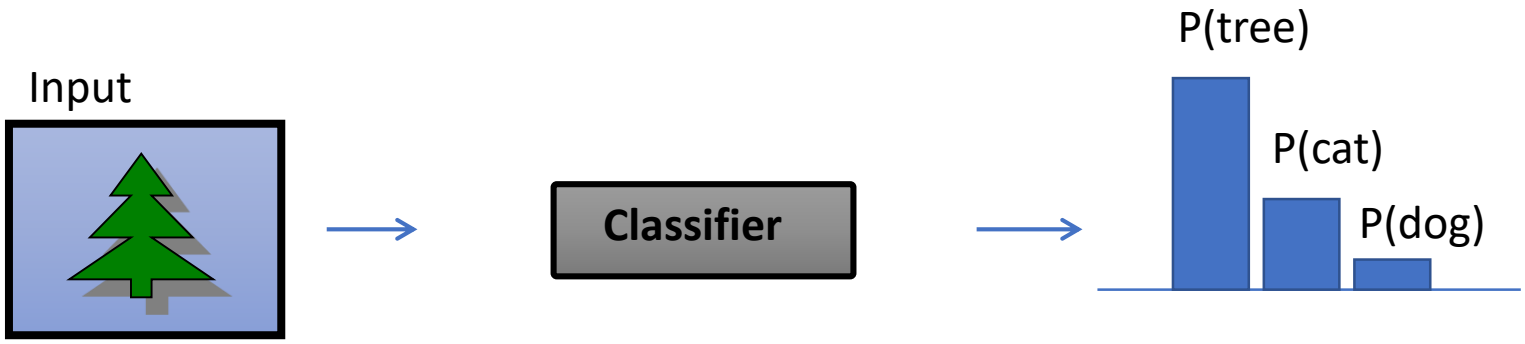
Introduction to Image Classification

Alessandro Giusti

What is image classification?



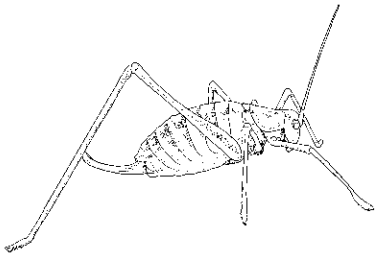
Part 1: a 10-minute recap on classification

The geometric intuition

The Classification Problem

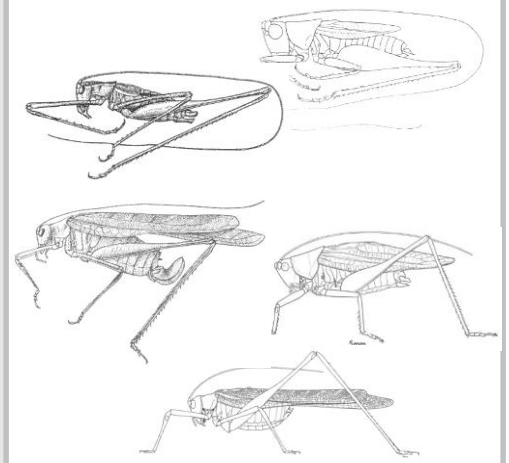
(informal definition)

Given a collection of annotated data, in this case 5 instances **Katydids** of and 5 of **Grasshoppers**, decide what type of insect the unlabeled example is.

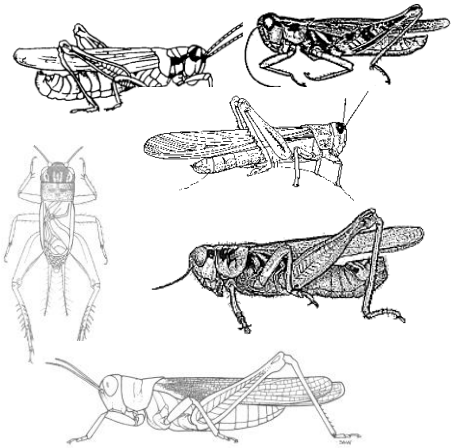


Katydid or **Grasshopper**?

Katydids

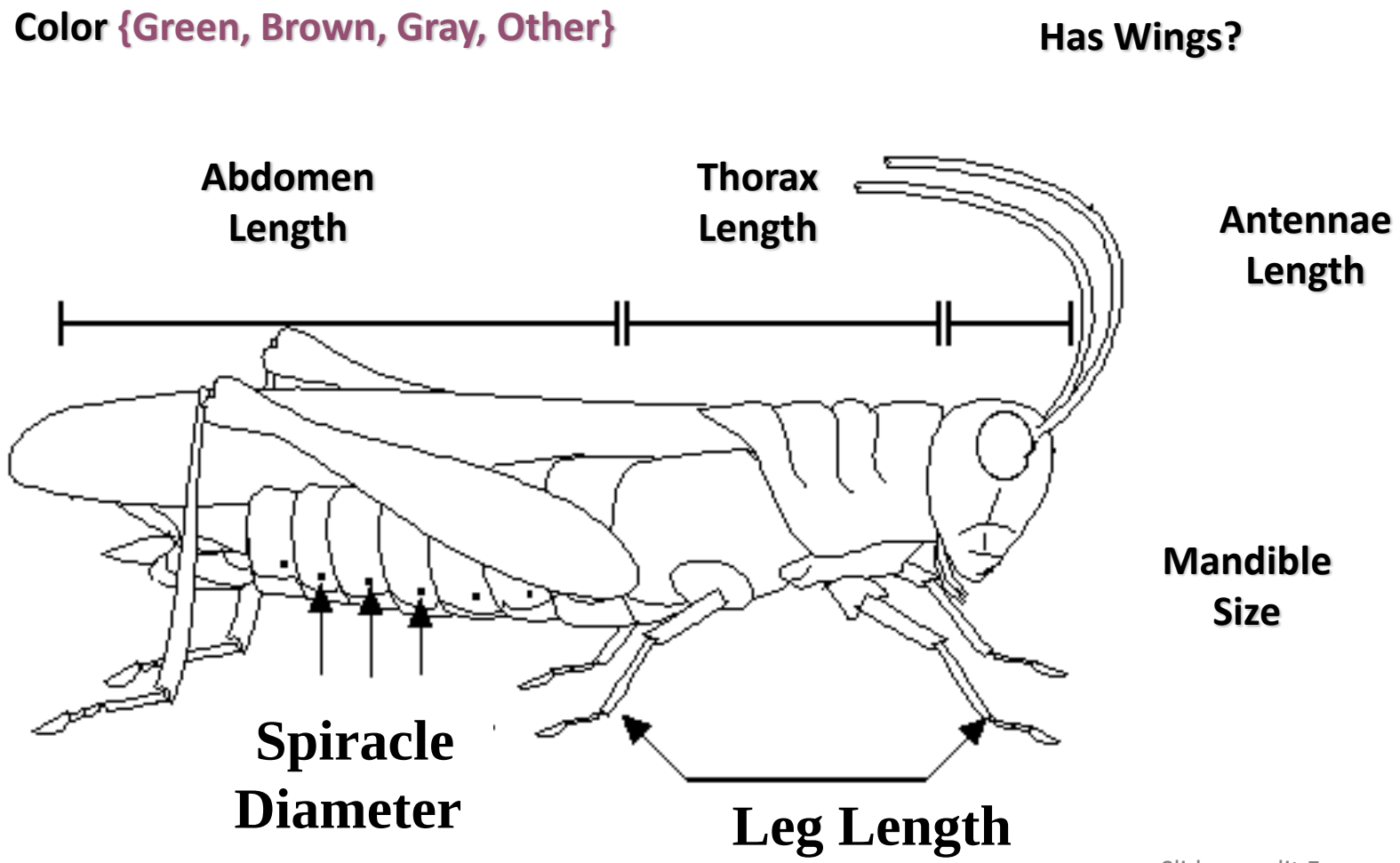


Grasshoppers



Slides credit Eamonn Keogh

For any domain of interest, we can measure *features*



Slides credit Eamonn Keogh

We can store features in a database.

The classification problem can now be expressed as:

- Given a training database (My_Collection), predict the class label of a previously unseen instance

My_Collection

Insect ID	Abdomen Length	Antennae Length	Insect Class
1	2.7	5.5	Grasshopper
2	8.0	9.1	Katydid
3	0.9	4.7	Grasshopper
4	1.1	3.1	Grasshopper
5	5.4	8.5	Katydid
6	2.9	1.9	Grasshopper
7	6.1	6.6	Katydid
8	0.5	1.0	Grasshopper
9	8.3	6.6	Katydid
10	8.1	4.7	Katydids

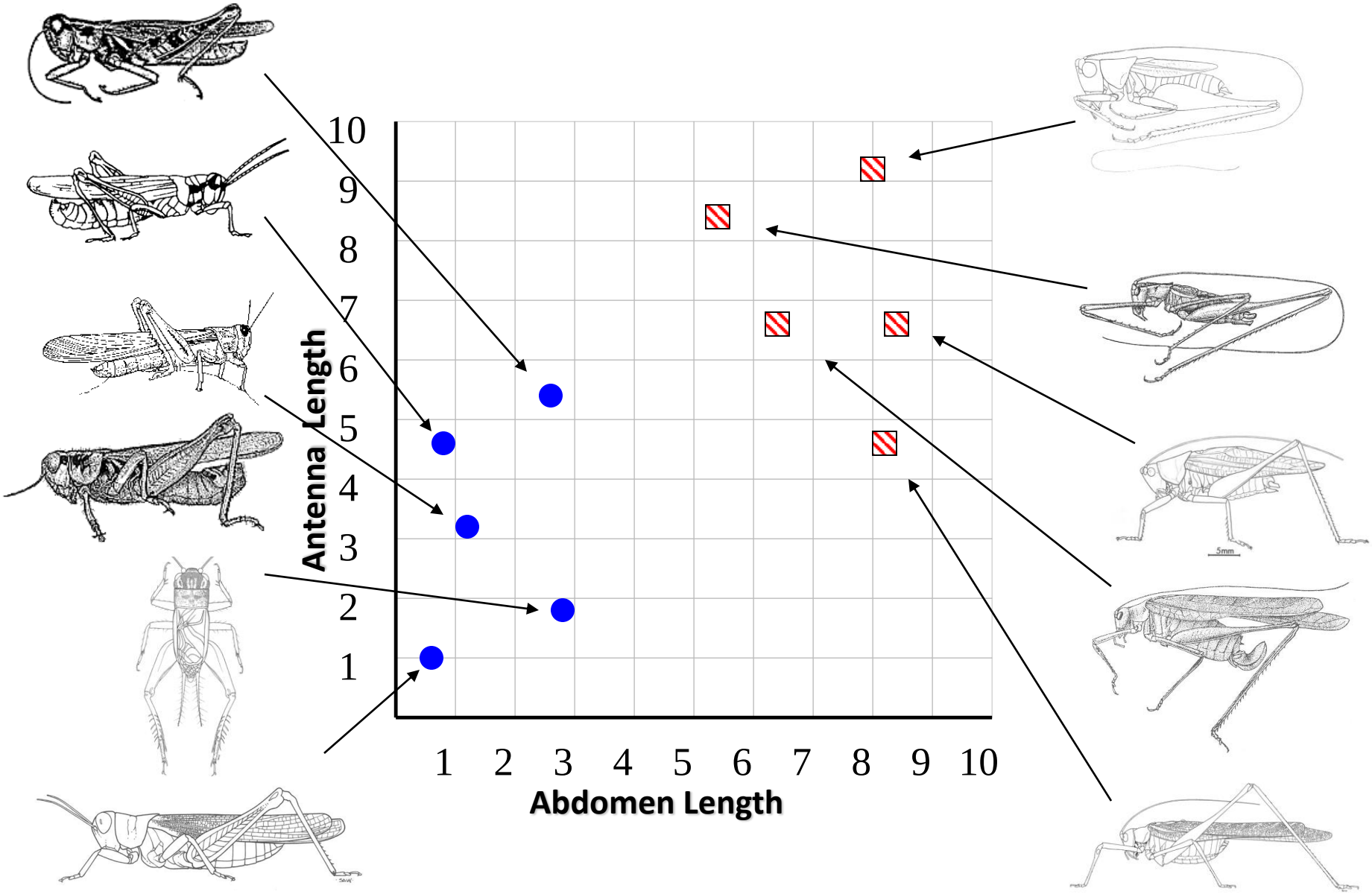
previously unseen instance =

11	5.1	7.0	???????

