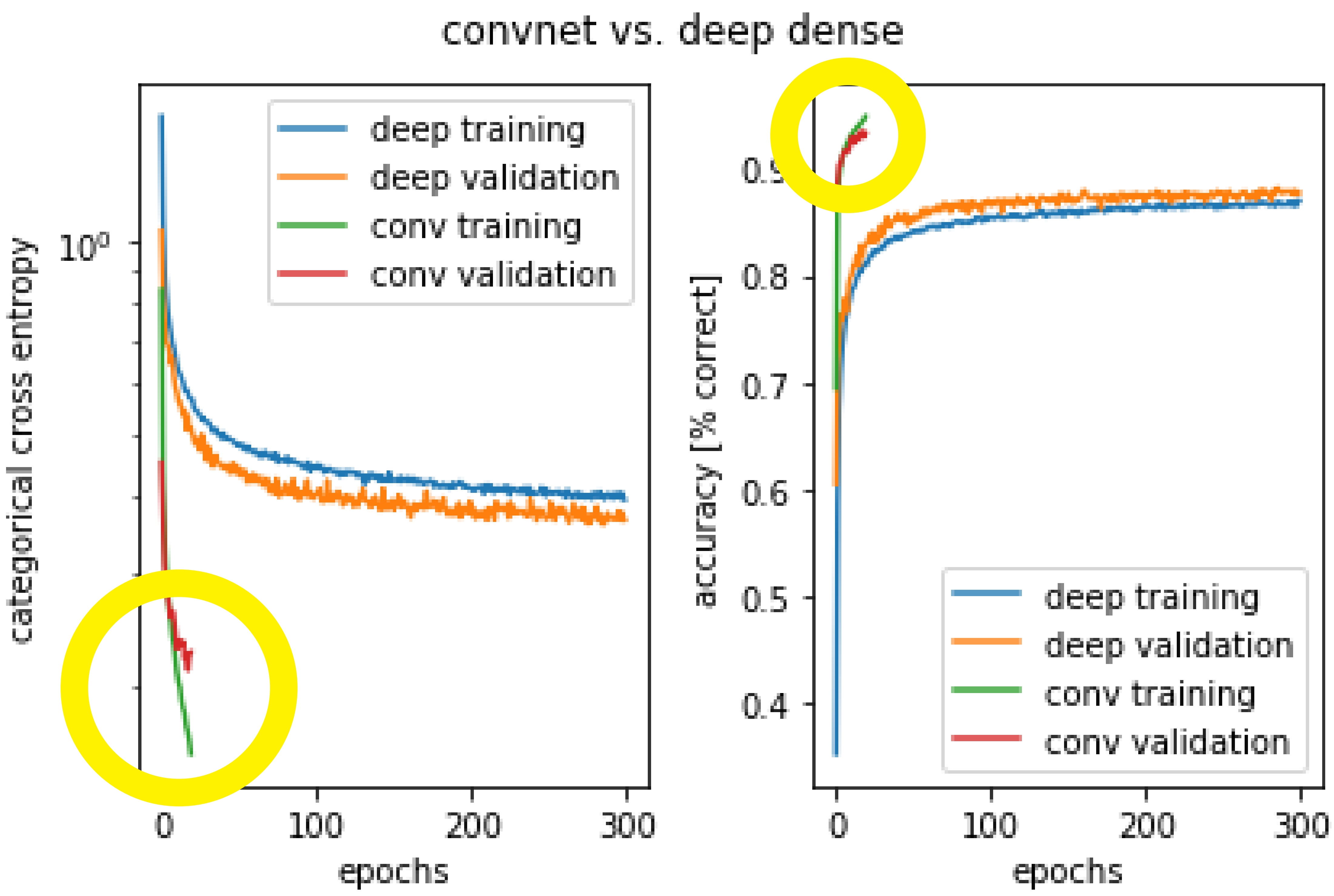
→ Use prior knowledge as ‘inductive bias’ in your algorithm!

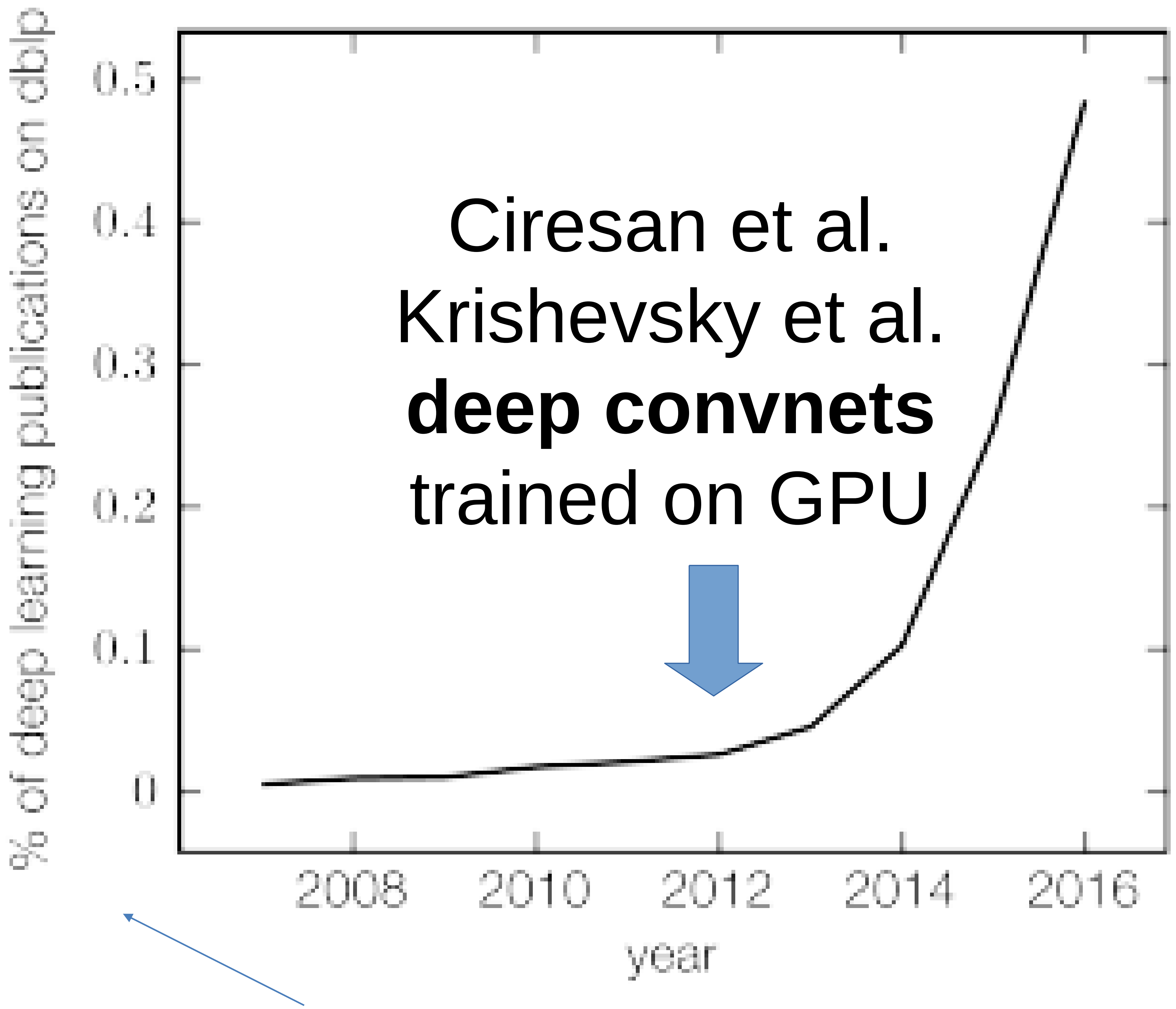
Previous slide.

Prior knowledge is important. We can use prior knowledge when we design the network architecture.

Motivation: Convolutional networks (convnet) work well!

An image recognition task





Ciresan et al.
Krishevsky et al.
deep convnets
trained on GPU

Fukushima (1982): Neocognitron
McClelland et al. (1996): Parallel Distributed Processing

Previous slide.

Left: Networks with convolutional layers reach significantly better training and test performances on image recognition than those without, and converge more rapidly.

Right: Convolutional networks have been around since 1982 (Fukushima: Neocognitron) and multi-layer networks since the early 1990ies. But the “deep learning revolution” started only around 2012, after the publication of several really deep convolutional networks that out-performed other approaches on e.g. difficult object recognition tasks. Afterwards the number of deep learning publications increased dramatically.

Convolutional networks (convnet) work well on images

Goal for today:

- why do they work well?
  → answer: induce a good inductive bias
- what is this inductive bias?
  → translation invariance of objects
- how can we construct convolutational networks?
- how can we train convolutational networks?

Previous slide.

We will work in several steps:

- First we will identify the role of inductive bias, and explain the term
- The we will show that translation invariance of objects leads to a strong inductive bias
- Then we will explain convolutational networks and indicate why they are a convenient way to

Artificial Neural Networks

Johanni Brea & W. Gerstner

EPFL, Lausanne, Switzerland

Inductive Bias and Convolutional Neural Networks

Part 2A: Inductive Bias in Machine Learning (Introduction)

1. No Free Lunch Theorem
2. **Inductive bias for images**

Previous slide.

This lecture has been prepared and taught several times by Dr. Johanni Brea (EPFL) for the class ‘Artificial Neural Networks’ at EPFL.

Convolutional networks and No free lunch theorem

The big success of Deep Neural Networks came when they worked on large image data bases.

Why are convolutional networks better than other networks on image tasks?

→ explicit inductive bias that reflects properties of images!

Previous slide.

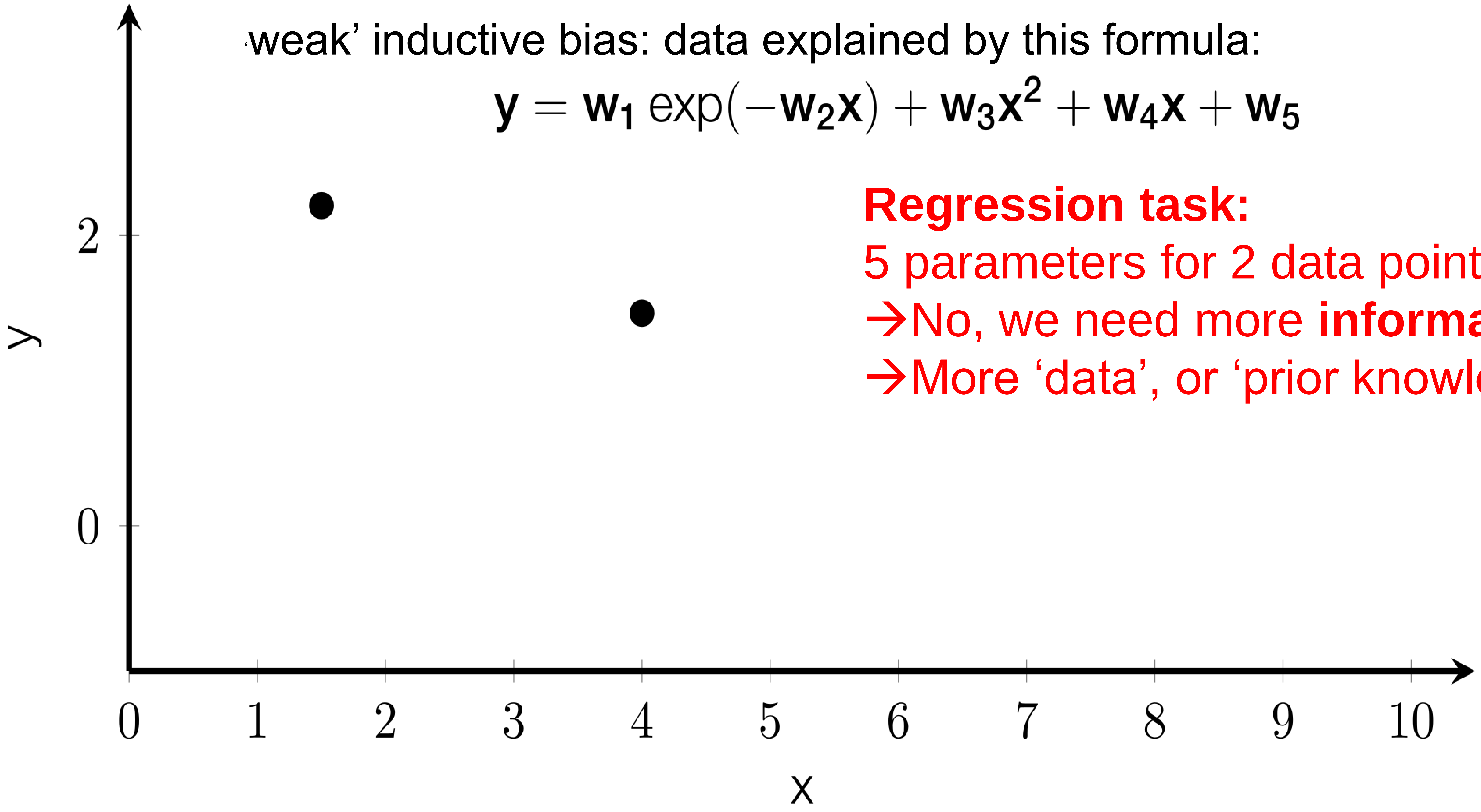
Why are convolutional networks much better than other approaches on these tasks? Because they have the right “inductive bias”.

Before we actually start looking at convolutional networks, we will illustrate the term “inductive bias” with a toy example and look at the different ways to find good inductive biases.

review: No Free-lunch Theorem and Regularization

‘weak’ inductive bias: data explained by this formula:

$$y = w_1 \exp(-w_2 x) + w_3 x^2 + w_4 x + w_5$$



Regression task:

5 parameters for 2 data points?

→No, we need more **information!**

→More ‘data’, or ‘prior knowledge’

Previous slide.

The notion of Inductive Bias has a strong link to the problem of regularization.

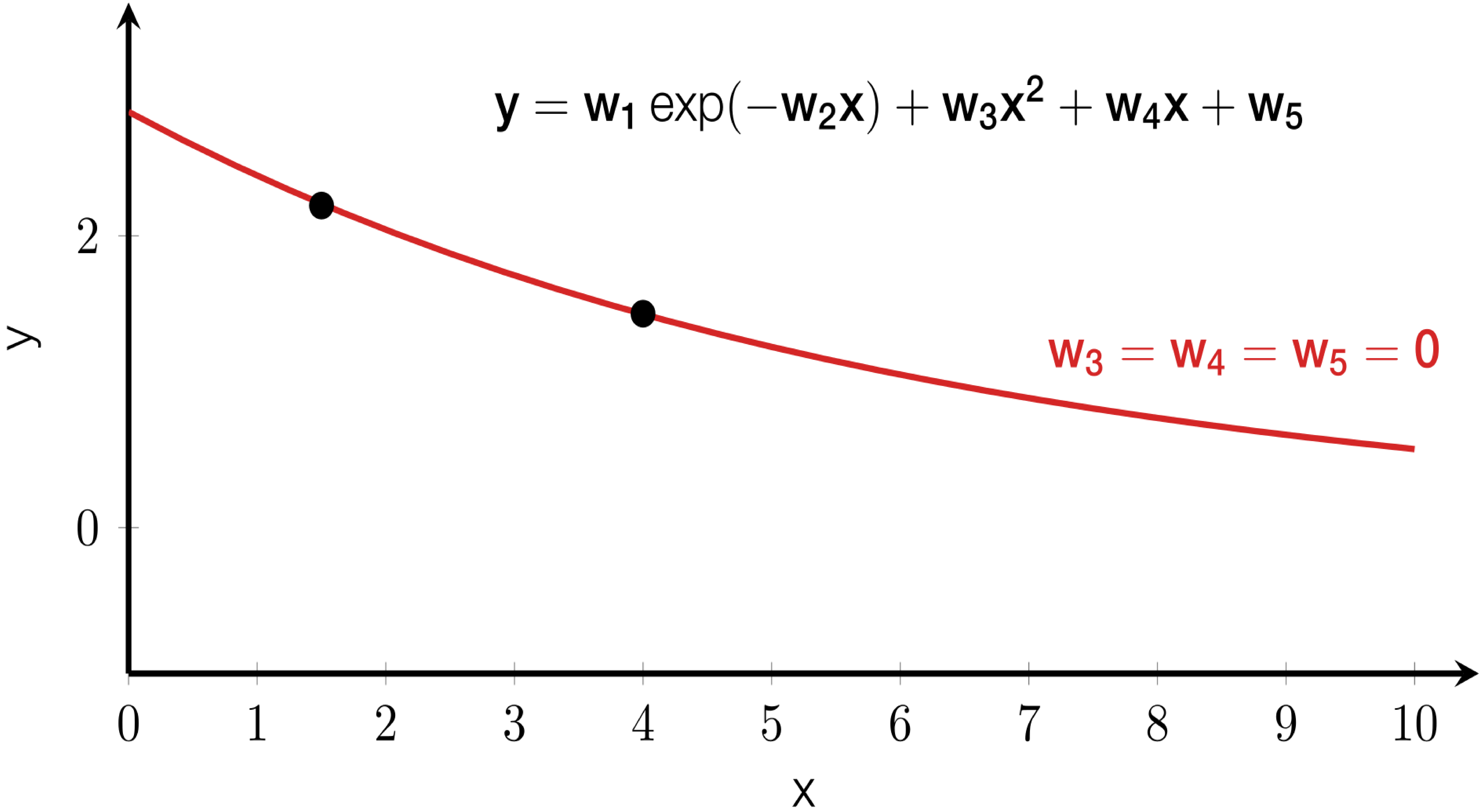
Let us suppose we have two data points in our training set. It is a regression task with input x and real-valued target y; if it were a classification task, y would be discrete.

There are infinitely many possibilities to fit these two data points, but let us assume that our initial inductive bias is that x and y can be linked through the displayed formula.

The fitting problem still is under-constraint, since we have more parameters than data points; we still have have infinitely many possibilities to fit these two data points with the displayed formula.

In other words our ‘inductive bias’ is ‘weak’, since the formula is offers too much freedom.

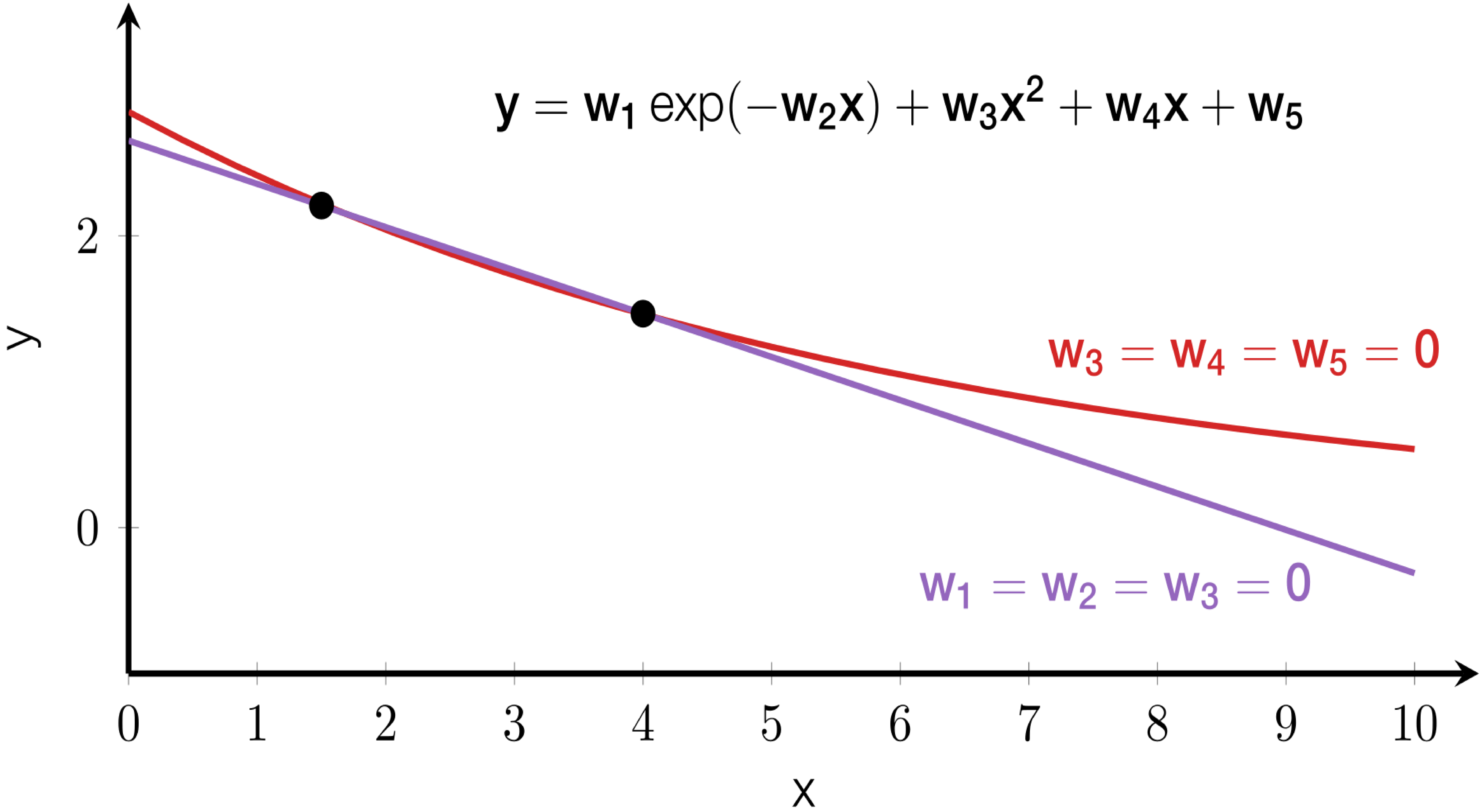
review: No Free-lunch Theorem (strong inductive bias 1)



Previous slide.

But let us assume we had reasons to consider a stronger inductive biases.
As a first example, exponential decay with $w_3 = w_4 = w_5 = 0$.

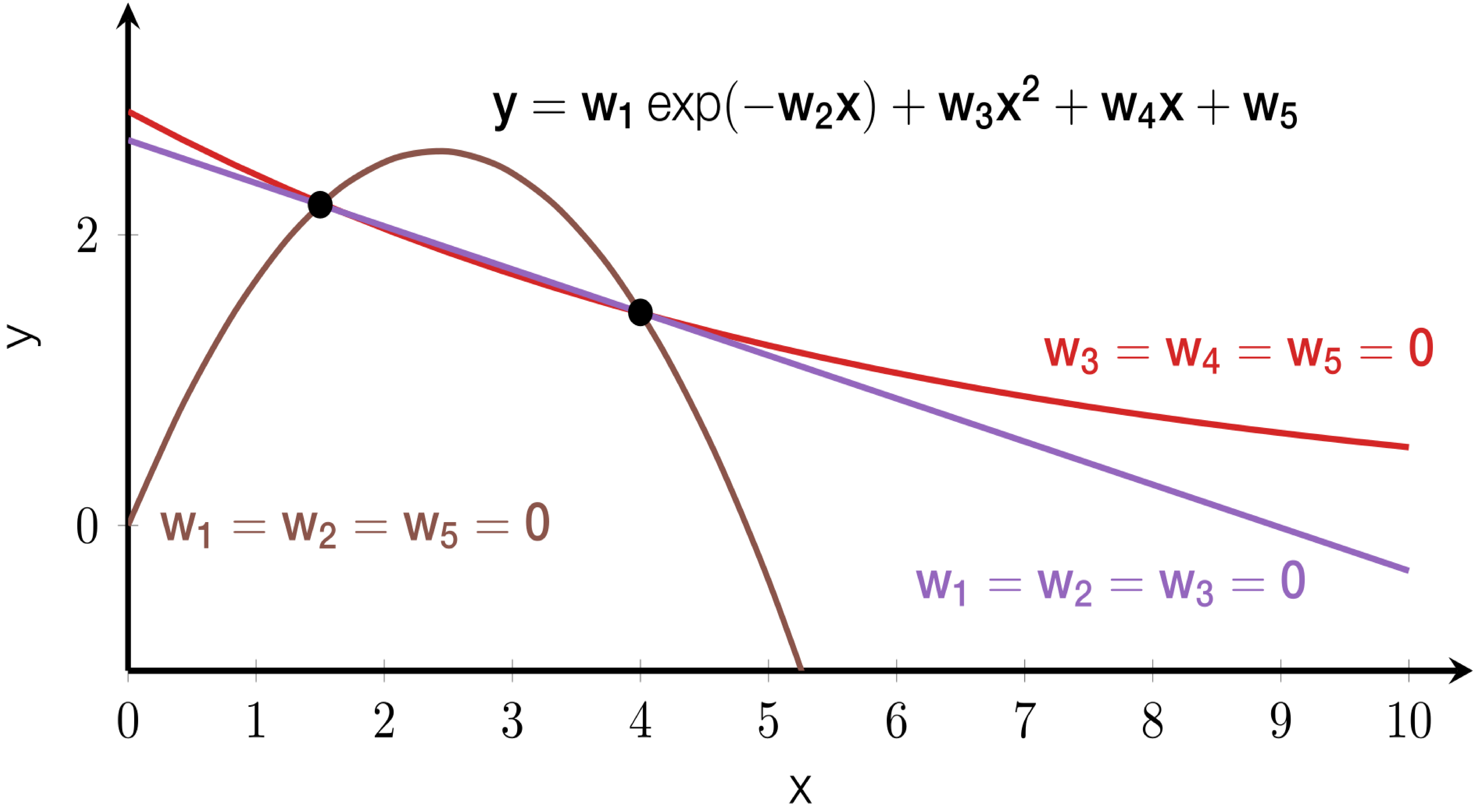
review: No Free-lunch Theorem (strong inductive bias 2)



Previous slide.

Second option for strong inductive bias: we want to fit by a straight line, so that $w_1 = w_2 = w_3 = 0$.

review: No Free-lunch Theorem (strong inductive bias 3)



Previous slide.

And third choice of a strong inductive bias, we assume that the data can be fit by a parabola through the origin with $w_1 = w_2 = w_5 = 0$.

Any other way of fixing at least three (of the five) parameters would be an valid alternative.

How should we choose between these inductive biases?

Inductive Bias: Use prior knowledge to constrain solutions

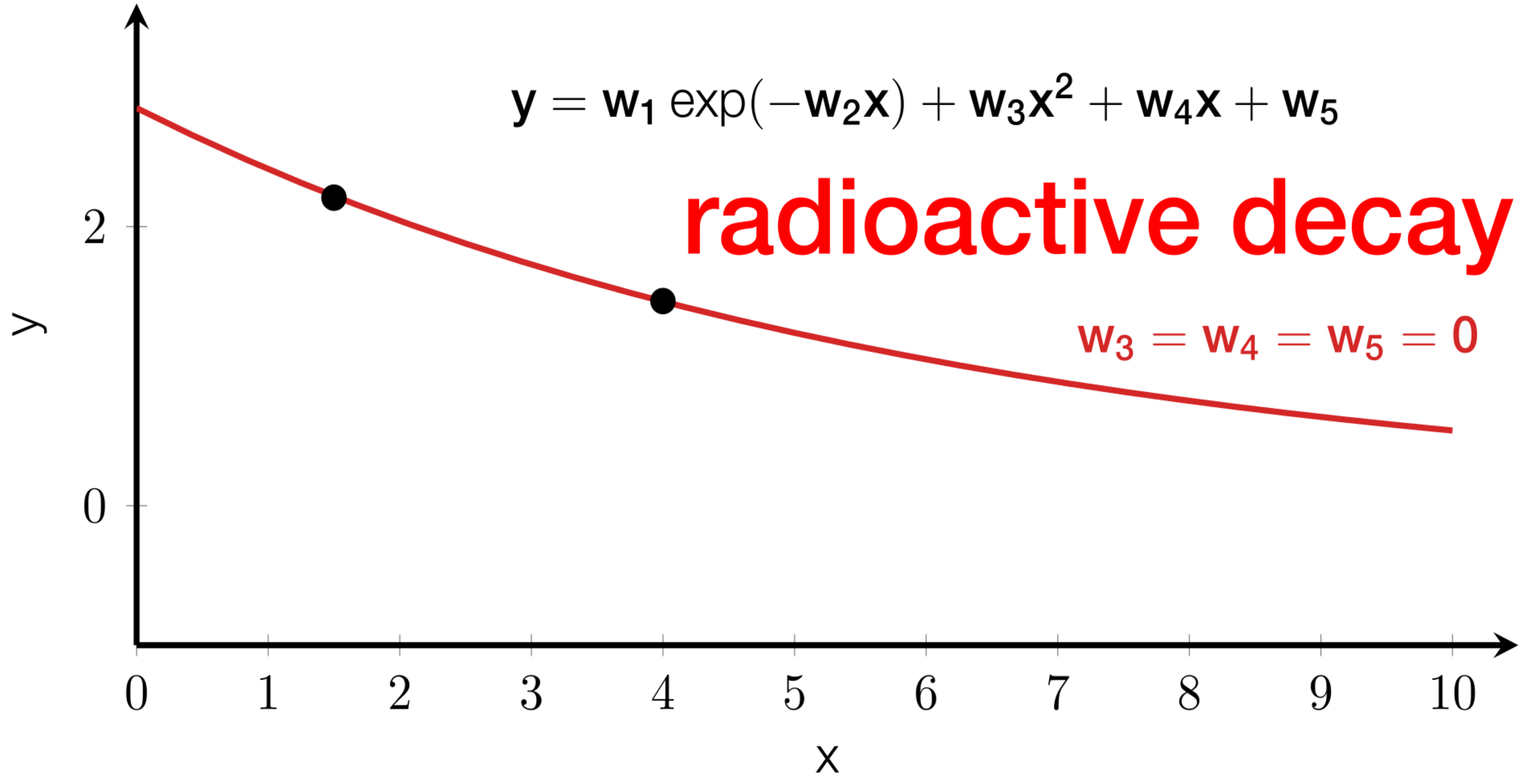
Question: What is a good inductive bias?

→ **Use your prior knowledge!**

Previous slide.

We always should choose our prior knowledge regarding the nature of the problem!

No Free-lunch Theorem: use inductive bias



How can we transfer our knowledge about the task to the problem at hand?

1.Data scientist approach: fit many similar problems

transfer knowledge by identifying good regularization schemes

→ force w_3 and w_4 and w_5 to be (close to) zero

2. Explicit knowledge from physics: known law of nature

→ $w_3 = w_4 = w_5 = 0$

3. Data augmentation

Previous slide.

If we have more information about the data, e.g. it comes from measurements of radioactive decay, we can transfer our knowledge about this type of problem, select a strong inductive bias and fit the other parameters.

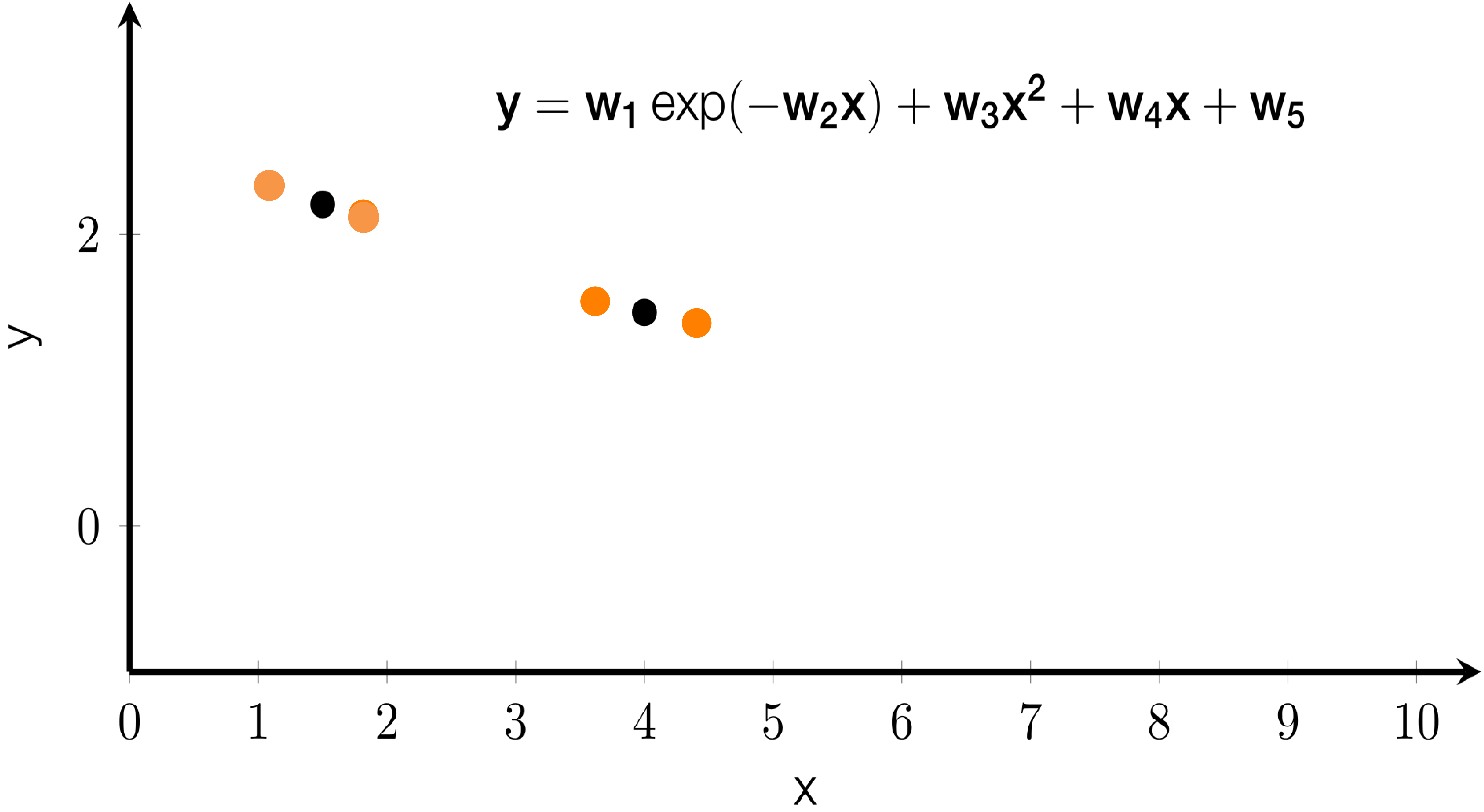
This transfer can happen in different ways.

1. (The data scientist approach) We have already several times fitted all parameters w_1 to w_5 to similar data and we have always observed that w_3 to w_5 were close to 0. Hence we decide to choose L1-regularization on w_3 to w_5 but leave w_1 and w_2 unconstrained.

2. (The physicist approach) There is some law that dictates a certain form of the function.

3. (Regularization by data augmentation) next slide.

review: No Free-lunch Theorem (data augmentation)



Previous slide.

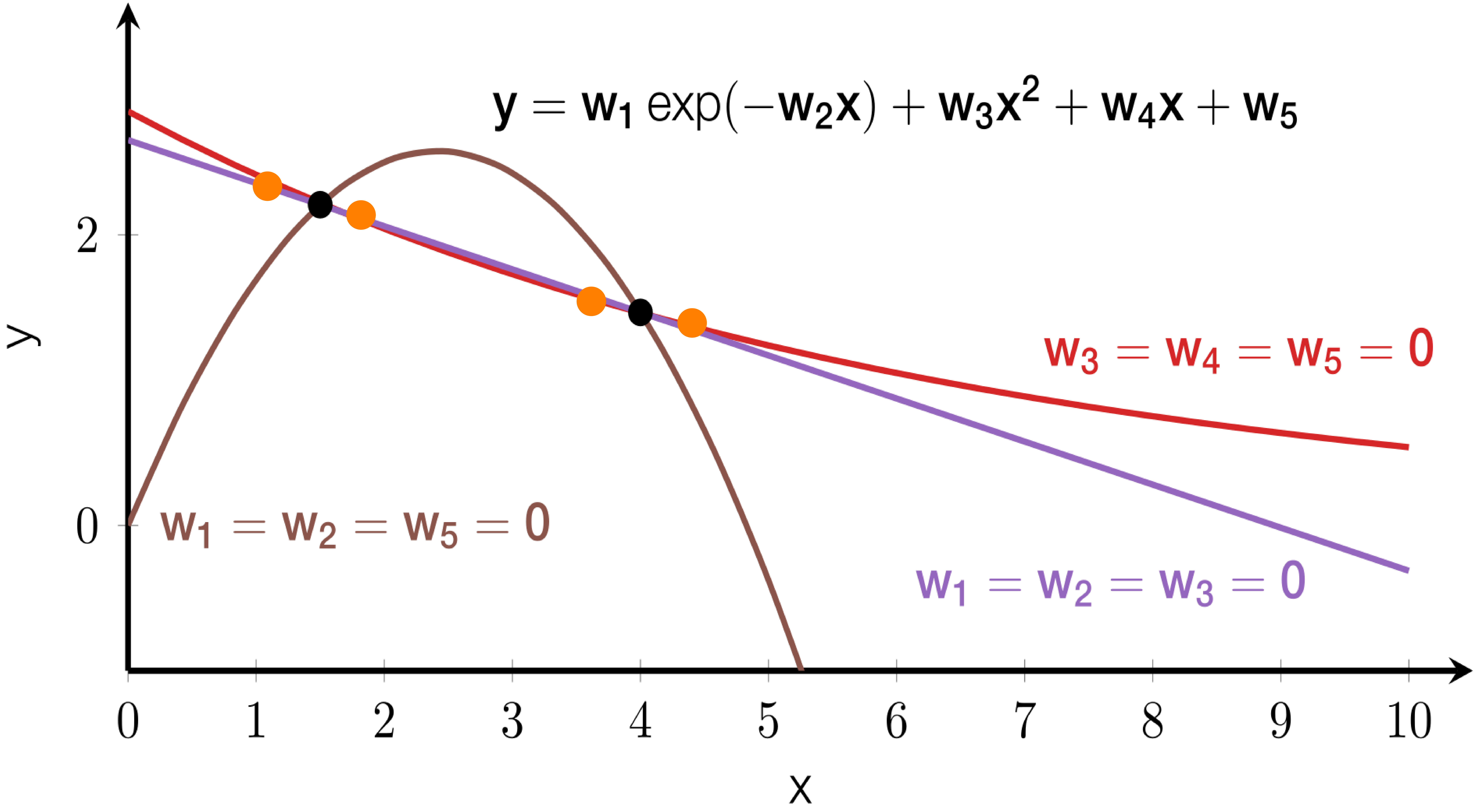
We may not know much about the data or find it difficult to define an explicit inductive bias but we have the intuition that

(i) the outcome should not change much if we transform the input in a certain way, e.g. we assume that if we would move the data points in the training set a bit to the left or to the right the outcome y should not be very different

(ii) The output should monotonically decrease as a function of x, as indicated with the orange data points.