Slides credit Eamonn Keogh

Grasshoppers

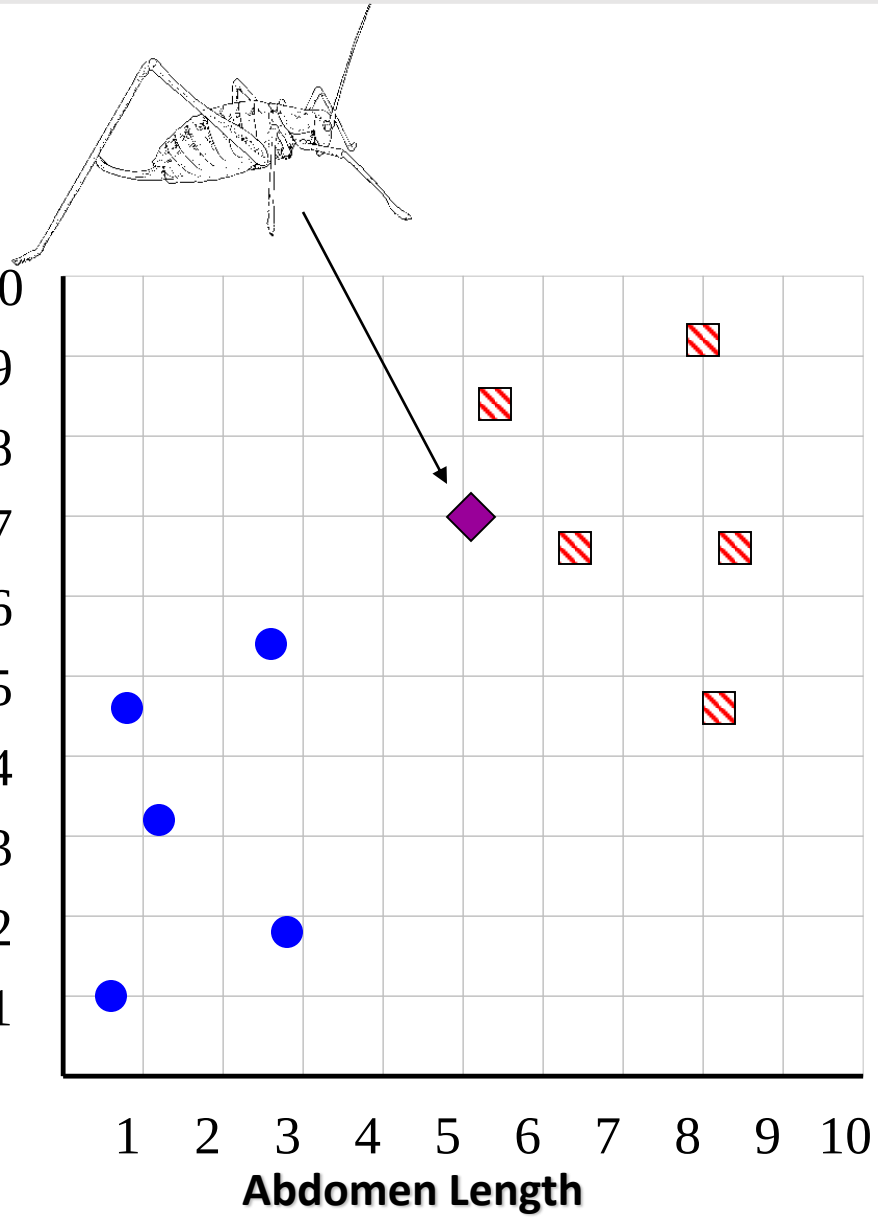
Katydids



Slides credit Eamonn Keogh

previously unseen instance =

11	**5.1**	**7.0**	**???????**



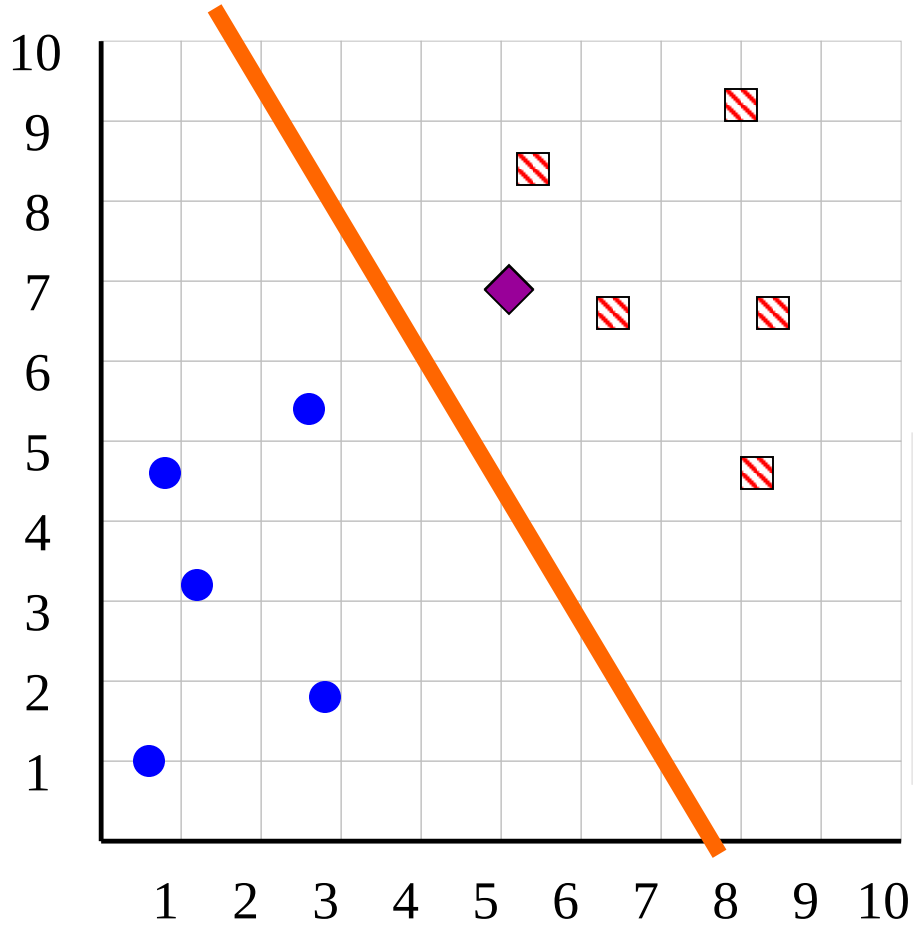
- We can "project" the previously unseen instance into the same space as the database.

- We have now abstracted away the details of our particular problem. It will be much easier to talk about points in space.

▨ Katydids

● Grasshoppers

Simple Linear Classifier

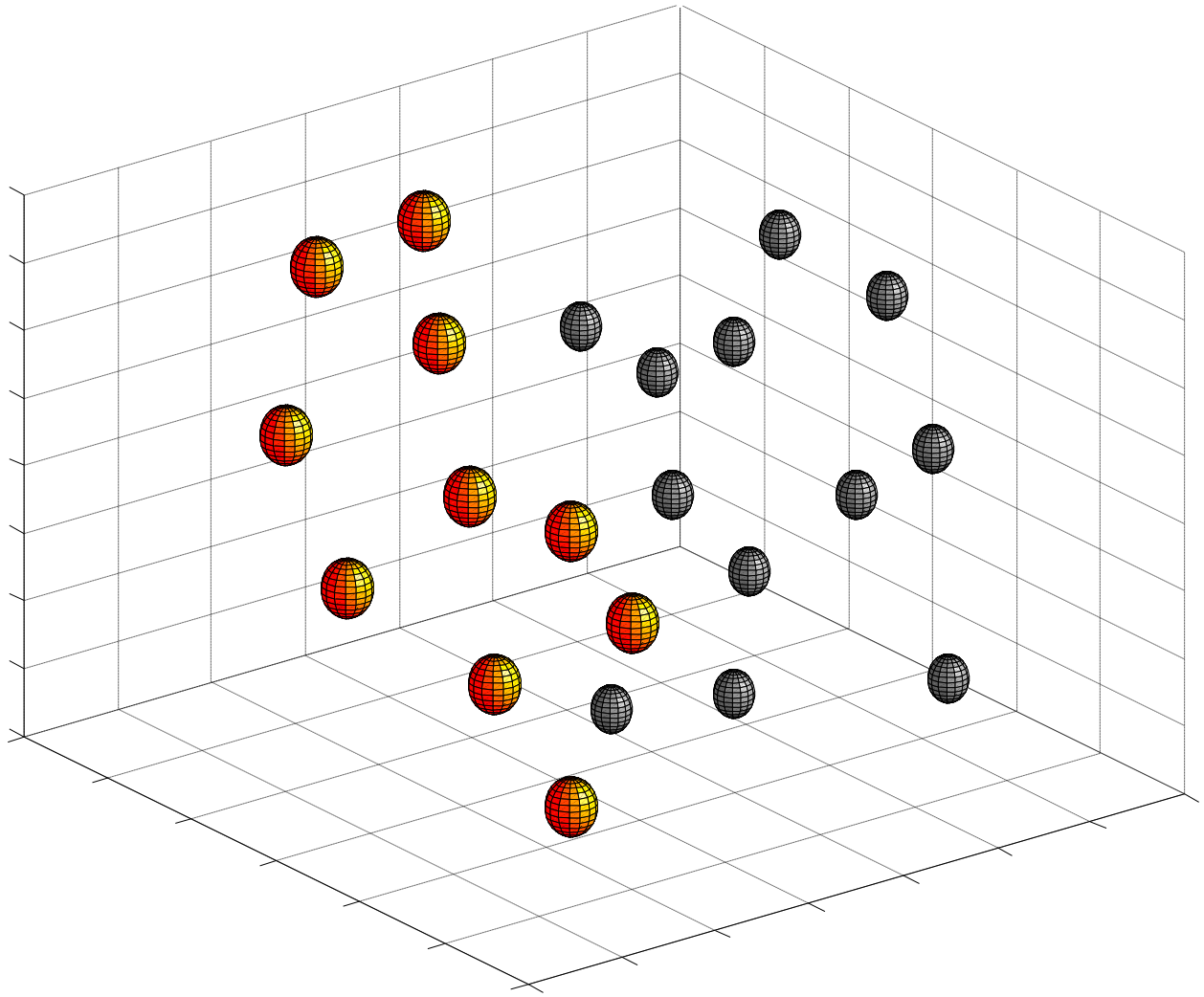


R.A. Fisher
1890-1962

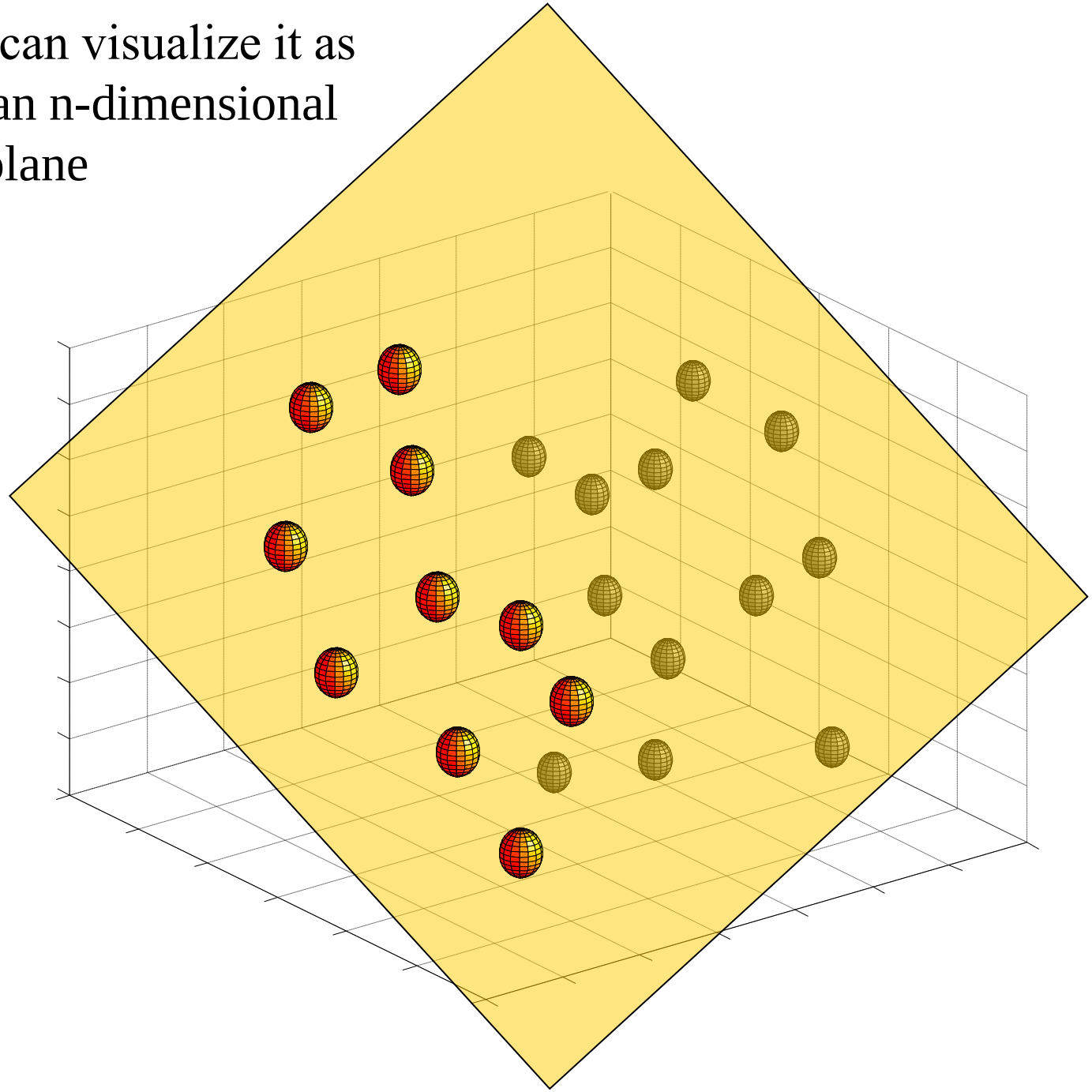
If previously unseen instance above the line
then
class is **Katydid**
else
class is **Grasshopper**