With this augmented data set we can fit all 5 parameters!

review: No Free-lunch Theorem (data augmentation)

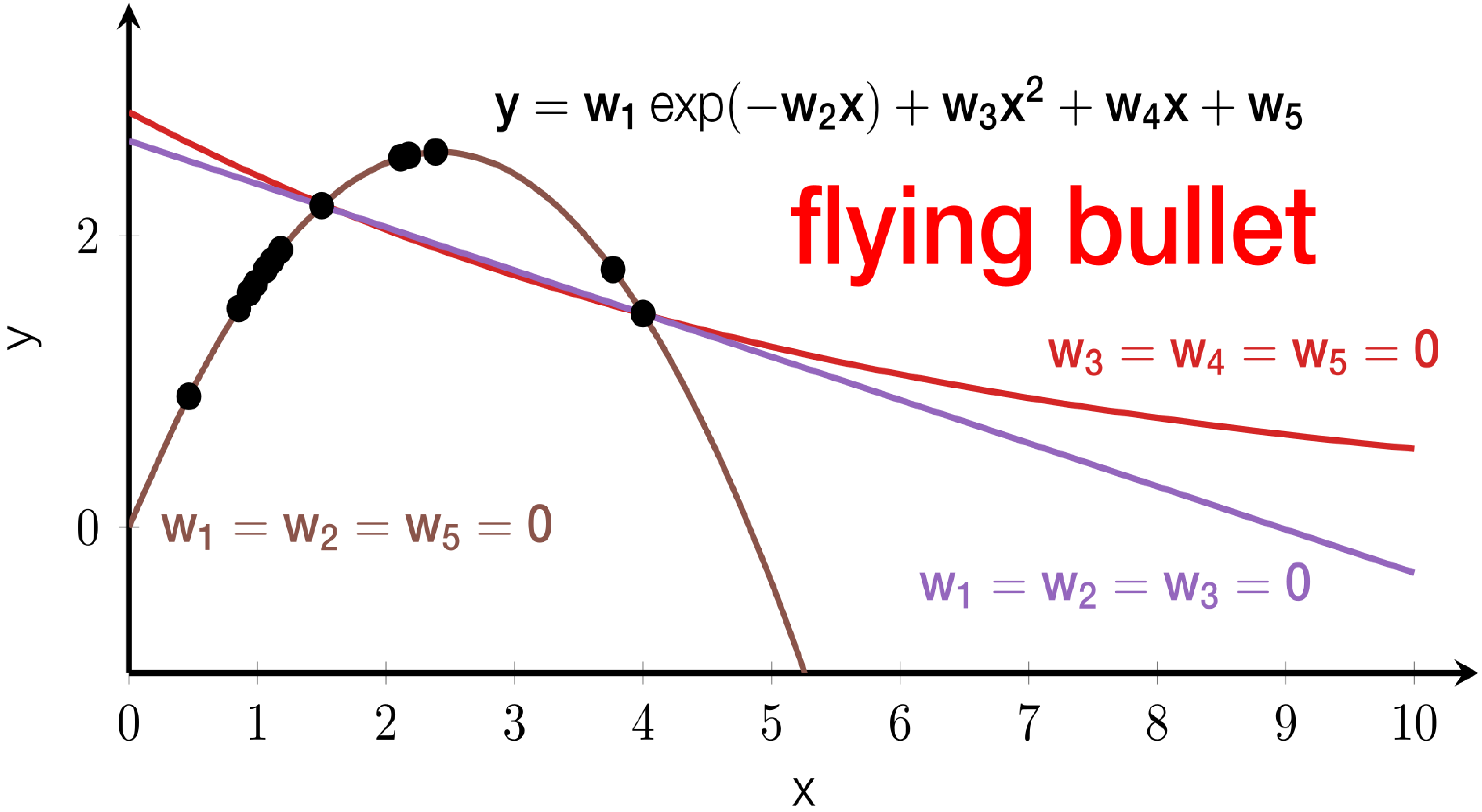


Previous slide.

Caveat: It may be difficult to find transformations that really leave the outcome invariant; getting more actual training data is typicall better than data augmentation.

Thanks to data augmentation (WITH SMART TRANSFORMATION!), or transfer from related problems we may be able to find a fit to only two data points that generalizes well if the data actually came from a measurement of radioactive decay.

review: No Free-lunch Theorem (the wrong inductive bias)



$$y = w_1 \exp(-w_2 x) + w_3 x^2 + w_4 x + w_5$$

flying bullet

$w_3 = w_4 = w_5 = 0$

$w_1 = w_2 = w_5 = 0$

$w_1 = w_2 = w_3 = 0$

Previous slide.

Even though the training error is zero for all three strong inductive biases considered here and we have the same degrees of freedom in each case (namely two parameters to be fitted) the performance on the test set can be terrible, if we pick the wrong inductive bias for the data at hand. If the data came from the measurement of the trajectory of a flying bullet the fit with the exponential decay or the straight line would not generalize well.
In other words, we don’t get good generalization for free; there is no free lunch here. Only if we choose the inductive bias that matches the data we get good generalization.

Inductive bias

Induction = finding a rule (function) from specific examples
Inductive bias = prior preference for specific rules (functions)

1) Explicit inductive bias (transfer reasoning)

“For radioactive decay I know that $w_3 = w_4 = w_5 = 0$”

2) Inductive bias through transfer learning

“I first train different models on data from other radioactive elements and choose the best model class for my current case”

3) Inductive bias through data augmentation

“For radioactive decay neighboring points have similar values and values can only go down”

Quiz: Inductive bias

🟦 [ ] With a strong (and correct) inductive bias, I can reach a low test error with very little training data.

🟦 [ ] With a strong inductive bias the test error will always be low.

🟦 [ ] Data augmentation is a heuristic method to get more training data.

🟦 [ ] In data augmentation there is an inductive bias in the form of our assumptions about reasonable transformations to be applied to the data

🟦 [ ] Choosing a specific neural network architecture is equivalent to choosing an explicit inductive bias.

Artificial Neural Networks

Inductive Bias and Convolutional Neural Networks

Johanni Brea & W. Gerstner

EPFL, Lausanne, Switzerland

Part 2B: Data Augmentation for Images

1. No Free Lunch Theorem
2. **Inductive bias for images**

Previous slide.

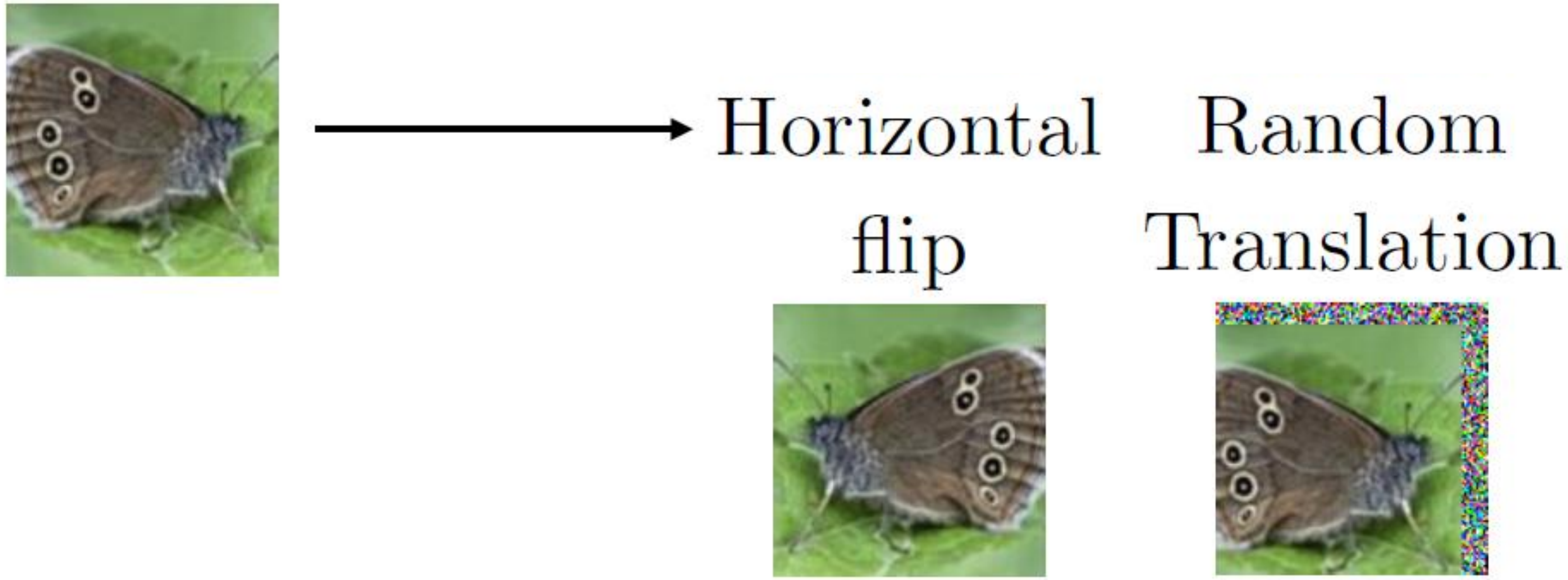
Data augmentation is a very effective regularization method and comes at relatively low cost. Data augmentation only work if we know ‘how to generalize’. This is why data augmentation corresponds to an inductive bias.

The videos for this part can be found on

https://lcnwww.epfl.ch/gerstner/VideoLecturesANN-Gerstner.htm

Deep Learning Lecture 2, part 4 (Data augmentation)

Dataset augmentation: Translations and Rotationas



Deep Learning, Goodfellow et al. 2016

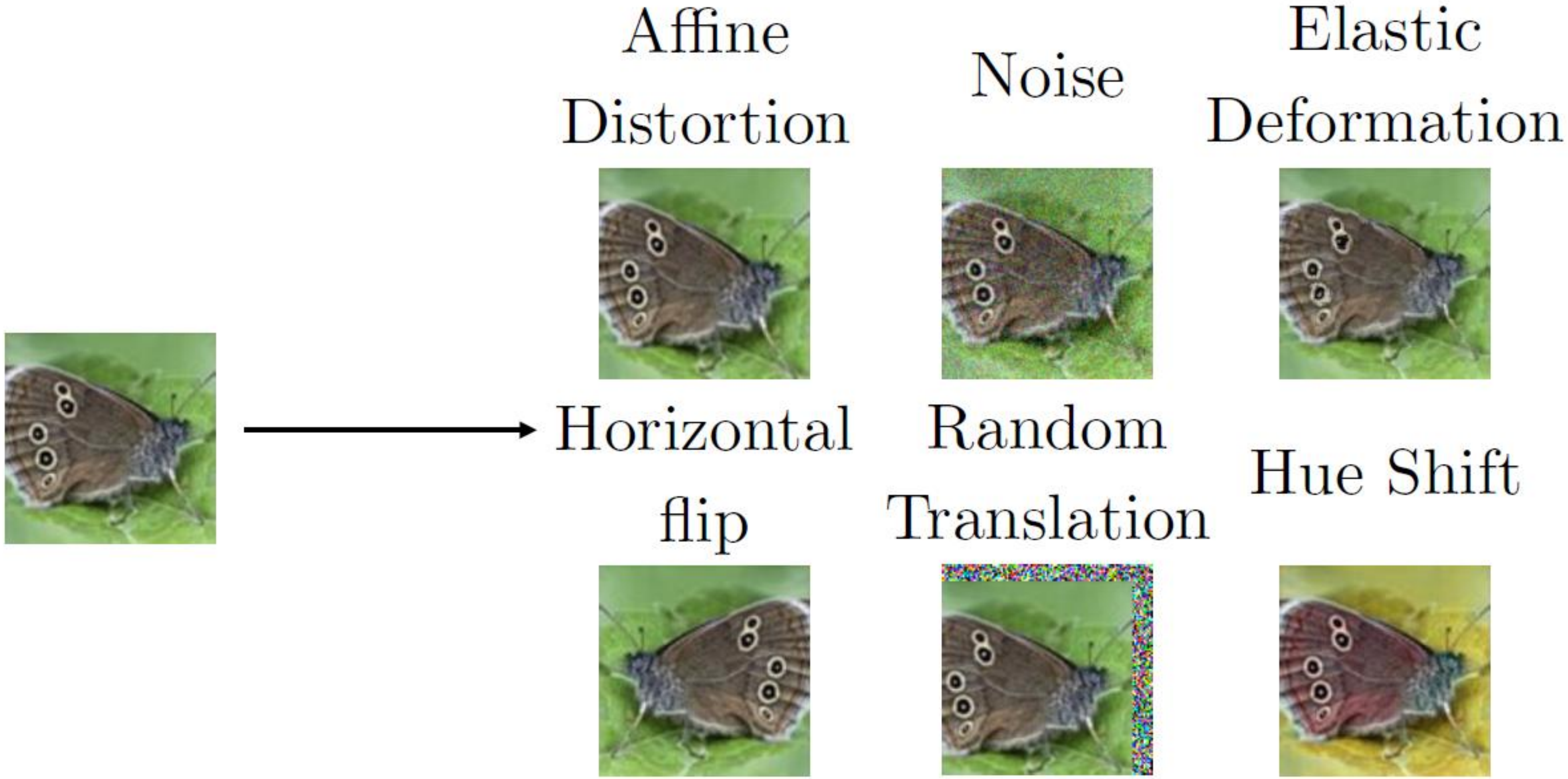
Previous slide.

Dataset augmentation is a simple regularization method. You start with a dataset of P data points.

For each data point you apply a few transformations. For the case of images, these are:

1. An image is laterally, vertically, or diagonally shifted (you need to fill in the background to do so). The new images are added to the data base (with the same label)
2. An image is flipped. The new image is added to the data base (the the same label).

Dataset augmentation: Translations and Rotations



Deep Learning, Goodfellow et al. 2016

Previous slide.

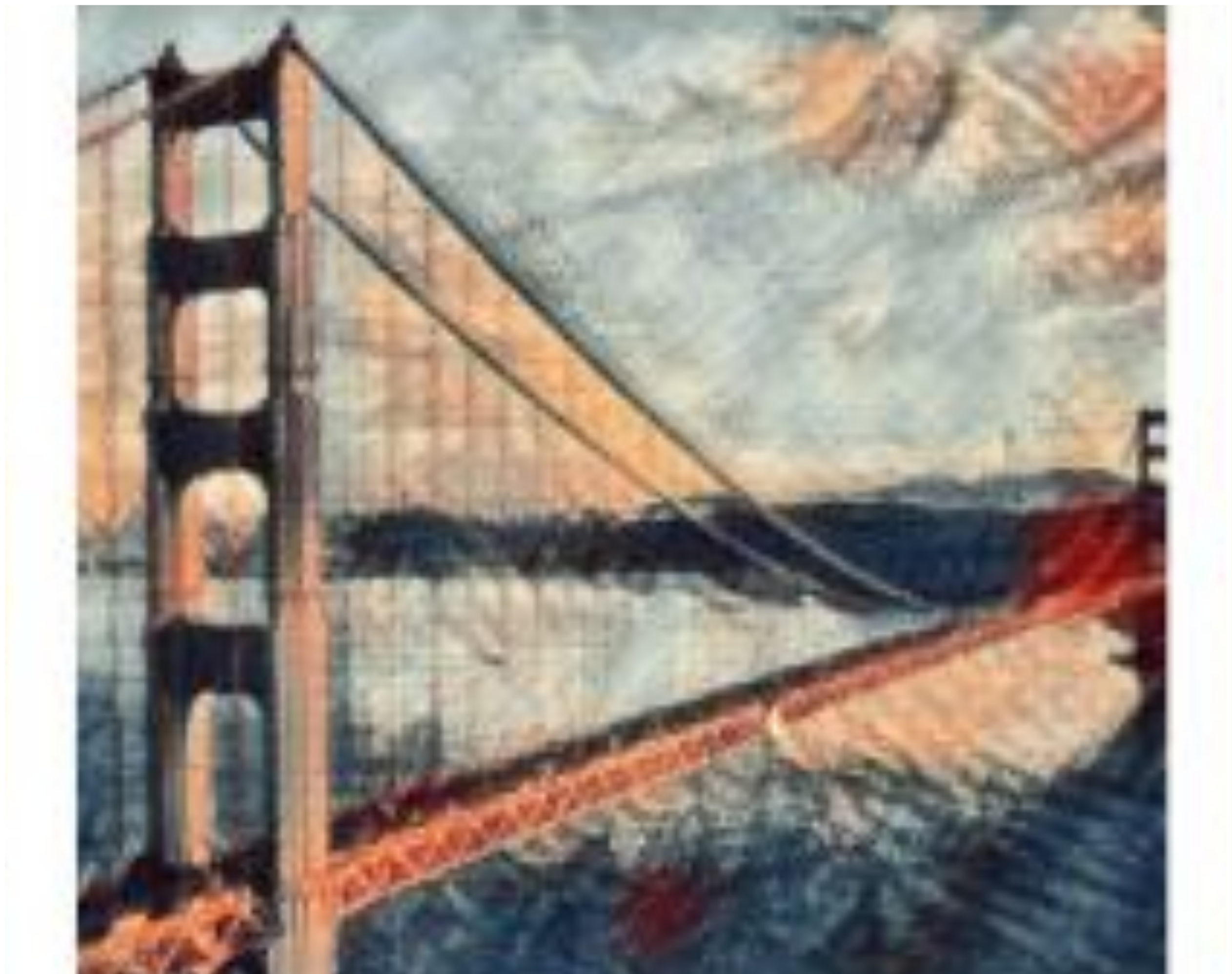
Dataset augmentation is a simple regularization method. You start with a dataset of P data points.

For each data point you apply a few transformations. For the case of images, these are:

1. An image is laterally, vertically, or diagonally shifted (you need to fill in the background to do so). The new images are added to the data base (with the same label)
2. An image is flipped. The new image is added to the data base (the the same label).
3. You add pixel noise (white or locally correlated). The new images are added to the data base (with the same label).
4. You apply one or several elastic deformations. The new images are added to the data base (with the same label).
5. You slightly shift the color scheme. The new images are added to the data base (with the same label).

Thus, a single image gives rise to twenty or more images. The transformations must correspond to the known invariances: a butterfly remains a butterfly if it is shifted, if the background illumination changes, if its shape changes slightly, etc.

Do you recognize this image?



https://deepart.io/

Previous slide.

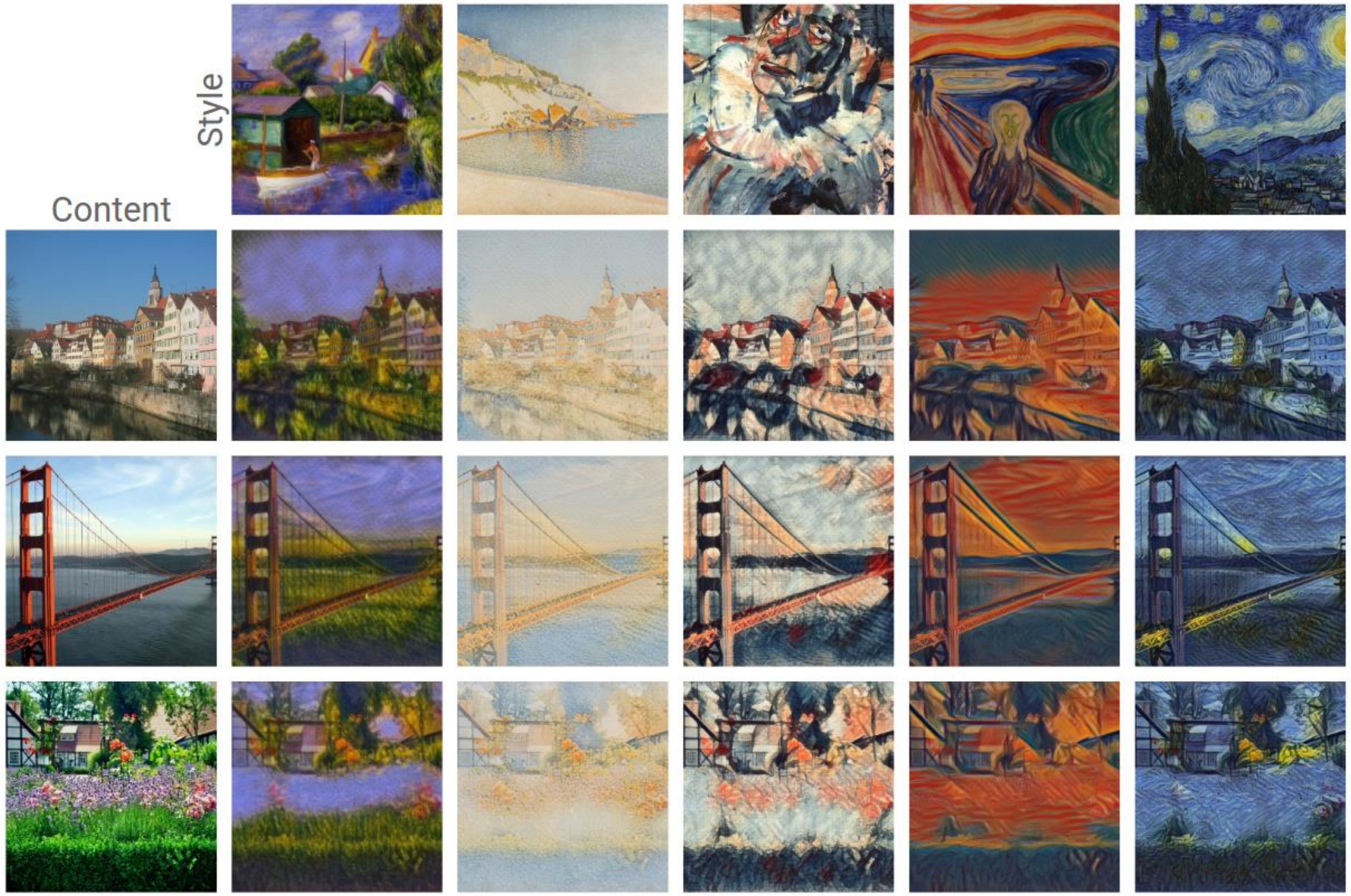
Dataset augmentation: artistic styles

Previous slide.

But there are other ways to do data augmentation on images. We may exploit that humans recognize not only objects and scenes from photographs, but also from artistic images.

For example, we recognize a boathouse in the country, or a face, or the sky in the night, even if these are stylized paintings.

Dataset augmentation: artistic styles



https://deepart.io/

Gatys, Ecker, Bethge "A Neural Algorithm of Artistic Style" (2015).

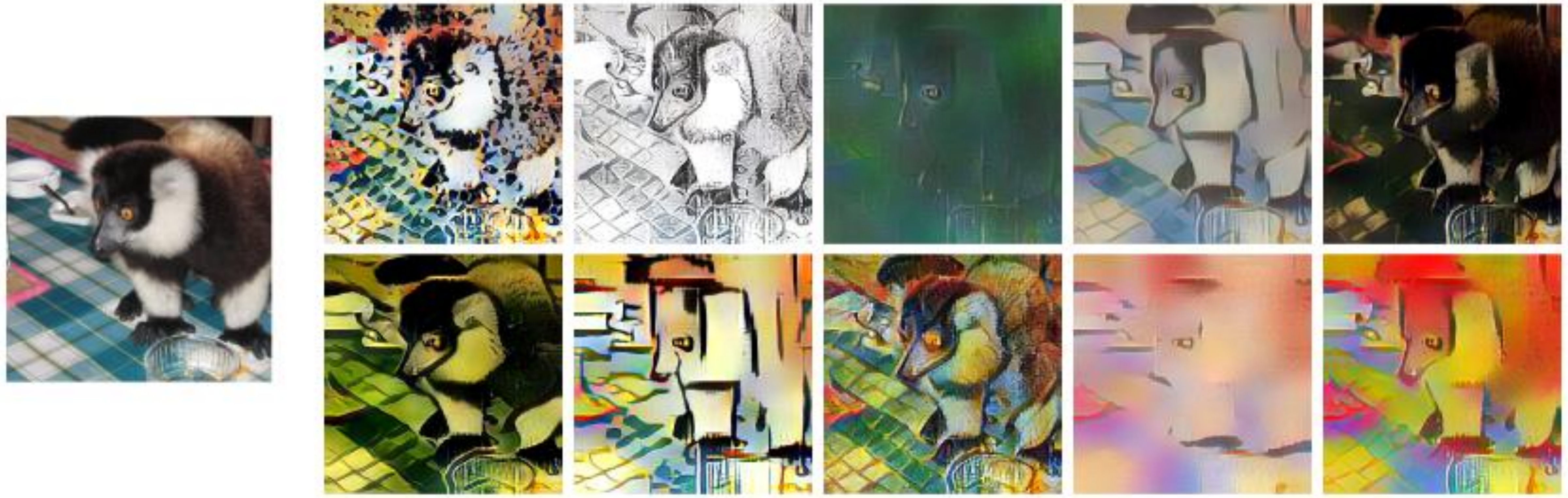
Previous slide.

Therefore, we can use as training data for object recognition not just original photos, but also transformed versions of these photos.
With AI methods, stylized versions of images can be automatically generated.

Dataset augmentation

Augment images from ImageNet
with stylized versions, generated automatically

Robert Geirhos et al.,
ICLR, 2019



thousands of potential styles for each image!

easy original images → difficult stylized images (same label)

Training with stylized images

→ improves generalization,

→ improves recognitition by form rather than texture

→ improves resistance to adversarial attacks

→ smart way to transfer prior knowledge (=construct a good inductive bias)

Previous slide.

When trained on stylized versions of images (automatically generated from the originals and a random artistic style), the object recognition of deep networks becomes much more stable, and more comparable to humans, compared to the case where only original image in ImageNet were used.

Summary: Data augmentation for images

- Data set augmentation is a **regularization method**

- Data set augmentation exploits known **‘inductive bias’** of data

- Data augmentation should reflect all **known invariances** of the task

- Labels for augmented images are ‘copied’ from originals

- Translations/symmetries/style transforms are generated automatically by algorithms

Previous slide.

Data set augmentation is a regularization method, because it avoids overfitting.
Data set augmentation is possible if we have prior knowledge, i.e., if we have a strong inductive bias about the kind of transformations under which the data is invariant.

Data set augmentation is effective, ant not costly in terms of human labor: we just use machine time to generate the additional images using some algorithms.

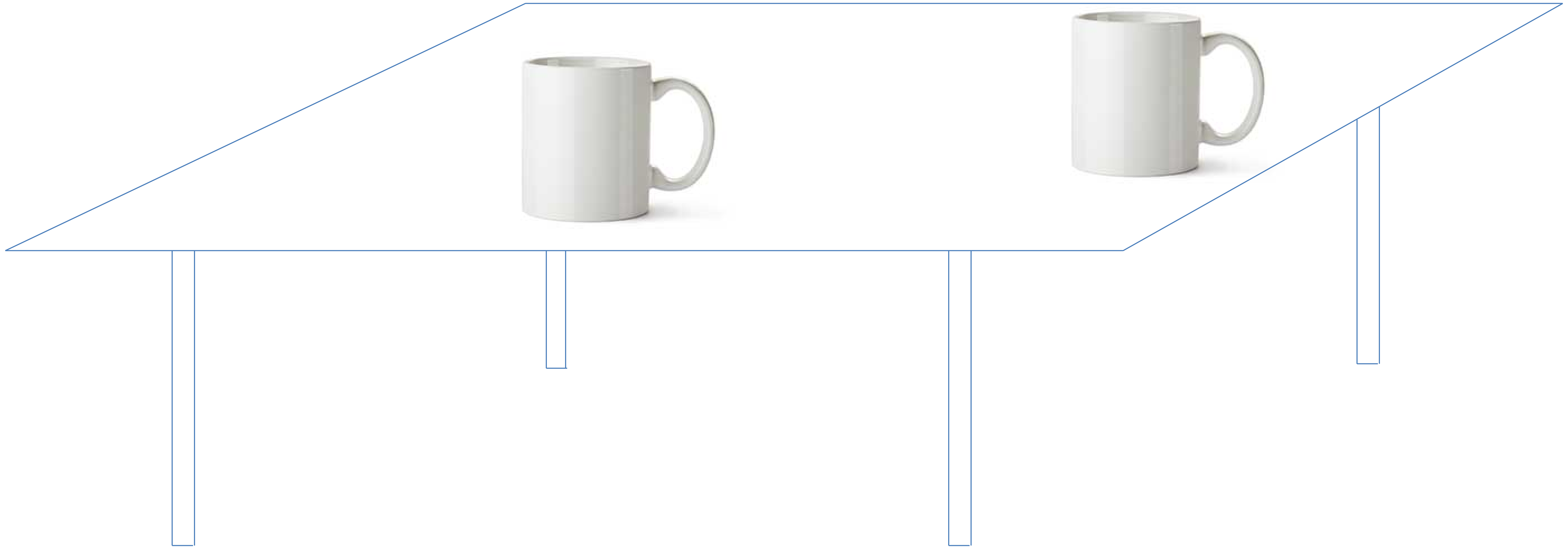
For images, there are many obvious transformations that keep the label unchanged.

And of course, it also is possible outside the domain of images as inputs.

Aim of the remaining parts of lecture:

Convolutional networks provide an excellent inductive bias for image recognition:
object invariance to (local) translation

Inductive bias via network architecture (rather than data augementation)



Previous slide

Translation invariance: a mug is a mug, wherever it is placed on the table.

The aim of convolutional networks is to exploit this inductive bias.

→ **Inductive bias for images**

Artificial Neural Networks

Johanni Brea & W. Gerstner
EPFL, Lausanne, Switzerland

Inductive bias and Convolutional Neural Networks

Part 3: Convolutional filters as inductive bias for images

1. No Free Lunch Theorem
2. Inductive bias
3. **Convolution filters as inductive bias for images**

Convolutional networks have been covered in Intro to Machine Learning (Profs Jaggi + Flammarion)
Convolutions are also standard in Signal Processing, Computer vision, Physics, and other fields
→ hence this part is marked 'review'

Previous slide.

For tasks involving natural images, like object recognition, we have now strong empirical evidence, that neural networks with convolutional layers work better than networks with only dense layers (all-to-all connectivity).

Thus we may conclude that convolutional layers provide a better **explicit inductive bias**

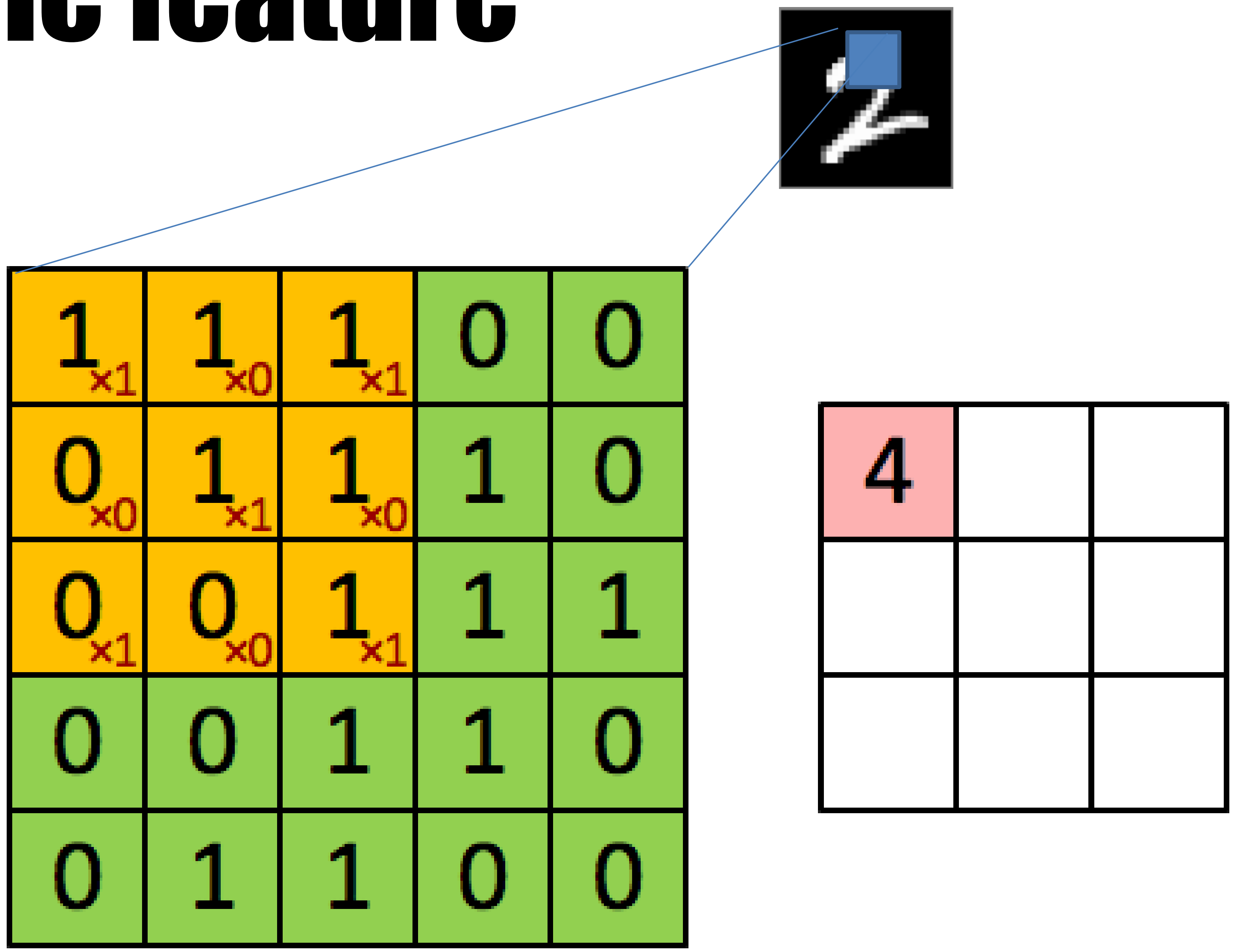
We will start now with a recap of convolution.

Convolution is a mathematical operation that is widely used in computer vision, signal processing, and physics.

Review of Convolution: one feature

1	0	1
0	1	0
1	0	1

Filter (or Feature)



$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

w_{xy}

I_{xy}

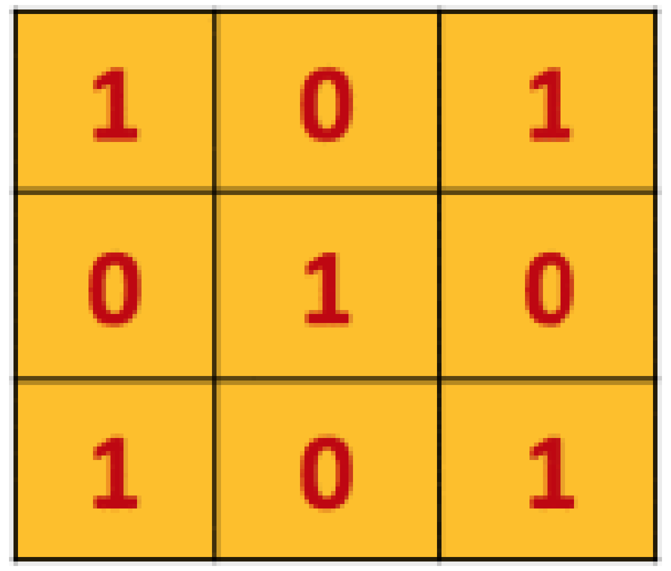
$$a_{ij} = b + \sum_{x=1}^{3} \sum_{y=1}^{3} I_{i+x-1,j+y-1} w_{xy}$$

http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Previous slide.

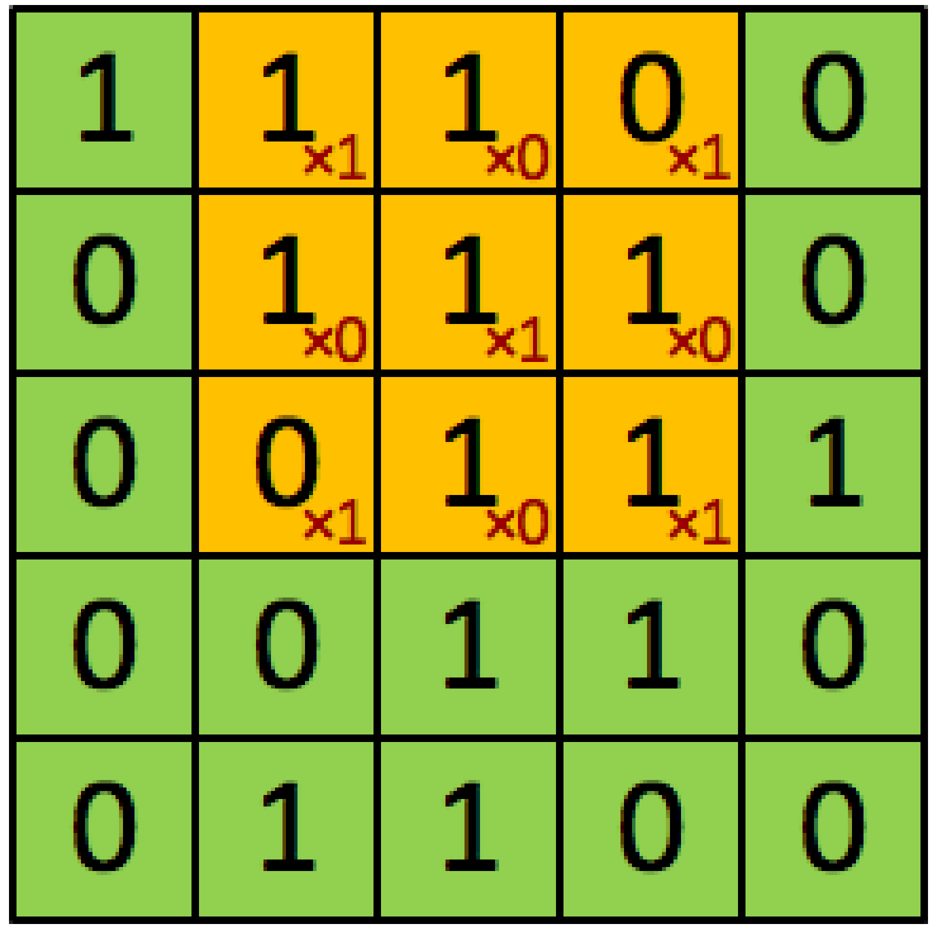
On the left we have a 3x3 filter. Applying it to the top-left corner of the 5x5 image in the middle means multiplying each filter weight with the corresponding pixel value and summing the results to get the activation a_{11} = 4 for bias b = 0.