Katydids
Grasshoppers

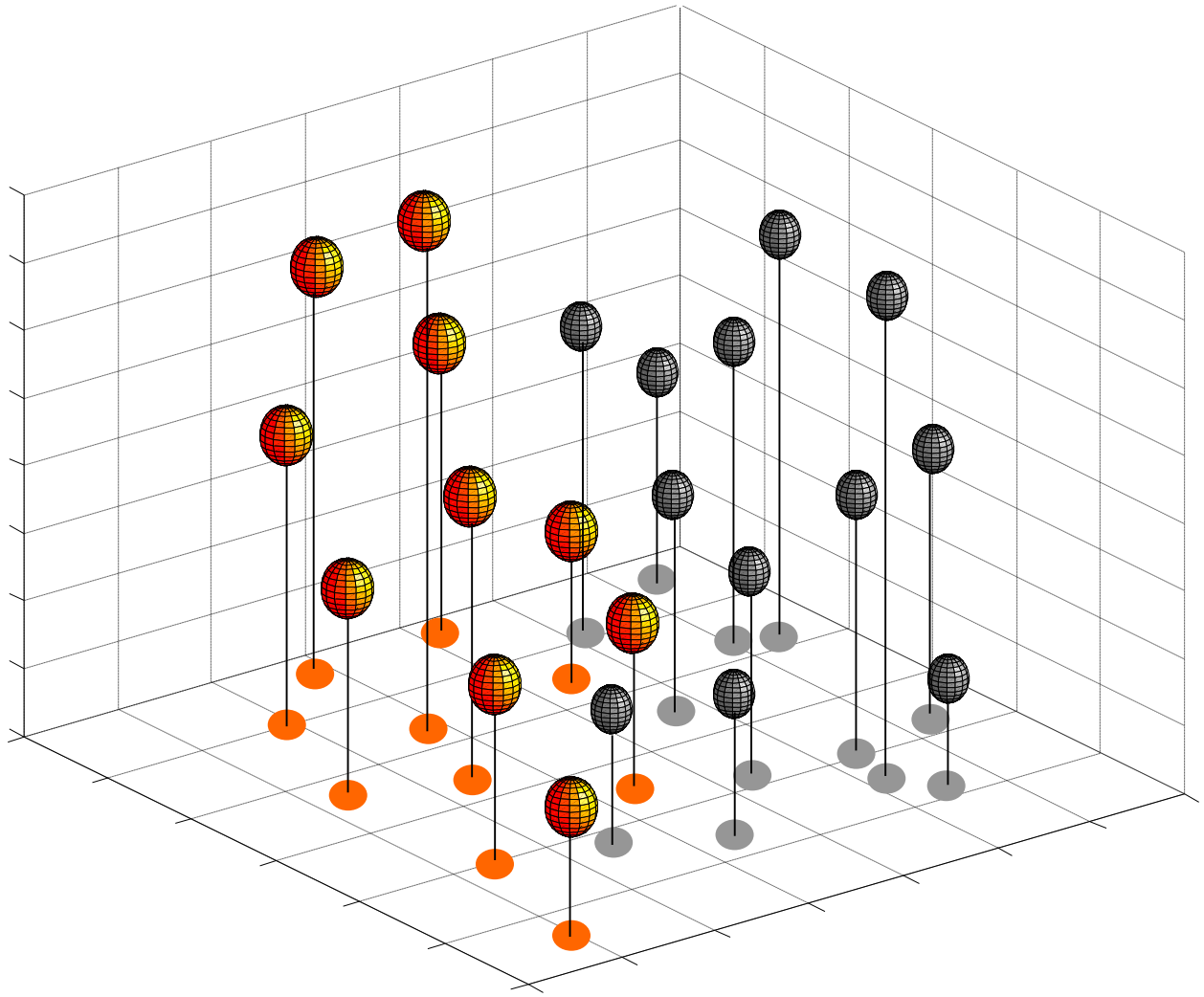
The simple linear classifier is defined for higher dimensional spaces…

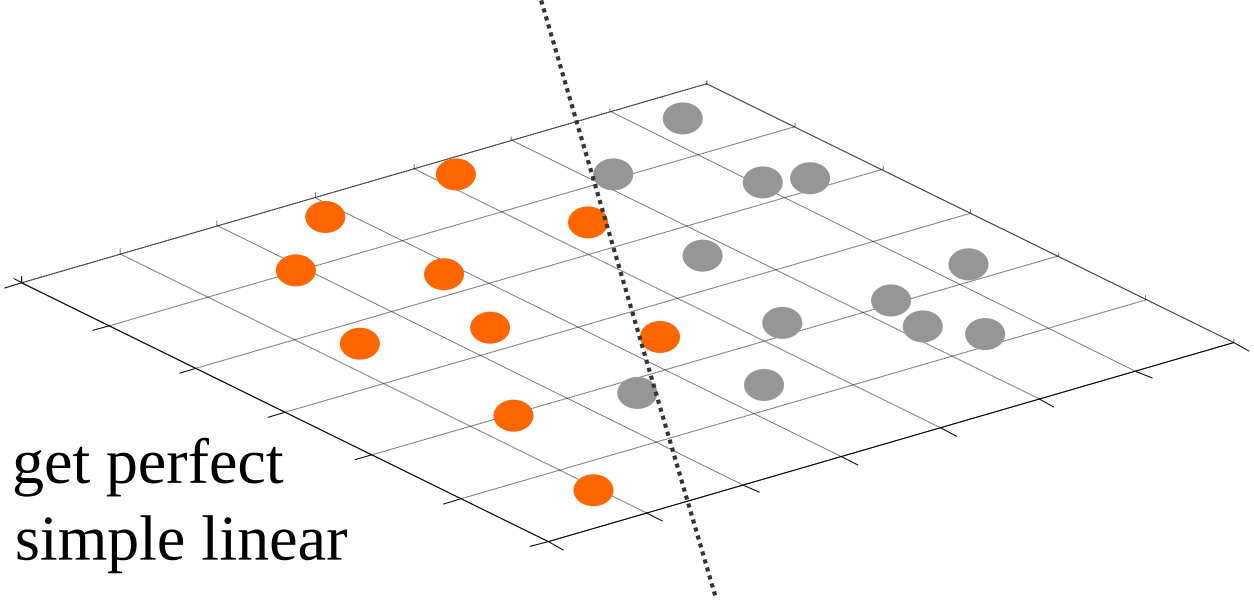


… we can visualize it as being an n-dimensional hyperplane



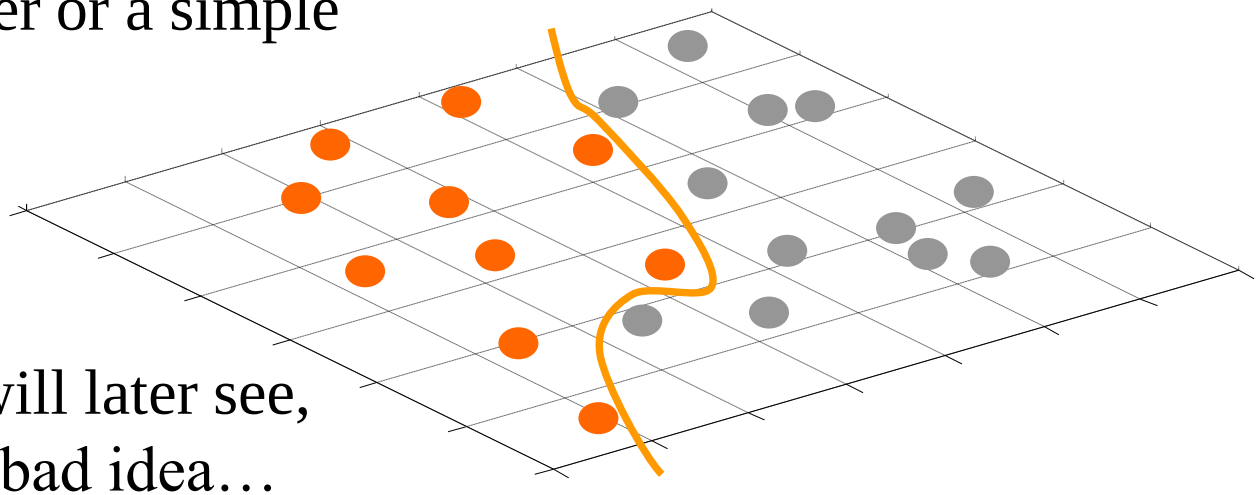
It is interesting to think about what would happen in this example if we did not have the 3rd dimension…





We can no longer get perfect accuracy with the simple linear classifier…

We could try to solve this problem by user a simple *quadratic* classifier or a simple *cubic* classifier..



However, as we will later see, this is probably a bad idea…

Predictive Accuracy

How do we *estimate* the **accuracy** of our classifier?
We need a VALIDATION or TESTING dataset that the classifier never saw before

On this dataset:

$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{Number of instances in our database}}$$

Predictive Accuracy

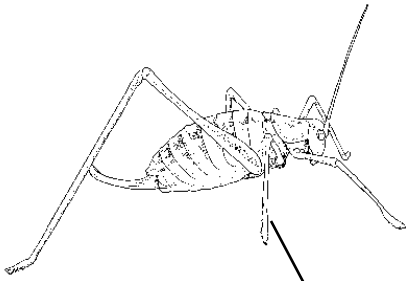
$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{Number of instances in our database}}$$

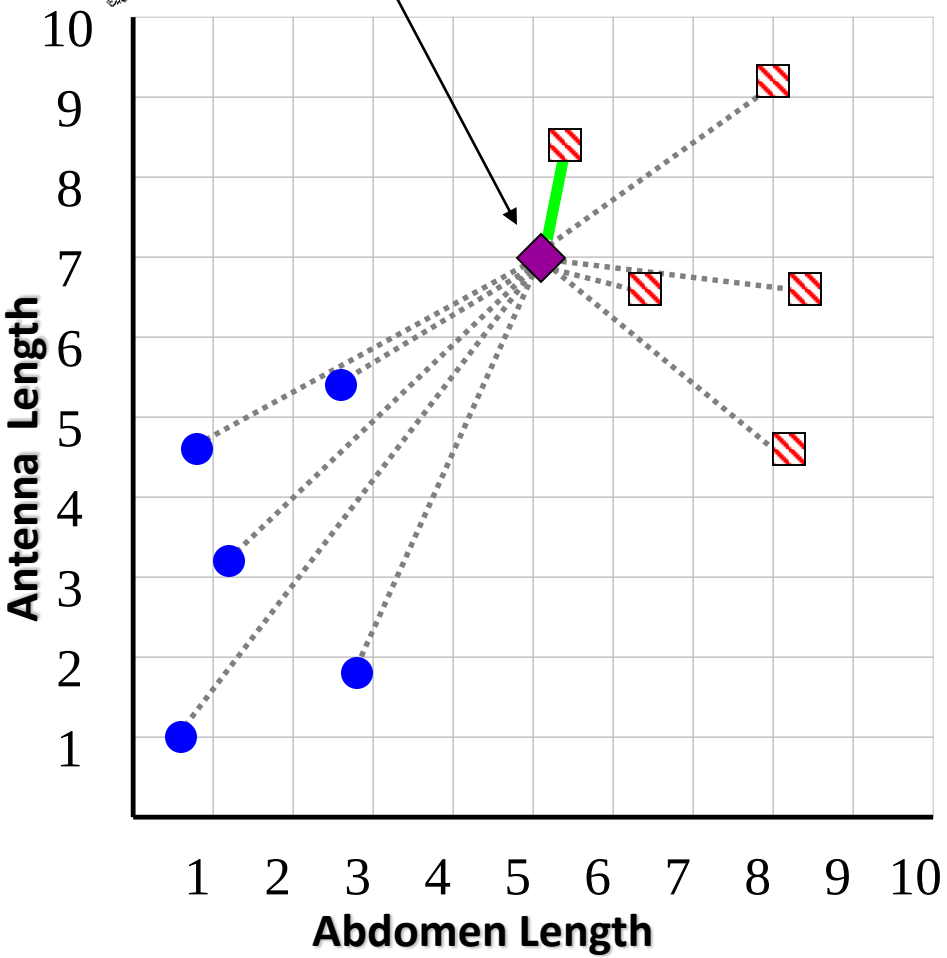
Accuracy is a single number, we may be better off looking at a **confusion matrix**. This gives us additional useful information…

Classified as a…

True label is…	**Cat**	**Dog**	**Pig**
Cat	**100**	**0**	**0**
Dog	**9**	**90**	**1**
Pig	**45**	**45**	**10**