Review of Convolution: one feature



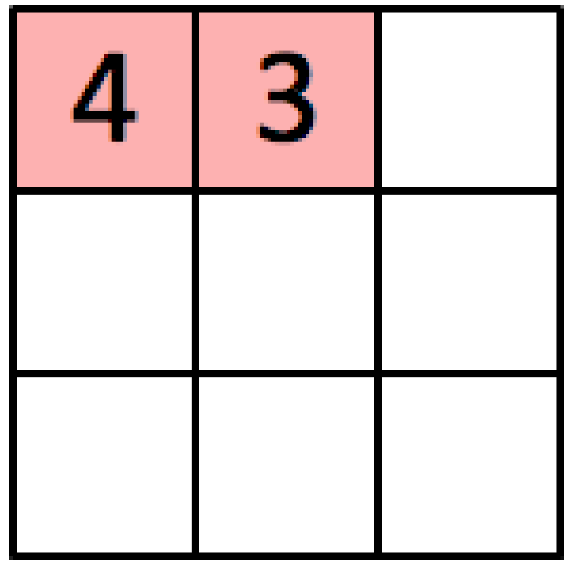
1	0	1
0	1	0
1	0	1

Filter (or Feature)



1	$1_{\times 1}$	$1_{\times 0}$	$0_{\times 1}$	0
0	$1_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	0
0	$0_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	1
0	0	1	1	0
0	1	1	0	0

Image



4	3	

Convolved Feature

$$w_{xy}$$

$$I_{xy}$$

$$a_{ij} = b + \sum_{x=1}^{3} \sum_{y=1}^{3} I_{i+x-1,j+y-1} w_{xy}$$

http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

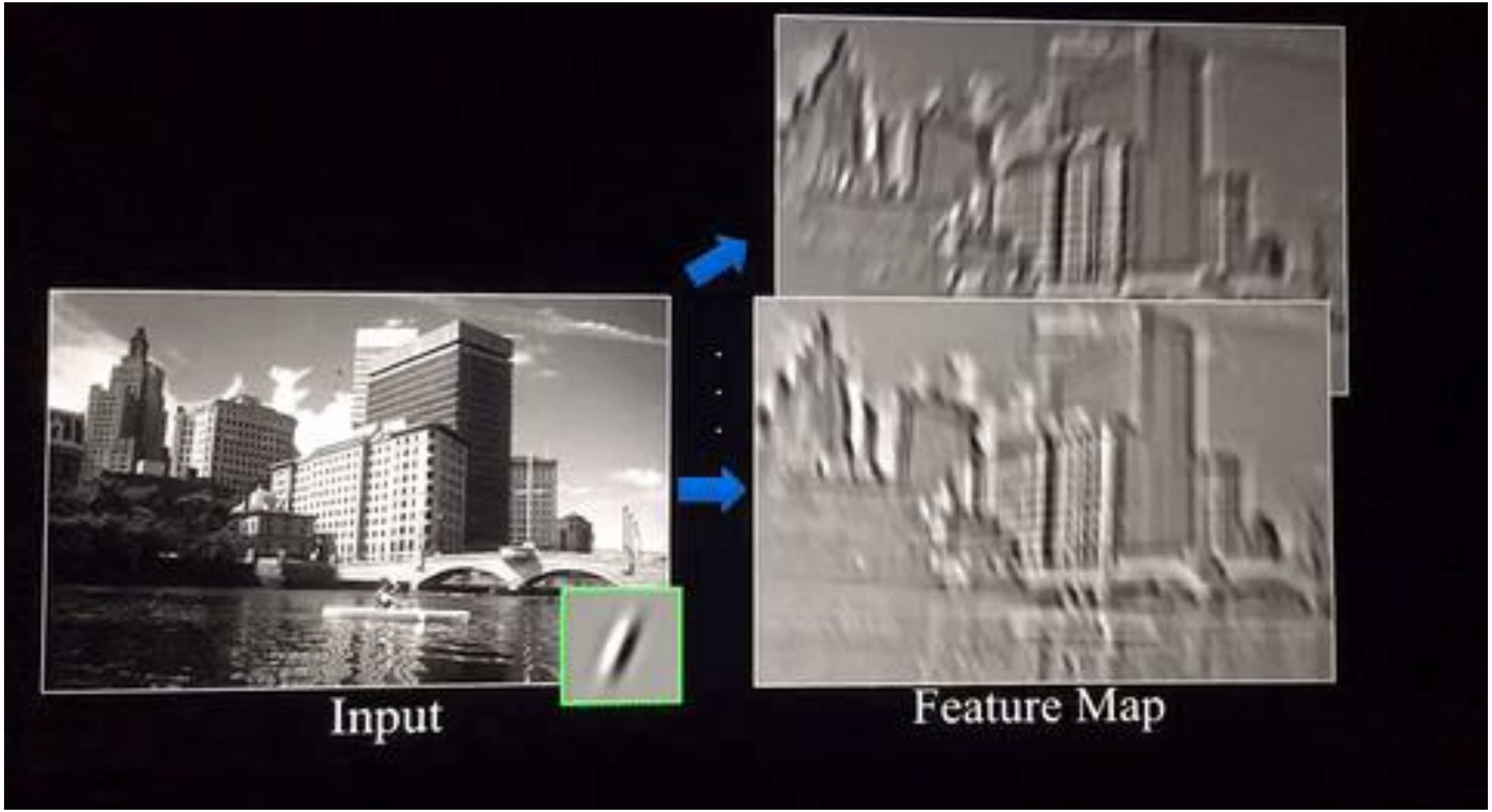
Previous slide.

Next we move the filter over the image and repeat the multiplication and summation at each position to get all values of the convolved feature matrix on the right.

Note that in contrast to a “standard neuron” that would take the full image as input and compute one activation value, we get multiple activation values with a “convolutional neuron” (filter): there is one value for each position at which the filter (neuron) is applied.

Review of Convolution: multiple features

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/



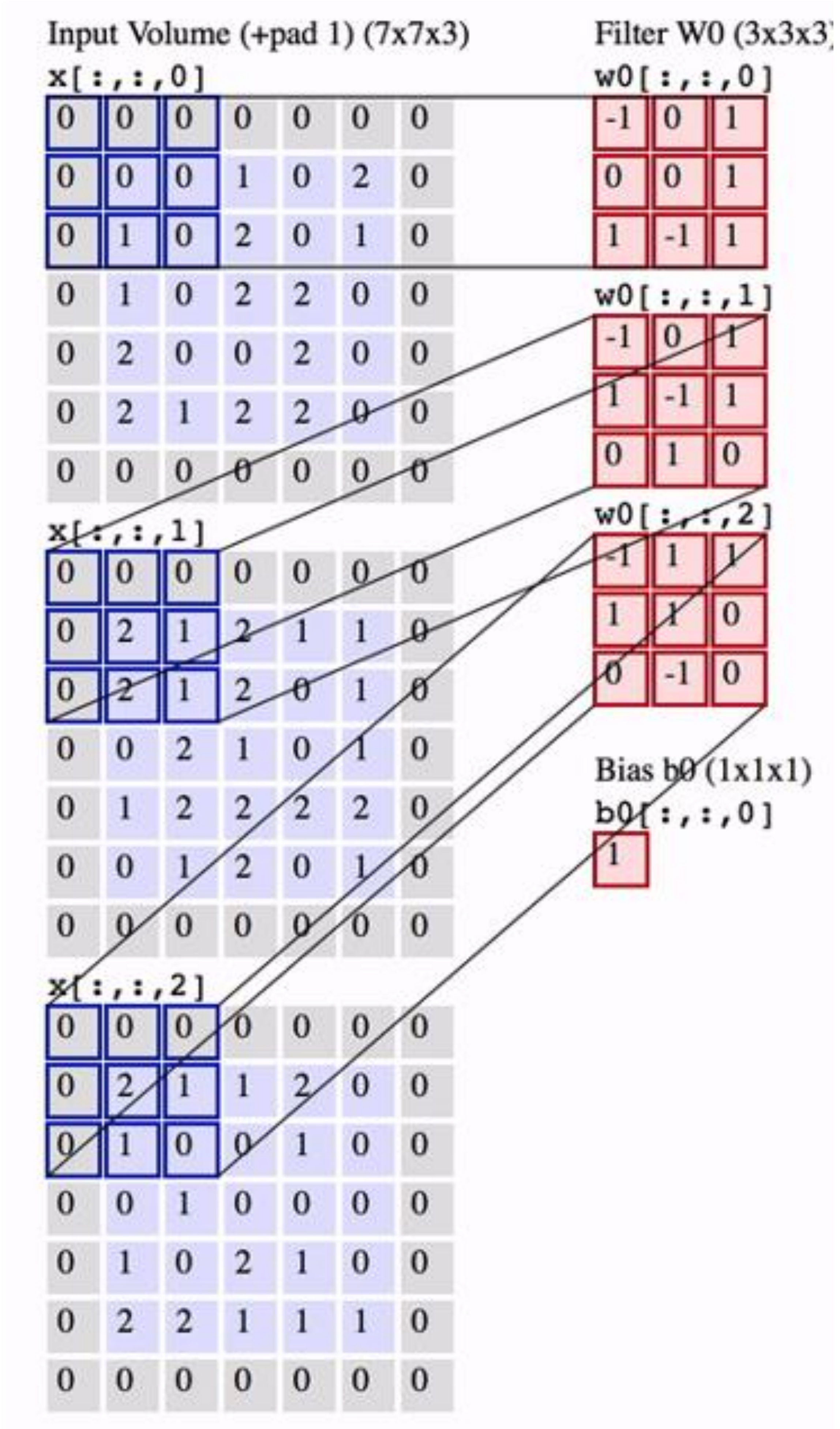
Each filter type generates one new processed imaged called ‘feature map’

Previous slide.

For multiple filters we get multiple convolved features, also called feature maps. Note how the different simple filters in this example extract edges of different orientation.

One convolutional layer is usually composed of multiple filters. Since each feature map is two-dimensional we can think of the output of a convolutional layer as a three-dimensional object. We will discuss this in more details on the next-but-one slide.

Review of Convolution: colors and padding



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
0	0	1
1	-1	1

w0[:,:,1]

-1	0	1
1	-1	1
0	1	0

w0[:,:,2]

-1	1	1
1	1	0
0	-1	0

Bias b0 (1x1x1)

b0[:,:,0]

1

- 3 color channels

- Filter 3x3 (spatial) in each color dimension
  → filter tensor W0 denoted as (3x3x3)

- Apply several filters: W0, W1, W2, W3, …

- Padding: extend images at borders with 0

Previous slide.

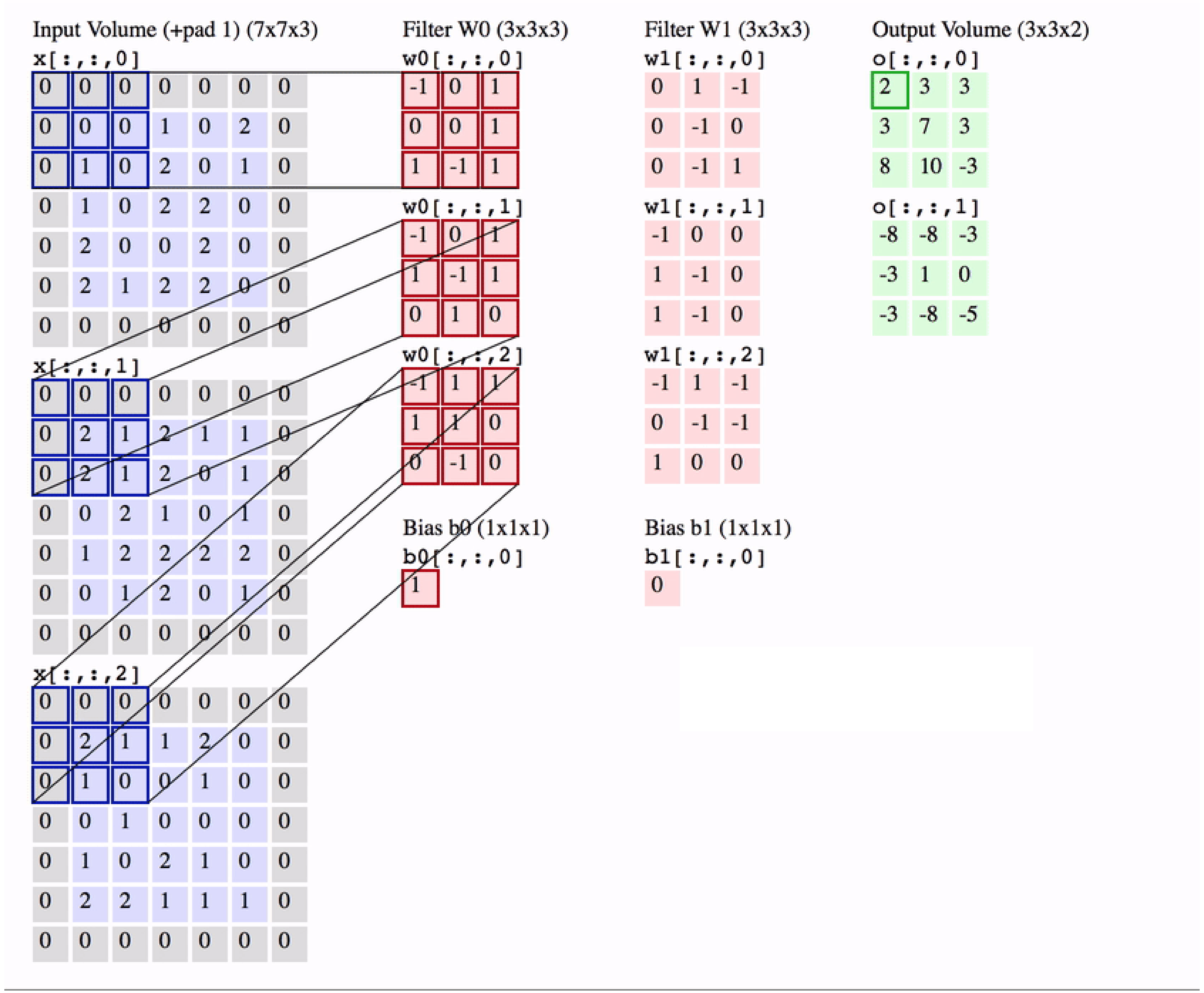
For color images, we have three color channels.
Therefore, the filter also works (potentially differently!) in each color domain.
A filter that has a spatial 3x3 layout becomes in total a filter with 3x3x3=27 parameters.

Usually we apply several filters of form (nxnx3).

For the way we move a filter across the image, there are two important parameters:

(i) Padding = extend the image at the borders
(ii) Stride n = jump by n pixels

Review of Convolution: Stride



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
0	0	1
1	-1	1

w0[:,:,1]

-1	0	1
1	-1	1
0	1	0

w0[:,:,2]

-1	1	1
1	1	0
0	-1	0

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

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

w1[:,:,1]

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

w1[:,:,2]

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

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

2	3	3
3	7	3
8	10	-3

o[:,:,1]

-8	-8	-3
-3	1	0
-3	-8	-5

- Stride 2

filters jump 2 pixels at a time as we convolve

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

Previous slide.
We can apply each filter starting at the top left-corner and moving it column by column, row by row over the image. This is known as convolution with padding 0 and stride 1. Here we consider two generalizations:

First, if filters are applied only at every n’th position, we speak of stride n. The example was done with stride 2.

Second, if one wants to position the center of a filter to the top-left pixel of an image, this can easily be achieved with padding the original image, i.e. appending rows and columns of zeros to the original image.

The green boxes show the result of the application of two filters (each one applied at 9 different positions, stride 2.).

Of course, stride and padding can in general be different in x and y direction.

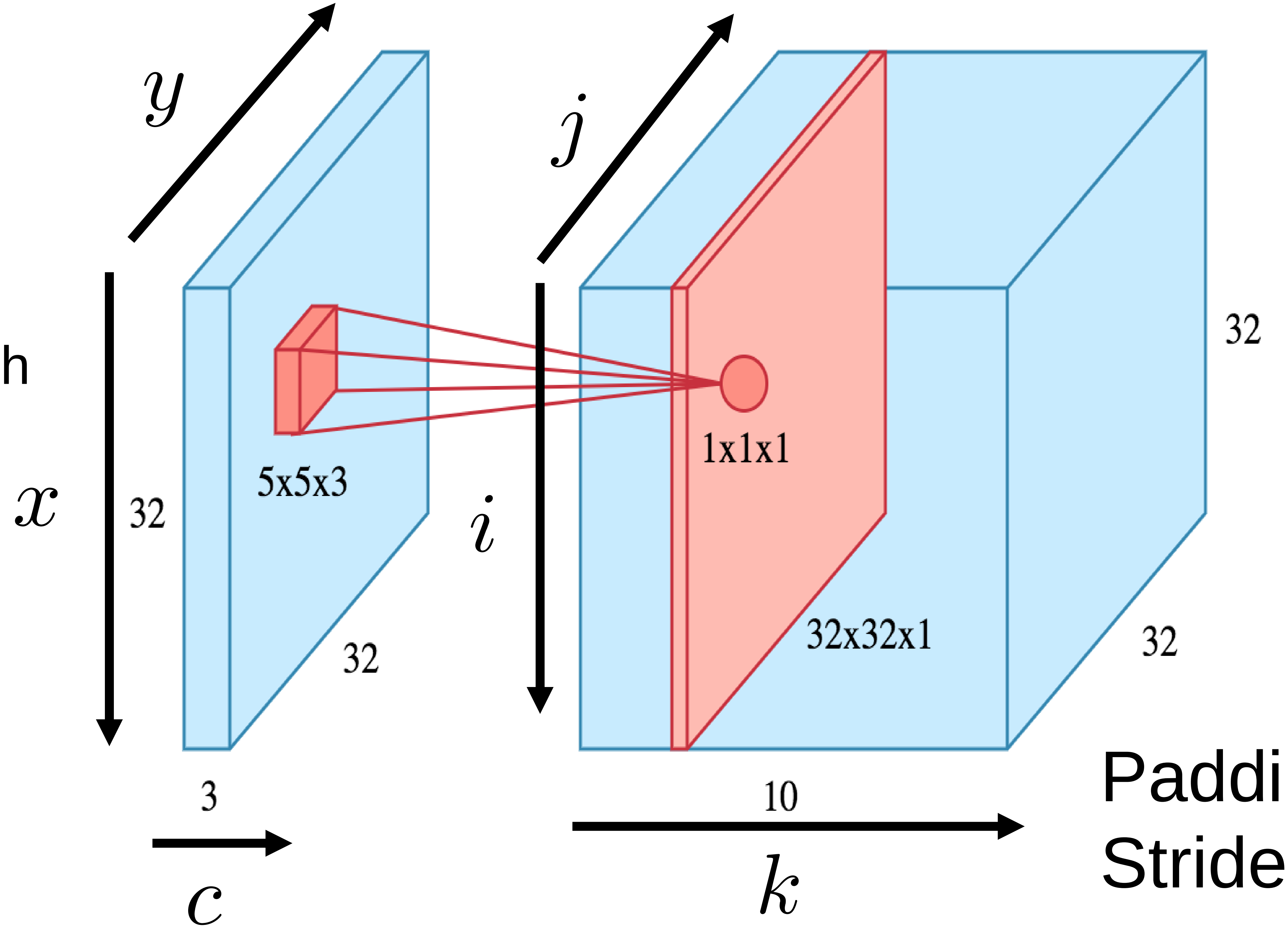
Note, that using a stride > 1 results in a reduction of the x-y dimensions of the feature map, but if we would use stride 1 in this example, the input without padding and the output would have the same x-y dimensions (5x5) thanks to padding.

Review of Convolution: one convolutional layer

color image

width x height x depth

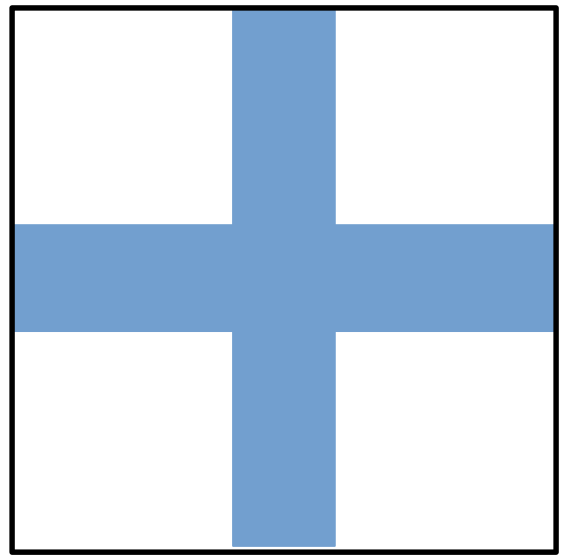
32 x 32 x 3



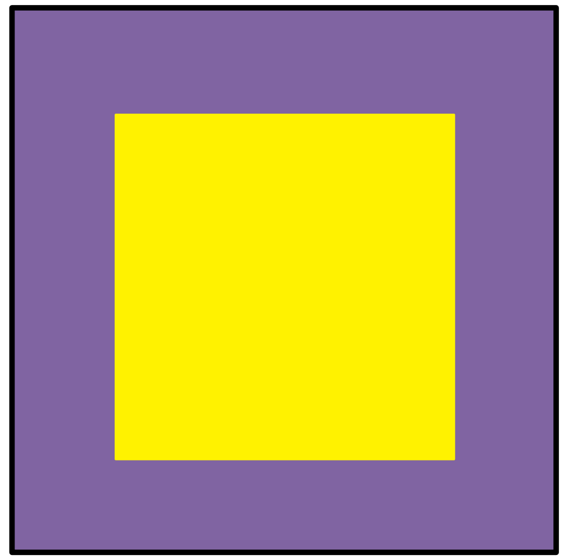
Padding 2

Stride 1

filters, e.g



w_{xyc1}



w_{xyc2}

I_{xyc}

$$a_{ijk} = b_k + \sum_{x=1}^{5}\sum_{y=1}^{5}\sum_{c=1}^{3} I_{i+x-1,j+y-1,c}\,w_{xyck}$$

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Previous slide.

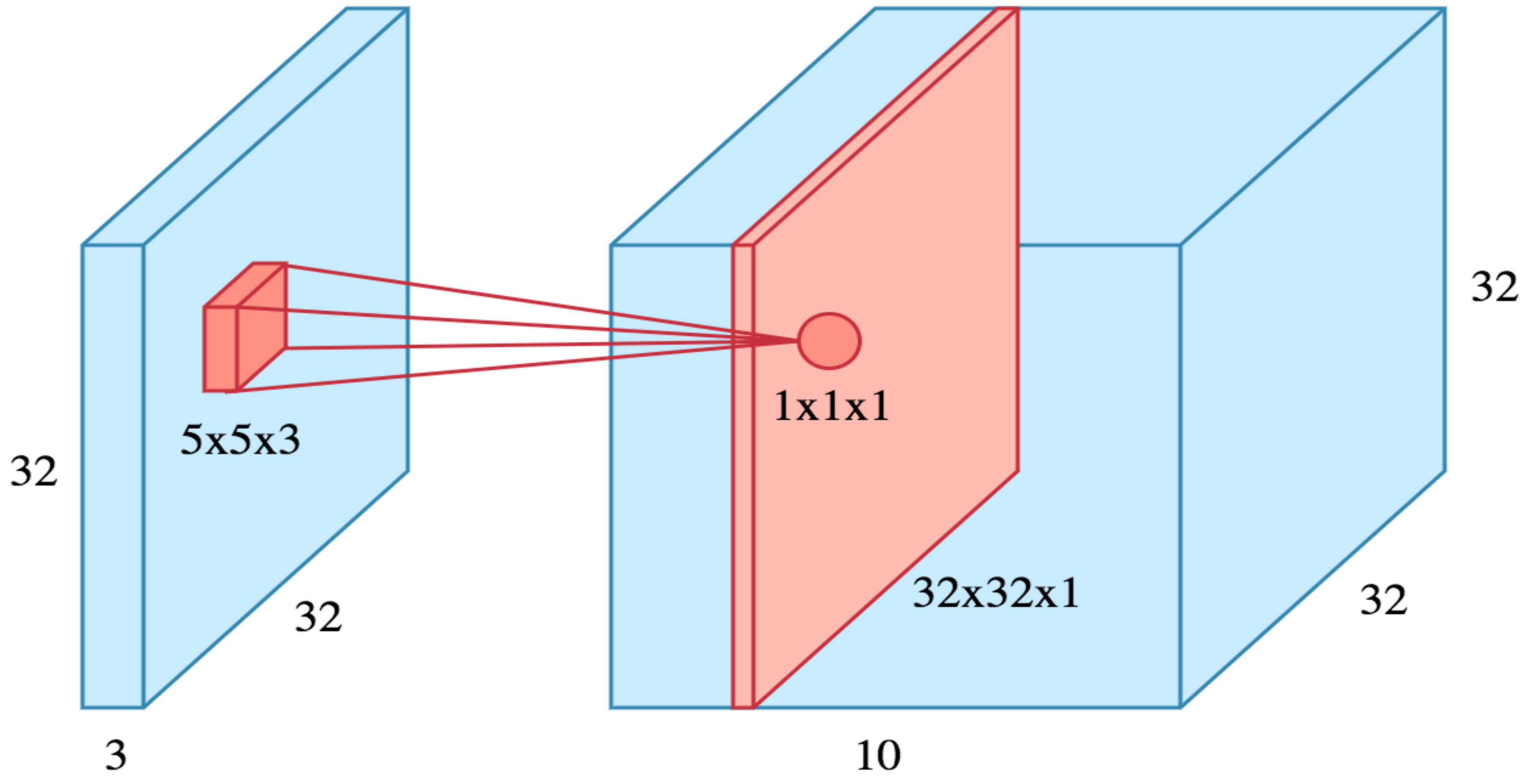
On the left we have a color image of 32x32 pixels with 3 color channels. We represent it as a volume of dimensions 32x32x3. If we convolve it with 10 filters of size 5x5x3 with padding 2 and stride 1 we get a volume of dimensions 32x32x10. One of those 10 filters could be a detector for blue crosses or for vertical yellow bars. (In case there is a single filter the volume would be 32x32x1, as indicated on the left).

What would be the output volume if we used padding 0 instead of padding 2?

What would it be if we used stride 2 instead of stride 1 for padding 2?

What would it be if we used 24 filters instead of 10? Volume is 32x32x24

Quiz: Convolutions



With padding 2, stride 1 and 10 filters we get in the above configuration a volume of 32x32x10.

□ [ ] With padding 0, stride 1 and 10 filters we will then get a smaller volume of 28x28x10

□ [ ] With padding 0, stride 2 and 10 filters we will then get a smaller volume of 14x14x10

□ [ ] With padding 0, stride 2 and 10 filters, the filter will be placed in all borders

□ [ ]The volume is proportional to the number of filters.

Previous slide.

Convolution: computing the volumes (Exercise 2)

n x n x c input volume (e.g. image)   k filters of size f x f x c

stride s
padding p

$$\left(\frac{n+2p-f}{s}+1\right)\times\left(\frac{n+2p-f}{s}+1\right)\times k$$

Note: combination of stride and filter size where the above formula does not give a natural number should be avoided, because corners are not reached.

Previous slide.

In general the size of the output volume can be computed with the above formula. You have to make sure that you choose a stride that (when starting in the left corner) brings you with a number of steps to the right corner (no fractions!).

If fractions appear: solution is biggest integer small than the calculated fraction.

→ See Exercise!!

Inductive bias of a convolutional layer

1) Independence to translation

“A feature detector (such as a vertical edge detector) that’s useful in one part of the image is probably useful in another part of the image.”

2) local feature detectors are useful

NOTE: A conv layer is a special case of a dense layer with

- many neurons having the same weights → 1)
- many weights zero, except in a small neighborhood, →2)

Previous slide.

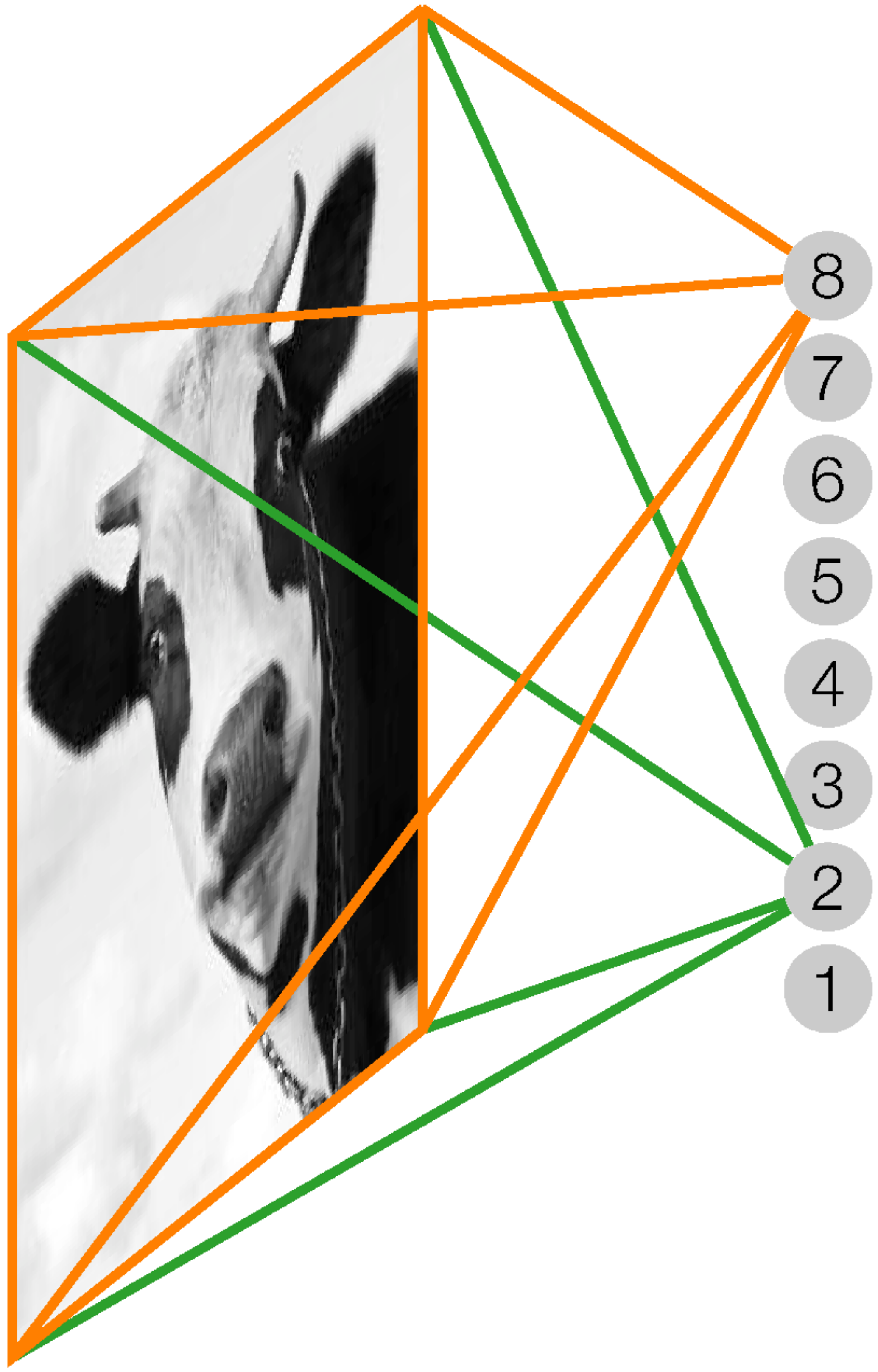
With a convolutional layer, even if the training set had a certain feature only in one part of the images – say, always in the upper part of the images – it will be detected in a test image also when it appears in another region.

Since the filters are typically much smaller than the full image, they are sensitive only to the configuration of pixels in a small neighborhood.

With dense layers one could achieve something similar to a convolutional layer with setting for each neuron all weights to zero except those in a small region of the input space and using data augmentation.

We will look at this in more details on the next slides.

From dense to convolutional layers

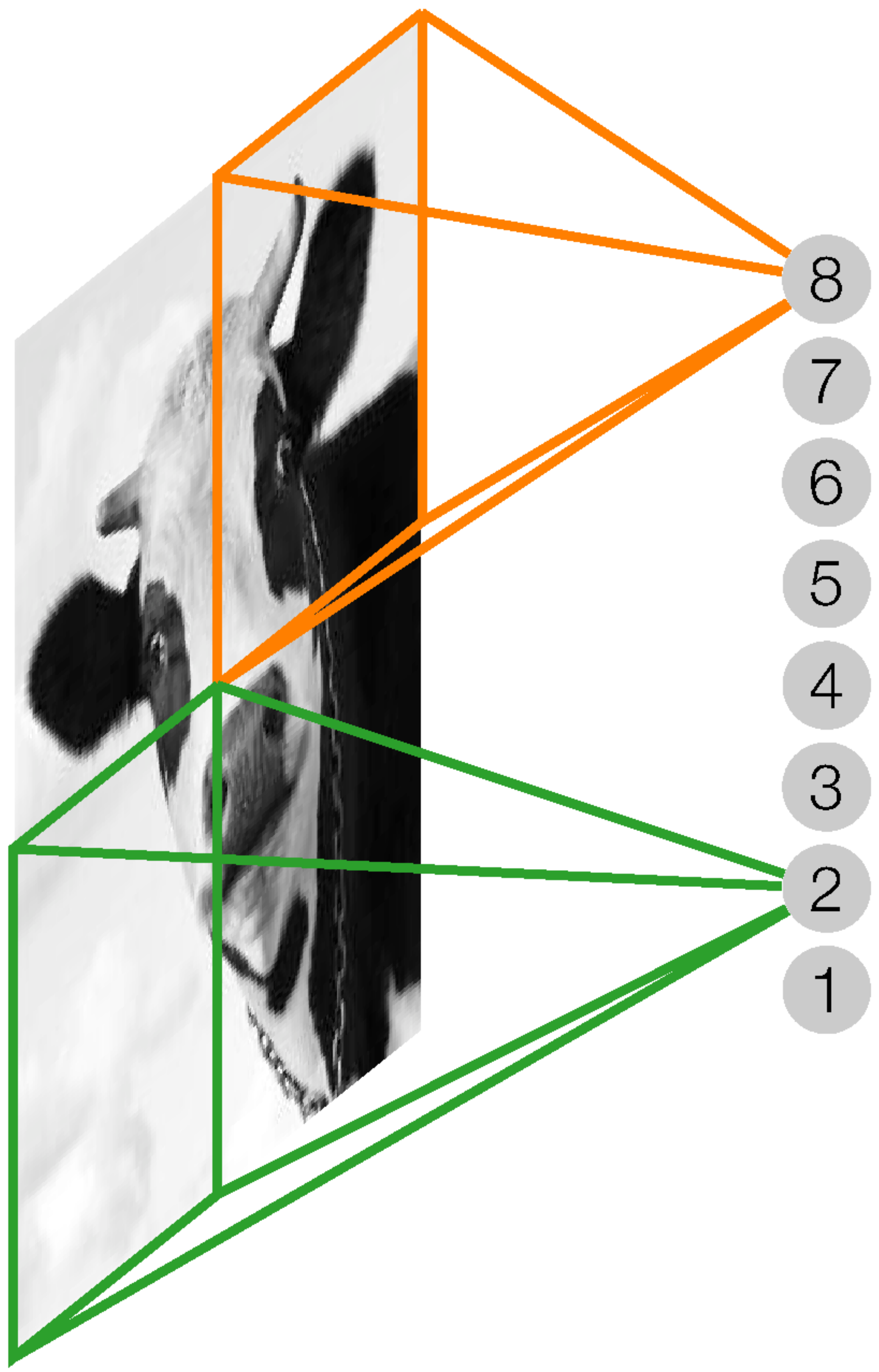


dense (all-to-all) connectivity

Previous slide.

Let us take the image on the left as input and a dense layer with 8 neurons that all connect to all pixels of the input image.

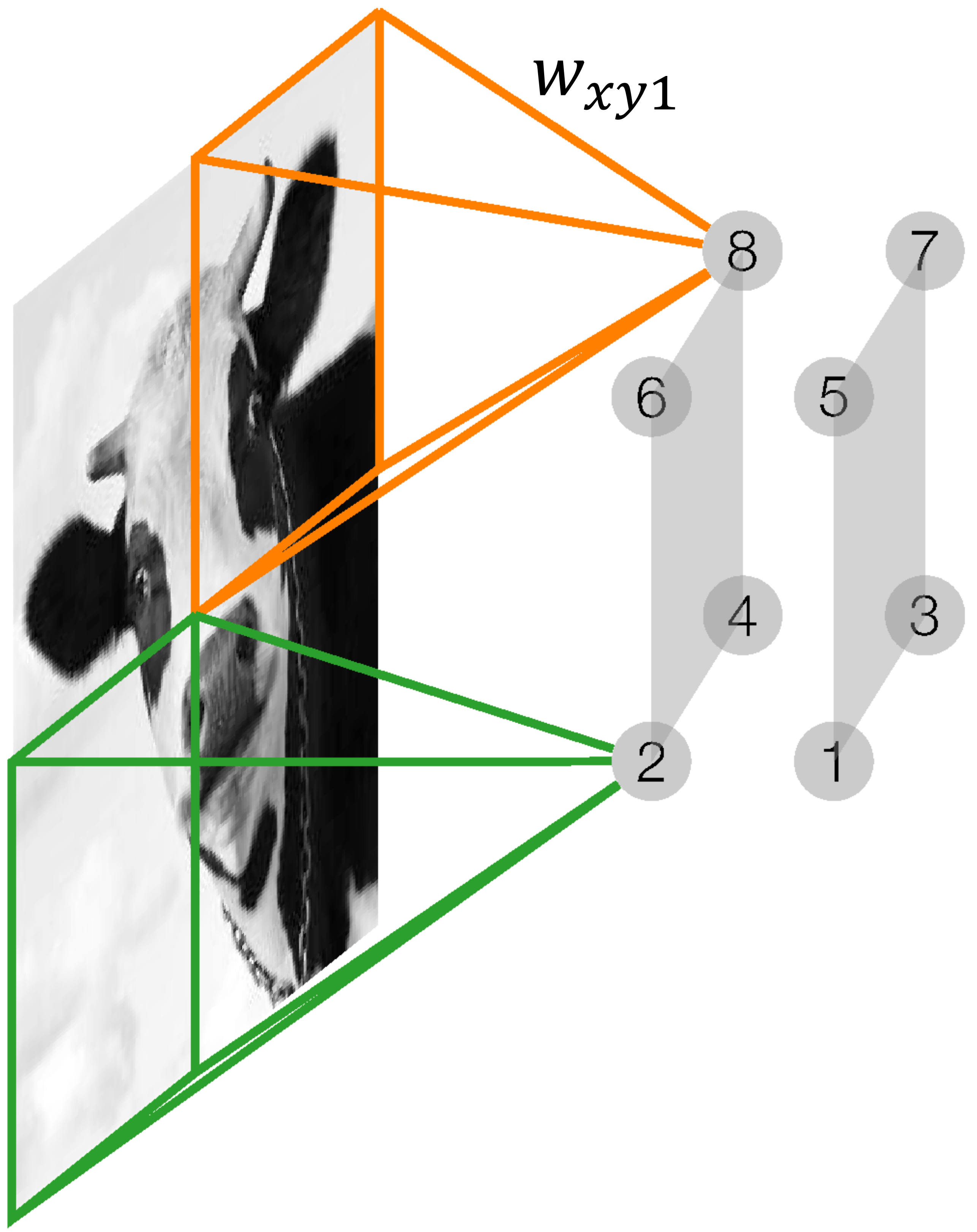
From dense to convolutional layers



Previous slide.

Now let us set most weights of these neurons to zero, such that neurons 1 and 2 only have non-zero weights to pixels in the bottom left quadrant of the image, neurons 7 and 8 only have non-zero weights to pixels in the top right quadrant etc.

From dense to convolutional layers



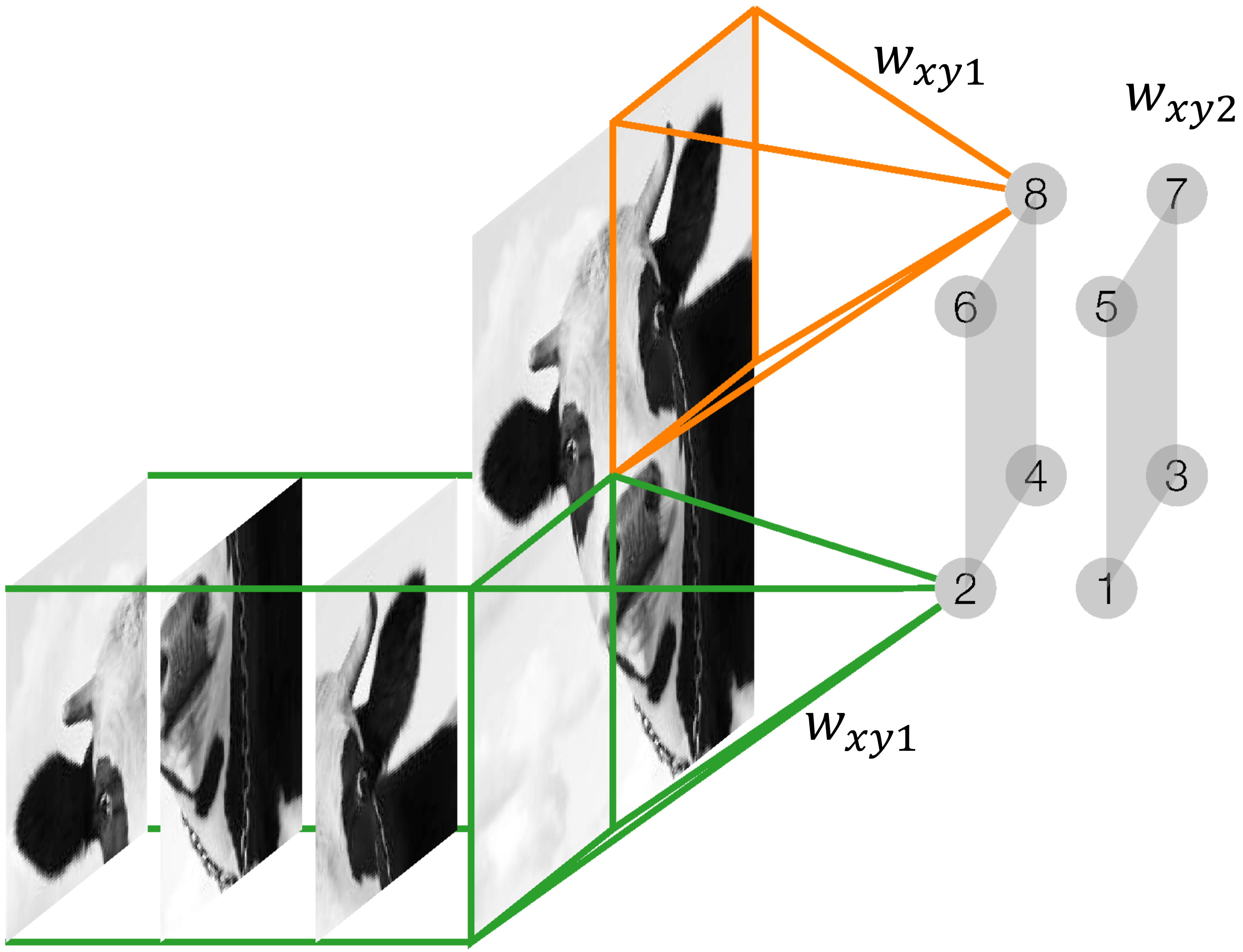
patchy connectivity

Previous slide.