A simple classifier: Nearest Neighbors





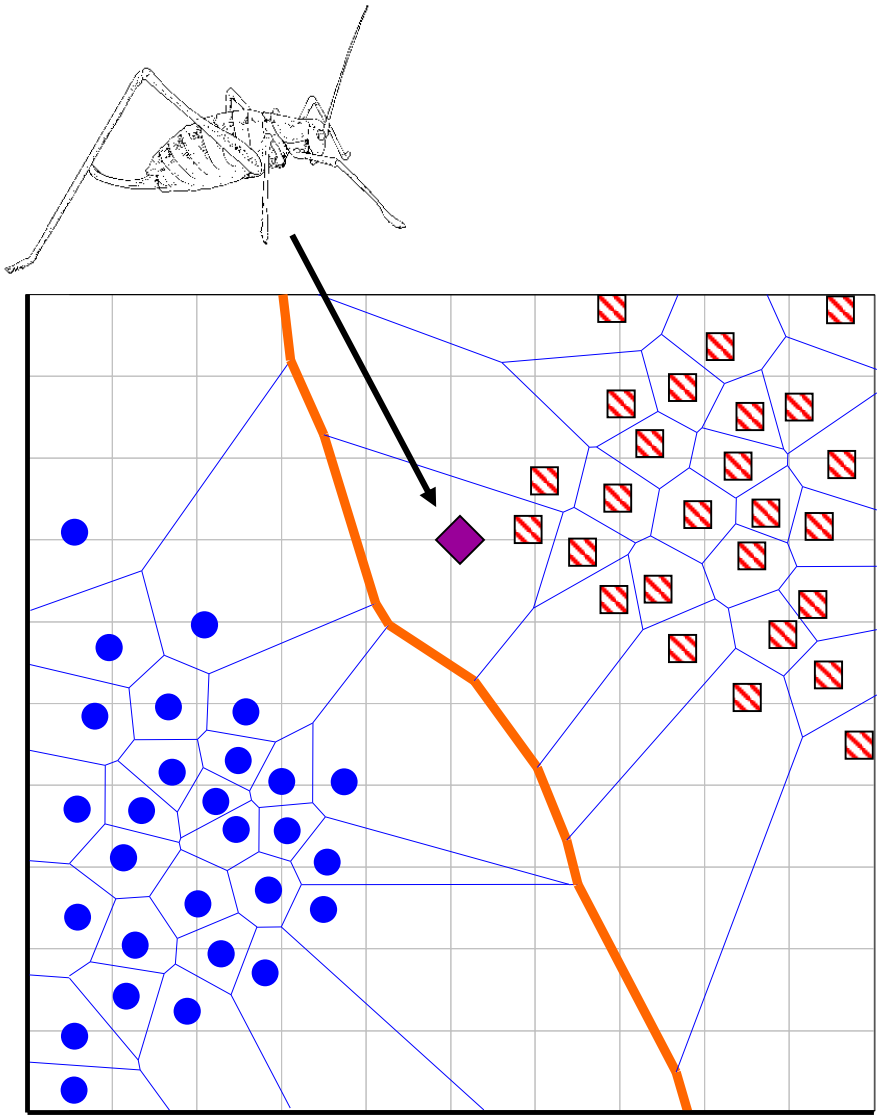
Evelyn Fix
1904-1965

Joe Hodges
1922-2000

If the **nearest** instance to the previously unseen instance **is a Katydid**
    class is **Katydid**
else
    class is **Grasshopper**

Katydids
Grasshoppers

We can visualize the nearest neighbor algorithm in terms of a decision surface…

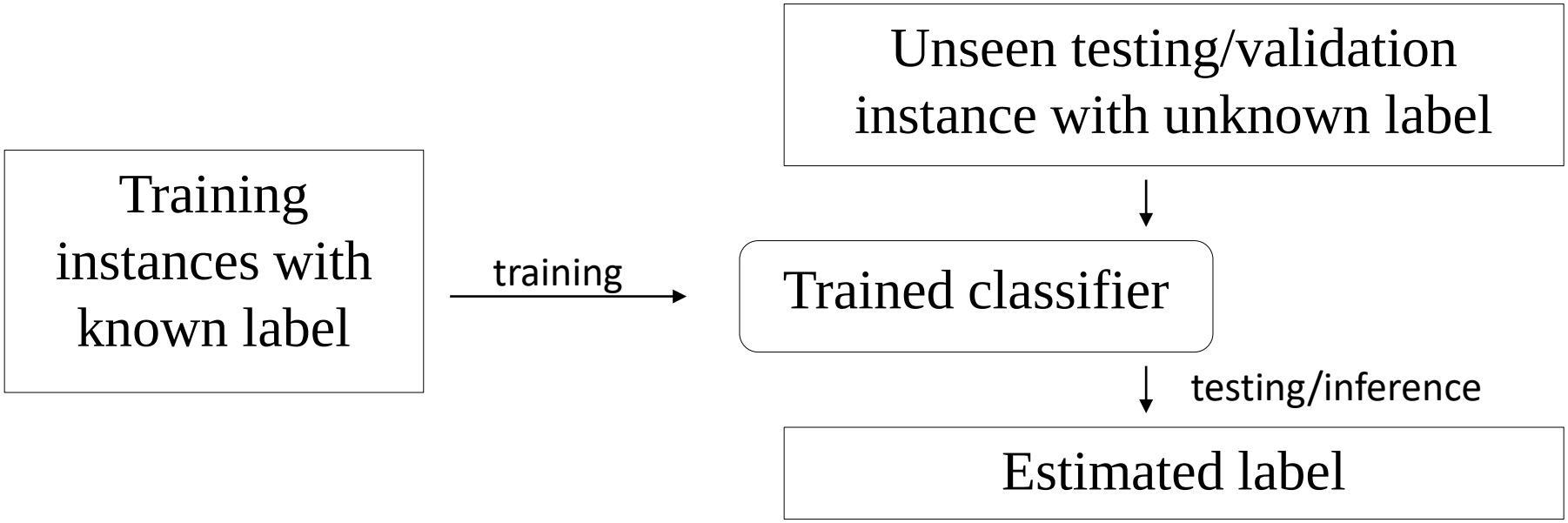


Note the we don't actually have to construct these surfaces, they are simply the implicit boundaries that divide the space into regions "belonging" to each instance.

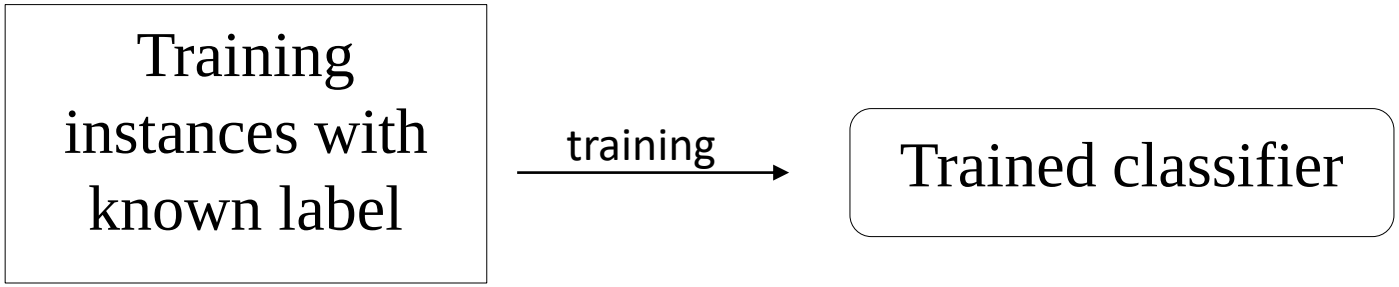
This division of space is called Dirichlet Tessellation (or Voronoi diagram, or Theissen regions).

Other classifiers are more complex, but we don't care

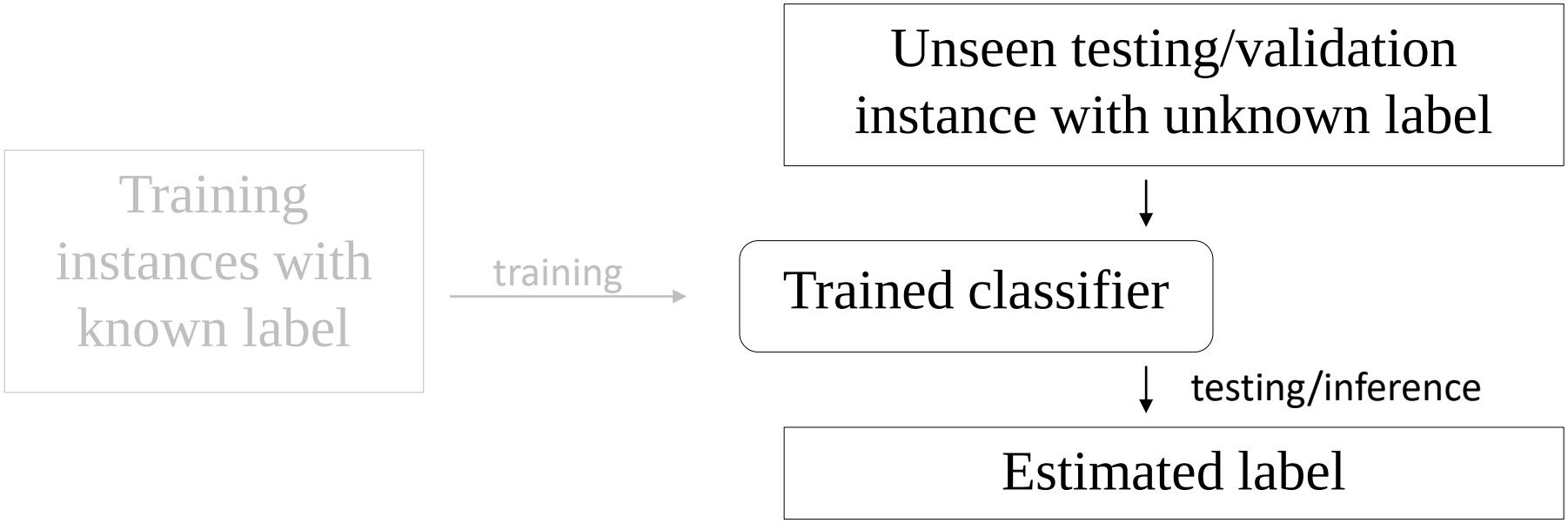
We'll consider them as a **black box** in this course



Step 1: training



Step 2: inference



Part 2

Three image classification approaches

Implementation

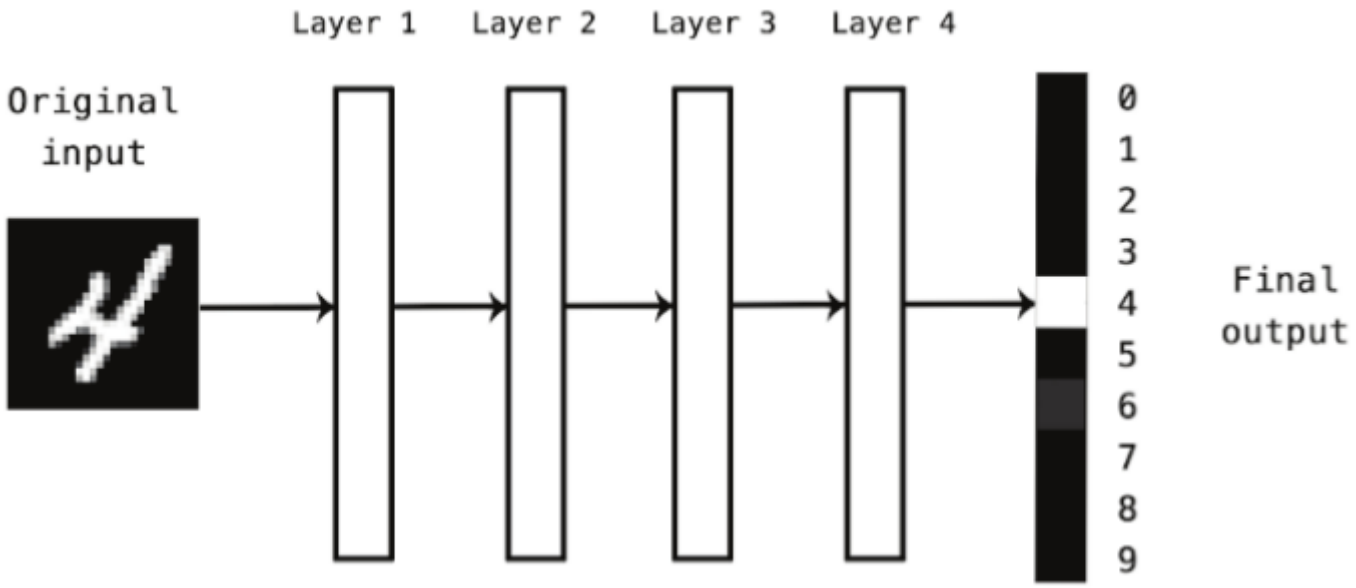
- Nearest-neighbor classifier using raw pixel values as features
- Nearest-neighbor classifier (or any other classifier) using handcrafted features
- Convolutional network using raw pixel values as features

Useful resources:
Andrej Karpathy, CS231n
http://cs231n.github.io/convolutional-networks/
Ian Goodfellow et al., Deep Learning
http://www.deeplearningbook.org/
F. Chollet, Deep Learning with Python
https://www.manning.com/books/deep-learning-with-python
Jonathan Hui, CNN Tutorial
https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/
Tim Demetters, Understanding Convolutions
http://timdettmers.com/2015/03/26/convolution-deep-learning/