Now we rearrange the neurons according to the region where they have non-zero input weights. This already resembles a lot the volumes we saw for convolutional networks.
We will call the left vertical plane the “even feature map” and the other one the “odd feature map”.

Let now all neurons in each feature map have exactly the same weight vector, e.g. the orange weight vector and the green weight vector are the same. Think of the green weight vector as the concatenation of the weights from each row of the patch.
But neurons in different feature maps shall have different weight vectors.

From dense to convolutional layers



patchy connectivity & data augmentation

Previous slide.

During training we now do data augmentation, such that each neuron sees each patch of the image equally often. Since the weight vectors of the neurons in each feature map were initialized to the same values and all neurons see the same training data, the weight vectors of all neurons in the same feature map will always remain the same.

You should realize now that we have total equivalence to a convolutional layer. But instead of keeping different neurons with the same weight vectors we replace all the neurons in one feature map with a single filter and instead of data augmentation, we move this filter over all patches of the image.

In the introductory example we discussed, how setting some weights to zero and doing data augmentation installs an inductive bias. Here we see how such an inductive bias can be absorbed in the architecture of the network.

Summary: Inductive bias of a convolutional layer

A convolutional layer is a special case of a dense layer with

- many neurons having the same weights
- many weights zero, except in a small neighborhood

→A normal deep network with dense connections should be able to learn the same tasks!

A convolutional layer brings two intuitions in the form of an inductive bias:

1) Independence to translation of filter

   "A feature detector that's useful in one part of the image is probably also useful in another part of the image."

2) Local features are useful to understand images

Previous slide.
Your notes.

Artificial Neural Networks

Johanni Brea & W. Gerstner

EPFL, Lausanne, Switzerland

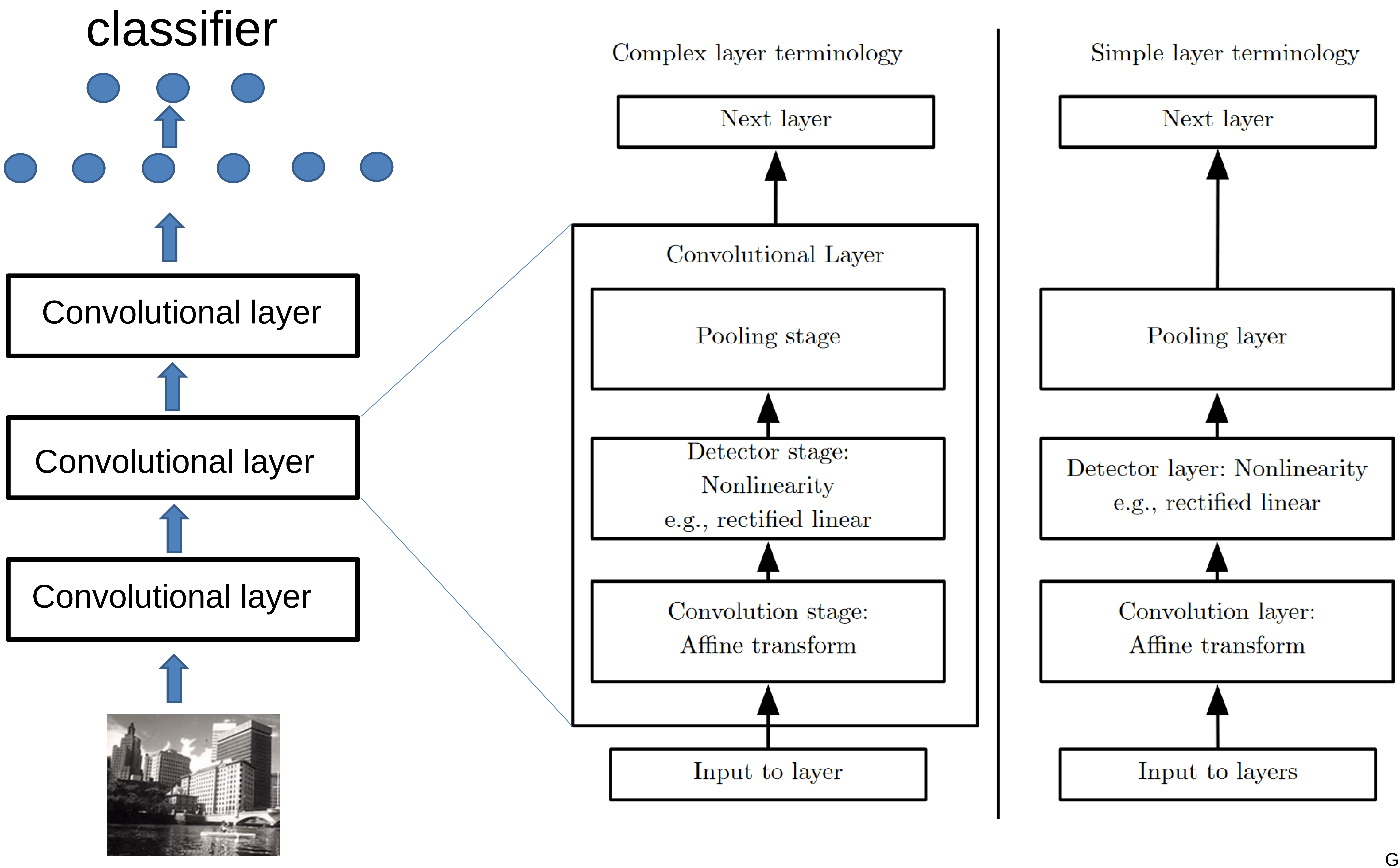
Inductive Bias and Convolutional Neural Networks

Part 4: MaxPooling as inductive bias for images

1. No Free Lunch Theorem
2. Inductive bias in machine learning
3. Convolution filters as inductive bias for images
4. **MaxPooling as inductive bias for images**

Previous slide.
After the convolutional layer follows a max-pooling layer.

Review: Components of a typical conv-net layer



classifier

Convolutional layer

Convolutional layer

Convolutional layer

Complex layer terminology

Next layer

Convolutional Layer

Pooling stage

Detector stage:
Nonlinearity
e.g., rectified linear

Convolution stage:
Affine transform

Input to layer

Simple layer terminology

Next layer

Pooling layer

Detector layer: Nonlinearity
e.g., rectified linear

Convolution layer:
Affine transform

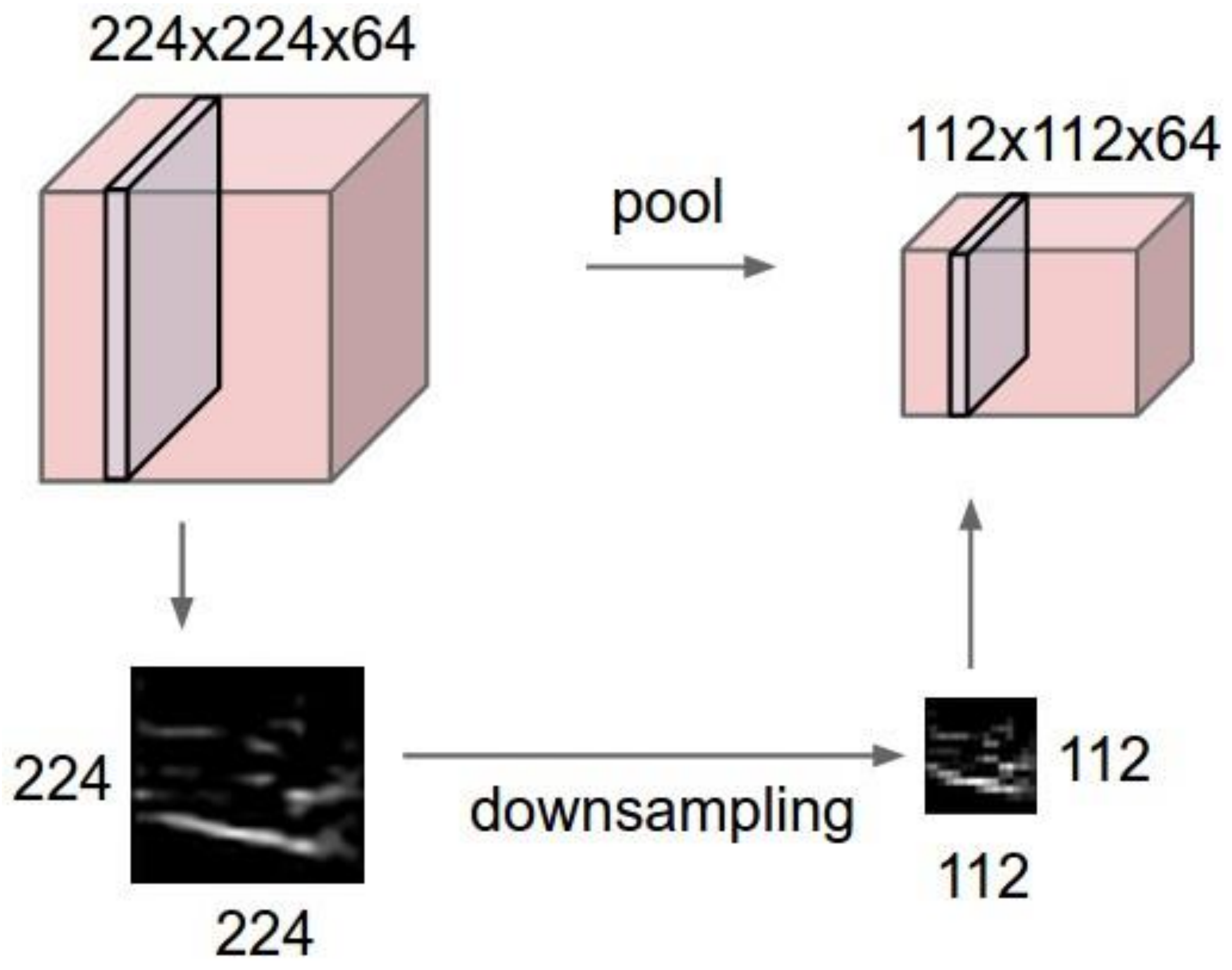
Input to layers

Goodfellow et al. Chapter 9.3

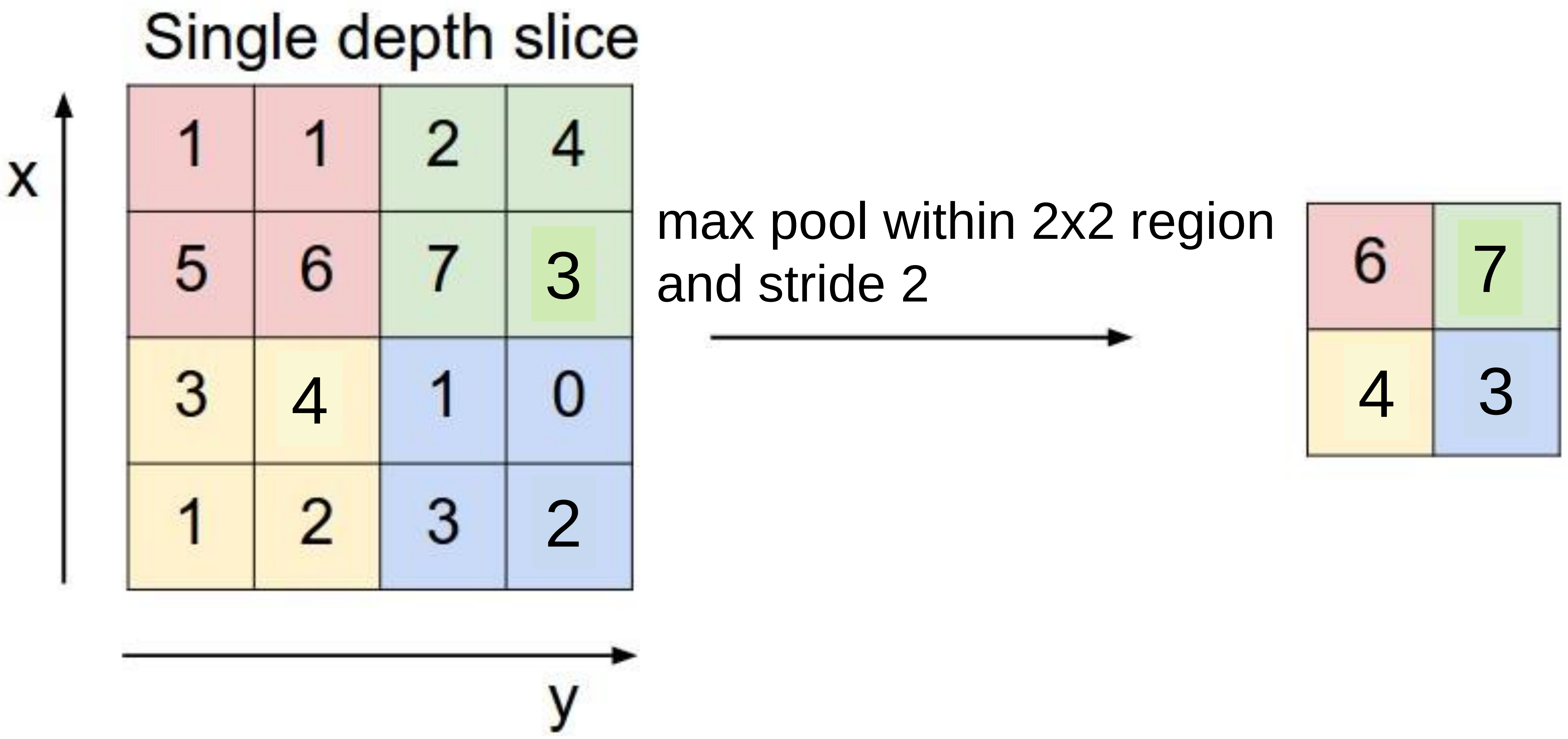
Previous slide.

Nevertheless, pooling was so popular that even in textbooks the definition of convolutional layers sometimes includes a pooling stage, as in the complex layer terminology on the left, where one layer consists of a convolution, nonlinearity and a pooling stage.

Pooling stage: Max Pooling



Response of filter No 17:



1	1	2	4
5	6	7	3
3	4	1	0
1	2	3	2

max pool within 2x2 region and stride 2

6	7
4	3

Previous slide.

Let us move on to a second component that is often used in convolutional networks: max pooling. Any kind of pooling reduces the x-y dimension of a volume; by a factor two in each dimension in the example on the left.

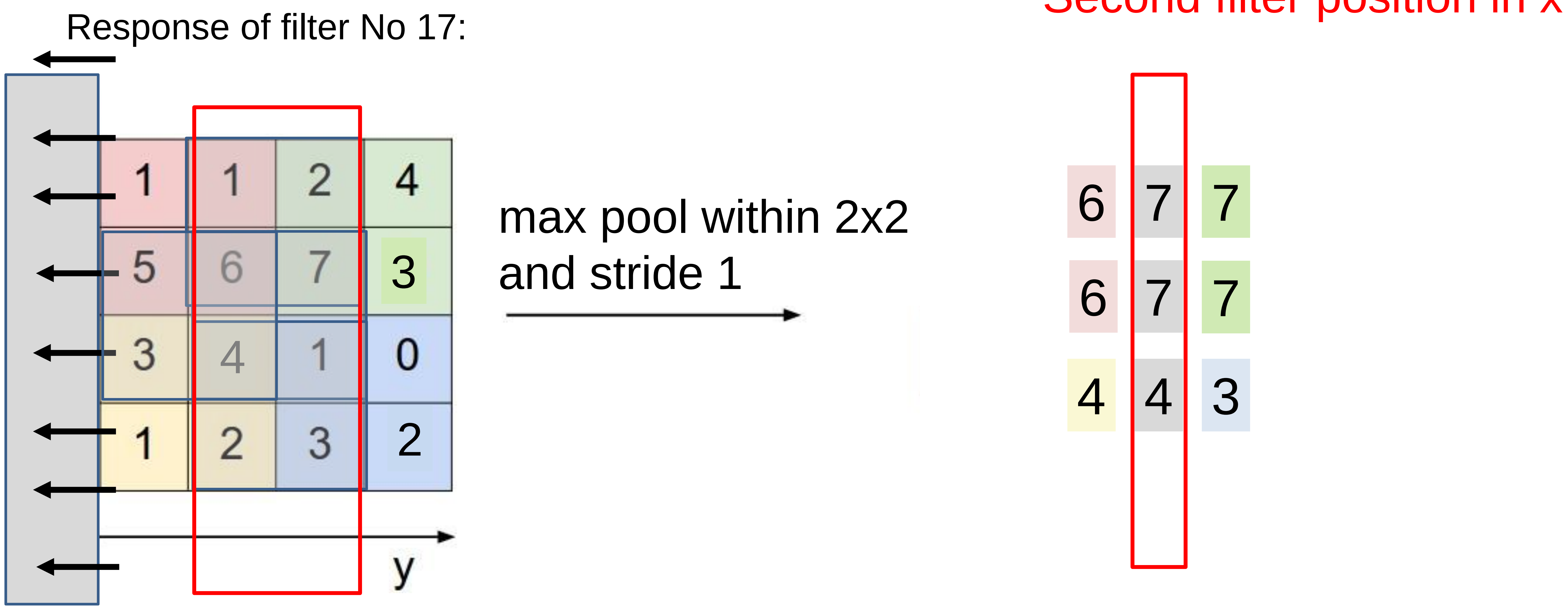
A max-pooling layer with a filter of size 2x2 and stride 2 picks the maximal number in groups of 4 neighboring pixels, as illustrated on the right.

If we move the input image on the right one pixel to the left and pad with zeros, the red, green and blue output would remain the same and the yellow output would switch from 3 to 2. A max-pooling layer thus implements an inductive bias that small translations should not have a large effect on the output of the neural network. Note, however, that the outcome changes quite a bit, if instead one moves the input one pixel to the right.

Instead of using the maximum, people sometimes use also the median or the mean.

Reducing the size of the output volume may be desirable from the perspective of computational resources. Pooling is however only one way to achieve this: increasing the stride in a convolutional layer achieves the same.

Max Pooling: stride one

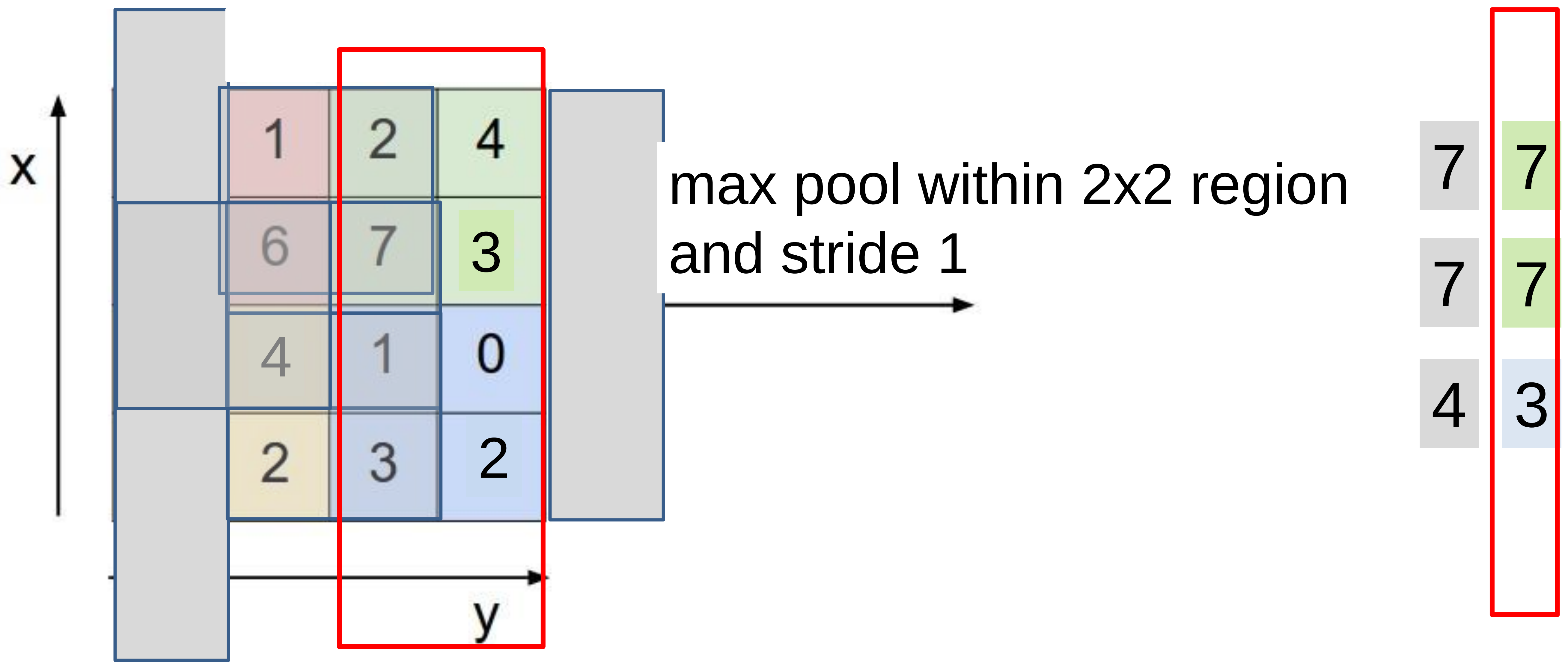


Focus on second filter position in x and
move the input image by one pixel to the left: many outputs keep the same value!

Max Pooling: stride one, small shift of image

Response of filter No 17:

Second filter position in x



1	2	4
6	7	3
4	1	0
2	3	2

max pool within 2x2 region and stride 1

7	7
7	7
4	3

Inductive bias of max pooling

Invariance to small translations with some probability

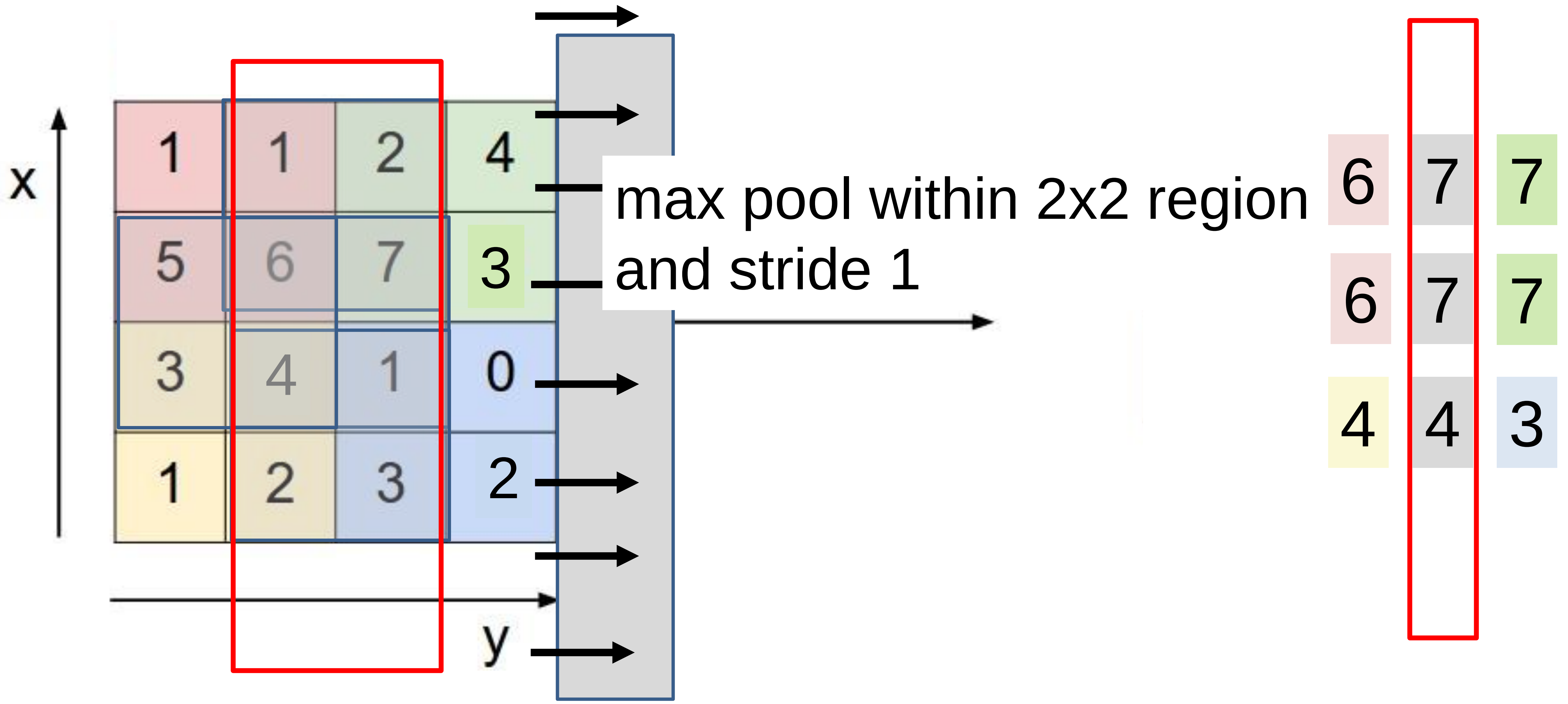
f(x) is invariant to local translation T if f(T(x)) = f(x)

“most neuronal outputs after pooling remain the same when wiggling the image”

Max Pooling: stride one, small shift of image

Response of filter No 17:

Second filter position in x



1	1	2	4
5	6	7	3
3	4	1	0
1	2	3	2

max pool within 2x2 region and stride 1

6	7	7
6	7	7
4	4	3

Inductive bias of max pooling

Invariance to small translations with some probability

f(x) is invariant to local translation T if f(T(x)) = f(x)

“most neuronal outputs after pooling remain the same when wiggling the image”

Previous slide.

The invariance to small translations is best implemented with a stride of one in the pooling layer

If we move the input image one step to the left and pad with zeros, quite a few responses remain identical (see the responses in the second column, highlighted by red rectangle).

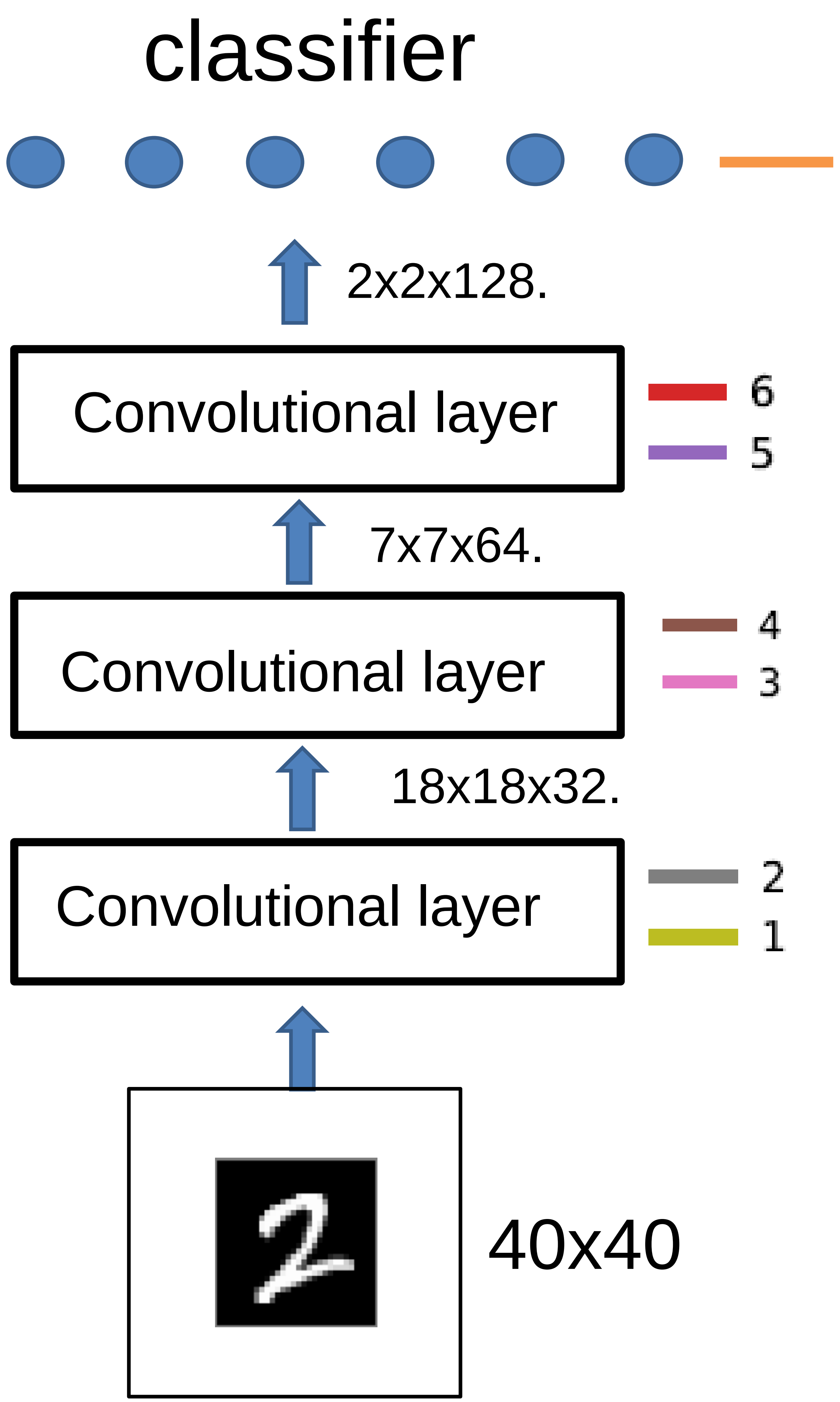
The one that changes (at the bottom of the second column) is in turn stable if we shift the image one step to the right.

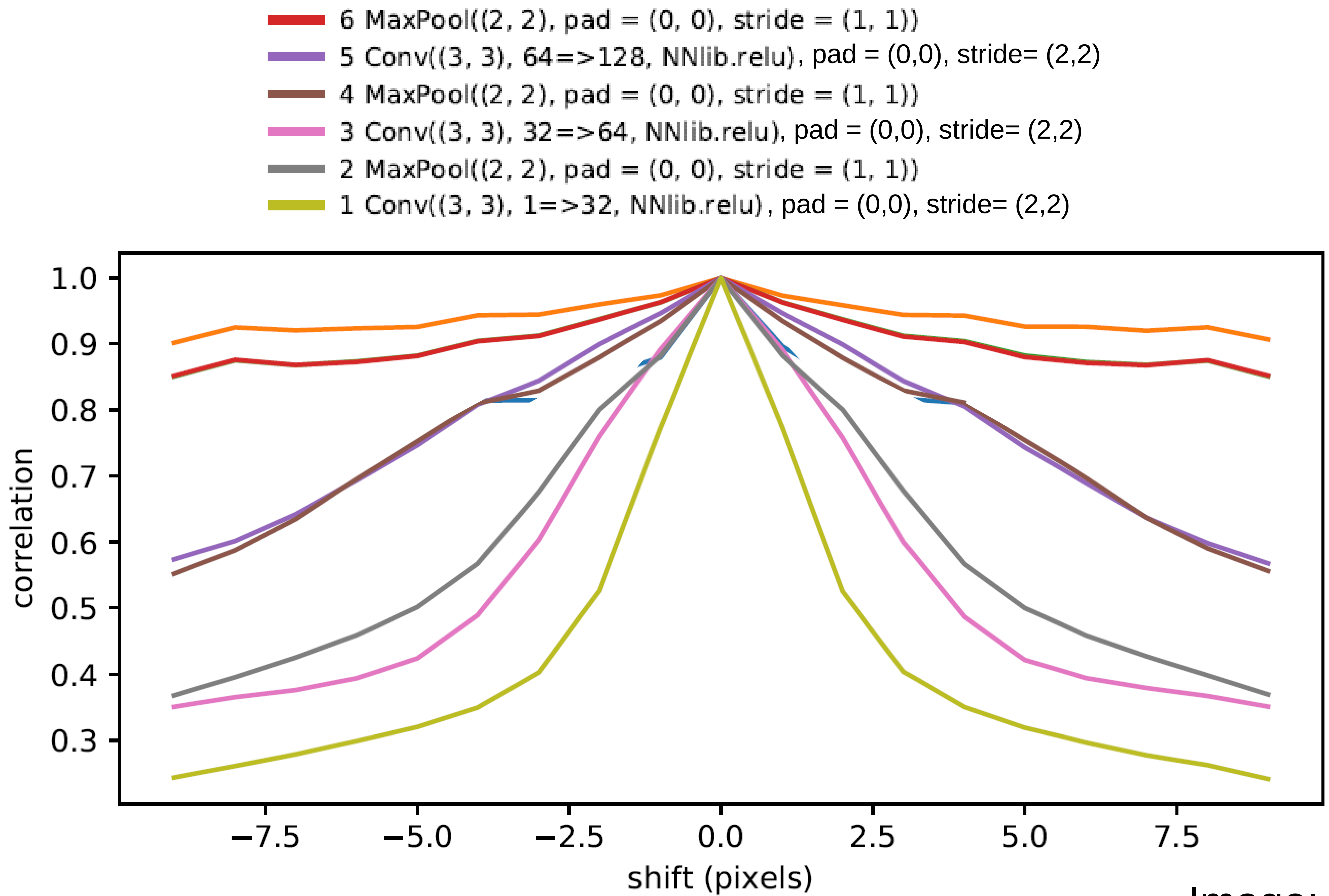
Note that when I say ‘shift by one step’ it really means ‘shift the original image at the entrance to the convolutional layer by an amount that corresponds to the stride in the convolutional layer’. So if the **first** convolutional layer has a stride of 2 then this means we shift the original image by 2 pixels. Thus a sequence of convolutional filter followed by nonlinearity followed by maxpooling implements an (approximate) local invariance to a translation by two pixels.

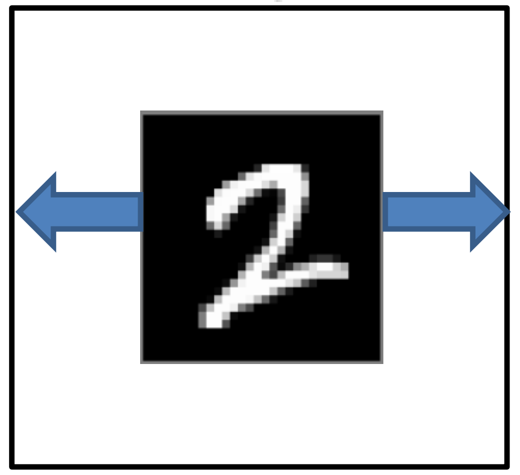
If in addition you also want to ensure translation invariance to a shift by a single pixel, you should use data augmentation where each image is jittered by one pixel to the left and to the right.

In the end we will stack many convolutional layers. If each time we get invariance by two pixels, then we have after 10 layers invariance across 1000 pixels!

ConvNet with Max Pooling stride one: global shift of image







40x40

Image:
Bernd Illing

Previous slide.

In the end we will stack many convolutional layers. If each time we get invariance by two pixels, then we have after 10 layers invariance across 1000 pixels!

In the example here, Bernd Illing (who was then a TA for the class and a PhD student in the lab) used 28x28 MNIST images embedded (by zero padding) into a 40x40 input field. He used 3 convolutional layers, each with a filtering stage, a ReLu nonlinarity, and a pooling stage. Filtering had a stride of 2, whereas pooling had stride 1. No padding.

The first convolutional stage used 32 different filters, applied with stride 2, makes 19x19x32. The pooling stage with stride 1 (region 2x2) reduces this to 18x18x32.

The second convolutional stage used 64 filters, applied with stride 2, makes 8x8x64. The pooling stage with stride 1 (but region 2x2) reduces this to 7x7x64

The red line, labeled 6, is the output of the third convolutional layer, after the pooling stage (2x2x128=512). If the image is now shifted to the left or to the right, the activity pattern changes only very little.

Critic of max-pooling in Deep Convolutional Networks

“The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.” *Geoff Hinton*
https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton/clyj4jv/

https://openreview.net/pdf?id=HJWLfGWRb https://www.youtube.com/watch?v=pPN8d0E3900

“We find that max-pooling can simply be replaced by a convolutional layer with increased stride without loss in accuracy on several image recognition benchmarks.”
Springenberg et al. https://arxiv.org/abs/1412.6806 See also https://openreview.net/forum?id=HJeuOiRqKQ

(In addition to max-pooling …) “Other popular pooling functions include the average of a rectangular neighborhood, the L2 norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel.” *Goodfellow et al., Deep Learning*

Support of max-pooling in Deep Convolutional Networks (DCN)

“we see that architectures and layer types commonly used in today’s DCNs can be derived from precise probabilistic assumptions that entirely determine their structure”

Patel, Nguyen, Baraniuk http://papers.nips.cc/paper/6231-a-probabilistic-framework-for-deep-learning

Previous slide.

The popularity of pooling layers in practical applications has decreased in recent years.