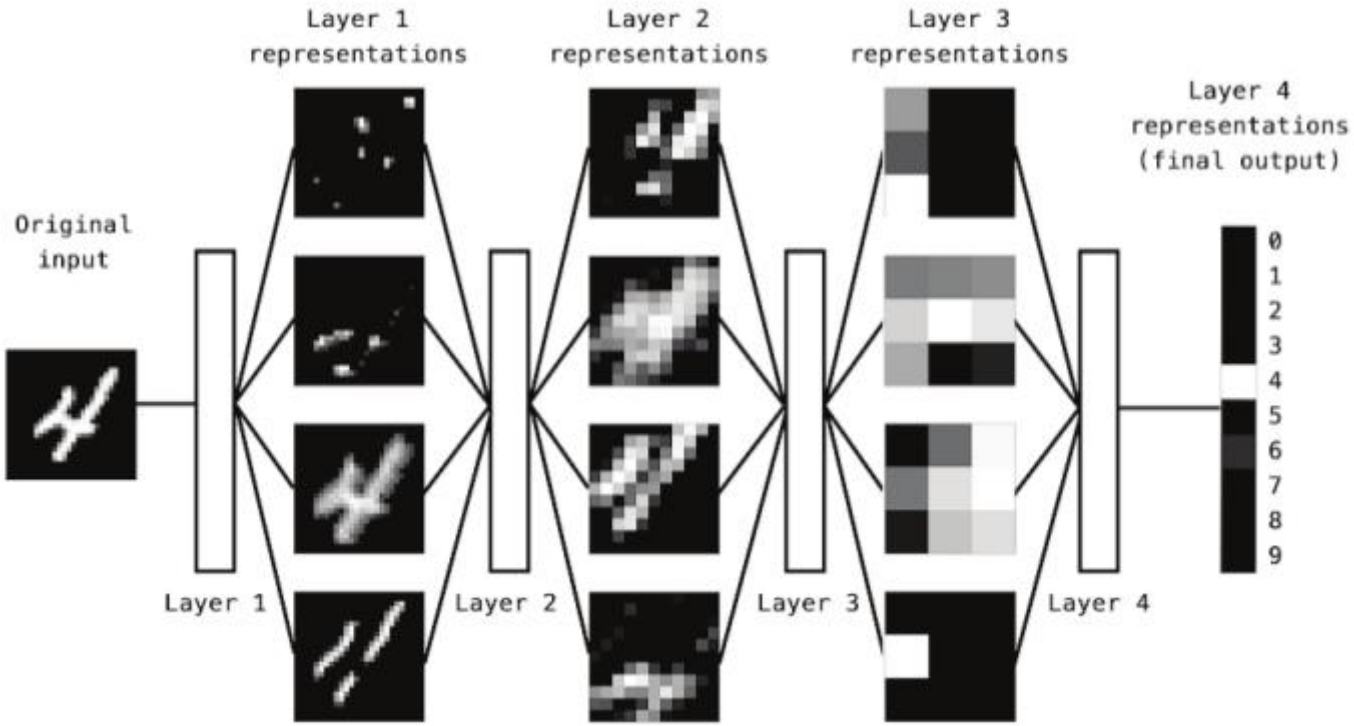
Part 3

An introduction to Convolutional Neural Networks

Layers in neural networks

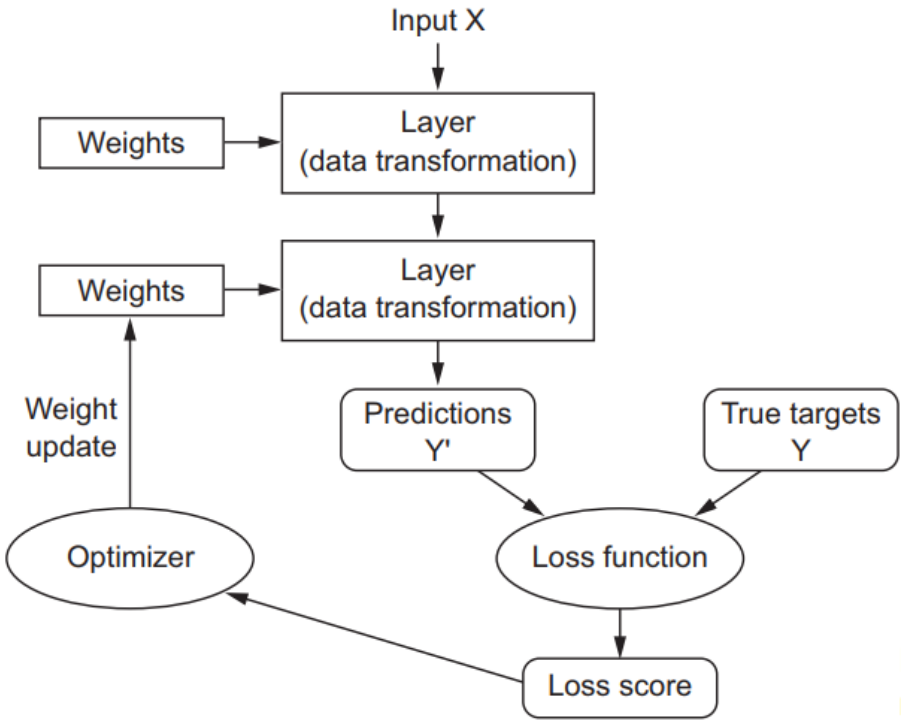


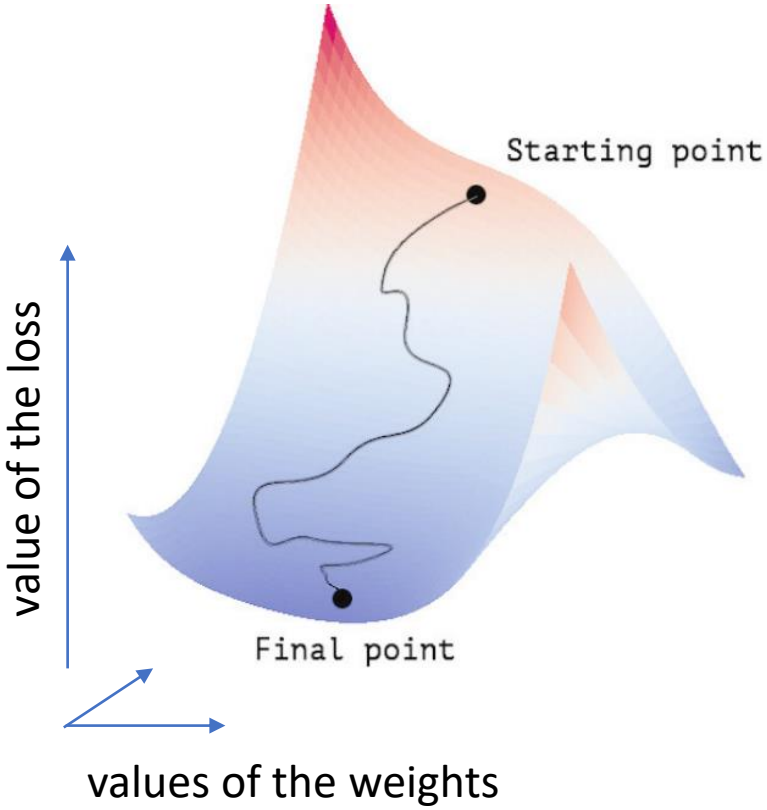
Successive layers learn intermediate representations of the image



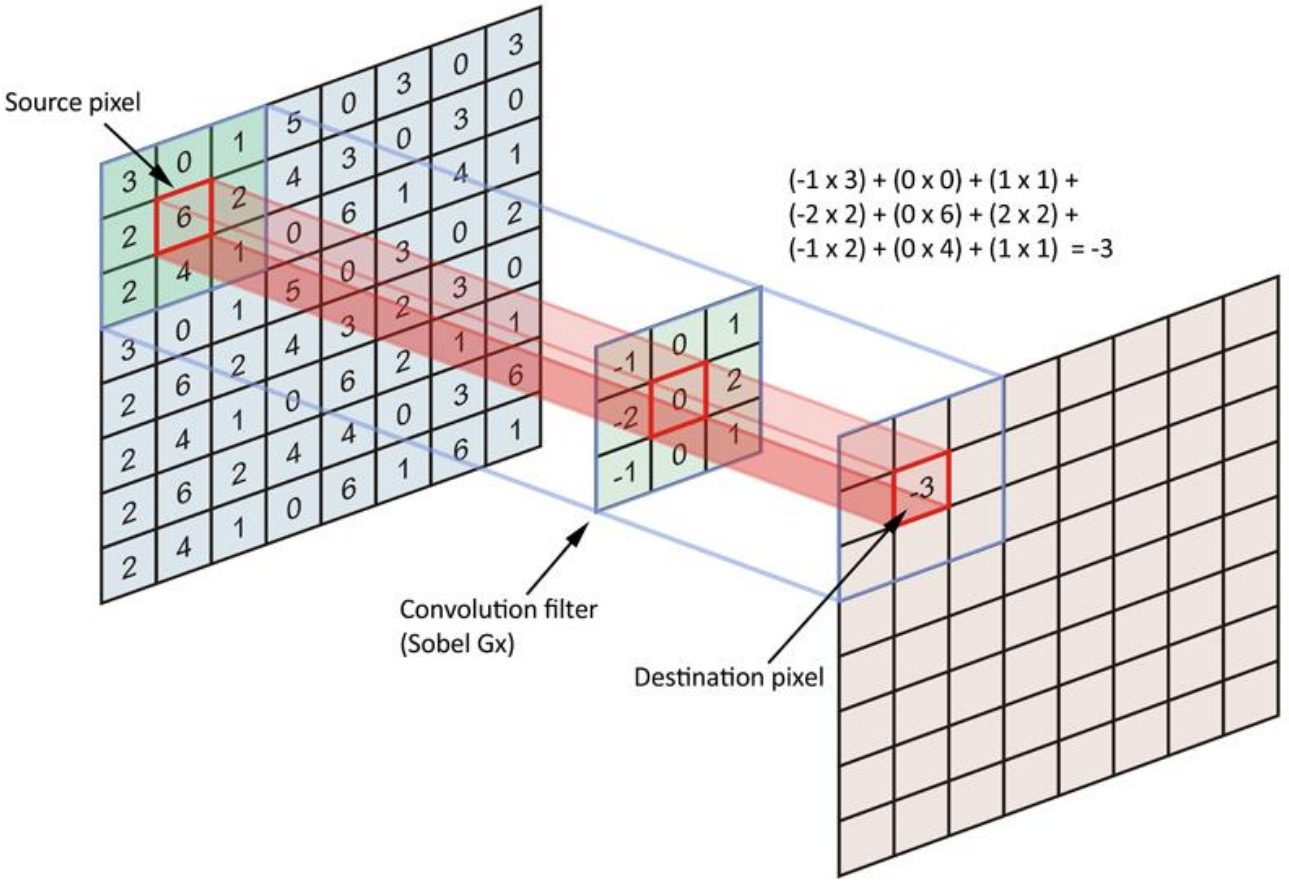
A neural network is a multi-stage information distillation operation

Training a neural network: finding a set of weights that solve the problem

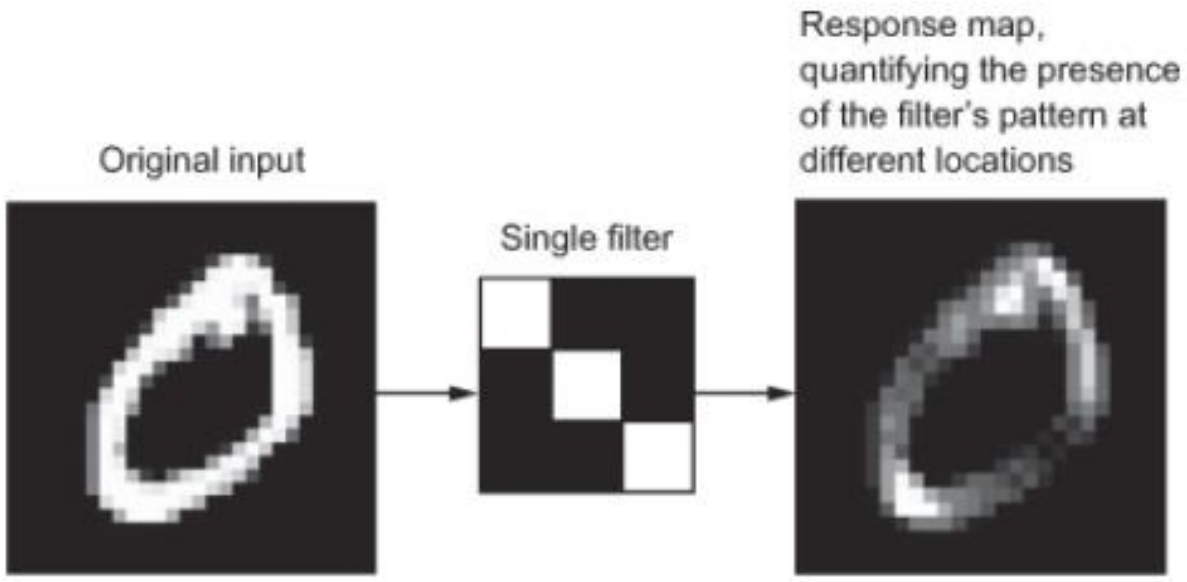




We already know what a convolution is

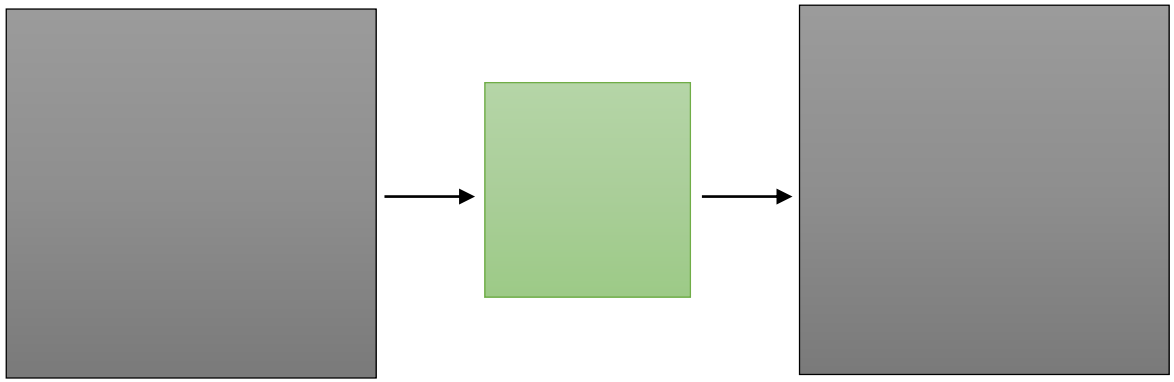


How does a convolution look like?



1 input map 1 3x3 kernel 1 output map

A basic convolutional layer



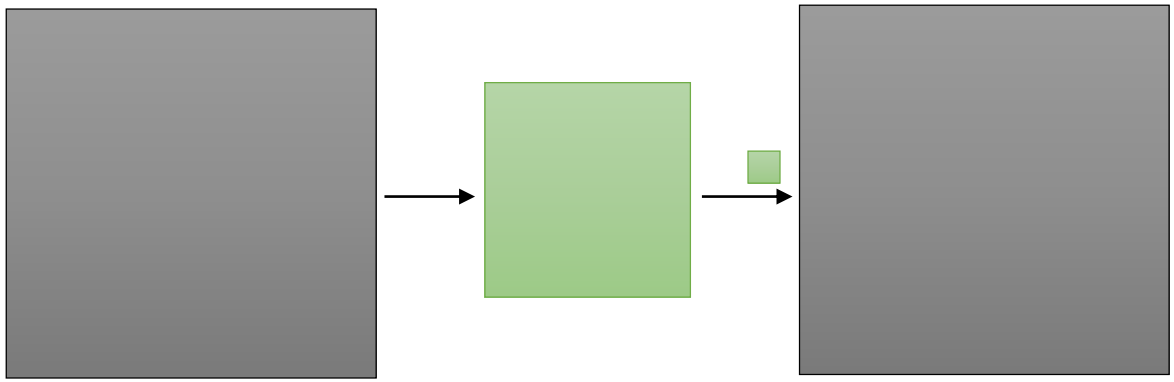
1 input map

1 3x3 kernel

1 output map

Output = input convolved with kernel

A basic convolutional layer



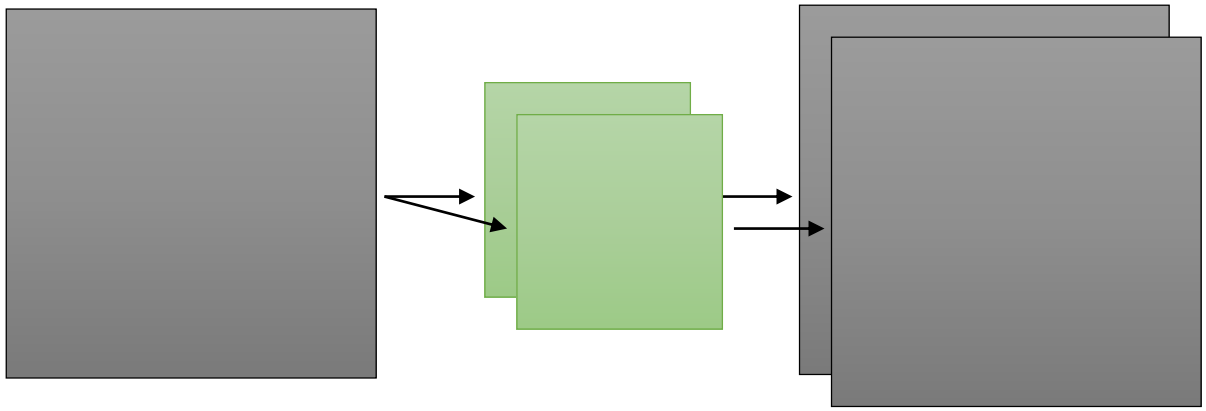
1 input map

1 3x3 kernel

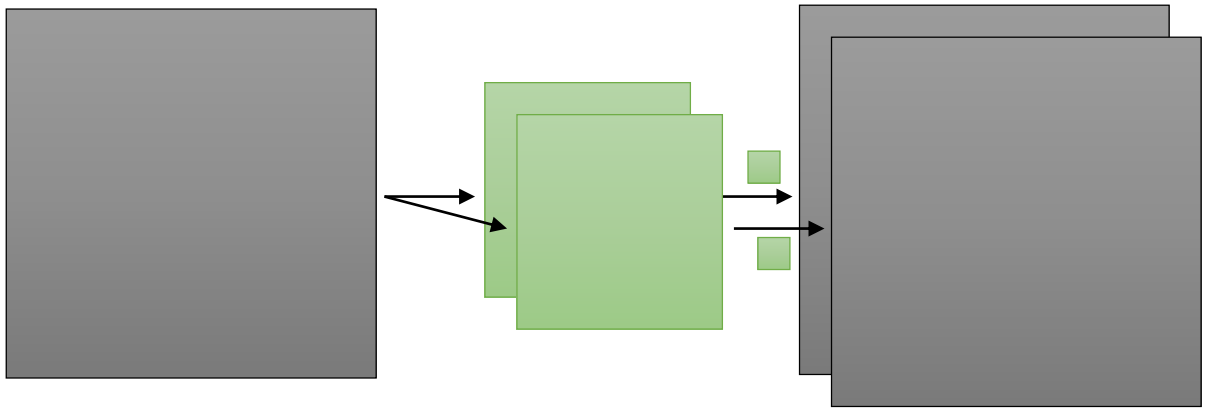
1 output map

1 bias parameter (optional)

Output = input convolved with kernel + bias



1 input map 2 3x3 kernels 2 output maps

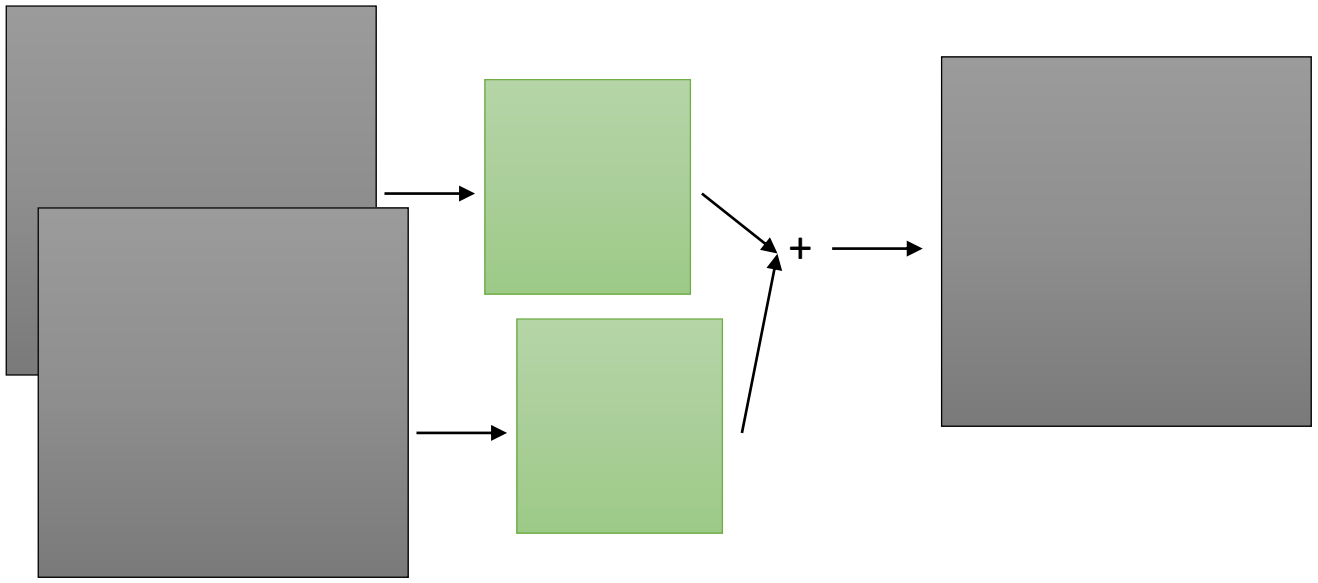


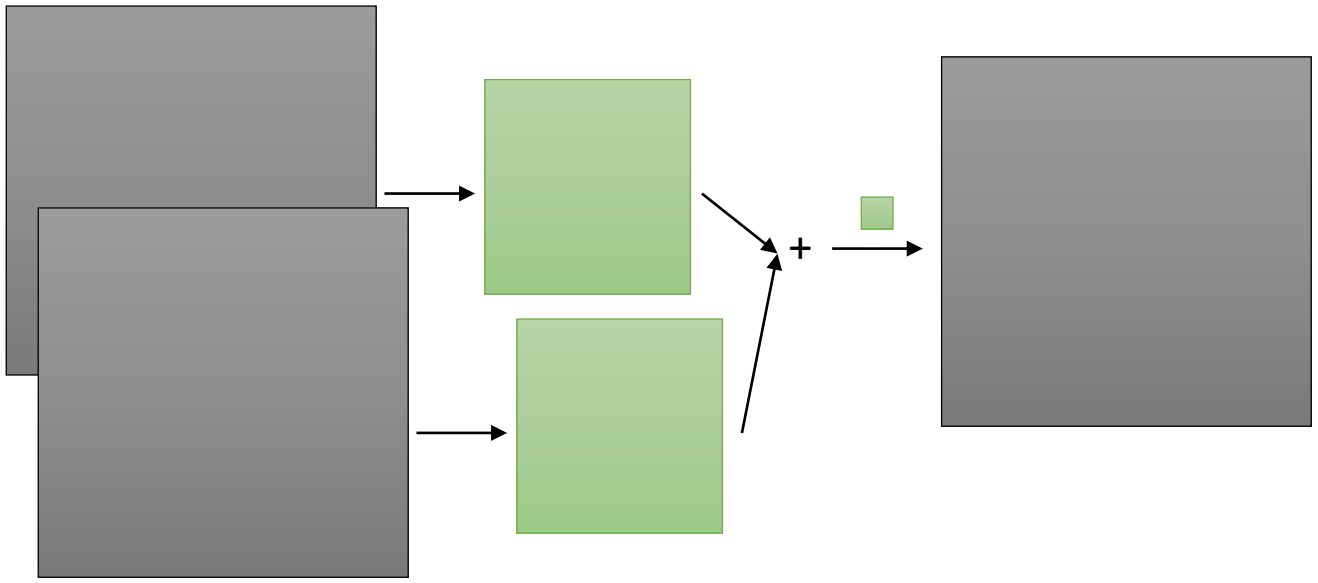
1 input map

2 3x3 kernels

2 output maps

2 bias parameters (optional)