However, theoretical studies have shown that Deep Convolutional Networks (DCN) arise naturally in the framework of probabilistic generative models (called DRMM) if local translation variance is part of the generative process:

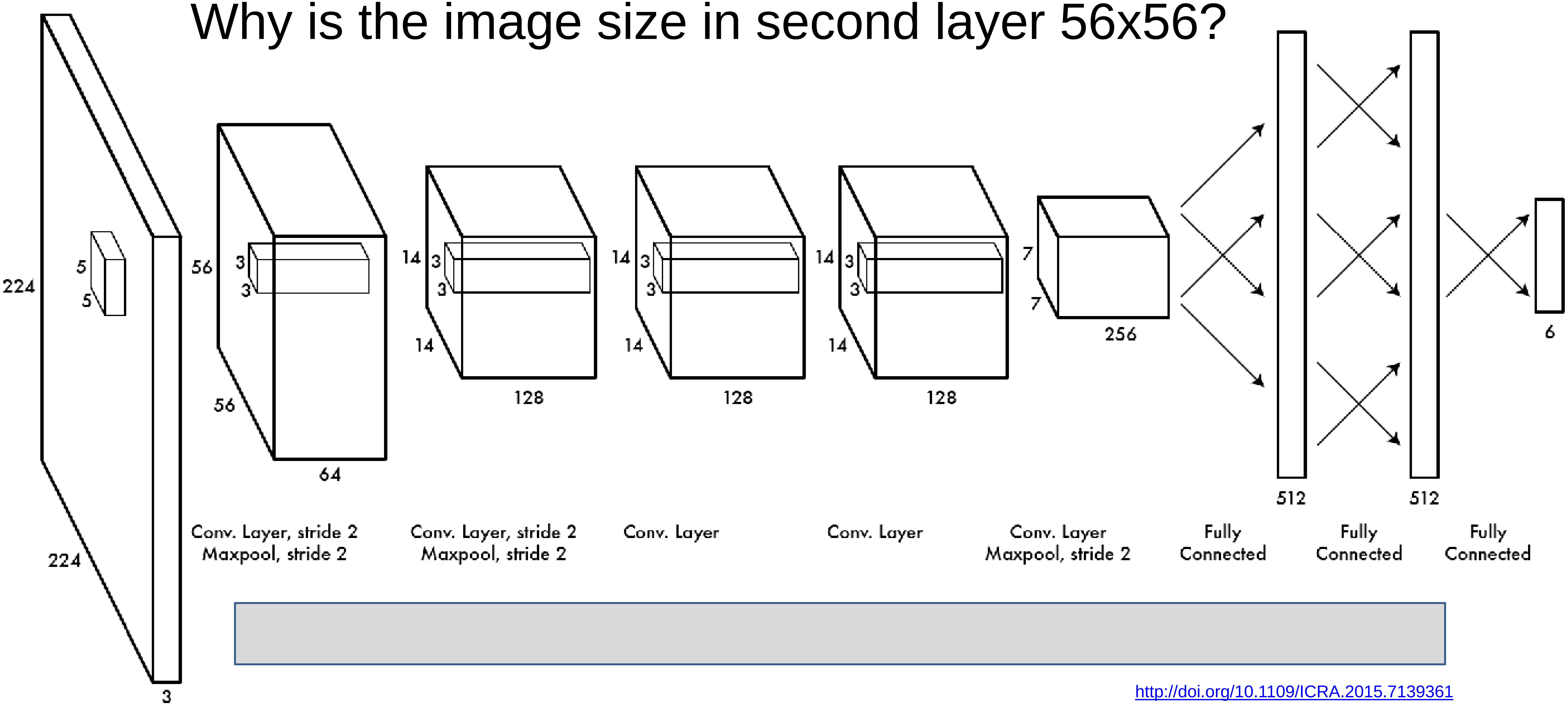
“The convolution, Max-Pooling and ReLu operations in a DCN correspond to max-sum/product inference in a DRMM. … Thus, we see that architectures and layer types commonly used in today’s DCNs can be derived from precise probabilistic assumptions that entirely determine their structure. The DRMM therefore unifies two perspectives—neural network and probabilistic inference”

Patel, Nguyen, Baraniuk, A probabilistic framework for deep learning,
30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.
http://papers.nips.cc/paper/6231-a-probabilistic-framework-for-deep-learning

Convolutional network

How many filters are used in first layer?

Why is the image size in second layer 56x56?



http://doi.org/10.1109/ICRA.2015.7139361

Previous slide.

Putting everything together we can draw nice images like these to describe the network architecture. You should now be able to understand such graphs.

E.g.
How many filters are there in the first convolutional layer? The answer is 64

What padding was used in the first convolutional layer? 2. Two times stride of 2 reduces each dimension by a factor 4. Since 224/56 = 4, we conclude that the center of the filters was placed also at all edges of the image, which for a filter of size 5x5 means that a padding of 2 is needed.

Summary: Convolutional network

1) Convolutional networks exploit (partial) translation invariance of objects in images.
2) The inductive bias (“local translation invariance”) is encoded in the network architecture (several “convolutional layers”)
3) Pooling with stride 2 is often used to reduce dimensionality. However, pooling with stride 1 is more compatible with the idea of local translation invariance and hence recommended.
4) Dimensionality reduction can be achieved by strides >1 in the convolutional filtering stage.
5) Maxpooling can be replaced in practice by mean-pooling.
6) People claim that in practice a pooling stage is not even necessary.

Your notes.

Quiz: Convolutional Networks

[ ] The number of adjustable weights in a convolutional layer with 10 filters of size 5x5x3 followed by max-pooling in a neighborhood of 2x2 is 750 (excluding biases).

[ ] The number of weights in a convolutional layer does not depend on the size of the x-y dimension of input layer.

[ ] The number of outputs of a convolutional layer with 10 filters of size 5x5x3 followed by max-pooling in a neighborhood of 2x2 cannot be larger than 750.

[ ] A deep convolutional network with a sufficient number of filtering and max-pooling layers is also invariant under rotations of the image.

[ ] You have a dataset of centered portraits (passport photos) on white background. Global translation invariance is a good inductive bias for this dataset.

[ ] Pooling by averaging over a rectangular neighborhood is a special case of standard deep multilayer networks with feedforward architecture, whereas max-pooling is not.

Your notes.

Teaching monitoring – monitoring of understanding

[ ] today, up to here, at least 60% of material was new to me.

[ ] up to here, I have the feeling that I have been able to follow (at least) 80% of the lecture.

(previous slide)

Artificial Neural Networks

Johanni Brea & W. Gerstner

EPFL, Lausanne, Switzerland

Inductive Bias and Convolutional Neural Networks

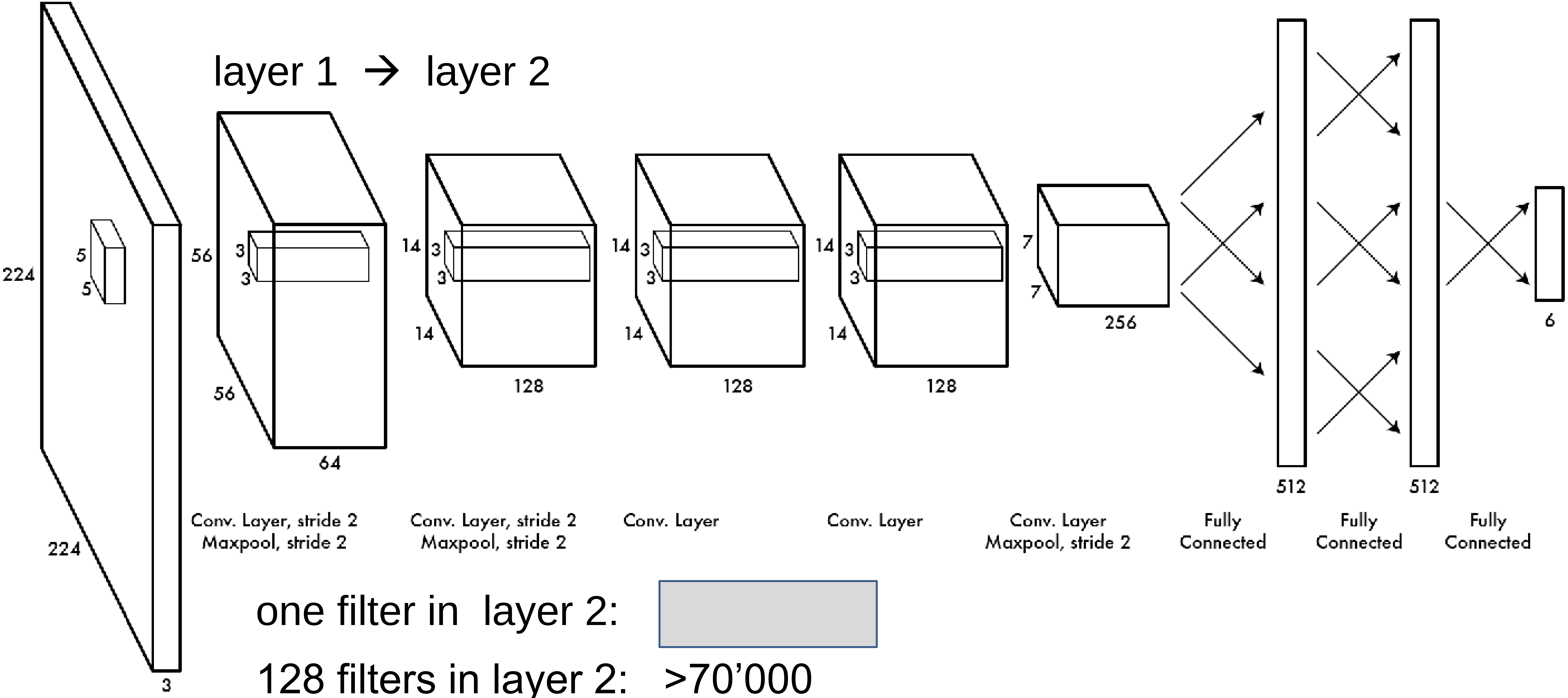
Part 5: The gradient of a convolutional layer

1. No Free Lunch Theorem
2. Inductive bias in machine learning
3. Convolution filters as inductive bias for images
4. MaxPooling as inductive bias for images
5. **The gradient of a convolutional layer**

Previous slide.
As in standard deep networks parameters are adjusted by gradient descent using Backprop. But BackProp is just a special case of automatic generalization.
We now sketch the framework of automatic differentiation as a generalization of Backprop.

Convolutional network: Filters are optimized by learning

How many weight parameters from layer 1 to layer 2?



layer 1 → layer 2

one filter in layer 2:

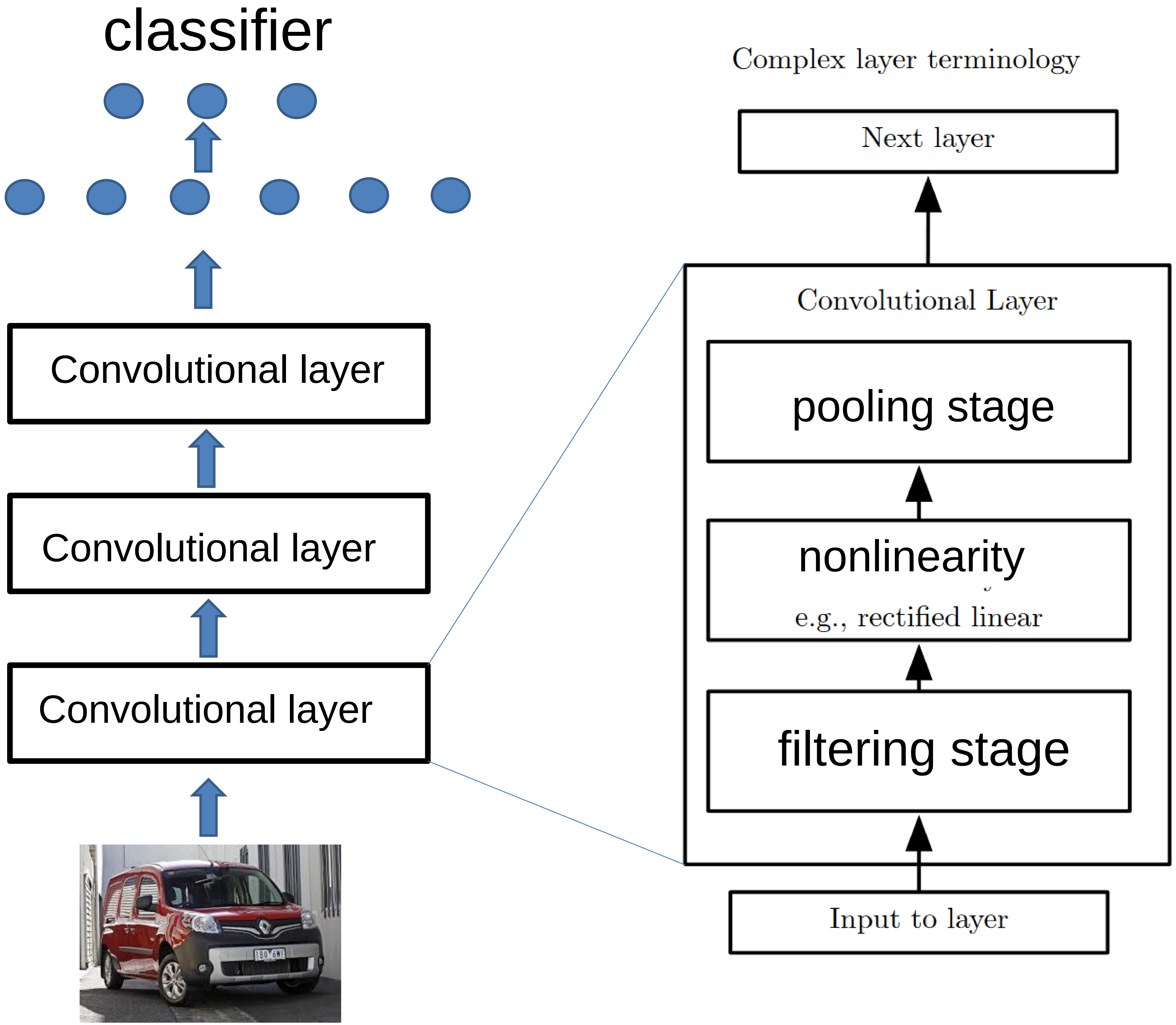
128 filters in layer 2: >70’000

http://doi.org/10.1109/ICRA.2015.7139361

There are many parameters.
The only novel aspects are that
(i) we have some weight sharing (easy to deal with) and
(ii) that the max-operation comes into play.

The following examples shows what this implies (see also Exercise 1)

Components of a typical conv-net layer



pooling in 2x2 region with stride 1

$$x^{(1,out)}_{ijk} = \max\{x^{(1)}_{i,j,k}, x^{(1)}_{i+1,j,k}, x^{(1)}_{i,j+1,k}, x^{(1)}_{i+1,j+1,k}\}$$

$$x^{(1)}_{ijk} = \sigma(a_{ijk})$$

$$a_{ijk} = b_k + \sum_{x=1}^{5}\sum_{y=1}^{5}\sum_{c=1}^{3} I_{i+x-1,j+y-1,c}\, w^{(1)}_{xyck}$$

Previous:
Just a reminder of the steps within one convolutional layer.

Filters are optimized by learning

Novel aspects: - weight sharing

- max-pooling

Exercise 1 and 2 now
Next Lecture at 14h15

Example: $1 \leq k \leq K$ convolutional filters 5x5x3 in layer 1 followed by nonlinearity σ (and max pooling) transmitted to a single unit (index *o*) in layer 2:

$$\hat{y}_o = \sum_{ijk} w^{(2)}_{ijko} x^{(1)}_{ijk}$$

$$x^{(1,out)}_{ijk} = \max\{x^{(1)}_{i,j,k}, x^{(1)}_{i+1,j,k}, x^{(1)}_{i,j+1,k}, x^{(1)}_{i+1,j+1,k}\}$$

$$x^{(1)}_{ijk} = \sigma(a_{ijk})$$

$$a_{ijk} = b_k + \sum_{x=1}^{5}\sum_{y=1}^{5}\sum_{c=1}^{3} I_{i+x-1,j+y-1,c} w^{(1)}_{xyck}$$

with max pooling over region 2x2, stride 1

Optimize weight w_{1135} of filter with filter index $k = 5$, input channel *c=3* and location (1,1). Quadratic loss L.

Exercise 1 (and 2) Now!

Next Lecture at 14h15

Consider a very simple convolutional neural network with 3 dimensional input (e.g. RGB image), one convolution layer with 5x5x3 filters, stride 1, non-linearity σ and one linear layer, i.e.

$$a_{ijk} = b_k + \sum_{x=1}^{5} \sum_{y=1}^{5} \sum_{c=1}^{3} I_{i+x-1,j+y-1,c} w^{(1)}_{xyck} \tag{1}$$

$$x^{(1)}_{ijk} = \sigma(a_{ijk}) \tag{2}$$

$$\hat{y}_o = \sum_{ijk} w^{(2)}_{ijko} x^{(1)}_{ijk} \tag{3}$$

a) With loss $L = \frac{1}{2} \sum_o (t_o - \hat{y}_o)^2$, compute dL/dw_{1135}

b) Add a max pooling layer of 2x2 region 4 that takes the outputs of equation (2) and transforms it into

$$x^{(1,out)}_{ijk} = \max\{x^{(1)}_{i,j,k}, x^{(1)}_{i+1,j,k}, x^{(1)}_{i,j+1,k}, x^{(1)}_{i+1,j+1,k}\} \tag{2b}$$

before sending it to the output, In other words, use Eq. (3) but with the upper index (1,out) instead of (1) and calculate again the gradient. What changes?

Derive the mathematical result and give an interpretation in words

c. Compare your result to the one you obtain with a dense (fully-connected) layer, i.e. $a_k = b_k + \sum_x \sum_y \sum_c I_{xyc} w^{(1)}_{xyck}$, $x^{(1)}_k = \sigma(a_k)$ and $\hat{y}_o = \sum_k w^{(2)}_{ok} x^{(1)}_k$.

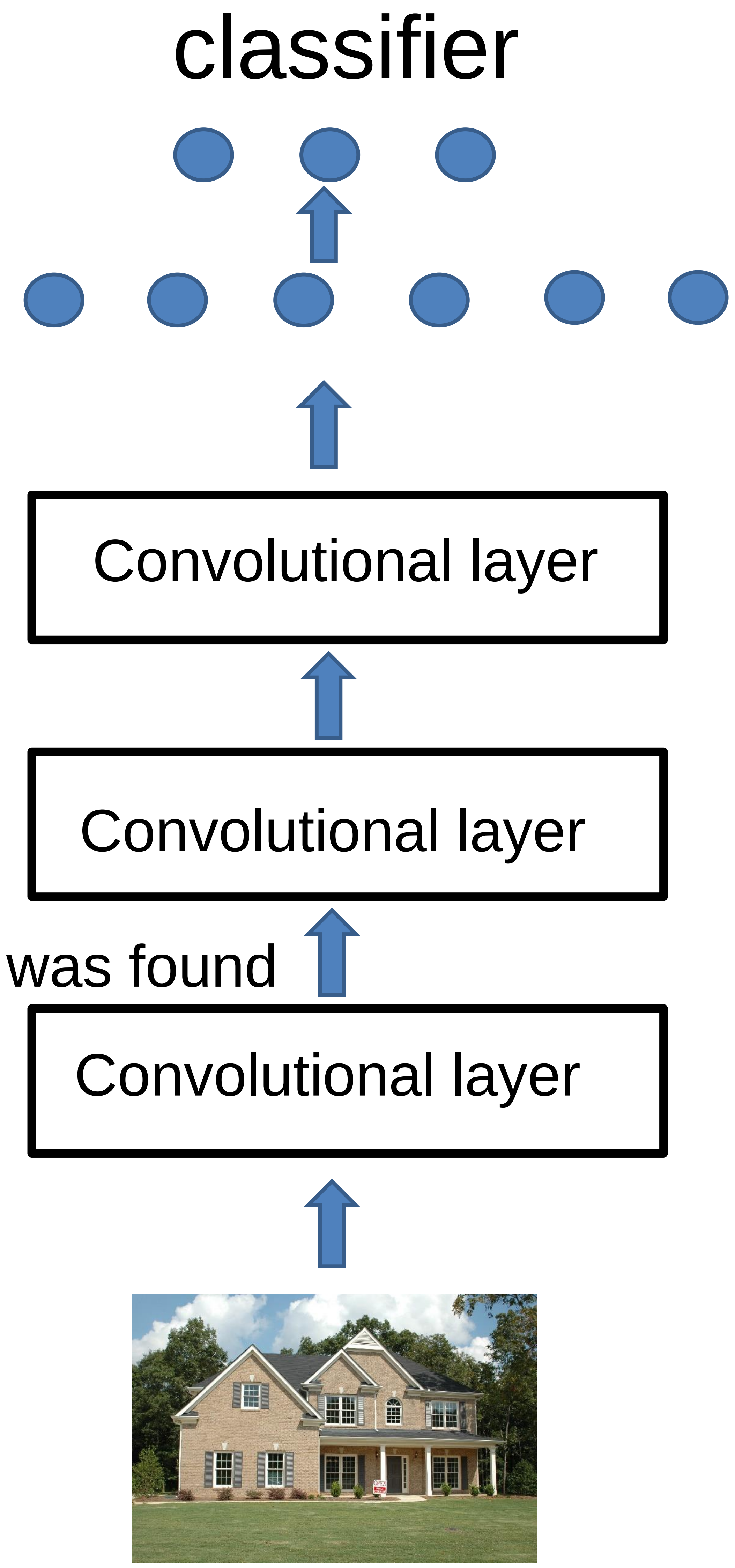
Give an interpretation in words regarding the differences between results from c and b or c and a

BackProp for Max-pooling: Questions?

Blackboard 1

with max pooling, stride 1

→selects those indices (i*,j*) that gave the argmax for some location.



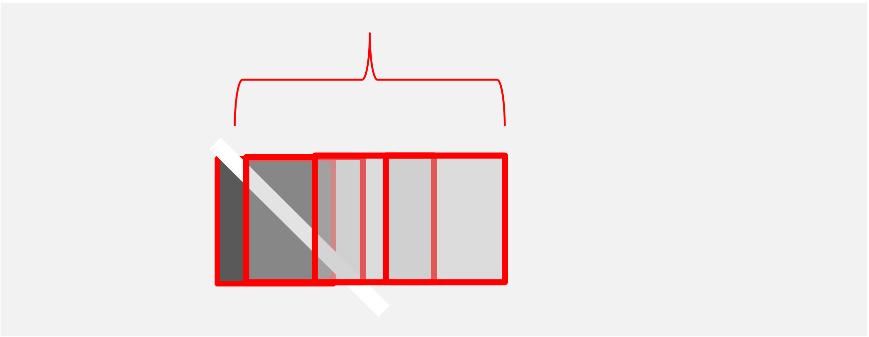
Gradient finds location where a specific feature was found

Your notes.

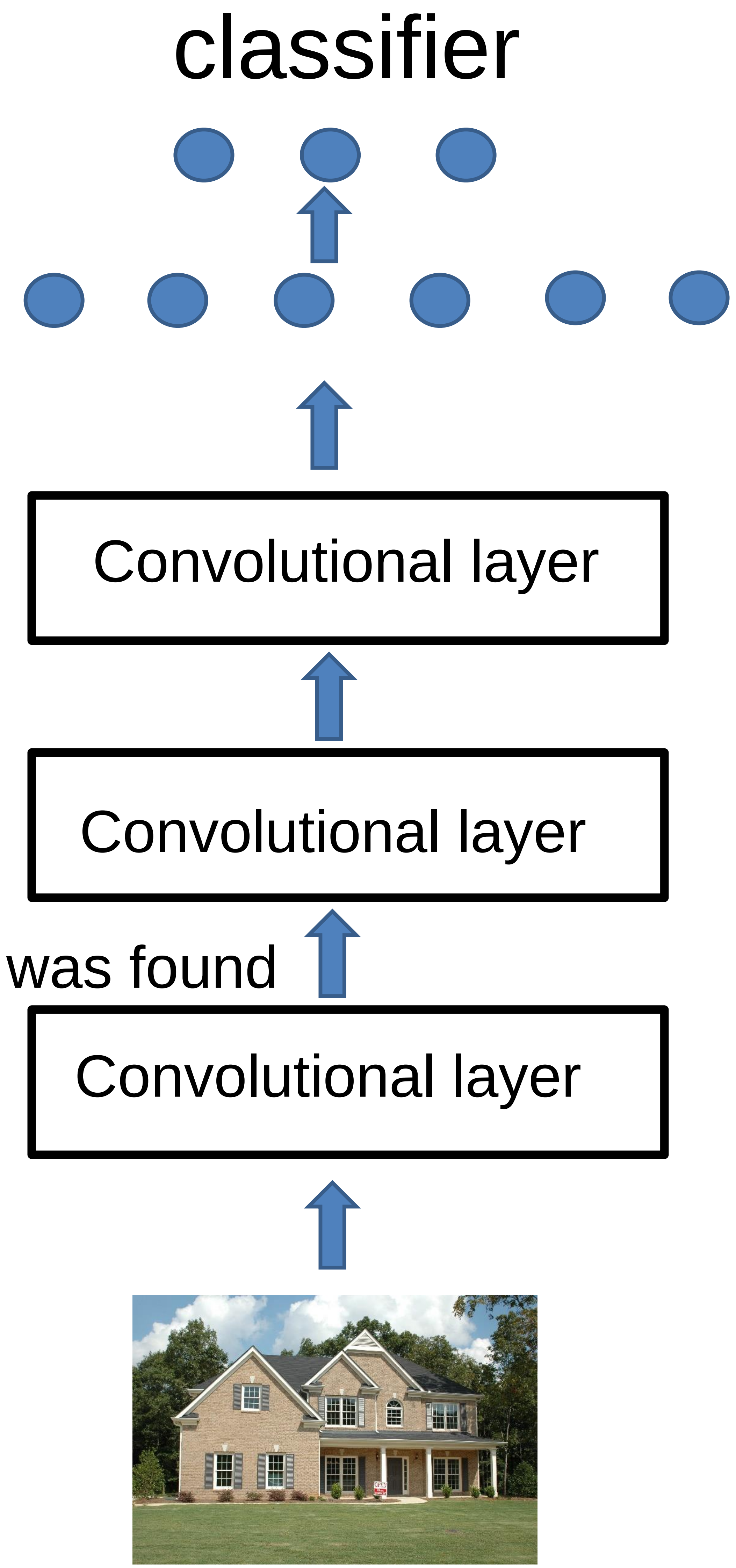
BackProp for Max-pooling: Attentional focus

with max pooling over region 5x5, stride 1

→selects those indices (i*,j*) that gave the argmax for some location.



Gradient finds location where a specific feature was found



classifier

Convolutional layer

Convolutional layer

Convolutional layer

Your notes.

Summary: Gradient across MaxPooling layer

Gradient is taken across max-pooling layer.

Backward path during backpropagation leads to a focusing on the location i*,j* that was the winner.

→ Selection of region/attentional effect

Previous slide.

The gradient across the max-pooling stage automatically focuses on the ‘relevant’ location, i.e., the location that was the winner (within the small region of max-pooling).

Over several layers this leads to a region-finding effect, similar to an attentional focus and can be used to find the relevant regions that decided in favor of a certain output (such as diagonal stripes for the decision of a house with a classic roof).

Artificial Neural Networks

Convolutional Neural Networks

Johanni Brea & W. Gerstner
EPFL, Lausanne, Switzerland

Part 6: Automatic Differentiation: BackProp revisited

1. No Free Lunch Theorem
2. Inductive bias in machine learning
3. Convolution filters as inductive bias for images
4. MaxPooling as inductive bias for images
5. The gradient of a convolutional layer
6. **Automatic Differentiation: BackProp revisited**

Previous slide.
As in standard deep networks parameters are adjusted by gradient descent using Backprop. But BackProp is just a special case of automatic generalization.
We now sketch the framework of automatic differentiation as a generalization of Backprop.

Shall we manually compute the gradients?

No. Use automatic (reverse mode) differentiation.

- Record computation tree on forward path to
- apply chain rule in reverse order.
  (Same idea as backprop in simple feedforward net)

- An implementation requires just a Tracker (to record the computation tree) and analytical expressions of primitive operations

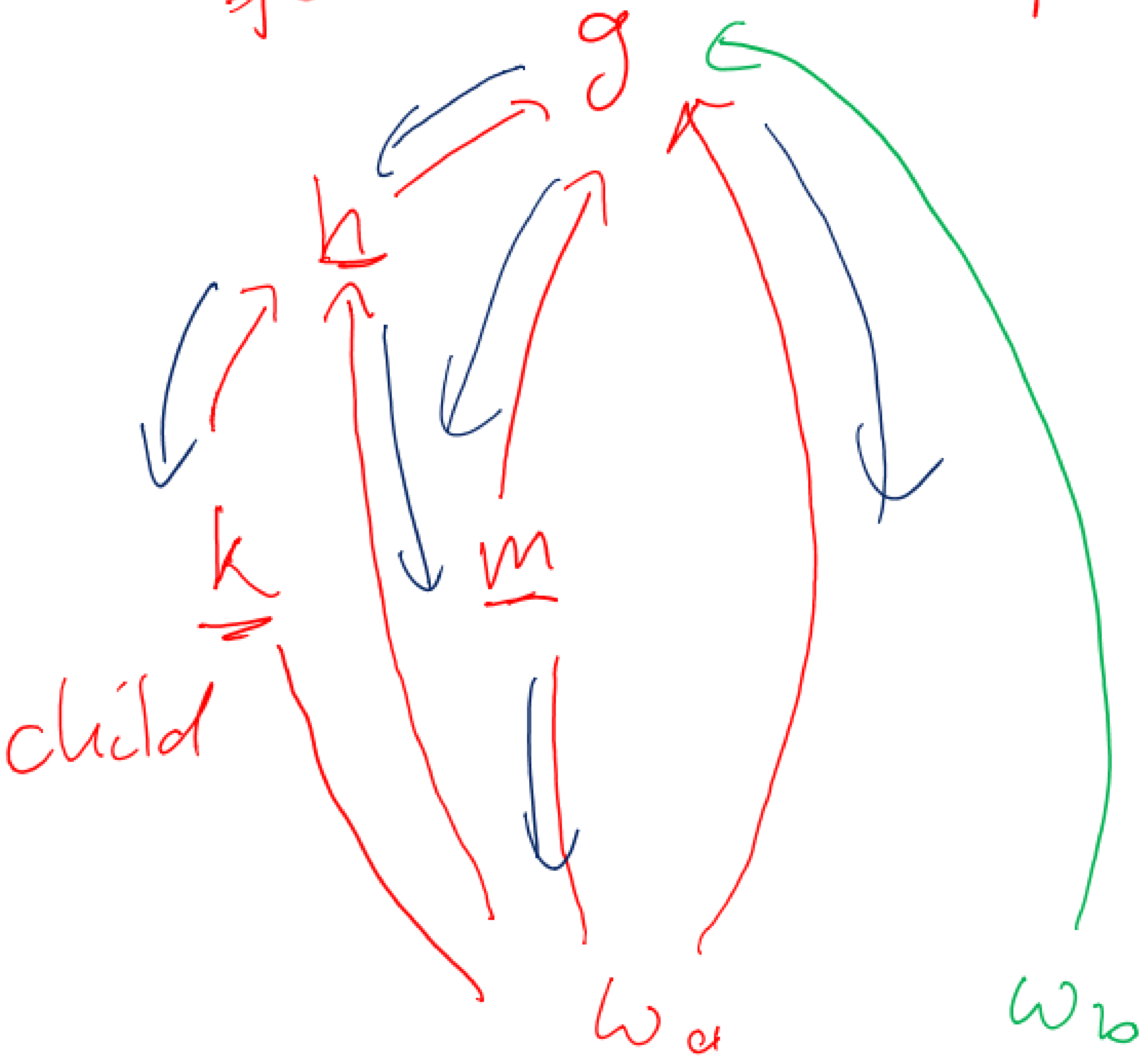
Your notes.

Example: Automatic differentiation

Blackboard 2

$$f(w_a^{\square}, w_b^{\square}) = g[h(k(w_a^{\square}), w_a^{\square}), m(w_a^{\square}), w_a^{\square}, w_b^{\square}]$$

forward calcul. path



$$\frac{\partial f}{\partial w_a} = \frac{\partial g}{\partial h}\left[\frac{\partial h}{\partial k}\frac{\partial k}{\partial w_a} + \frac{\partial h}{\partial w_a}\right]$$

$$+ \frac{\partial g}{\partial m}\cdot\frac{\partial m}{\partial w_a}$$

$$+ \frac{\partial g}{\partial w_a}$$

w_a has 4 children

Previous slide.

In the simple example, the parameter wa has 4 children.
This gives rise to 4 terms in the summation induced by the chain rule.

None of the intermediate functions like m or h has more than one child. Therefore no additional summations are generated.

Summary: Automatic Differentiation (generalized BackProp)

1. Determine children nodes of weight variable.

2. Find a backward (revers ancestral) schedule:
   all children scheduled before node itself.

3. Start with top node and run through reverse schedule

4. Look up primitive operations. Define intermediate variables.

5. Sum over children and multiply (→chain rule)

→Takes care of max pooling, weight sharing, batch normalization etc
→Takes care recurrent connections

Your notes.

Teaching monitoring – monitoring of understanding

[ ] today, up to here, at least 60% of material was new to me.

[ ] up to here, I have the feeling that I have been able to follow (at least) 80% of the lecture.

(previous slide)

Artificial Neural Networks

Convolutional Neural Networks

EPFL

Johanni Brea & W. Gerstner

EPFL, Lausanne, Switzerland

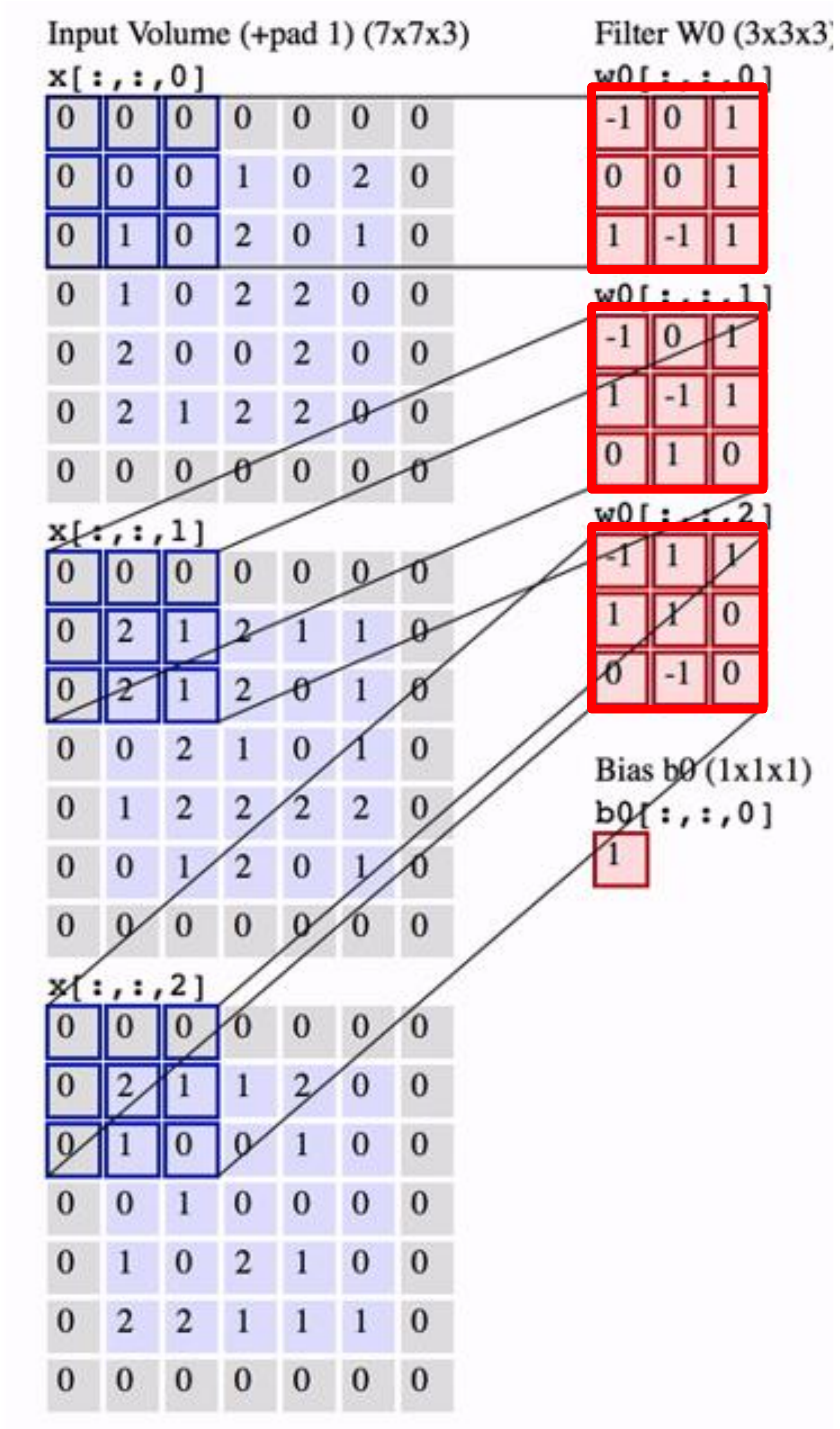
Part 7: Reducing the number of parameters: space and depth

1. No Free Lunch Theorem
2. Inductive bias in machine learning
3. Convolution filters as inductive bias for images
4. MaxPooling as inductive bias for images
5. The gradient of a convolutional layer
6. Automatic differentiation: BackProp revisited
7. **Reducing the number of parameters: space and depth**

Previous slide.

We have a large number of parameters. With some additional assumptions (inductive bias!) we can reduce the number significantly.

Convolution: colors and padding

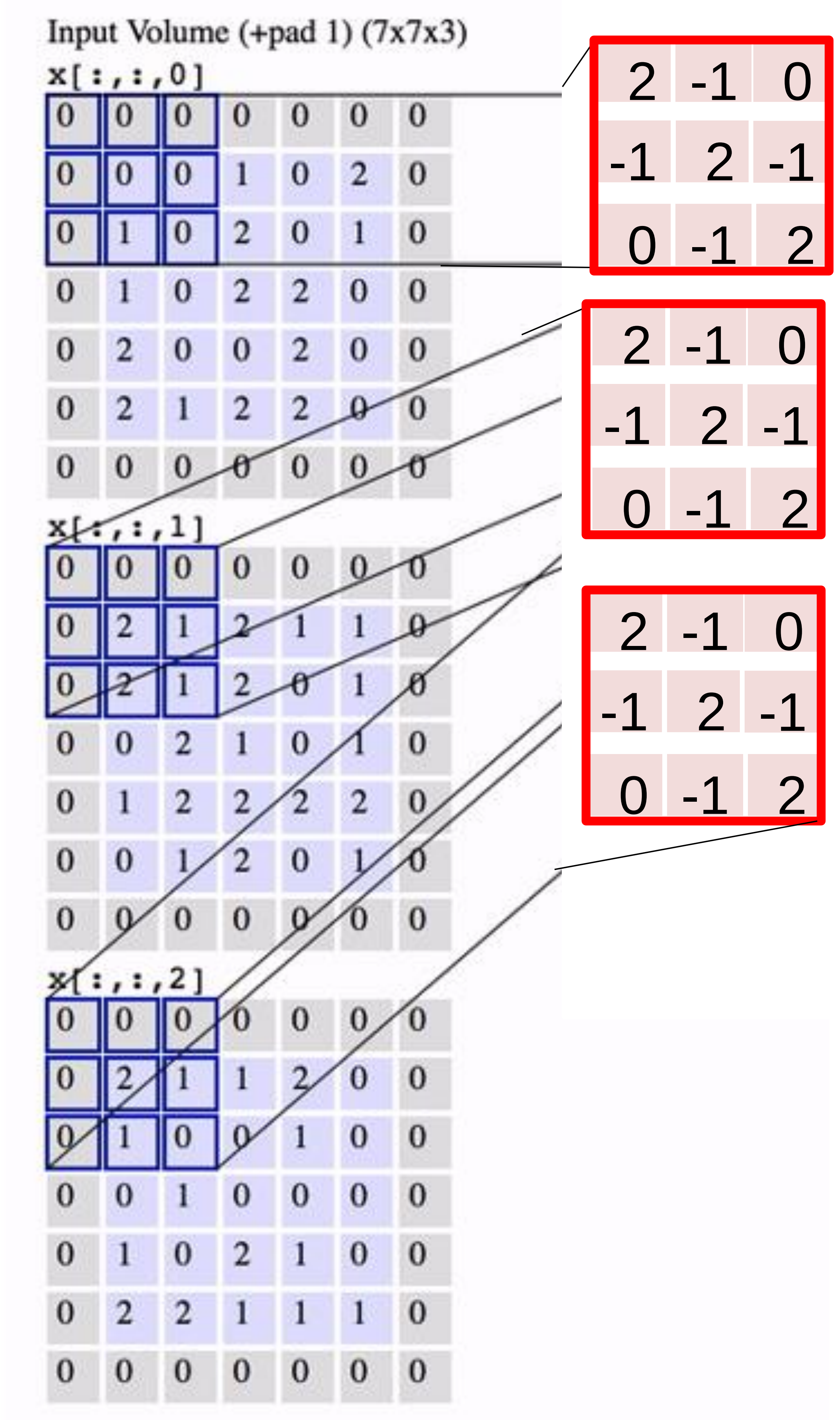


- 3 color channels

- Filter 3x3 (spatial) in each color dimension
  → filter tensor W0 denoted as (3x3x3)

- everal filters: W0, W1, W2, W3, … index k

Convolution: colors and padding



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

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

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

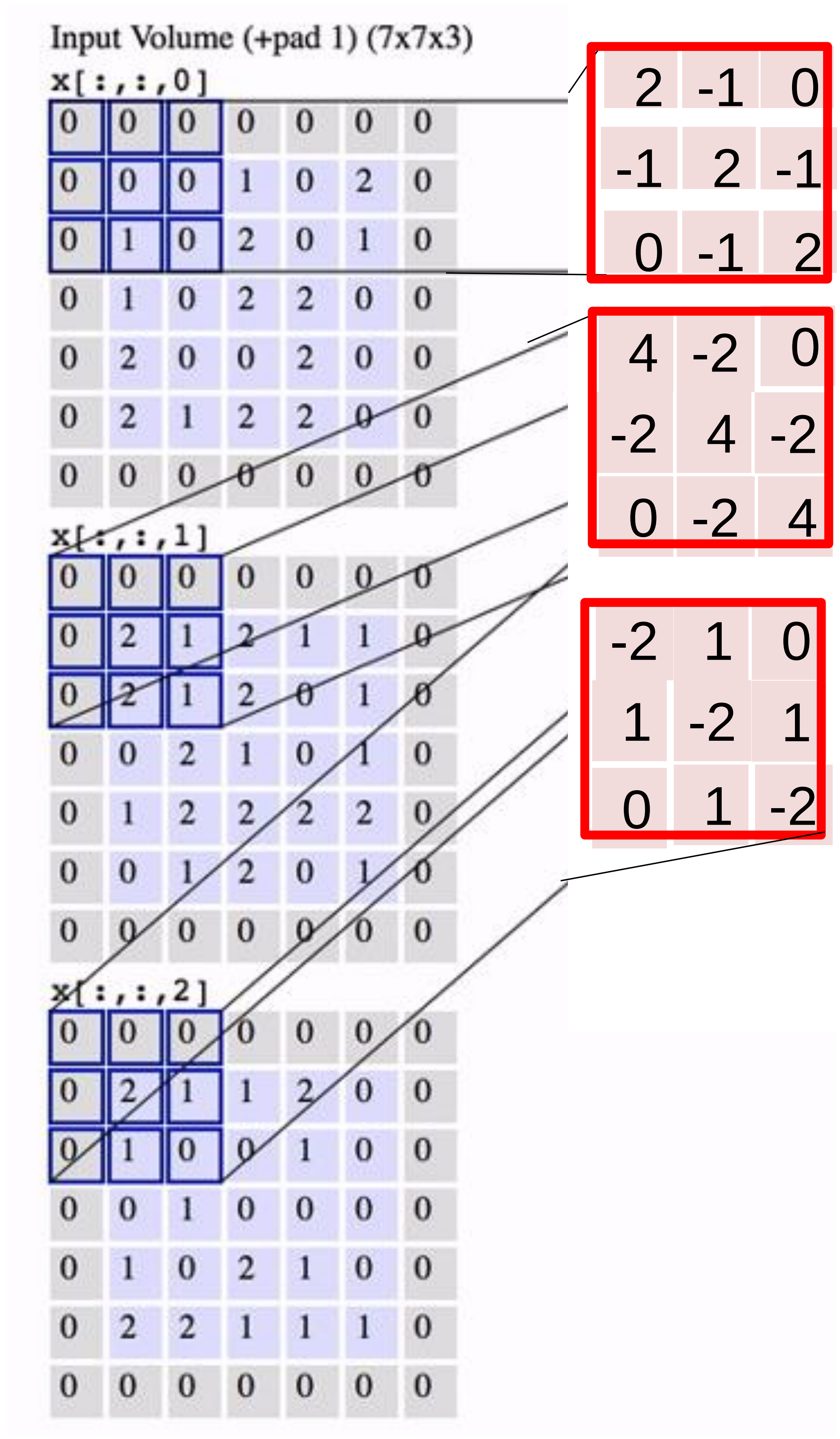
2	-1	0
-1	2	-1
0	-1	2

- 3 color channels

- Filter 3x3 (spatial) in each color dimension
  → filter tensor W0 denoted as (3x3x3)

But why not use the SAME spatial filter for 3 channels?