+
2 input maps
2 3x3 kernels
1 output map

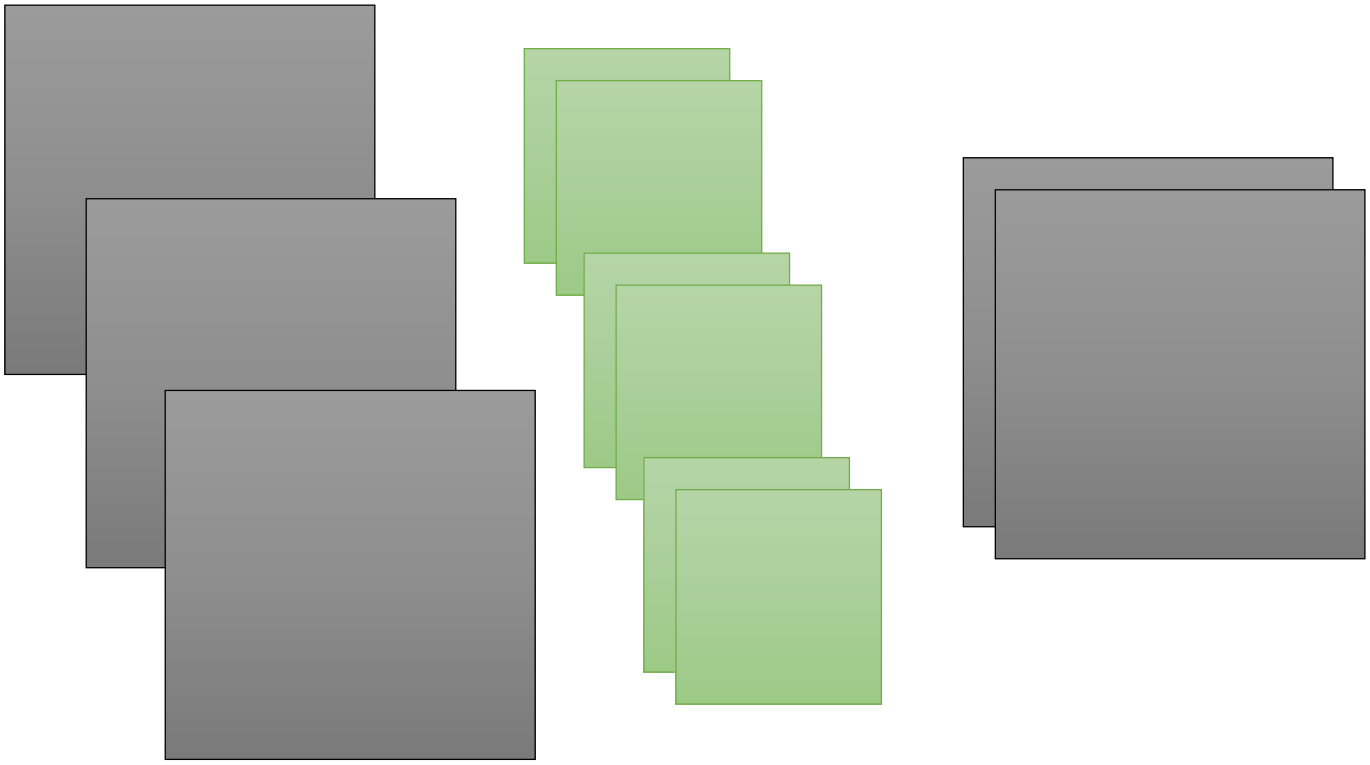


2 input maps

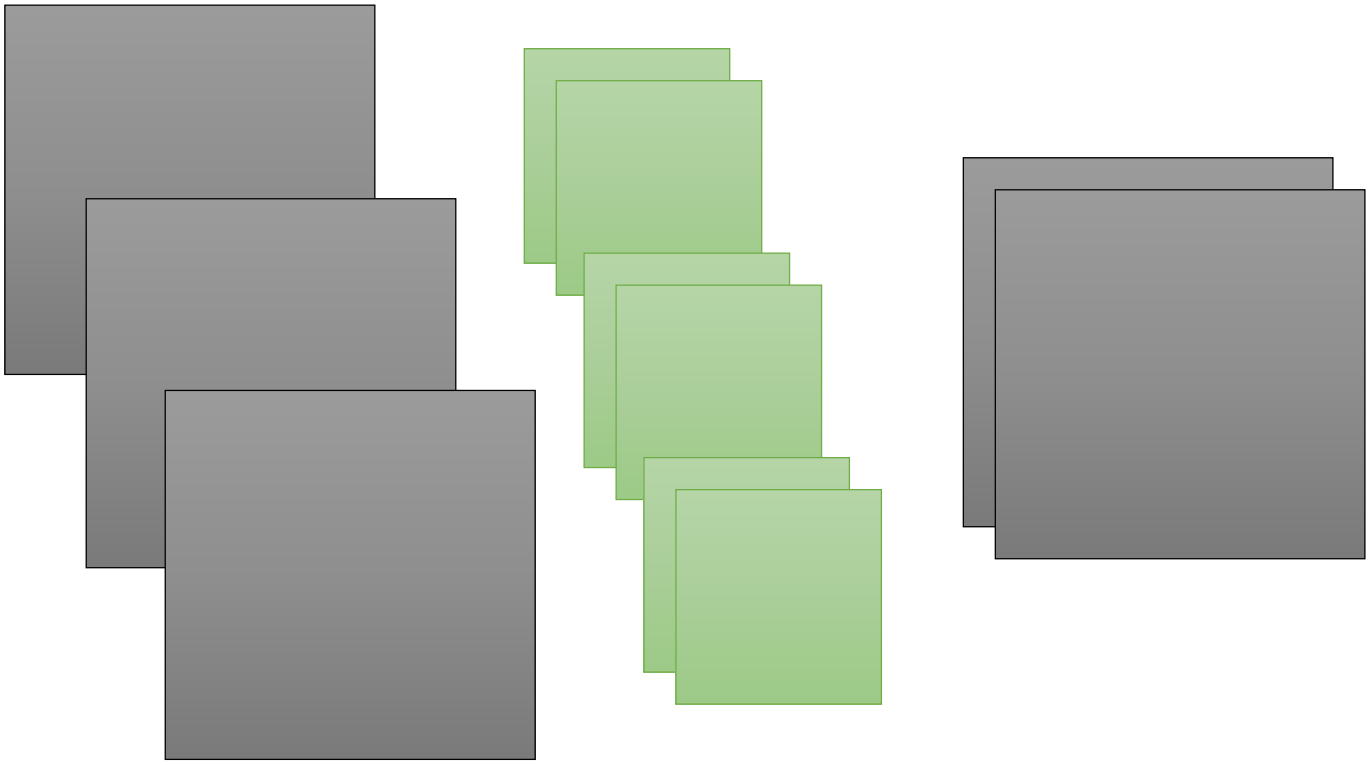
2 3x3 kernels

1 output map

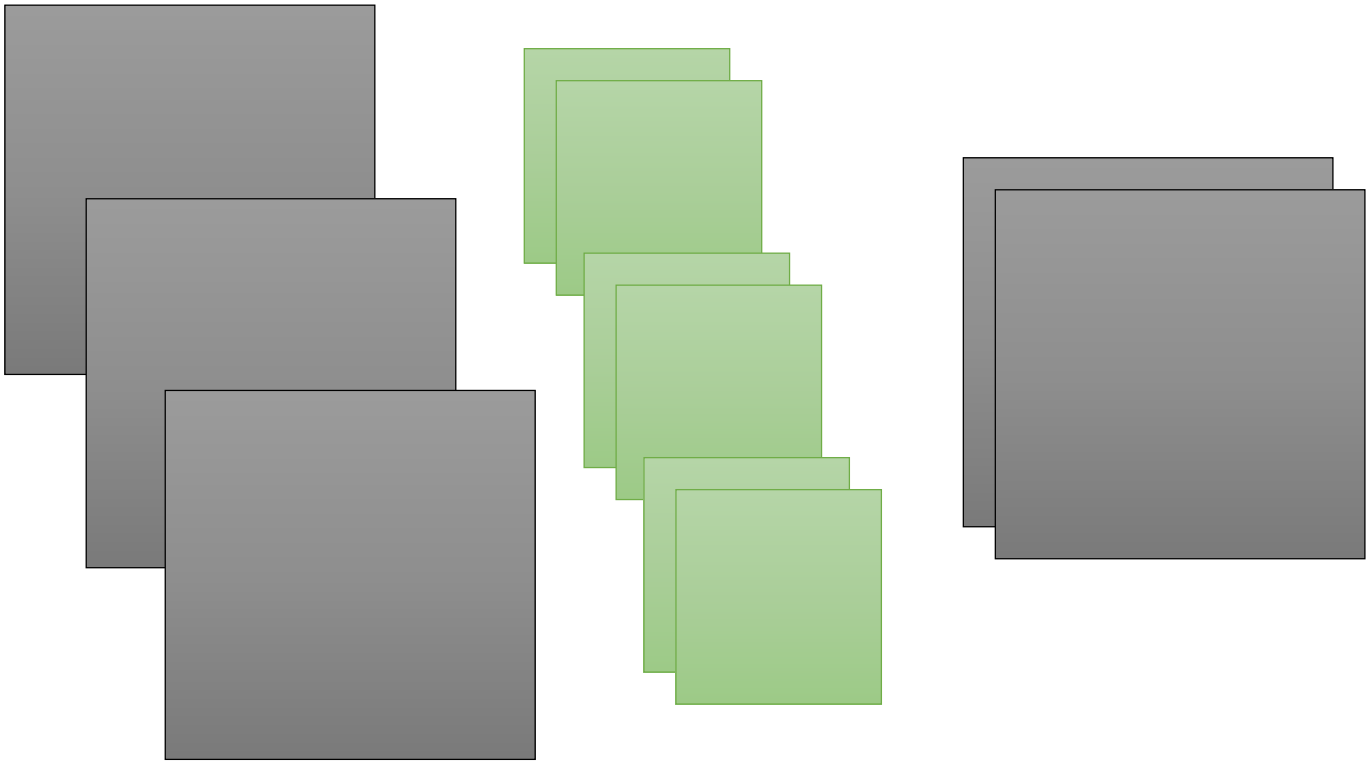
1 bias parameter (optional)



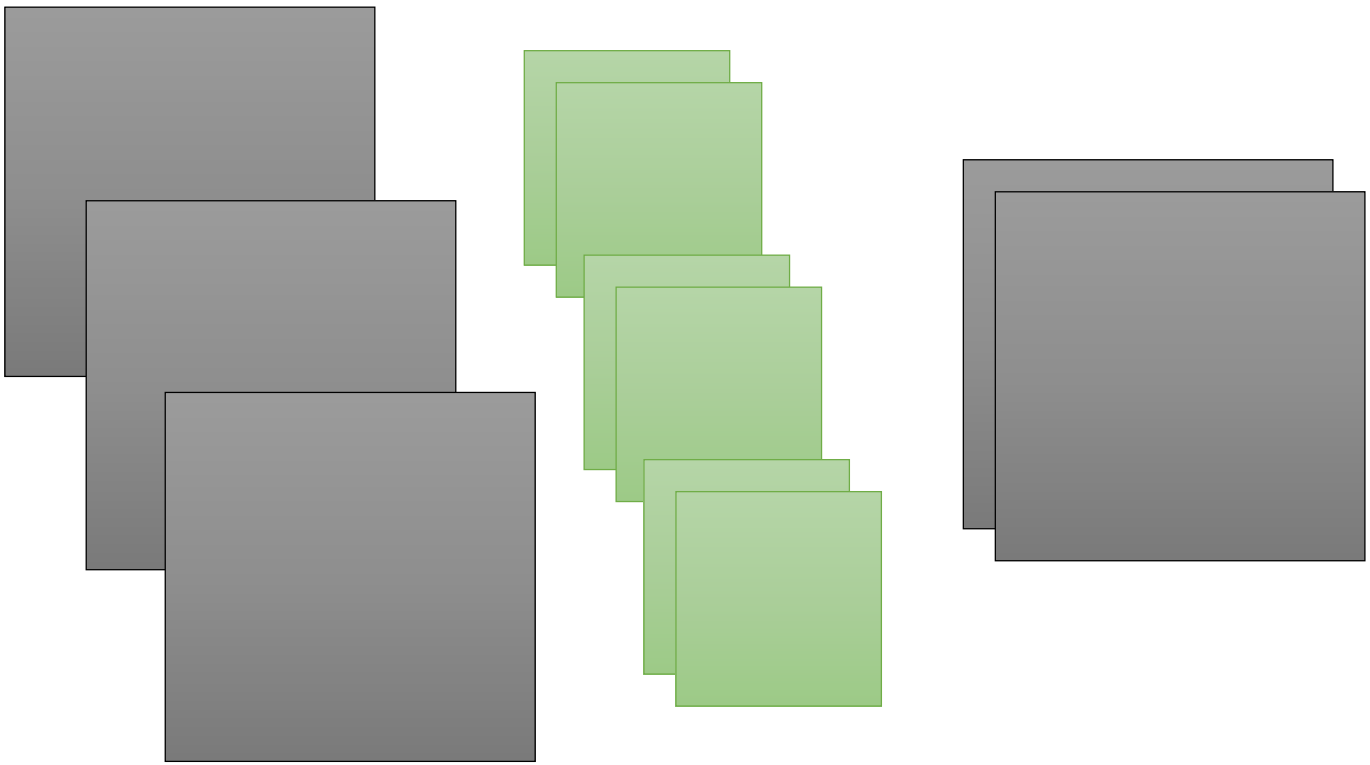
3 input maps 3x2 3x3 kernels 2 output maps