Convolution: colors and padding



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

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

4	-2	0
-2	4	-2
0	-2	4

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

- 3 color channels

- Filter 3x3 (spatial) in each color dimension
  → filter tensor W0 denoted as (3x3x3)

But why not use the SAME spatial filter for 3 channels?

And may be with different scaling?

Previous slide.

Basic idea:
Let us separate the spatial dimension from the color dimension.

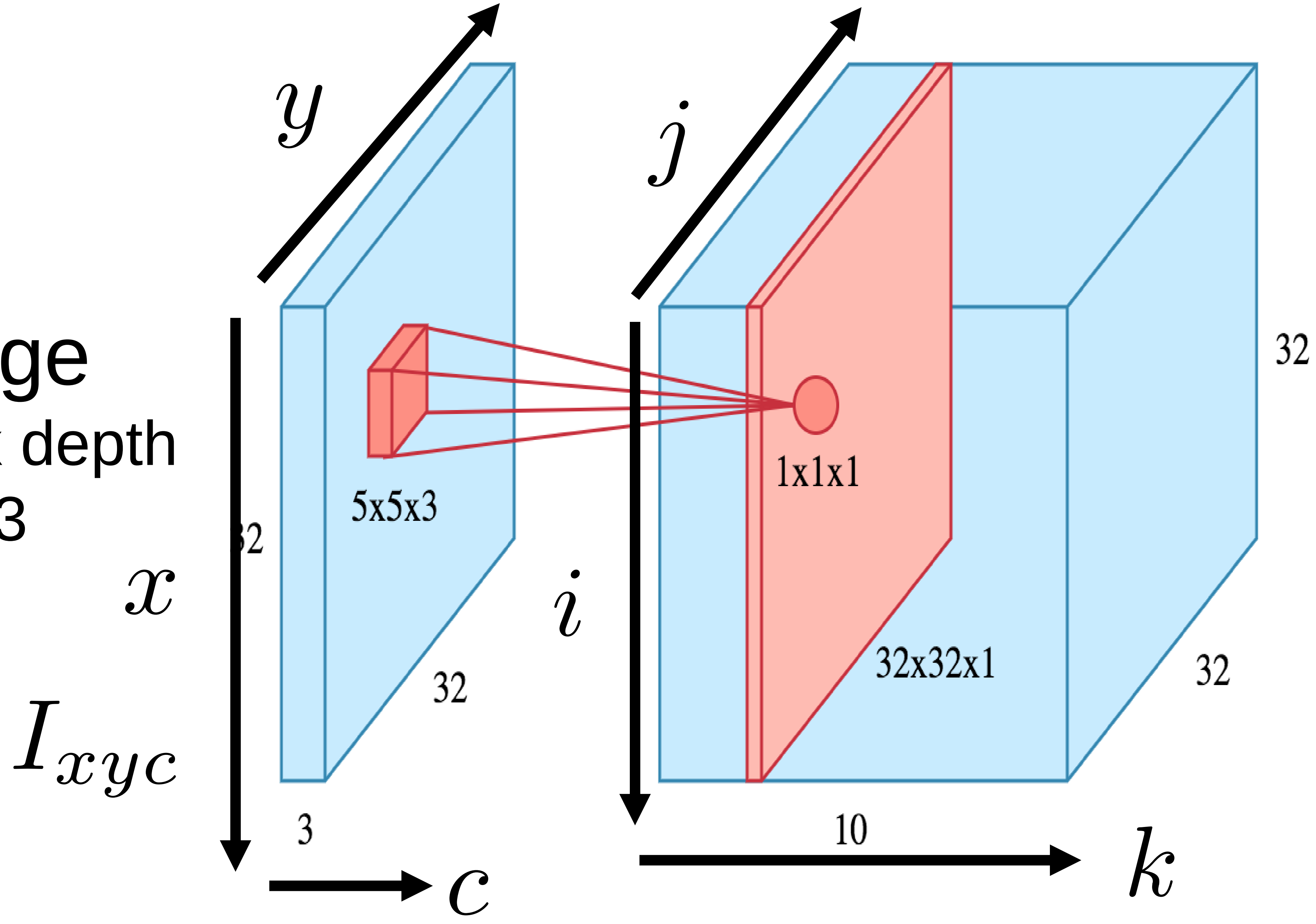
More generally:
Let us separate the spatial dimension from the depth dimension.

Convolution: outer product

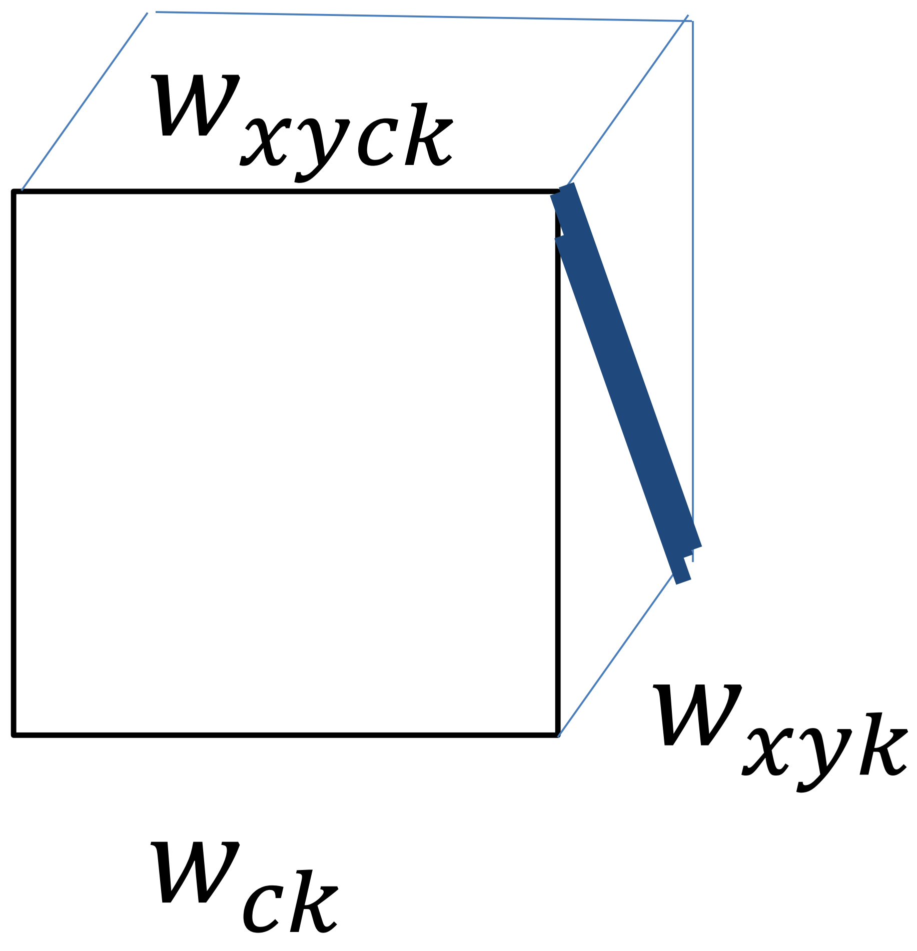
color image

width x height x depth

32 x 32 x 3



Filter k



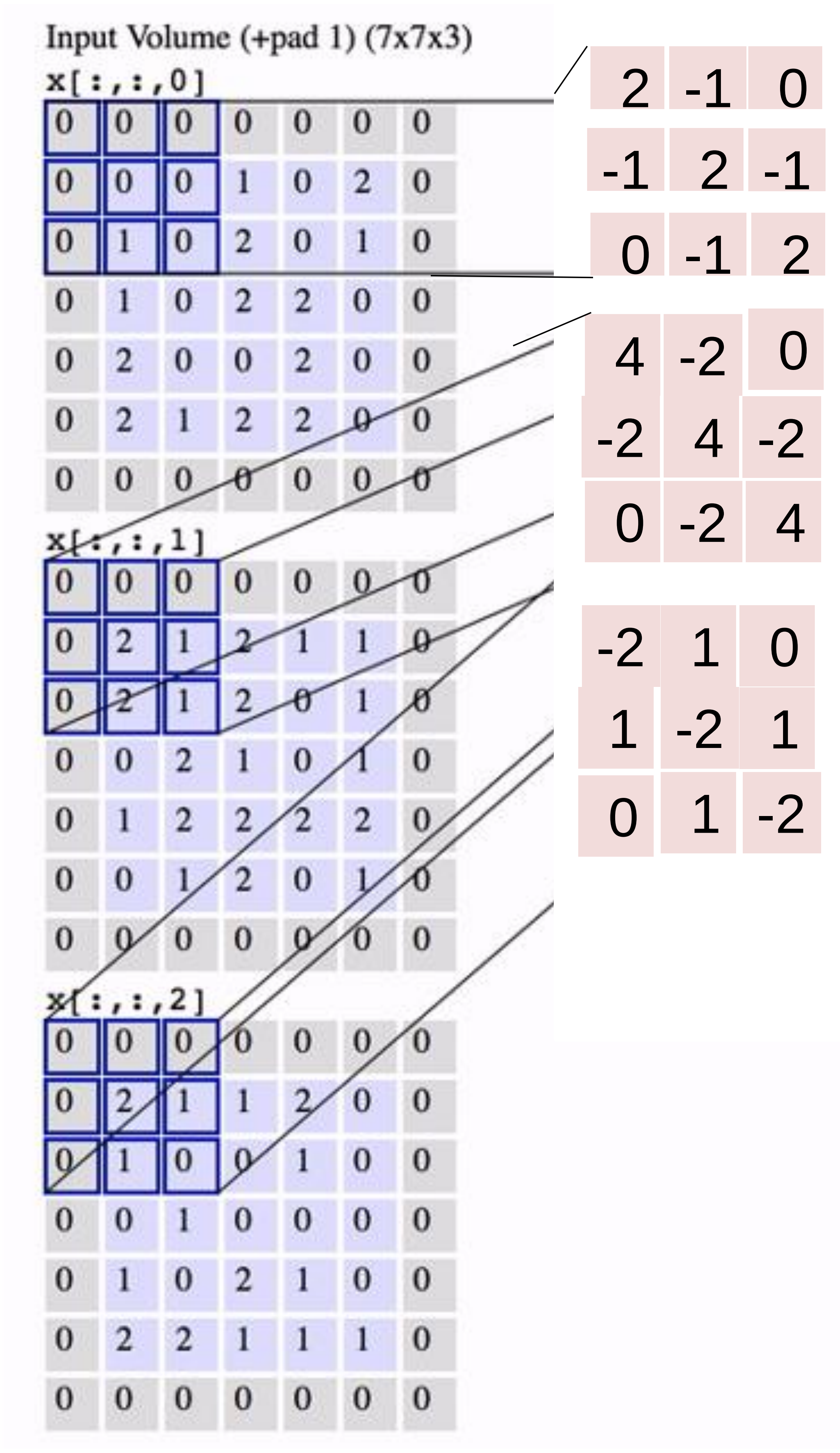
$$w_{xyck} = w_{xyk} w_{ck}$$

Blackboard 3

$$a_{ijk} = b_k + \sum_{x=1}^{5} \sum_{y=1}^{5} \sum_{c=1}^{3} I_{i+x-1,j+y-1,c} w_{xyck}$$

Your notes (calculation).

Convolution: outer product representation



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	0	1	0	2	0
0	1	0	2	0	1	0
0	1	0	2	2	0	0
0	2	0	0	2	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	1	2	1	1	0
0	2	1	2	0	1	0
0	0	2	1	0	1	0
0	1	2	2	2	2	0
0	0	1	2	0	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	1	1	2	0	0
0	1	0	0	1	0	0
0	0	1	0	0	0	0
0	1	0	2	1	0	0
0	2	2	1	1	1	0
0	0	0	0	0	0	0

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

4	-2	0
-2	4	-2
0	-2	4

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

Filter k $$w_{xyck} = w_{xyk} w_{ck}$$

- 3 color channels

- Filter 3x3 (spatial) in each color dimension
  → filter tensor W0 denoted as (3x3x3)

SAME spatial filter for all color channels
= calculate first a (shaded) grey value

→ Efficient numerical evaluation of the convolution

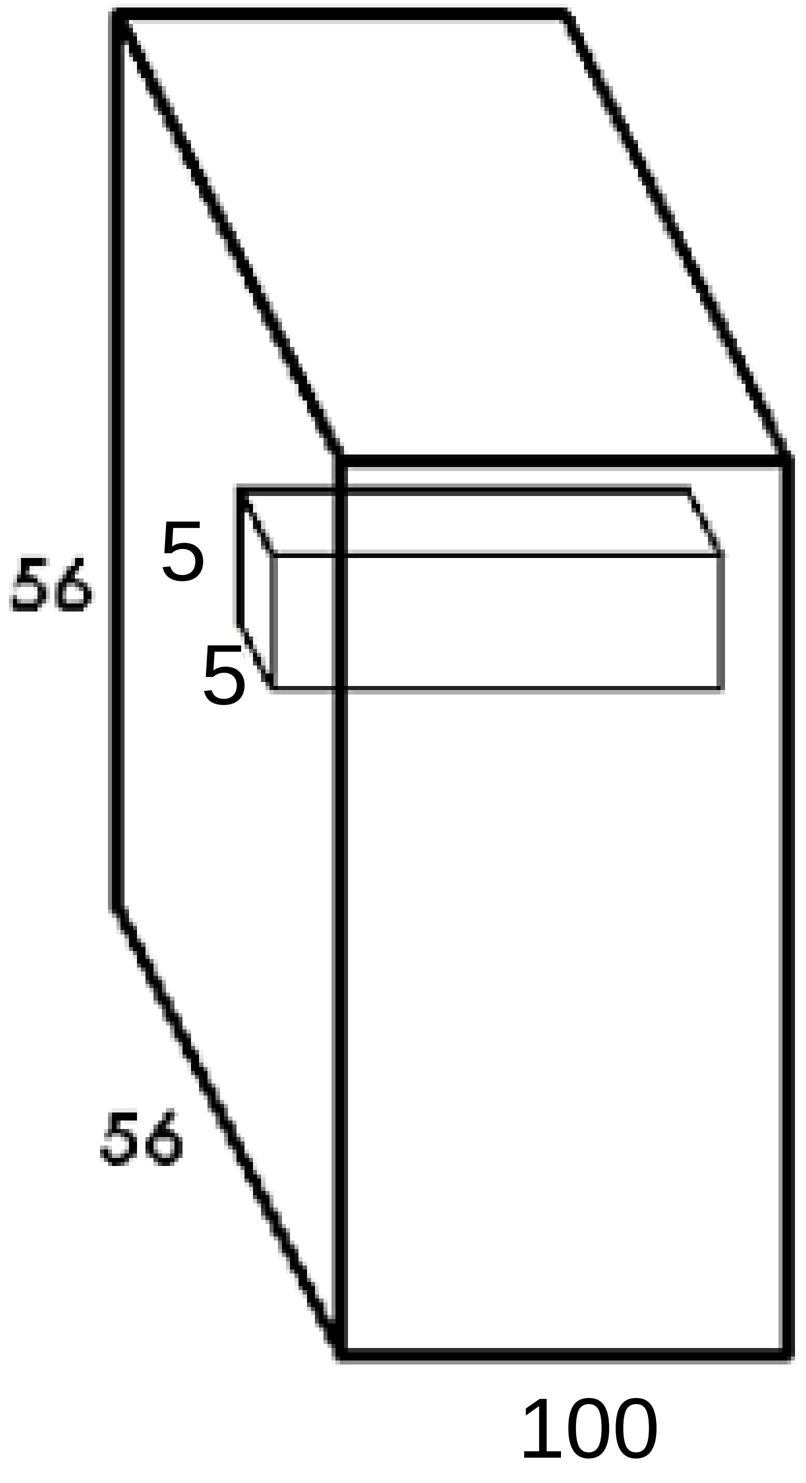
Previous slide.
The calculation shows that the separation of the filter into a spatial component and a depth component can be used to FIRST integrate out the depth: this amounts to a weighted average of all the values in a given xy position.
Then the spatial filtering can be performed later with this ‘average’.

In the specific case of three color channels, the averaging with equal weights would amount to first calculating the grey-scale image (getting rid of the color information). Since it is a weighted average it amounts to a grey with, e.g., a redish or greenisch shading.

Note that instead of the interpretation of the depth filter (a vector of length d) as a weighted averaging filter, we can also say that we apply a tensor of 1x1xd.

Then the full filter of size 5x5xd in outer product representation, can be calculated as the sequence of two filter applications:
First the filter 1x1xd (the depth filter), followed by a filter 5x5x1.
This idea is used later in the inception module.

Parameter reduction with outer product representation



For single filter k

- 100 depths channels: 1 ≤ c ≤ 100

- Filter 5x5 (spatial)
  → filter tensor W denoted as 5x5x100

Parameters:

SAME spatial filter for all color channels

$$w_{xyck} = w_{xyk} w_{ck}$$

Parameters: (5x5)+100

→ Enormous reduction of number of parameters (per filter)

Previous slide.

Reduction of parameters by more than a factor of 10!

The outer-product representation is used in many fields of science as a means to reduce the number of parameters in a large filter tensor. It is not restricted to ANN.

Artificial Neural Networks

Convolutional Neural Networks

Johanni Brea & W. Gerstner

EPFL, Lausanne, Switzerland

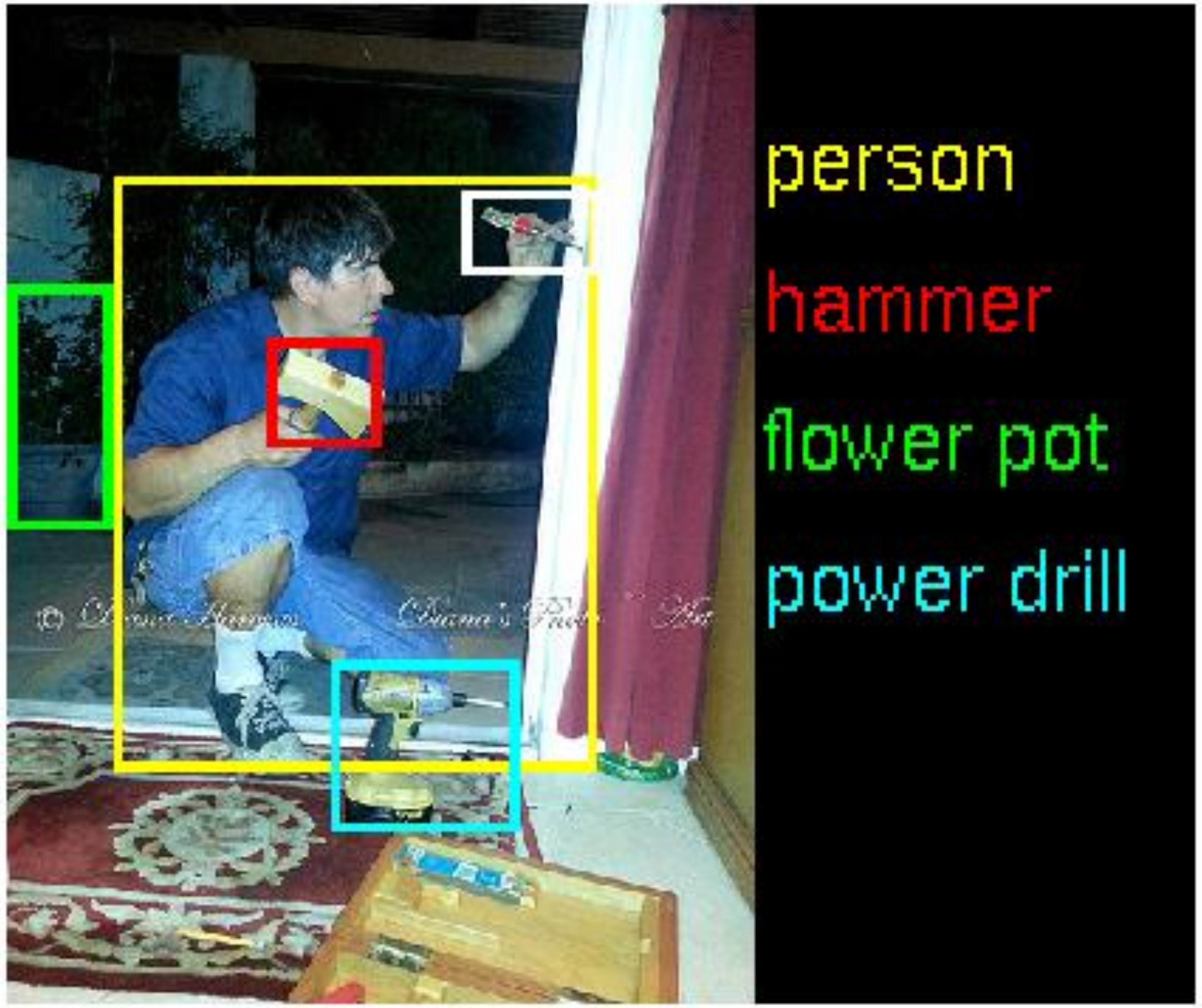
Part 8: Modern Convolutional Networks and Image Tasks

1. No Free Lunch Theorem
2. Inductive bias in machine learning
3. Convolution filters as inductive bias for images
4. MaxPooling as inductive bias for images
5. The gradient of a convolutional layer
6. Automatic differentiation: BackProp revisited
7. Reducing the number of parameters: outer-product
8. **Modern ConvNets and Image Recognition Tasks**

Previous slide.

We now look at recent developments around Convolutional Networks.

Recent Imagenet challenges: object detection task



http://www.image-net.org/
https://www.kaggle.com/c/imagenet-object-detection-challenge

Previous slide.

The task here is to find several objects in the image, and tell where these are.

Imagenet challenge 2012

example images of australian terriers

- 1.2 million images training set
- 1000 categories
- 50 thousand validation set
- 100 thousand test set

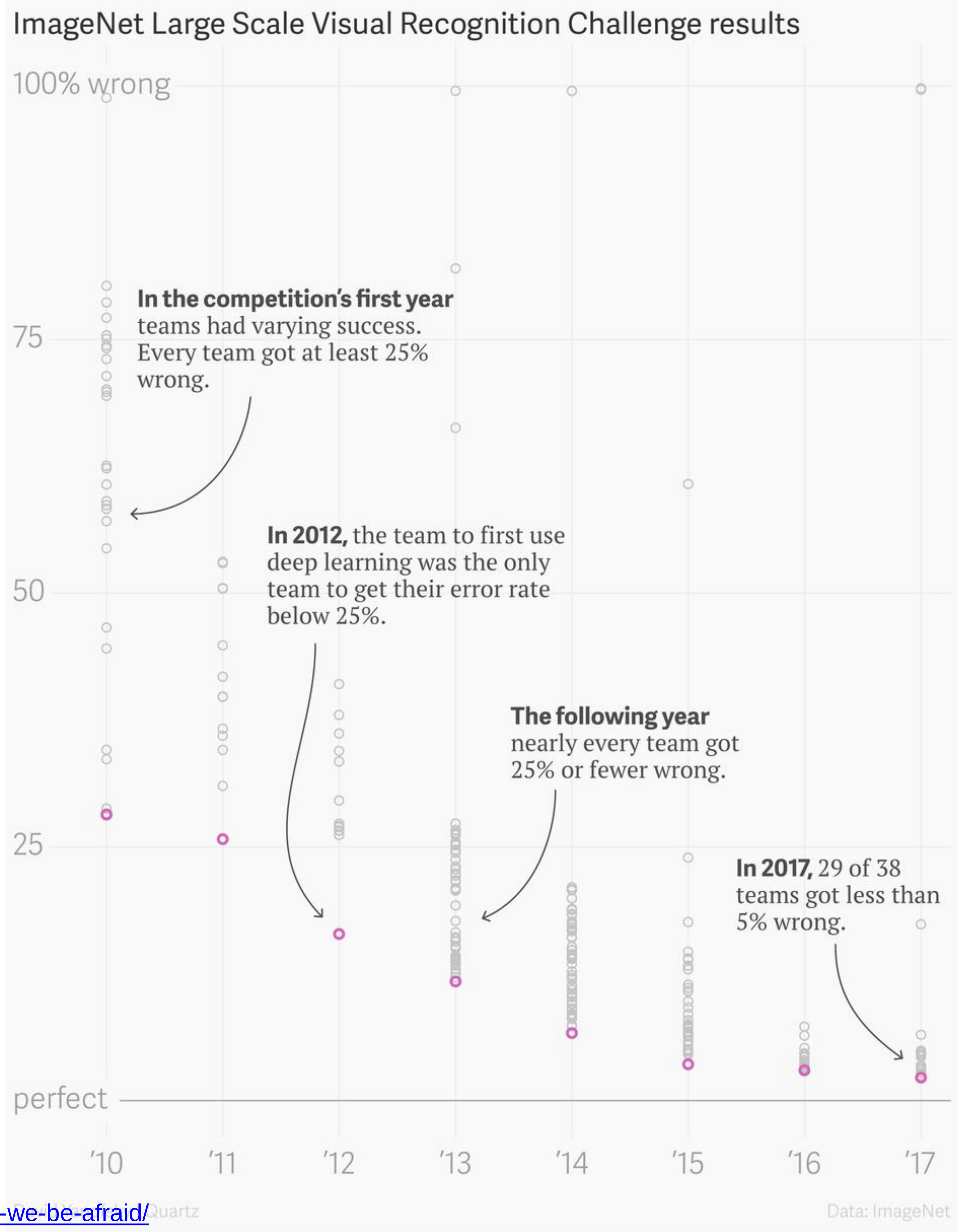
http://www.image-net.org/

Previous slide.

In the classic ImageNet challenges, the task was to find a single object. Note that the data base had a sizable fraction of different dog breeds.

The imagenet competitions have extensively been used to demonstrate progress in deep learning. The competition in 2012 contained 1.2 Million hand-labeled images containing objects in 1000 classes; these numbers have increased in the meantime by at least an order of magnitude.

Imagenet challenge progess



ImageNet Large Scale Visual Recognition Challenge results

In the competition's first year teams had varying success. Every team got at least 25% wrong.

In 2012, the team to first use deep learning was the only team to get their error rate below 25%.

The following year nearly every team got 25% or fewer wrong.

In 2017, 29 of 38 teams got less than 5% wrong.

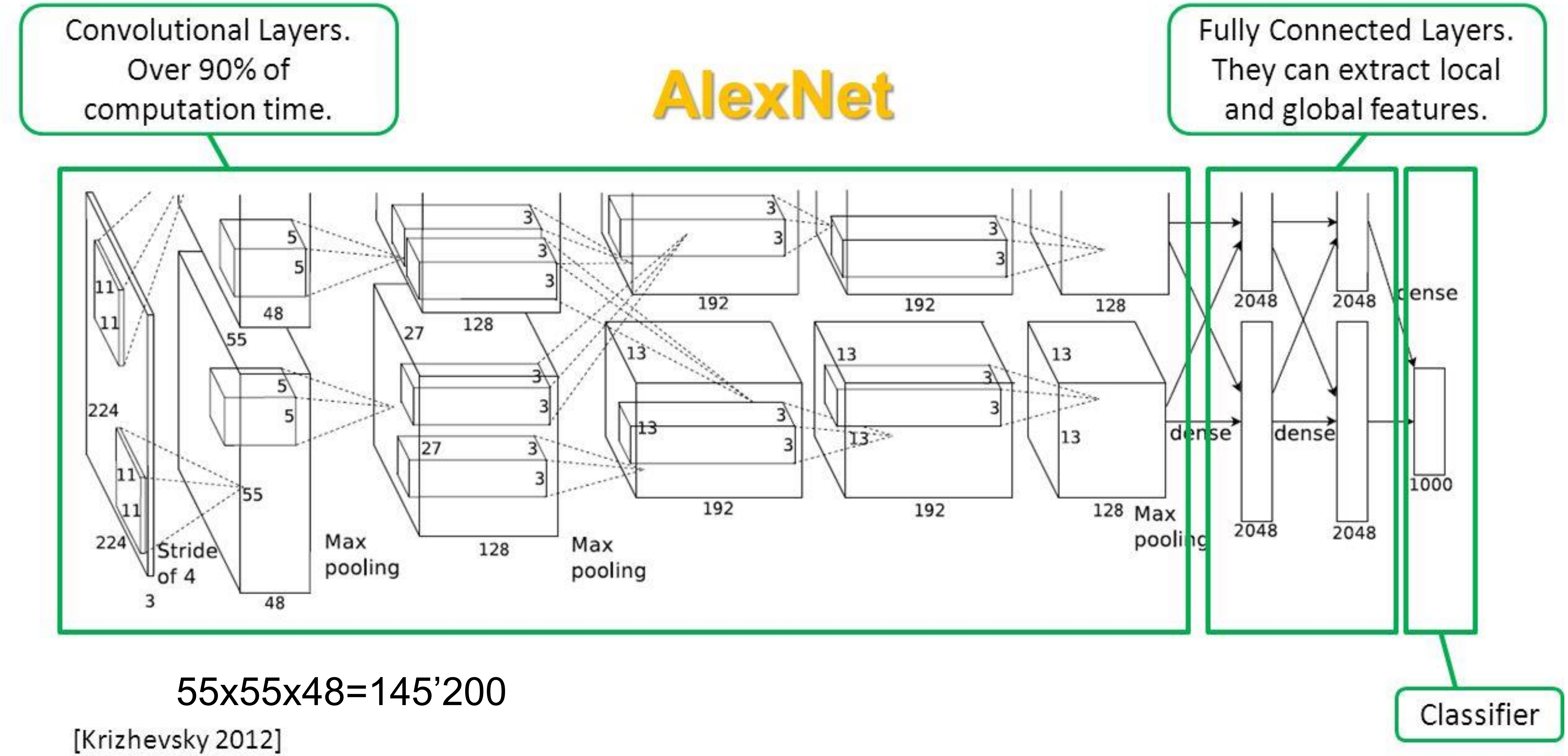
Data: ImageNet

https://qz.com/1046350/the-quartz-guide-to-artificial-intelligence-what-is-it-why-is-it-important-and-should-we-be-afraid/

Previous slide.

Right: Within only a few years the error rate decreased from 25 to 5 percent thanks to advances in the design and training of convolutional networks.

AlexNet: winner of Imagenet challenge 2012



55x55x48=145’200

[Krizhevsky 2012]

http://slideplayer.com/slide/10259573/34/images/5/AlexNet+Convolutional+Layers.+Over+90%25+of+computation+time..jpg

Previous slide.

The best method of the 2012 competition, now called AlexNet, still had a pretty conventional architecture with elements that you have seen up to now in this lecture.

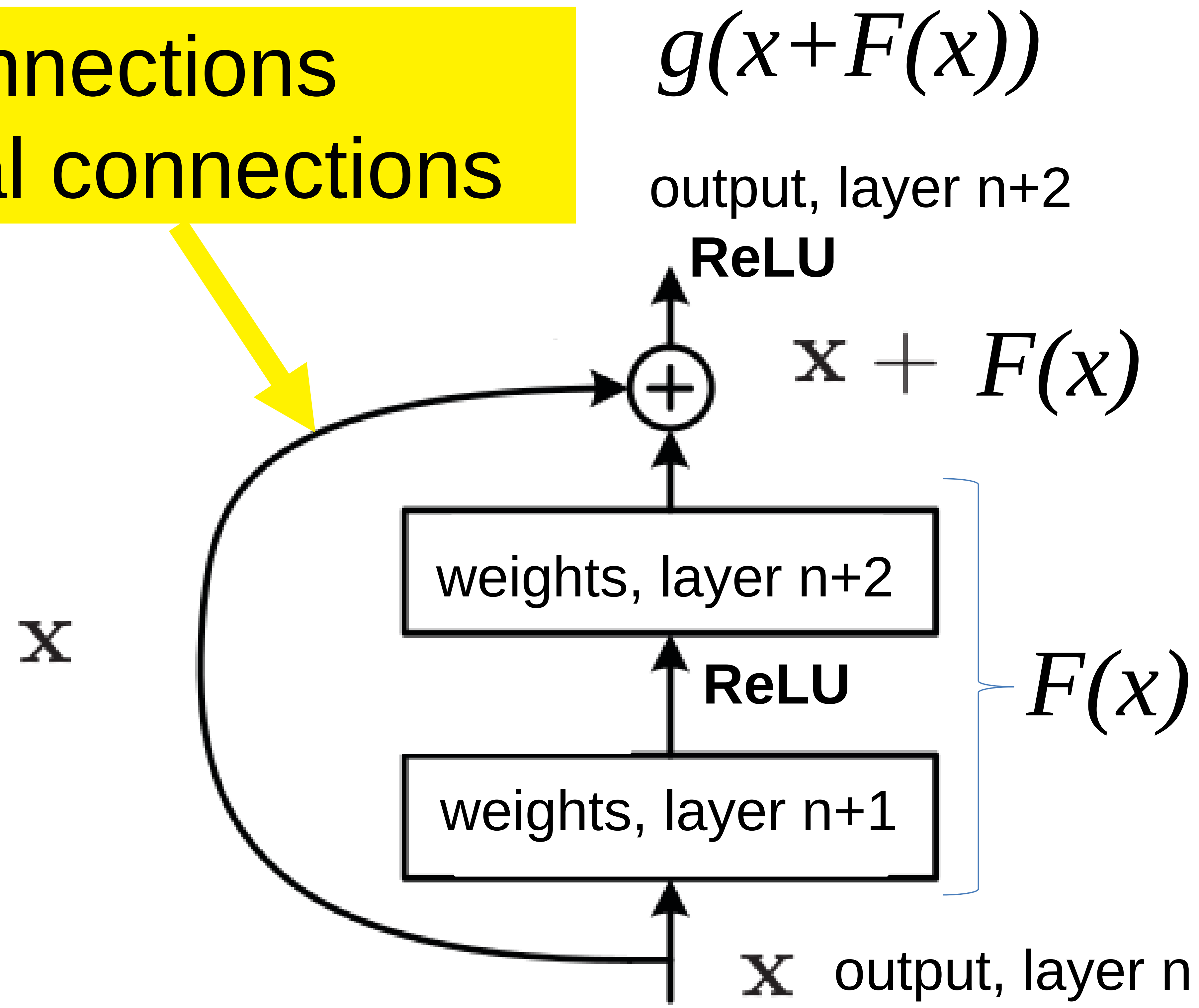
In the first layer, there are about 250’000 neurons. The two processing streams at top and bottom correspond to the implementation on two GPUs.

We will now start to look at newer features that helped to further improve convolutional networks.

ResNet

Better training in very deep (>100 layers) networks.

Skip connections
Residual connections



$g(x+F(x))$

output, layer n+2

ReLU

$\mathbf{x} + F(x)$

weights, layer n+2

ReLU

weights, layer n+1

$F(x)$

$\mathbf{x}$

$\mathbf{x}$ output, layer n

How many layers?
→ Let the data decide!

Blackboard 4

https://arxiv.org/pdf/1512.03385.pdf

Previous slide. In theory, the training loss should decrease with deeper and deeper networks, simply because they have more parameters. In practice, however, it was observed that successfully training very deep convolutional networks is difficult.
A possibly reason are vanishing gradients, but it is not clear whether this is the true reason.

The idea of skip connections is to allow the network to “dynamically choose the number of layers” (see blackboard). The layers within the skip-connections learn the residual F(x), the part that is not yet learned by the network up to layer n (which has the output x). Thus, the function F(x) can be used to adapt to special cases, exceptions, fine-tuning etc – without affecting the main network function (which is just copied to layer n+2 via the skip connections).

When calculating the gradients, the skip connection contributes no derivative g’, so that multiplication of small gradients is avoided along the skip connections: the skip connection = identity/copy has always a gradient of one. Therefore the combined vanishing gradient problem/linearity problem/bias problem is avoided: the skip connection is just a copy. But a copy is a trivial linear transformation; the parallel F(x) avoids the linearity problem because for some data the nonlinearity F might come into play. With small weights, the F makes a small contribution: like in a Taylor expansion, the dominant term is the copy through the skip connection.
Note that the skip connection could cover many more than just 2 layers!
Note also that the following nonlinear transformation g(x+F(x)) has the usual vanishing gradient/bias problem.

Summary: ResNet and Skip Connections

4 different interpretations of ResNet:

- Reduces the vanishing gradient problem
  → training possible with > 100 layers

- Forward pass corrects is copy + corrections
  → can only get better

- Network implicitly chooses the necessary number of layers
  → If layers are not useful, the function F is zero (and only copy survives)

- ResNets are similar to recurrent networks
  → recurrent networks correspond to same F in several ResNet layers

Previous slide.

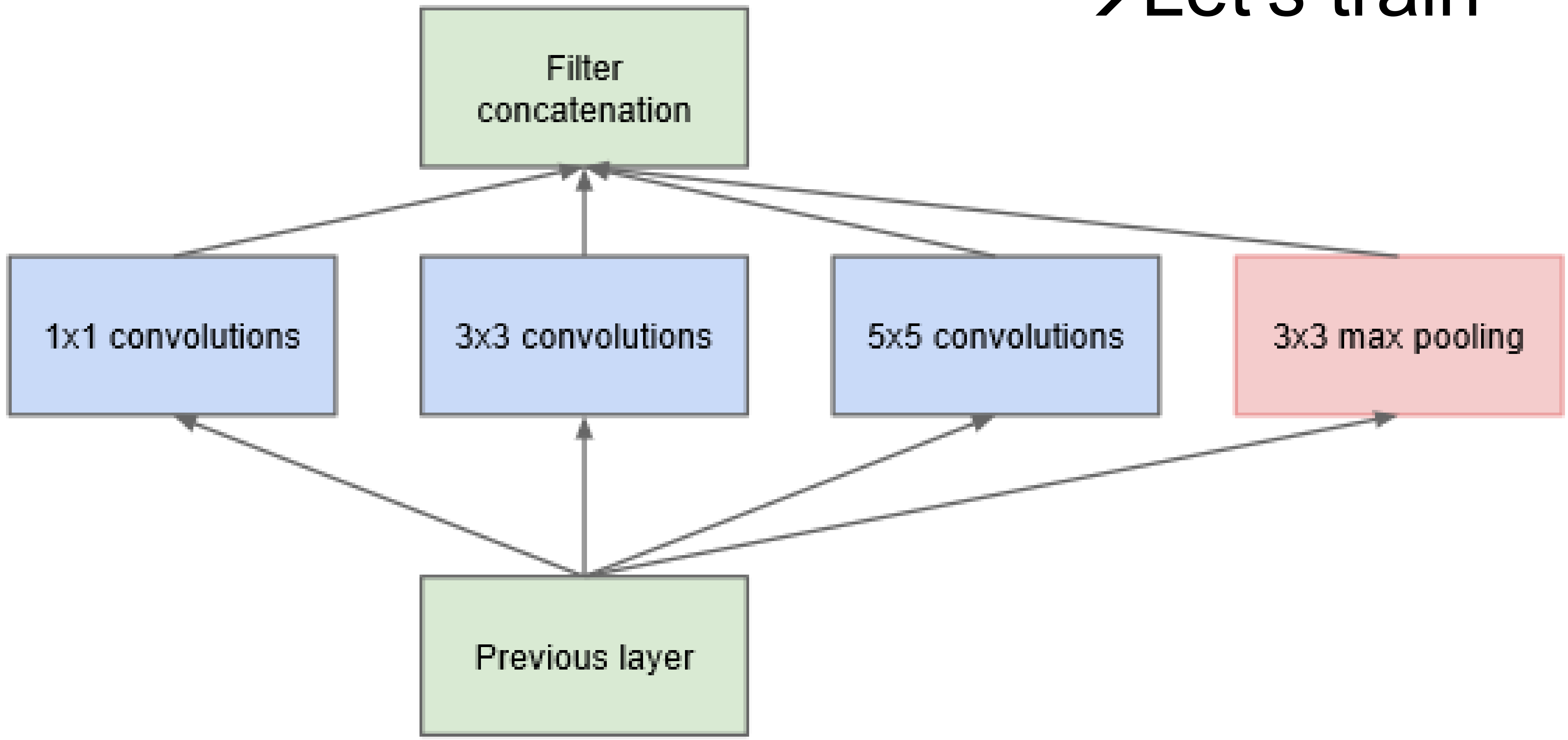
The first 3 points have been shown here.

Point number 4 will come in the lecture on Recurrent Networks.

Inception module (naïve version)

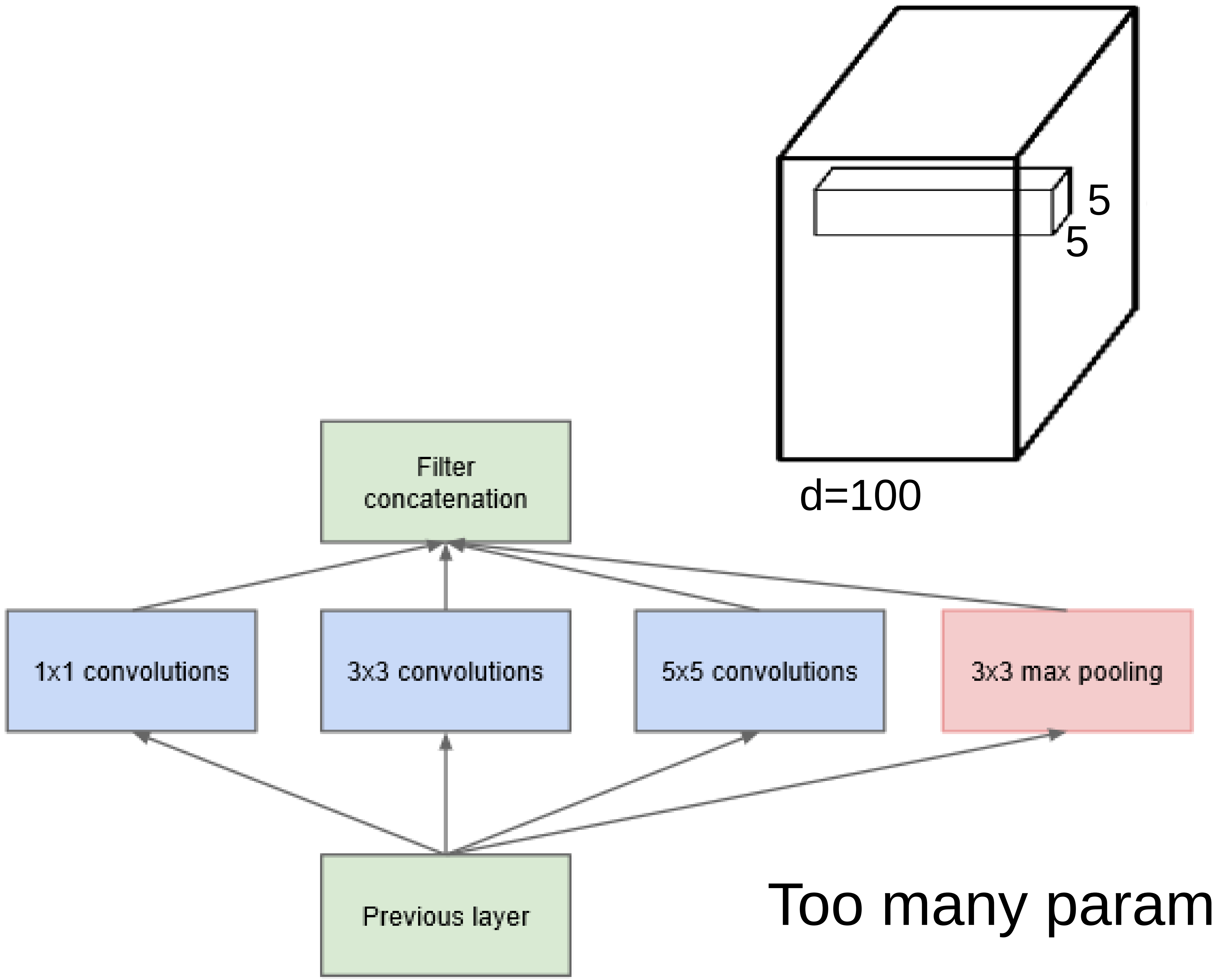
Are 3x3 convolutions better or 5x5 convolutions?
→Let the data decide
→Let's train

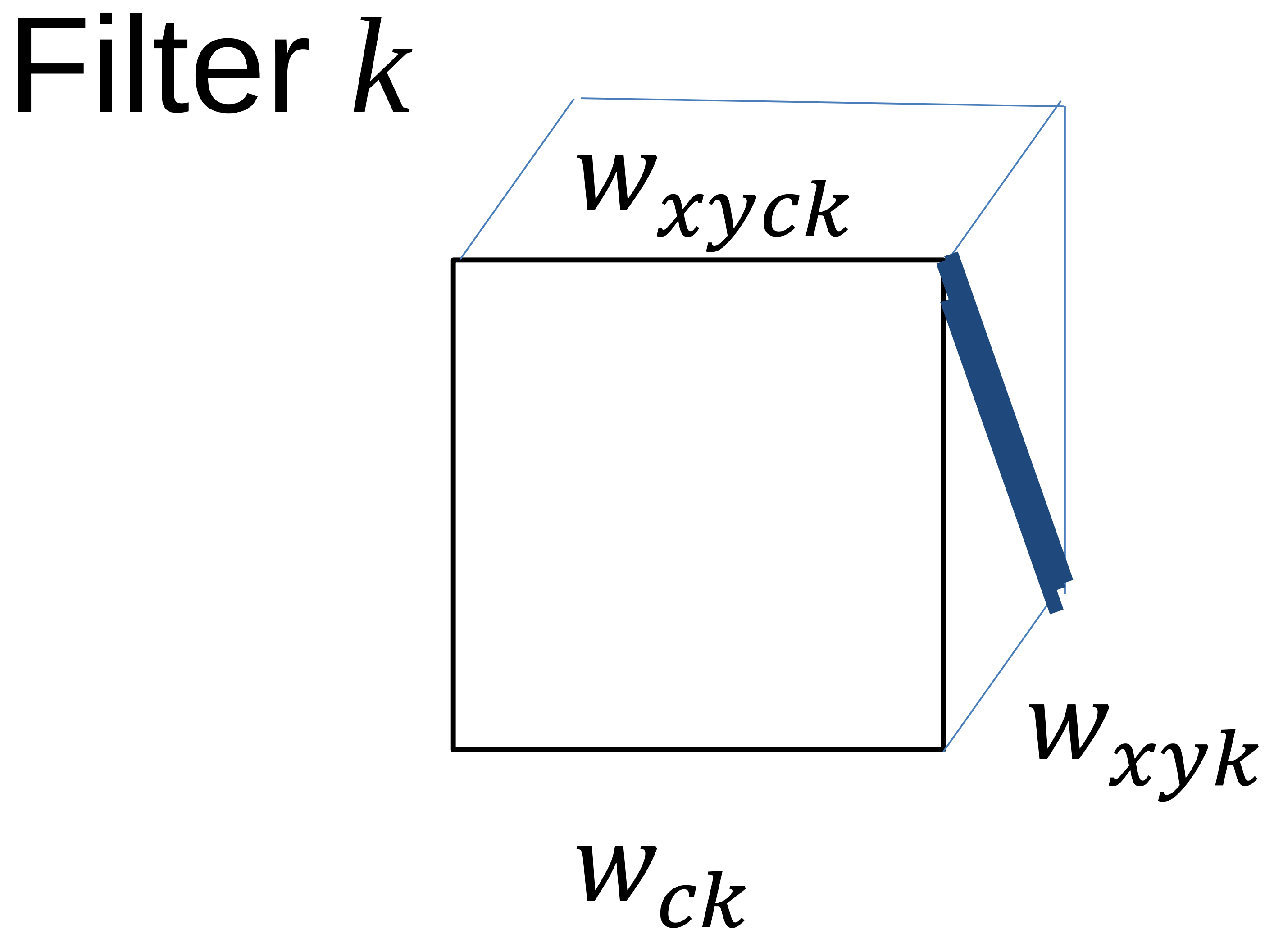
Do we need max-pooling? Let the data decide
→Let's train



https://arxiv.org/pdf/1409.4842.pdf

Inception module (naïve version)



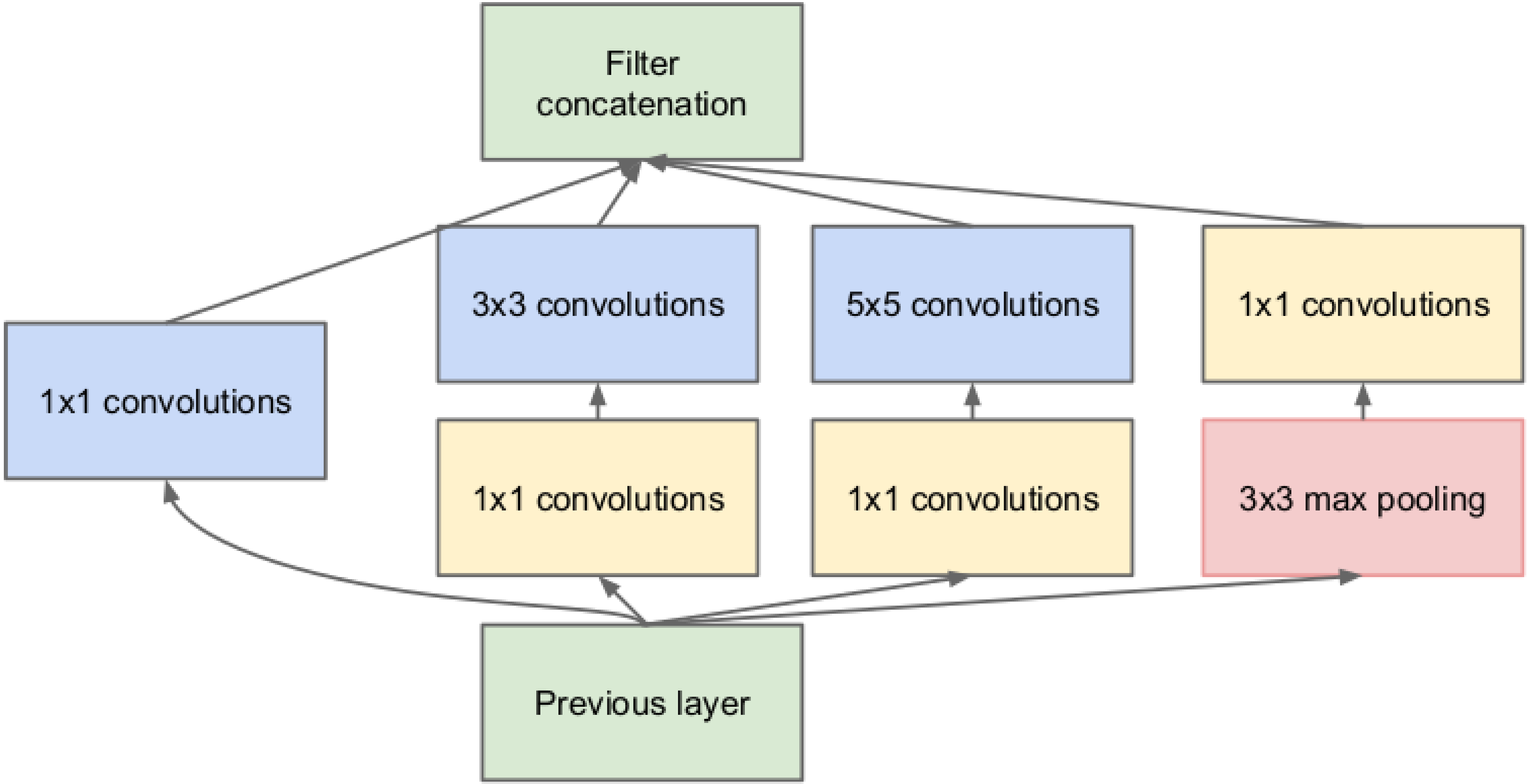


$$w_{xyck} = w_{xyk} w_{ck}$$

Too many parameters! Let’s use outer product

https://arxiv.org/pdf/1409.4842.pdf

Inception module



https://arxiv.org/pdf/1409.4842.pdf

Previous slide.

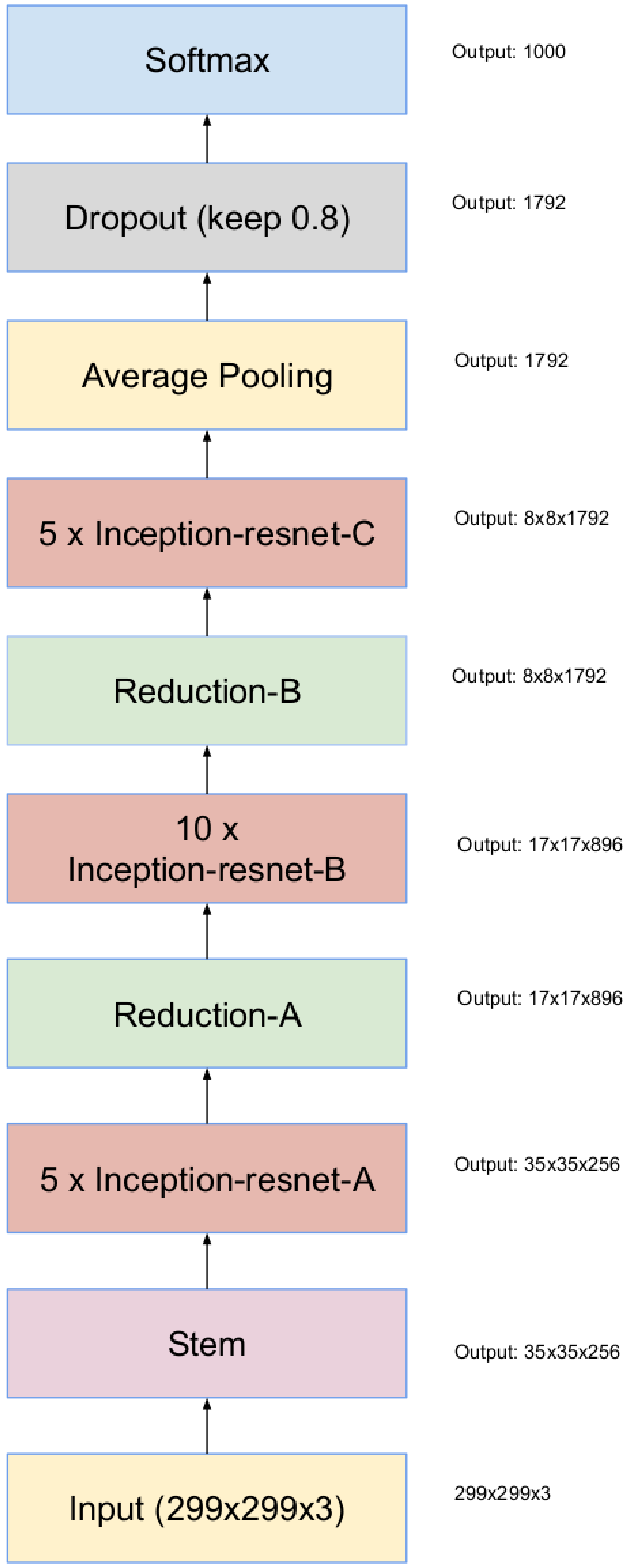
For AlexNet the designers decided where to place convolutional layers with filters of certain sizes and where to place pooling layers. But how should we know what the best architecture is? The inception module leaves this choice to the network itself:

Each layer contains convolutions with different filter sizes and pooling operations in parallel. Depending on the relative weight of each block, the layer can be used as max-pooling layer, or as convolution layer, or as combination of both.

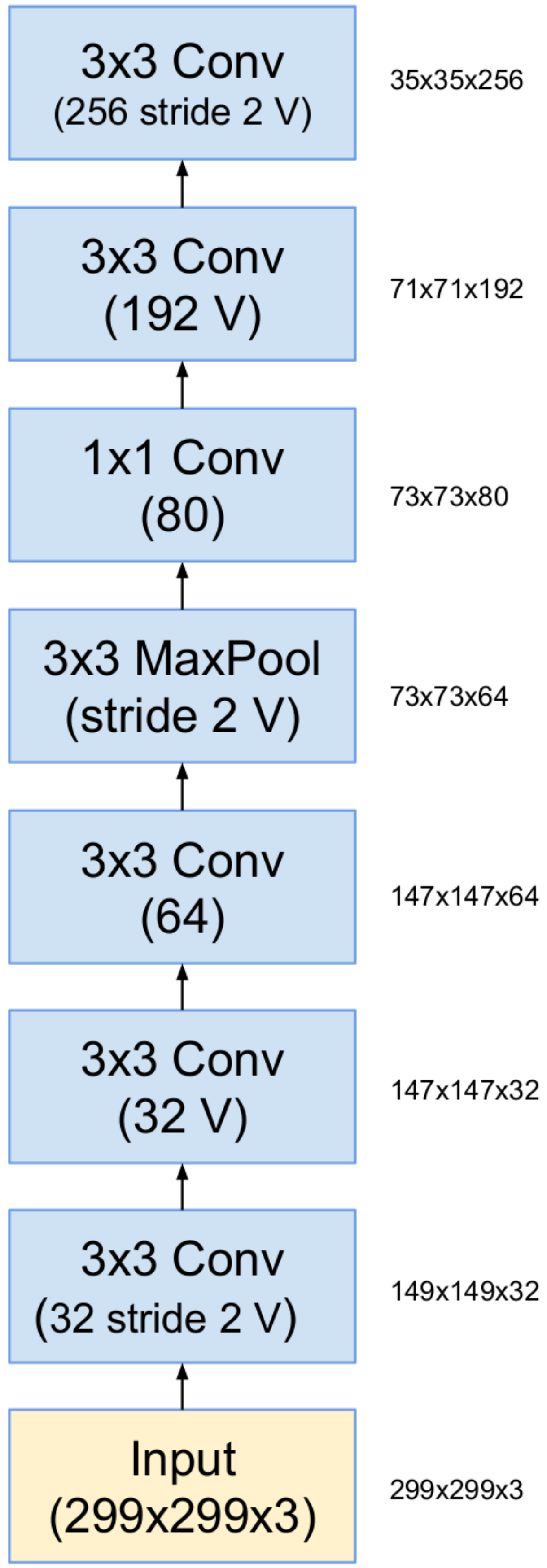
The 1x1xd filters learn to select the relevant features or mixture of features from the concatenation of the outputs. The 1x1xd filters also help to reduce the computational load and the number of parameters (outer-product representation).

Note that the convolution and max-pooling layers have to use the same stride, i.e. if stride 1 is used for the 3x3 and 5x5 filters, than also stride 1 should be used for the max-pooling layer.

Inception-ResNet-v2



Full model



Stem

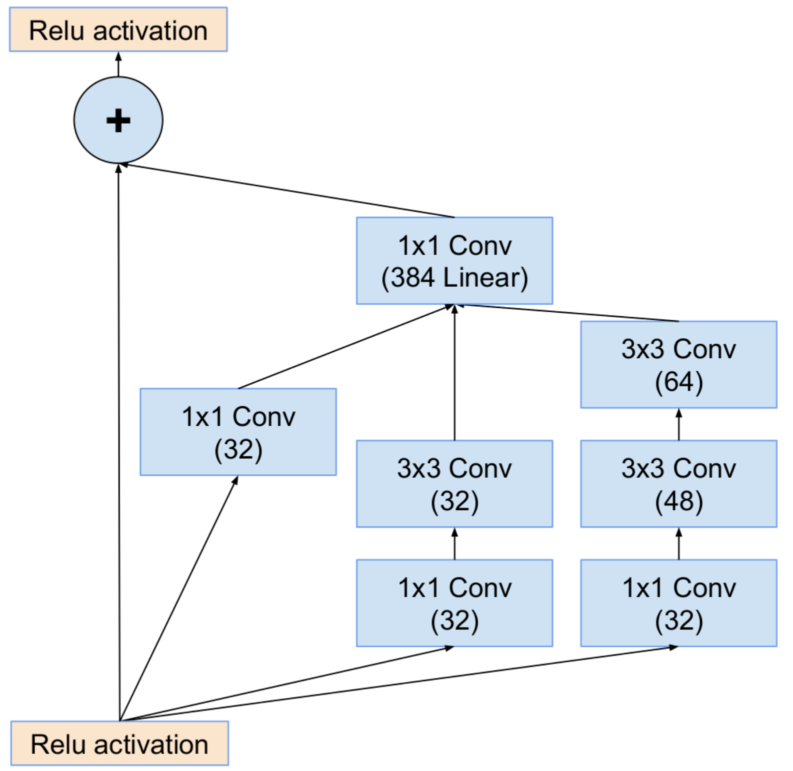


Figure 16. The schema for 35 × 35 grid (Inception-ResNet-A) module of the Inception-ResNet-v2 network.

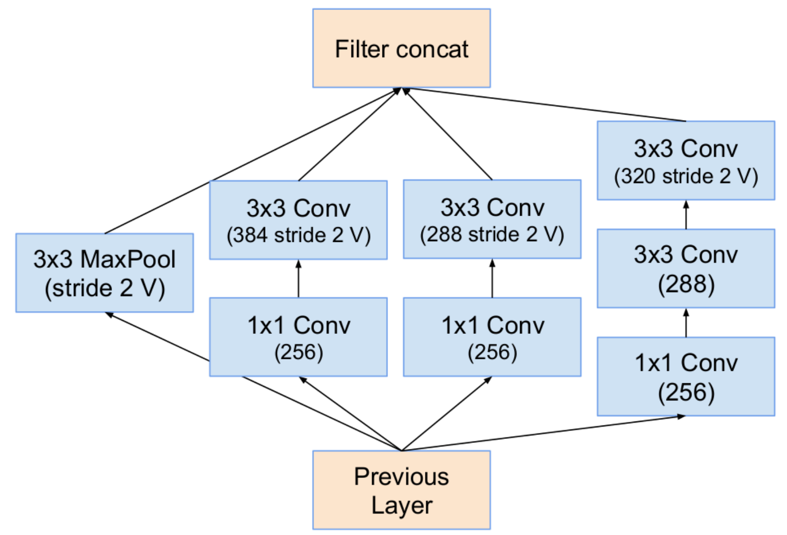


Figure 18. The schema for 17 × 17 to 8 × 8 grid-reduction module. Reduction-B module used by the wider Inception-ResNet-v1 network in Figure 15.

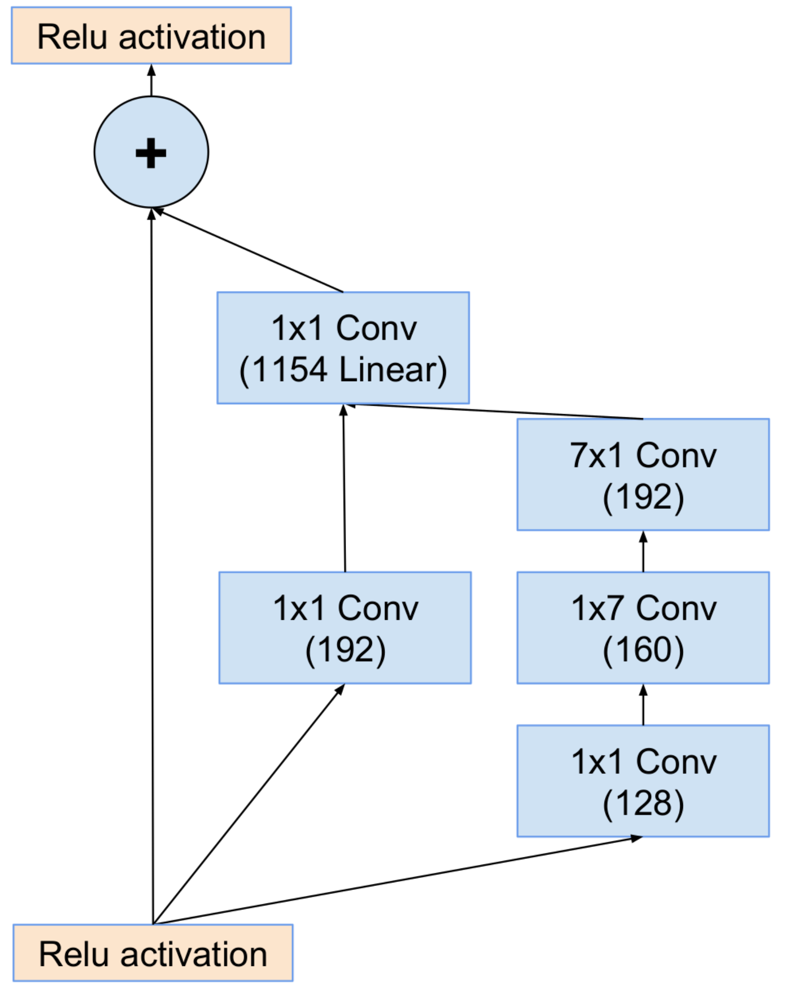


Figure 17. The schema for 17 × 17 grid (Inception-ResNet-B)

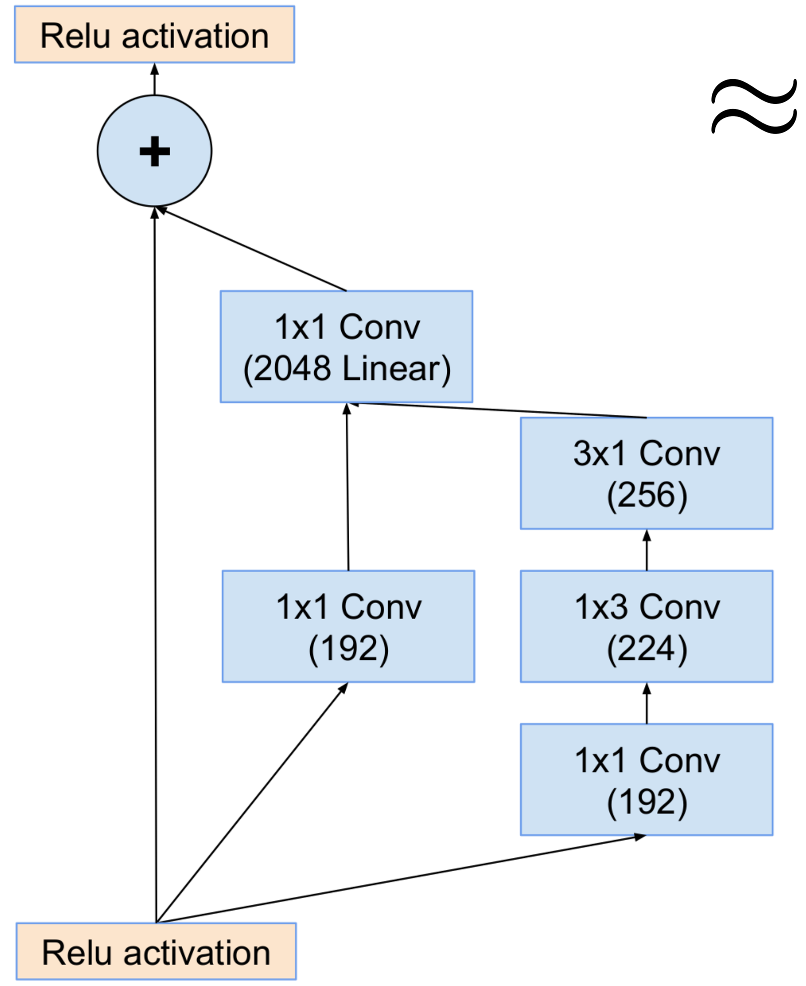
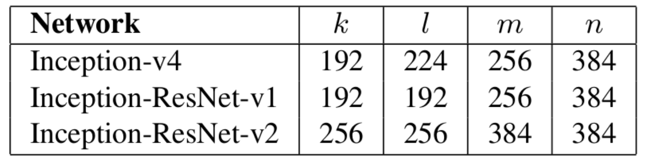


Figure 19. The schema for 8×8 grid (Inception-ResNet-C) module of the Inception-ResNet-v2 network.



Network	k	l	m	n
Inception-v4	192	224	256	384
Inception-ResNet-v1	192	192	256	384
Inception-ResNet-v2	256	256	384	384

≈ 60 million parameters

Special Layers

https://arxiv.org/pdf/1602.07261.pdf

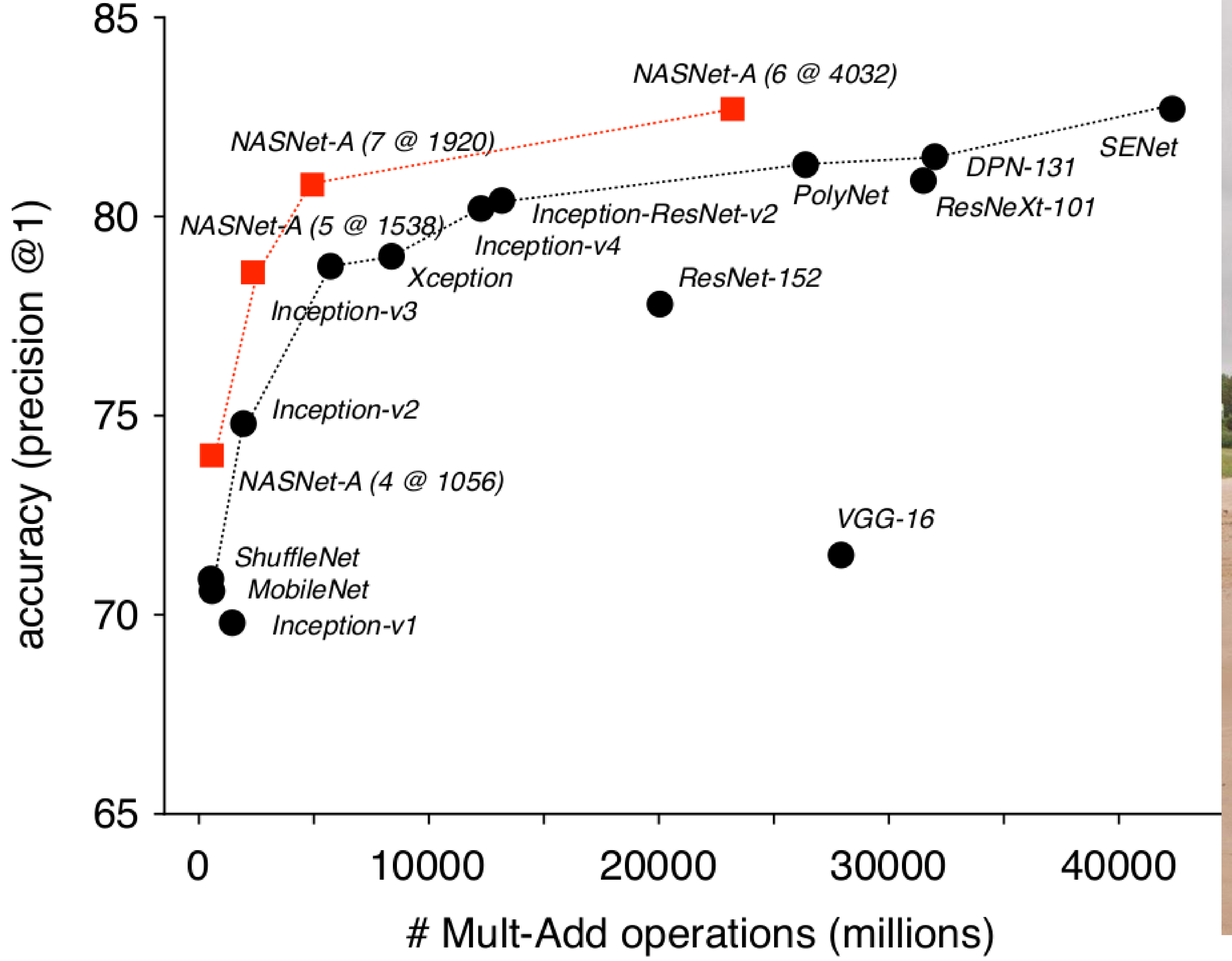
Previous slide.

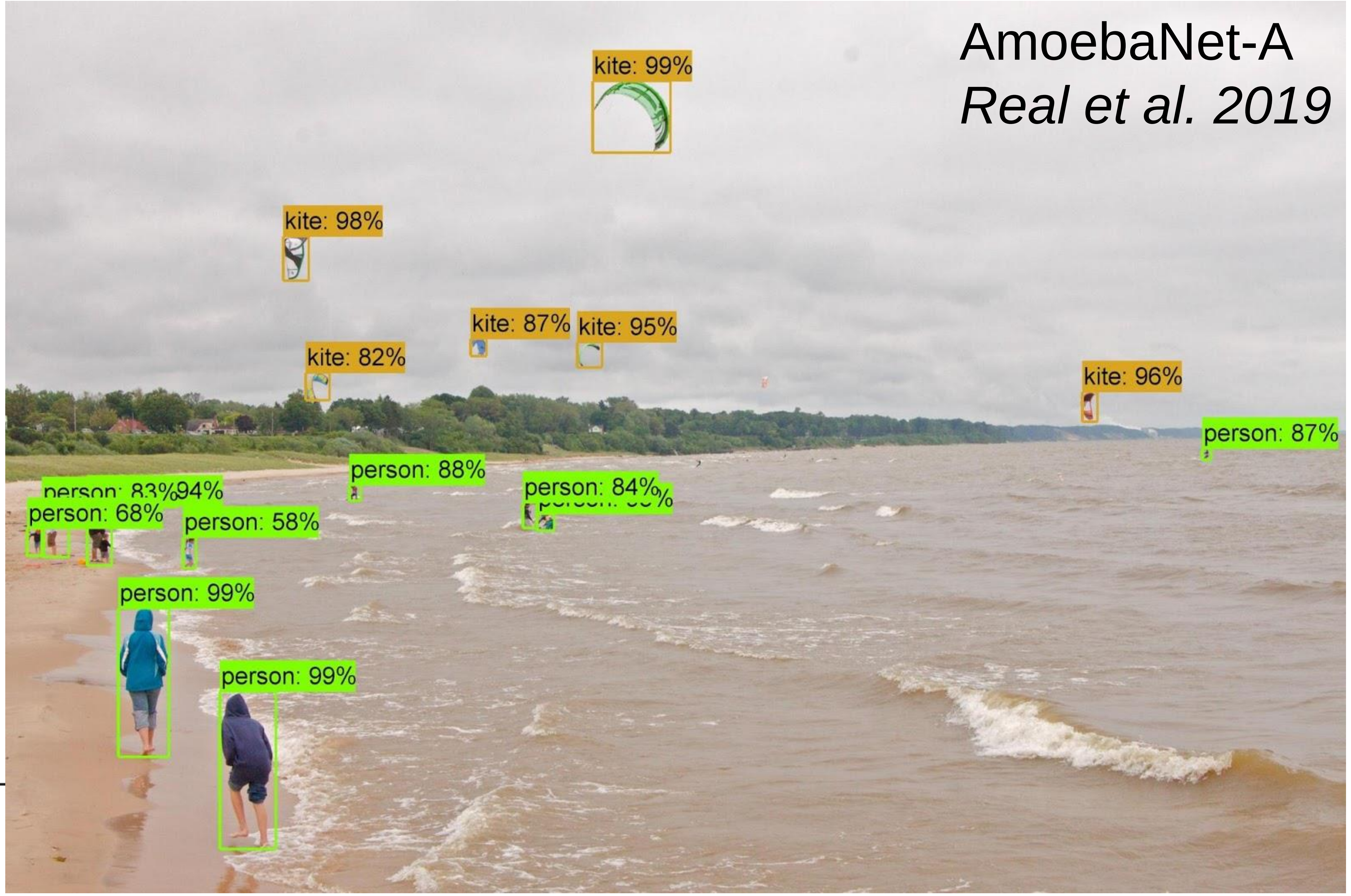
Inception modules have been the basis for GoogLeNet.

A full network may look quite scary and use these inception modules several times.

Transfer Learning for Image Recognition

- Find network architecture with reinforcement learning or evolutionary programming on a small dataset
- Use the same architectural elements on Imagenet





AmoebaNet-A
Real et al. 2019

https://arxiv.org/pdf/1707.07012.pdf https://arxiv.org/abs/1802.01548

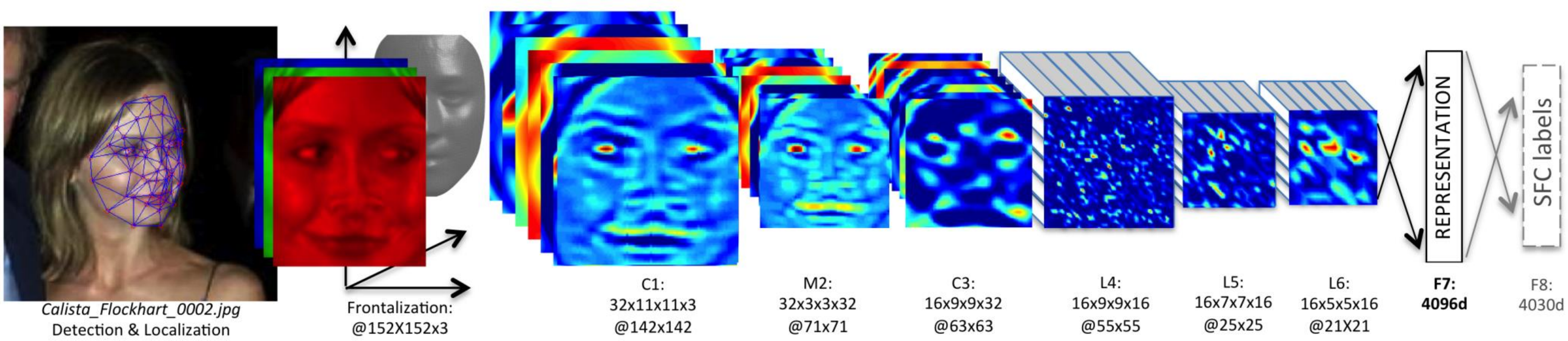
Previous slide.

With skip connections and Inception modules, allowed significant improvements over AlexNet. Yet, the search for the optimal network architecture, i.e. the best inductive bias, continues. Can we automatize this process? Yes, with sufficient computing power one may search for architectures, e.g. with reinforcement learning methods or evolutionary programming. State-of-the-art architectures, like NASNet and AmoebaNet were found like this.

On the left we see that NASNet reaches higher performance on imagenet than hand-crafted variants. In particular the NASNet variants reach consistently higher performance than alternatives with the same number of basic operations. VGG-16 is a famous hand-crafted network with a simple stucture of 13 convolutional layers with 3x3 filters and 3 fully connected layers.

On the right we see an impressive example of what can nowadays be achieved with these networks.

DeepFace and Transfer Learning



Closing the Gap to Human-Level Performance in Face Verification (2014)

- 4 million user - labeled faces on FaceBook images (for free!)
- 4000 individuals
- Retrain fully-connected layers at the top on Labeled Faces in the Wild (LFW) dataset reaching (human level) accuracy of 97.35%

Previous slide.

Even though good inductive biases help us reduce the amount of training data needed, it is still crucial to have a lot of labeled data. I find the example of Facebooks face recognition network interesting, because some years ago, millions of users worked for Facebook by labeling their images, without ever getting paid for it and probably without knowing that they helped Facebook to create a state-of-the-art face recognition tool.

Also interesting about this work is the demonstration of transfer: although the network was trained on the faces of Facebook users, it was sufficient to just retrain the fully connected layers at the top to beat all benchmarks on a dataset of images of celebrities.

These days the privacy problem of face recognition has been discussed in the newspapers. Indeed, with the data available on the net, it is possible to do high-level automatic face recognition in all sorts of situations.

Interestingly, staff from Google have no image online. They know why!

Take your own consequences!

Summary

- Inductive bias by ConvNet structure

- ResNet makes ConvNet structure more flexible

- Outer product reduces flexibility (and number of parameters)

- Needs a lot of data

- Energy hungry (societal problem!)

- Transfer learning:
    - basic representation is stable across tasks
    - train a few output layers for specific task

Previous slide.

Training deep networks consumes a huge amount of energy. Think about this before your train your own deep network from scratch.

Transfer learning also helps to reduce energy consumption.

Summary

1) Use good explicit inductive biases together with transfer learning and data augmentation.

2) Convolutions (and max-pooling) are reasonable inductive biases for natural images.

3) Residual connections help to learn in very deep networks.

4) Modern architectures like Inception-Resnet-v2, DeepFace and NASNet reach human level performance on recognition tasks

   → face recognition poses societal problems!

5) AutoDiff is a flexible tool to train different architectures.

Teaching monitoring – monitoring of understanding

[ ] today, up to here, at least 60% of material was new to me.

[ ] up to here, I have the feeling that I have been able to follow (at least) 80% of the lecture.

The end

Artificial Neural Networks

Convolutional Neural Networks

Johanni Brea & W. Gerstner
EPFL, Lausanne, Switzerland

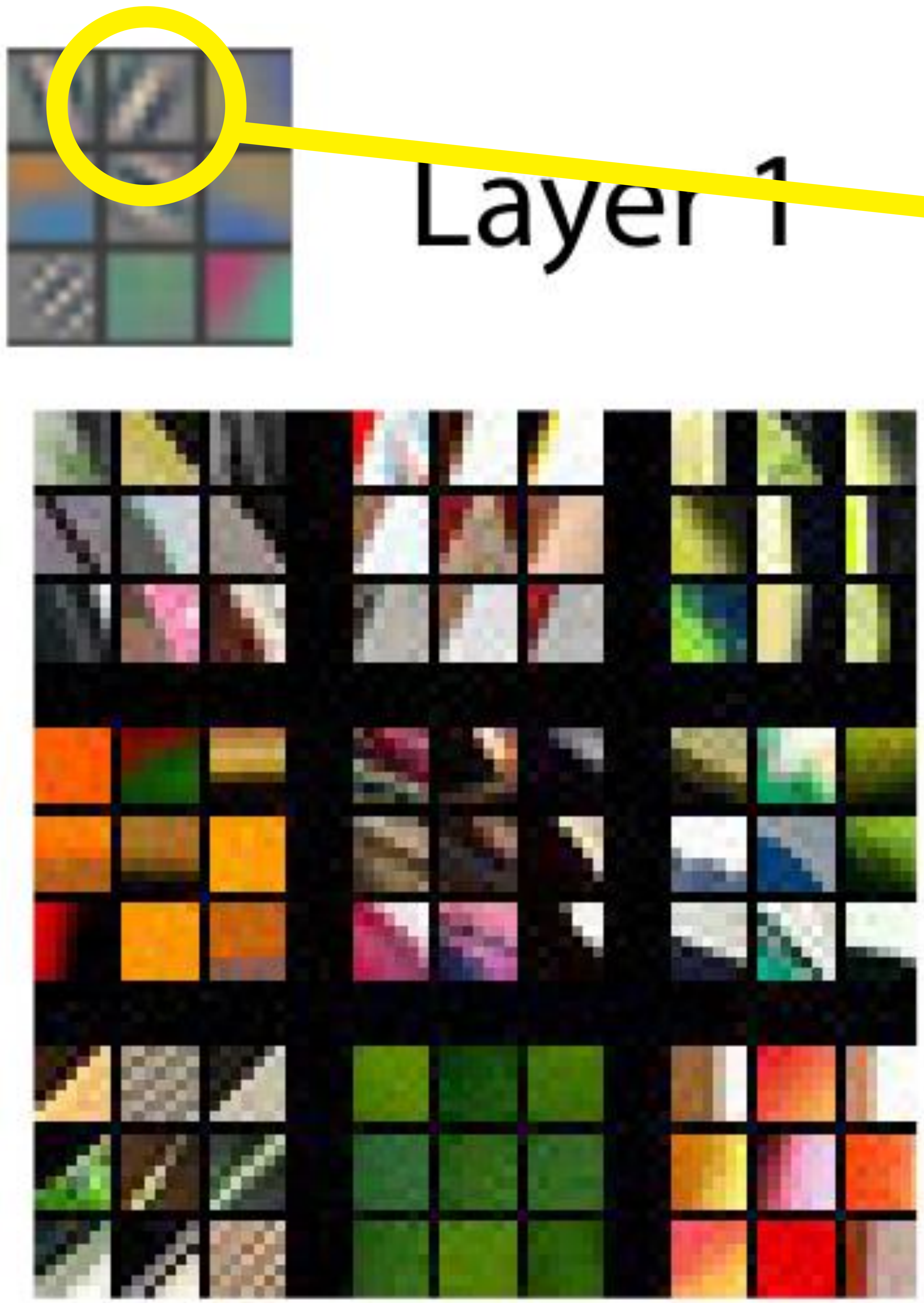
Part 9: Beyond object recognition

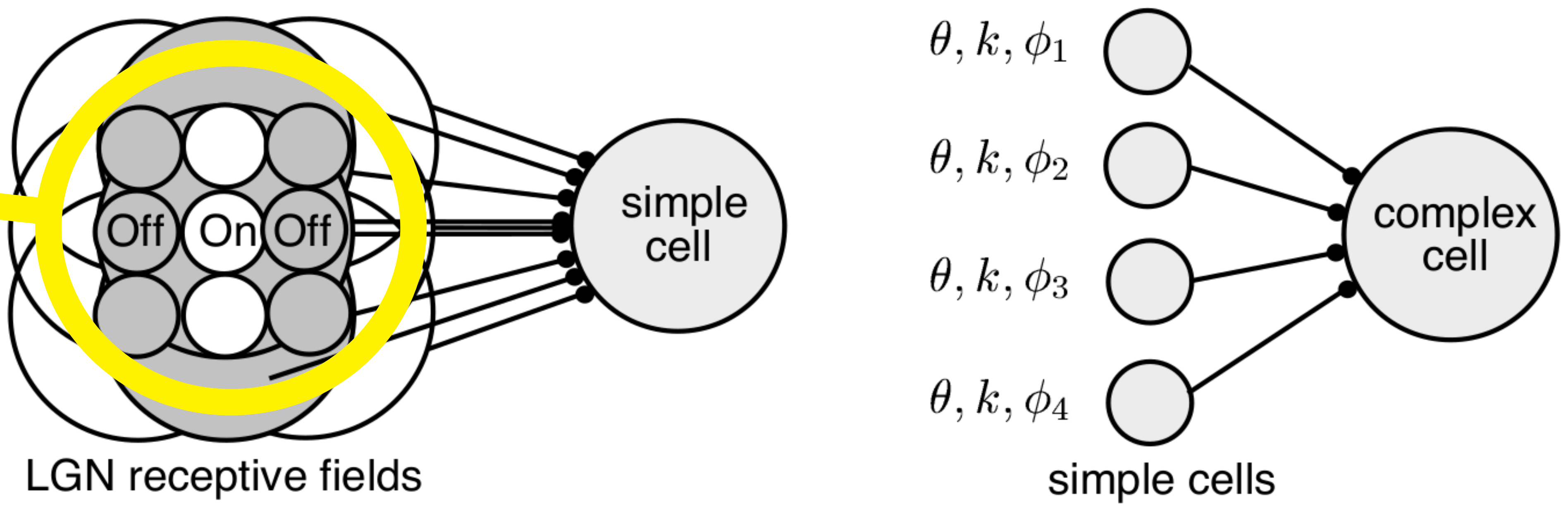
1. No Free Lunch
2. Inductive bias in machine learning
3. Convolution filters as inductive bias for images
4. MaxPooling as inductive bias for images
5. Modern ConvNets and ImageNet competitions
6. Gradient of a max-pooling layer
7. Automatic Differentiation
8. Modern Deep Networks and image recognition
9. **Applications beyond object recognition**

Next 4 slides – not shown in class. Will probably be shifted to the final lecture.

Convolutional neural networks share some features with biological neural networks.
In particular neurons in higher layers tend to respond to more abstract features than neurons in lower layers. In layer 1 we find features like those of complex cells, described in the famous work of Hubbel and Wiesel that studied experimentally the visual system of cats. For layers 2 to 5 on the following slides, we find the top 9 image patches that maximally activated a trained neural network together with a reconstruction of the feature within each patch that was responsible for the high activation. We see that neurons in the second layer respond to basic features like colors or edges of different orientation. In layer 5 for example, we see neurons that respond to the face of a dog with a high degree of invariance, i.e. the feature is not “distracted” by different backgrounds of kinds of dogs.

Neuroscience



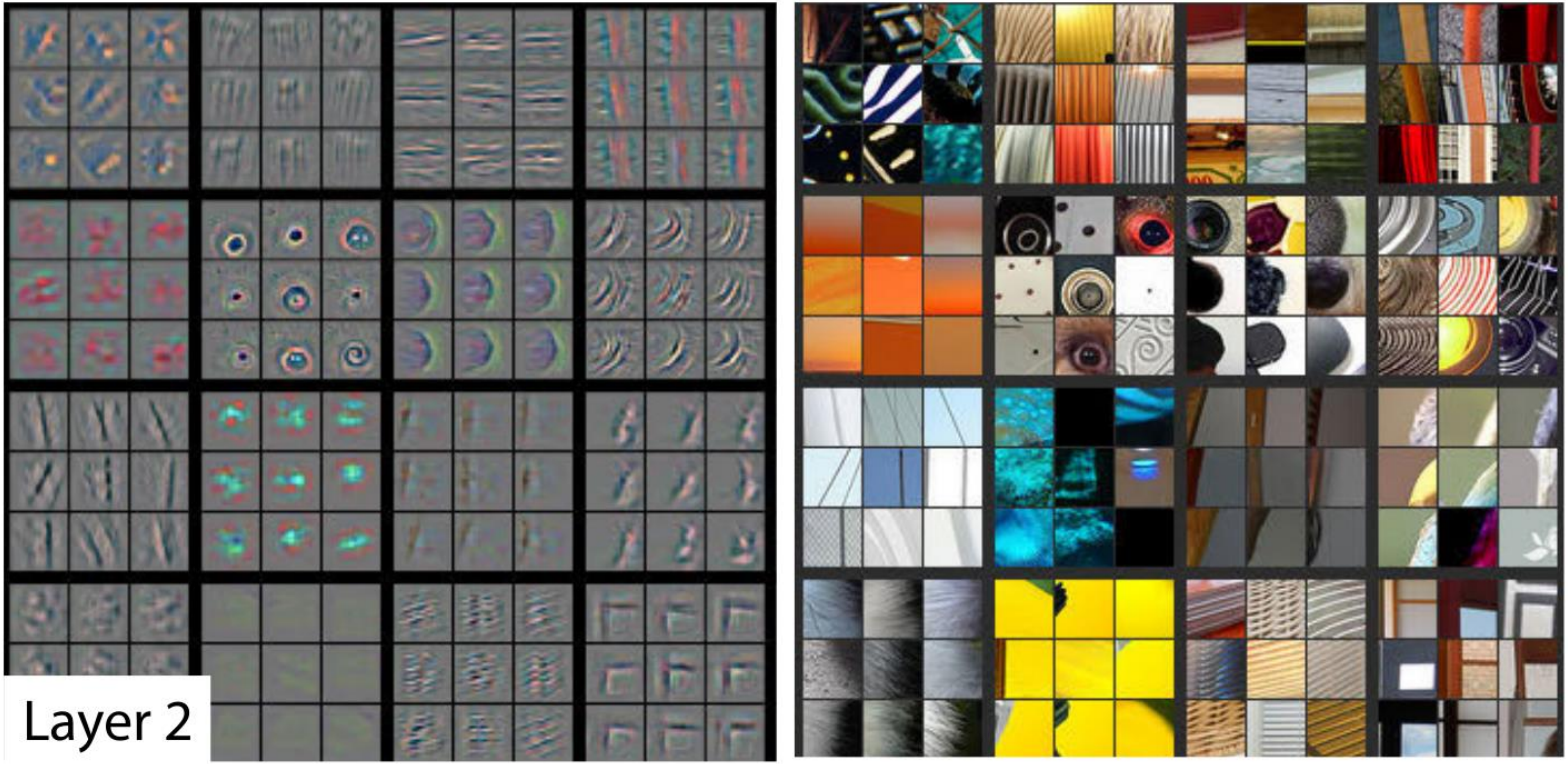


- ConvNets have similar architecture and similar features as the visual system of animals and humans

Dayan and Abbott, Theoretical Neuroscience, 2001

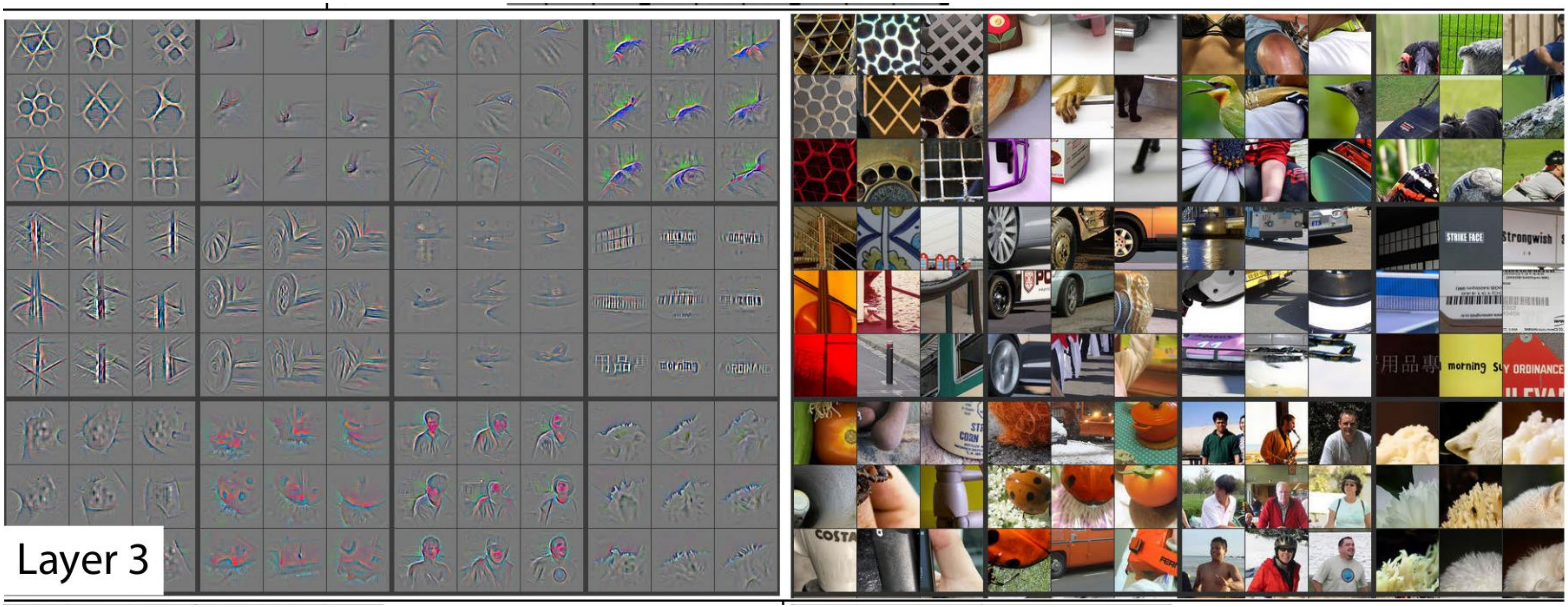
https://arxiv.org/pdf/1311.2901.pdf

Visualizing convolutional networks



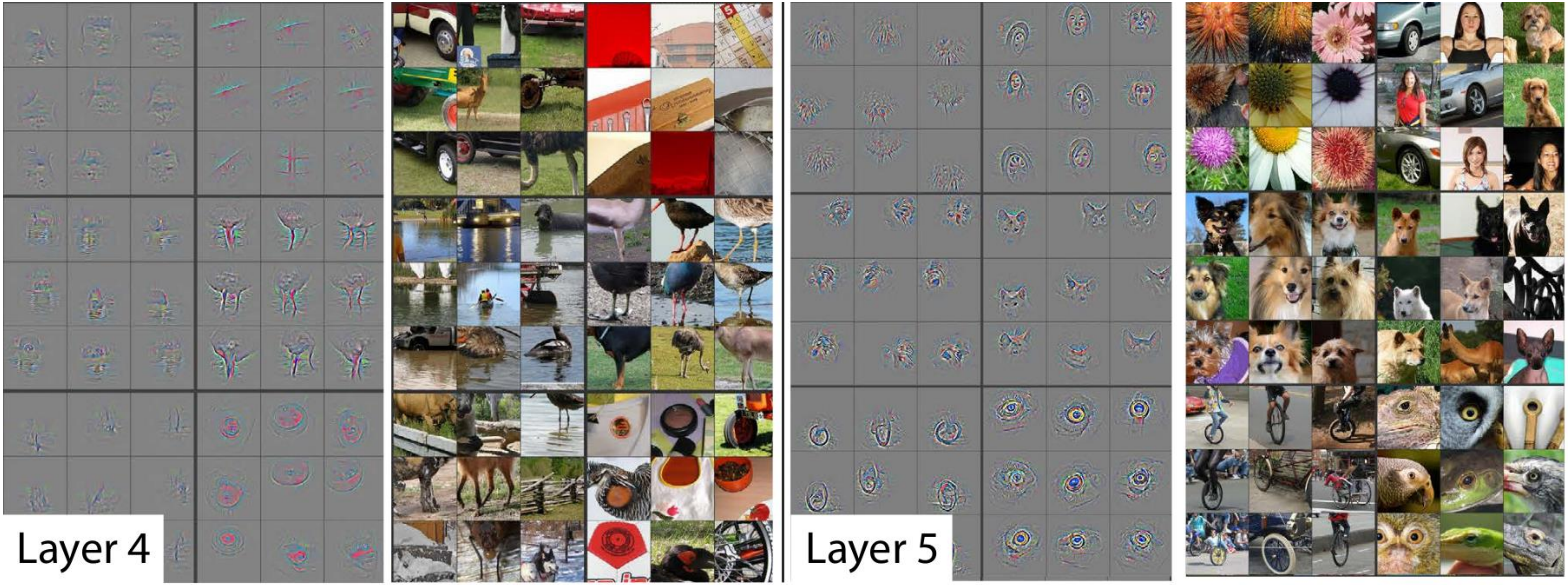
https://arxiv.org/pdf/1311.2901.pdf

Visualizing convolutional networks



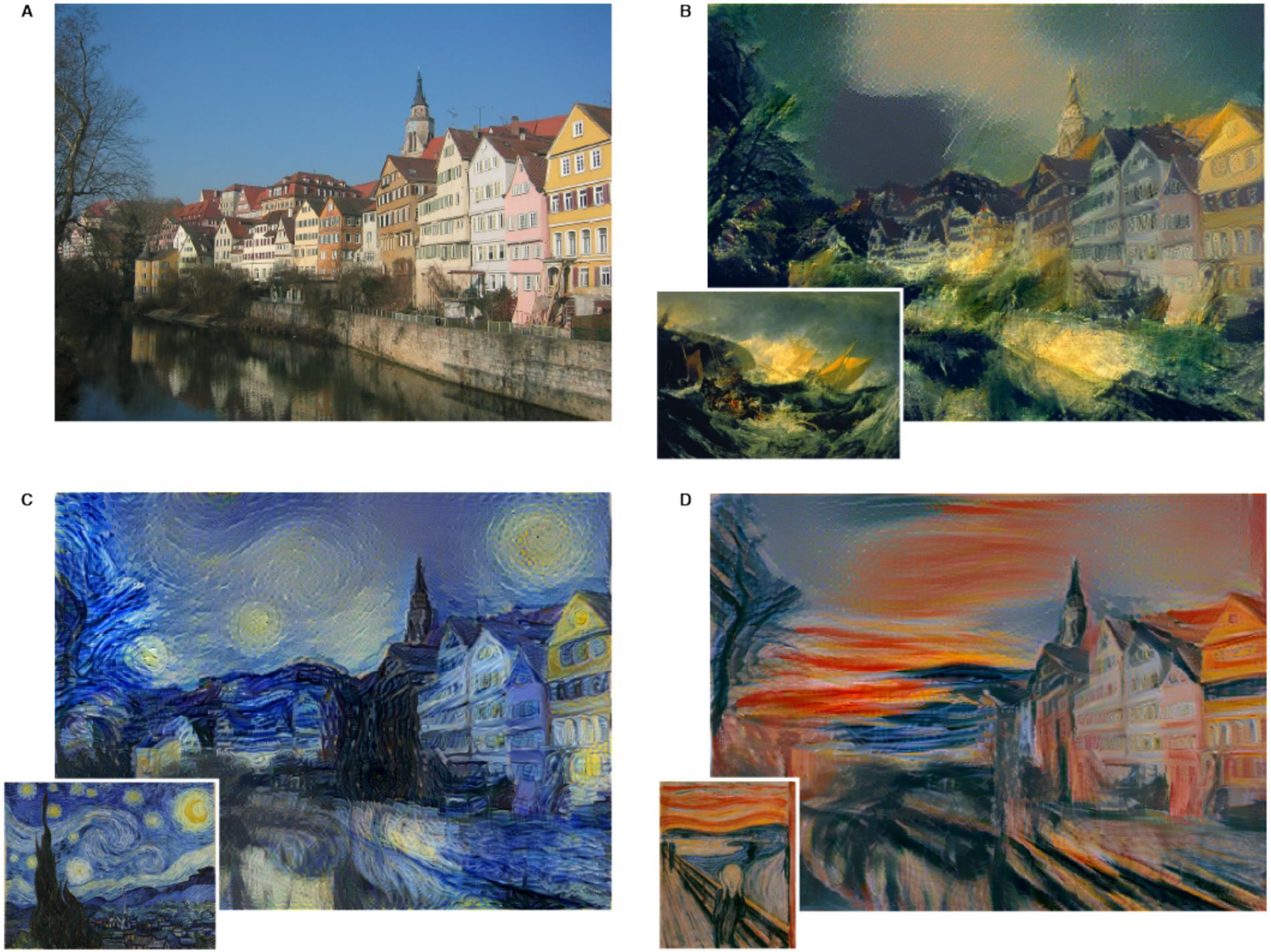
https://arxiv.org/pdf/1311.2901.pdf

Visualizing convolutional networks



https://arxiv.org/pdf/1311.2901.pdf

Neural style

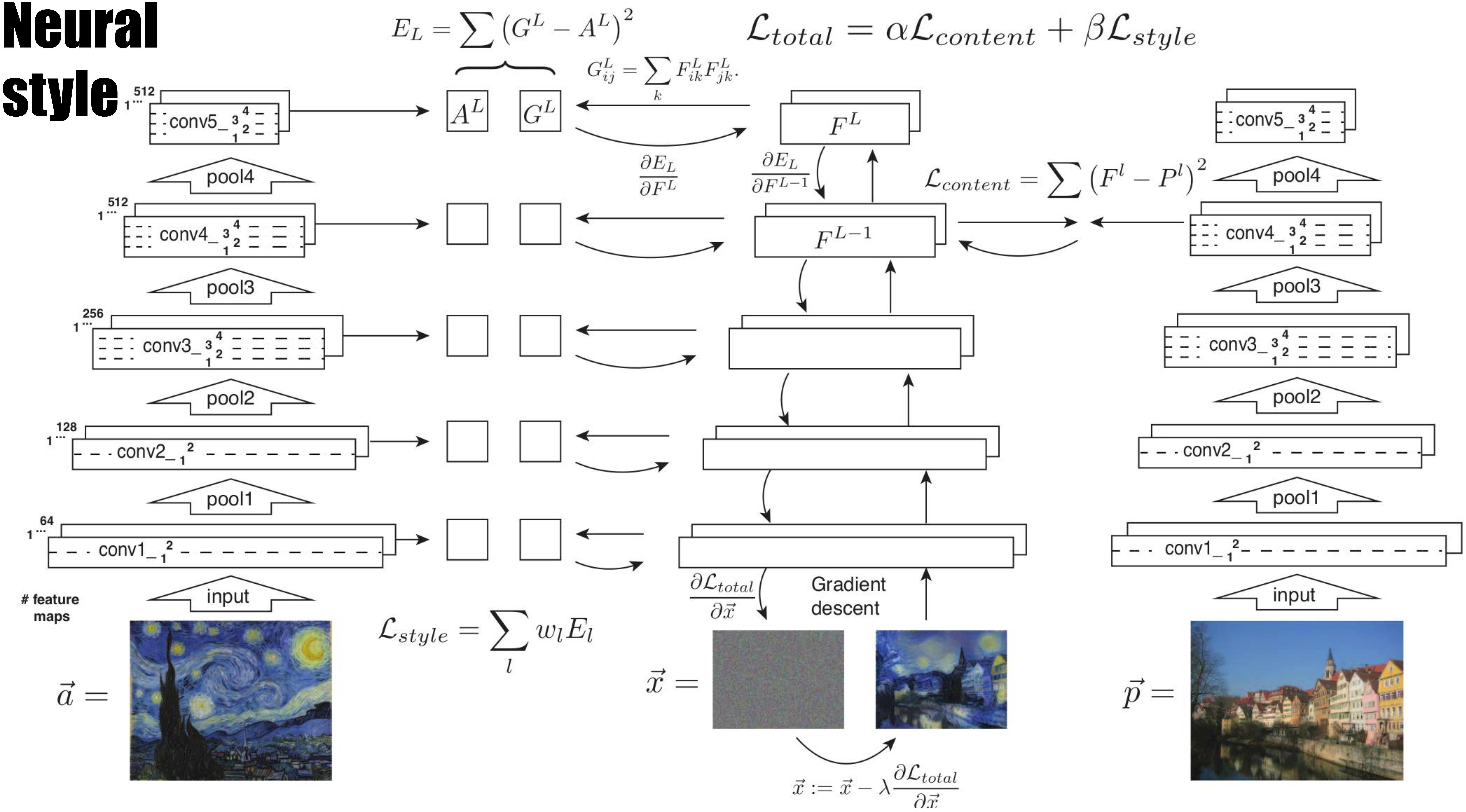


https://arxiv.org/pdf/1508.06576.pdf

Previous slide.

Neural style transfer is a now famous application of convolutional neural networks. Shown is a photo of Tübingen, regenerated by the network in 3 different styles, where the style was extracted from the small images in the bottom left corner of each panel.

Neural style



$$E_L = \sum \left(G^L - A^L\right)^2$$

$$\mathcal{L}_{total} = \alpha\mathcal{L}_{content} + \beta\mathcal{L}_{style}$$

$$G^L_{ij} = \sum_k F^L_{ik}F^L_{jk}.$$

$$\frac{\partial E_L}{\partial F^L} \qquad \frac{\partial E_L}{\partial F^{L-1}}$$

$$\mathcal{L}_{content} = \sum \left(F^l - P^l\right)^2$$

$$\mathcal{L}_{style} = \sum_l w_l E_l$$

$$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$$

Gradient descent

$$\vec{x} := \vec{x} - \lambda\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$$

Previous slide.

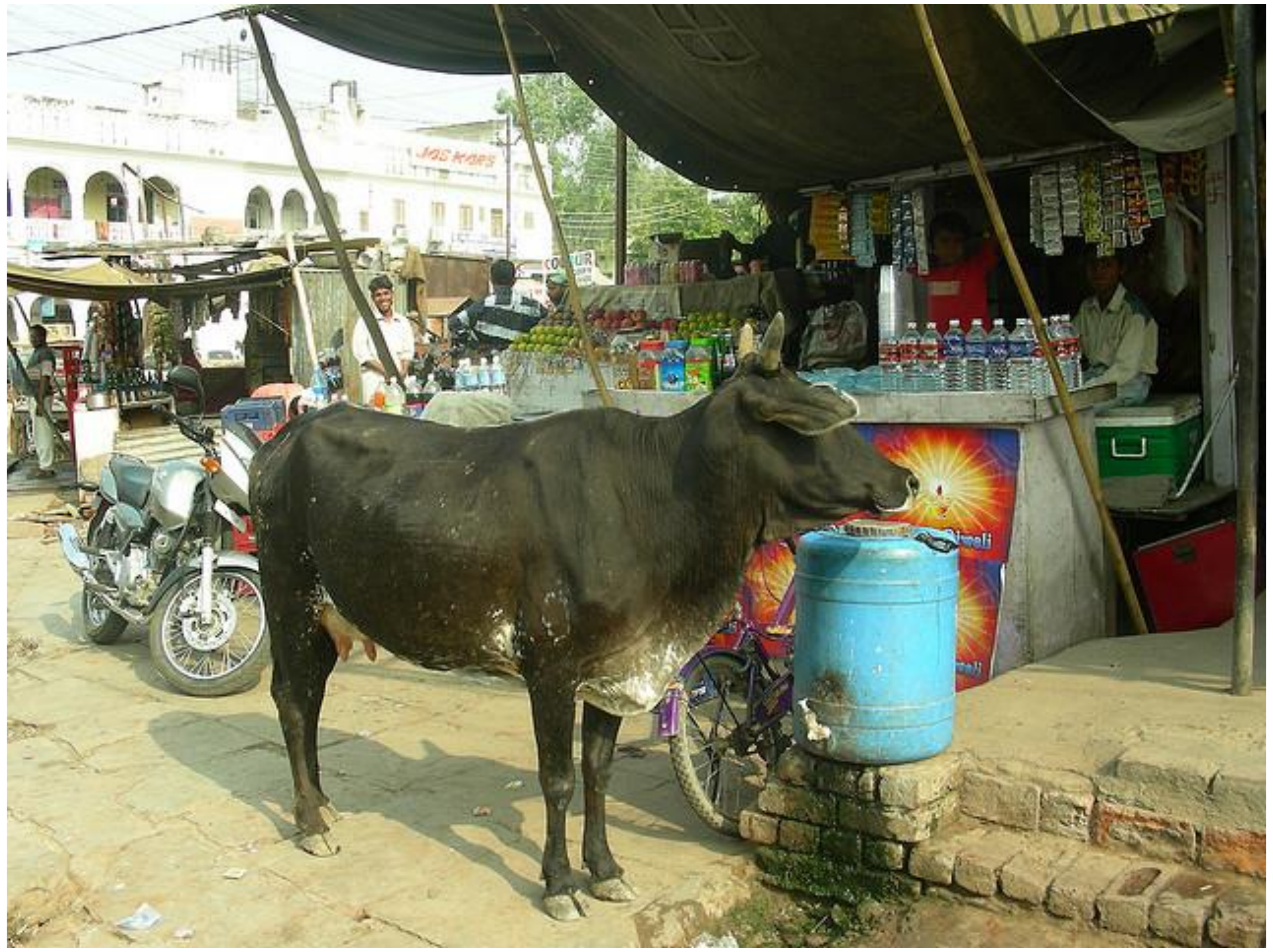
This works the following way:
A trained convolutional neural network is activated with an input image p. The activity P^l in each feature layer l in response to this input can than be used to reconstruct the image.

This is done in the following way: start with a new input of random pixel values x and compare the feature response F^l to P^l. Now we minimize the difference between F^l and P^l with gradient descent to find back a reconstruction of p. When using the features in the lower layers the reconstruction is almost perfect.

The style of an image can be reconstructed by following a very similar procedure but instead of comparing the responses, the difference between feature correlations G^l and A^l are minimized.

If the losses of style reconstruction and content reconstructions are mixed, the resulting image will have the content of input image p with the style of input image a.

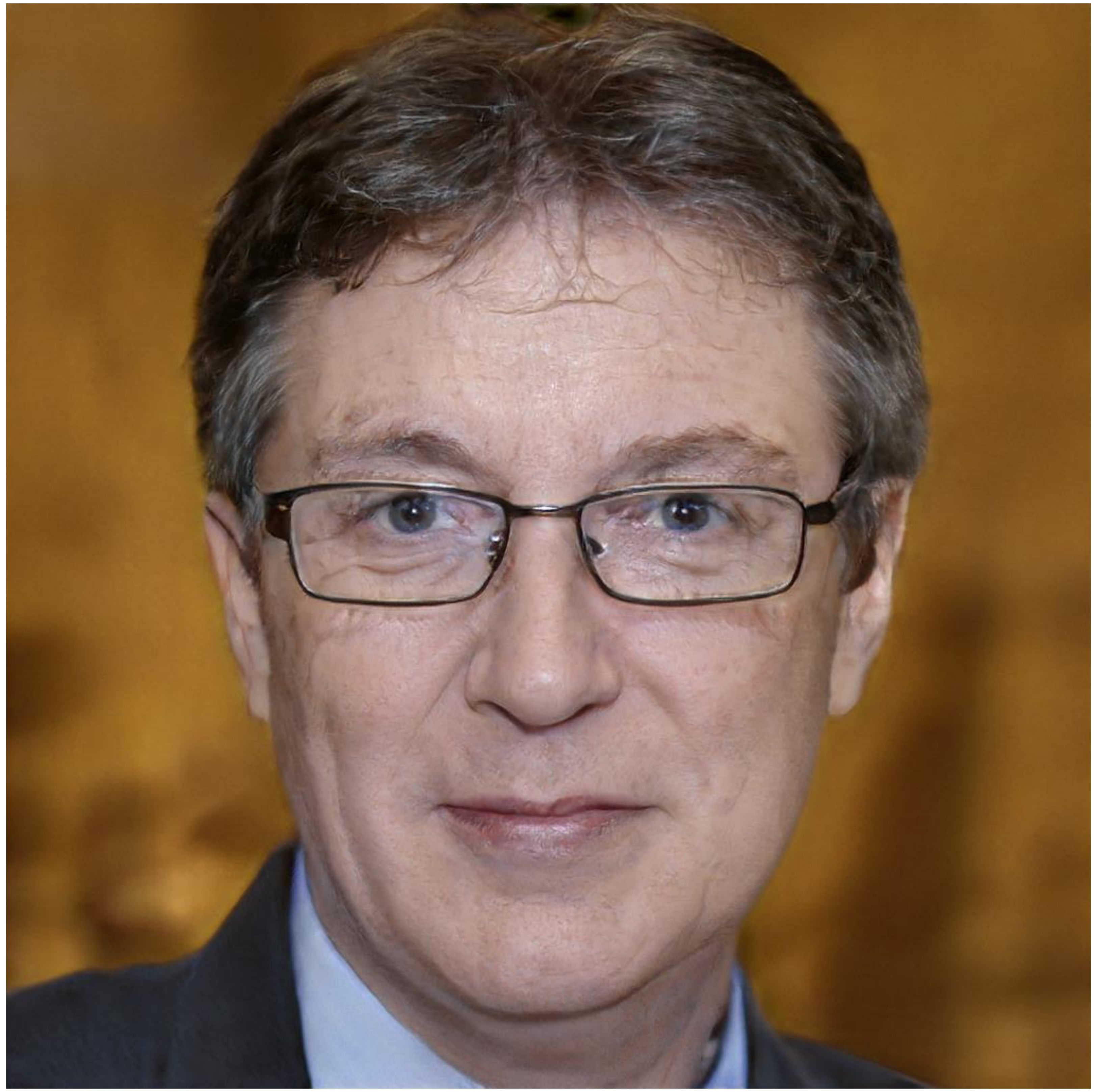
Caption generation and generative models of images



a cow is standing in the middle of a street

https://cs.stanford.edu/people/karpathy/deepimagesent/

ConvNet + bidirectional RNN



https://thispersondoesnotexist.com/

ConvNets in generative adversarial networks (GANs)

Previous slide.

Convolutional networks together with recurrent neural networks have also enabled better automatic caption generators. On the left the image is given as input and the caption is generated as the output from recurrent neural network that received as input abstract features extracted with a convolutional neural network.

You have seen already in the previous lecture the images of persons that don’t exist that where generated with generative adversarial networks that use convolutional neural networks.

Summary

1) Use good explicit inductive biases together with transfer learning and data augmentation.

2) Convolutions (and max-pooling) are reasonable inductive biases for natural images.

3) Residual connections help to learn in very deep networks.

4) Modern architectures like Inception-Resnet-v2, DeepFace and NASNet reach human level performance on recognition tasks.

5) AutoDiff is a flexible tool to train different architectures.

6) ConvNets and neurons in the visual systems have similar receptive fields.