3 input maps 3x2 3x3 kernels 2 output maps

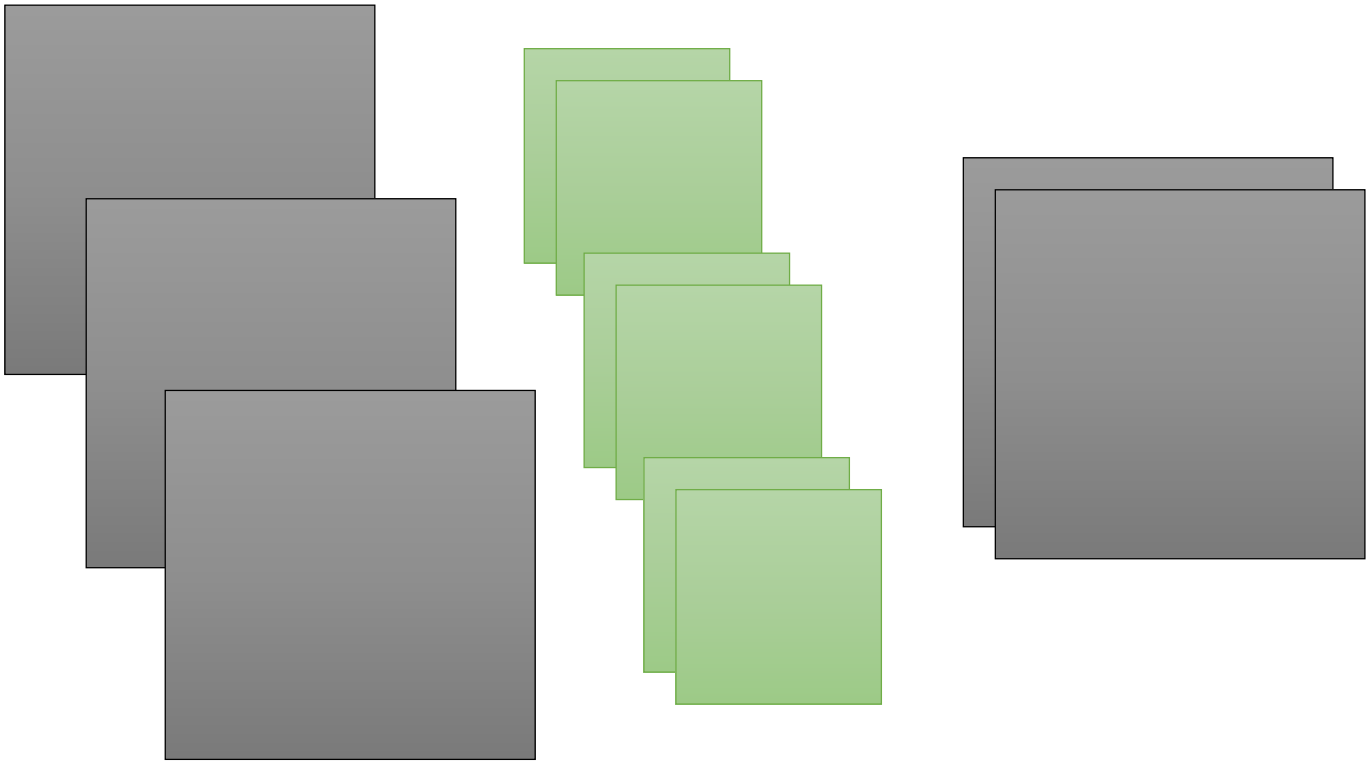


3 input maps 3x2 3x3 kernels 2 output maps



3 input maps 3x2 3x3 kernels 2 output maps

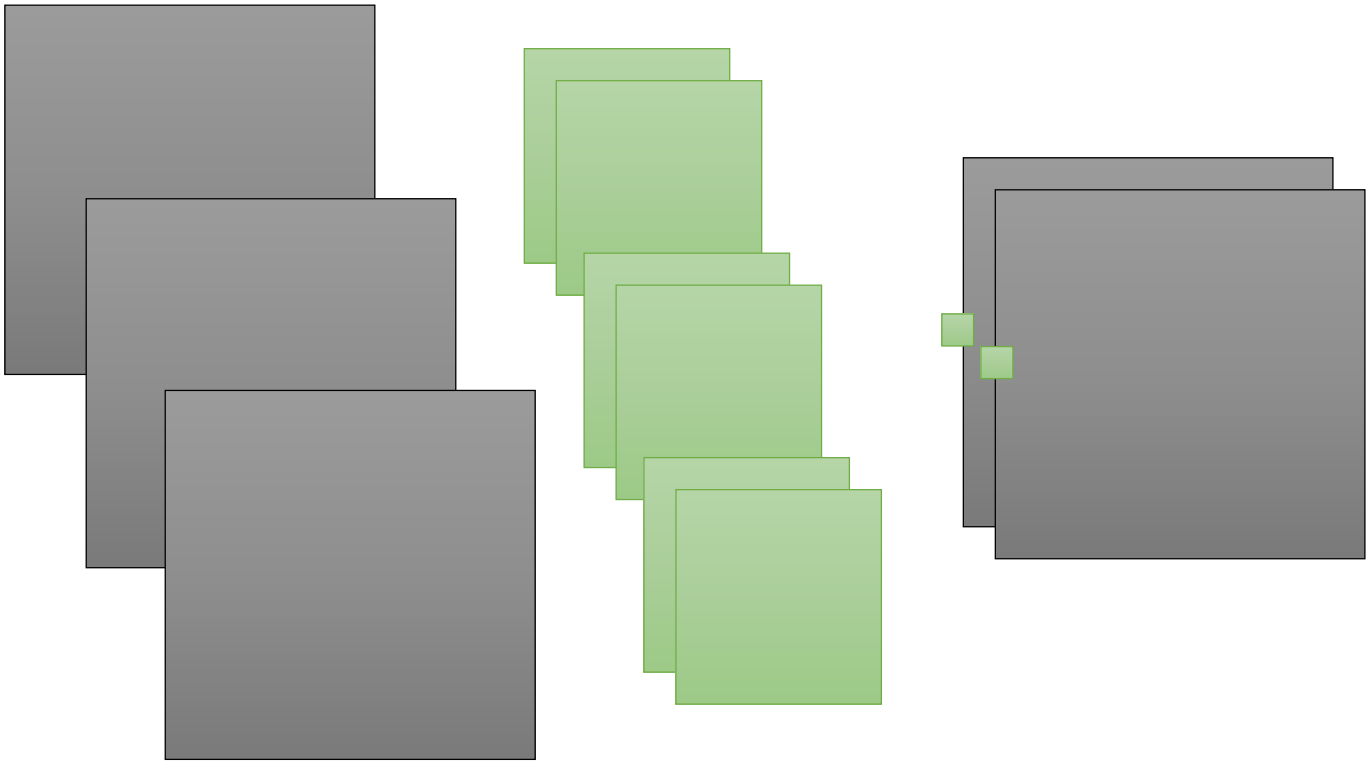
Quiz: how many parameters
does this layer have?



3 input maps

3x2 3x3 kernels
= 54 ...

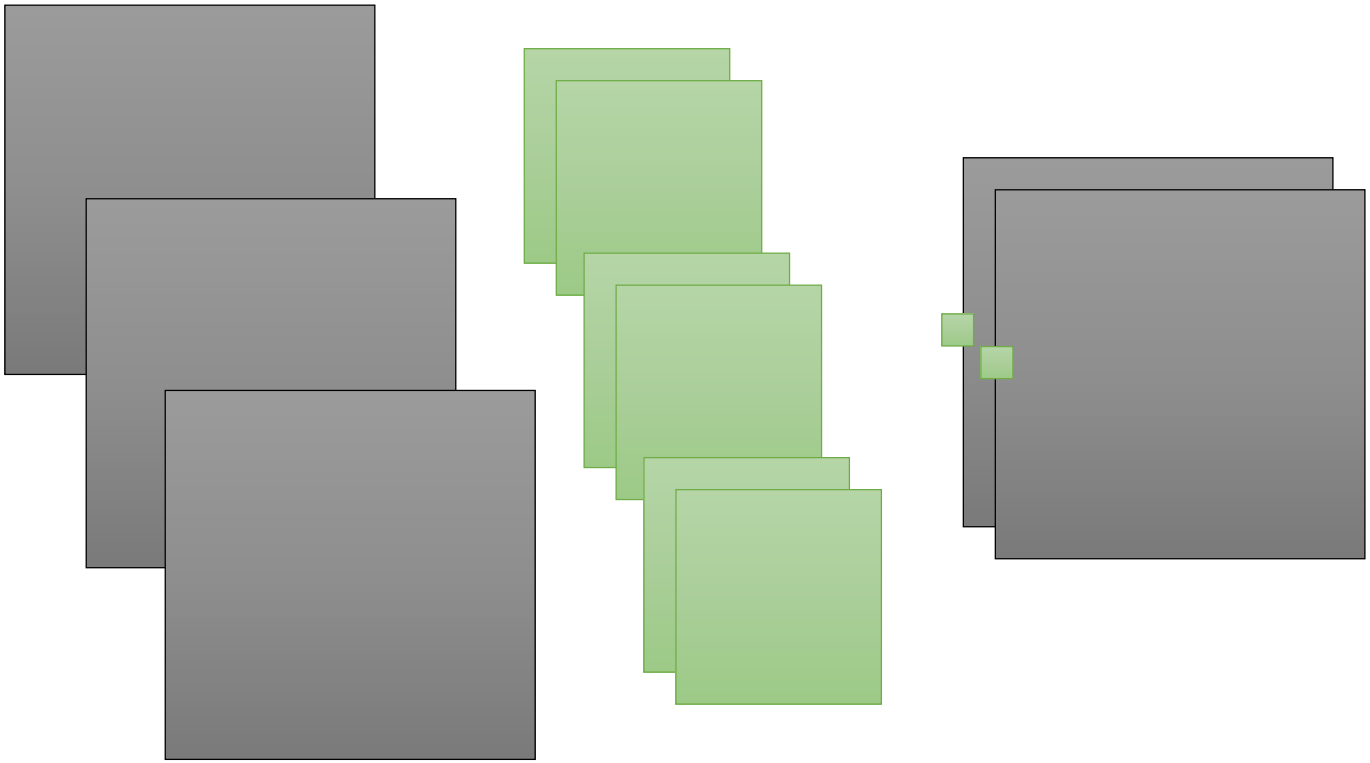
2 output maps



3 input maps

3x2 3x3 kernels
= 54 ...

2 output maps
+ 2 biases



3 input maps

3x2 3x3 kernels

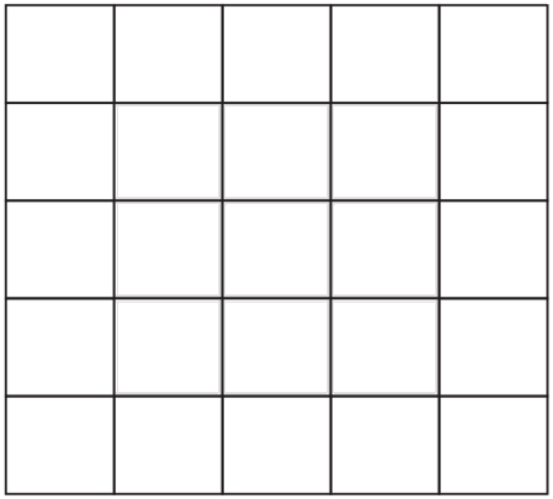
2 output maps

= 54 ...

+ 2 biases

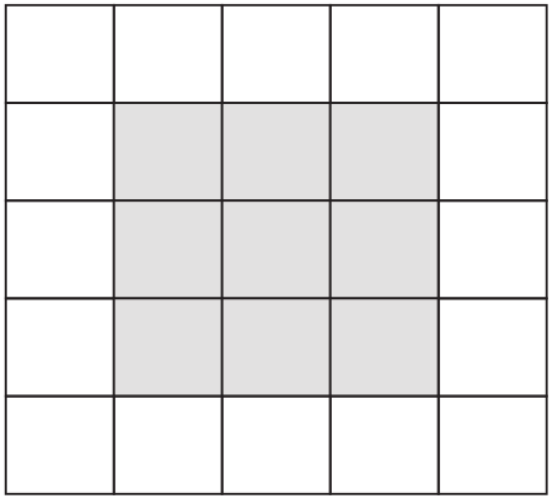
= 56 trainable parameters (weights)

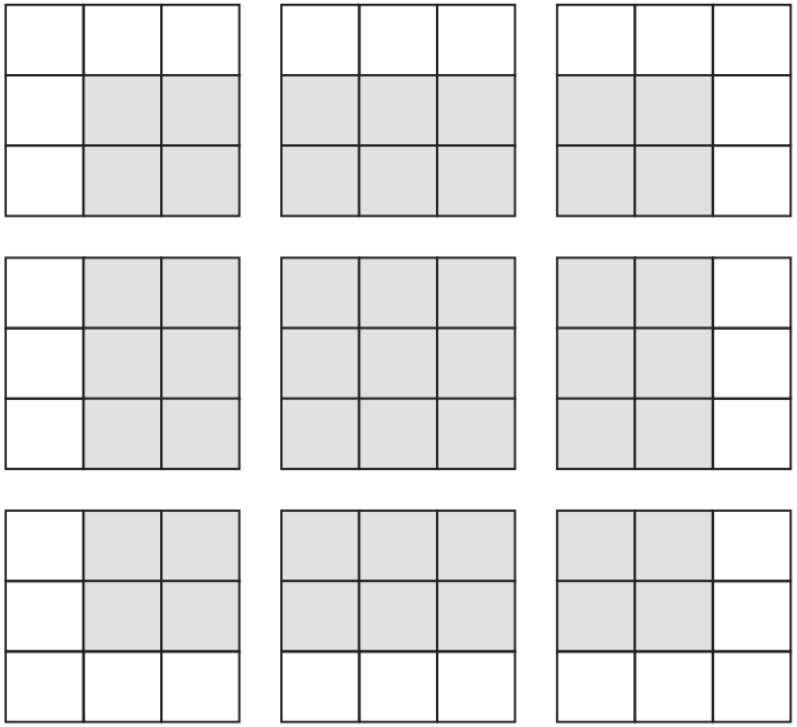
Details: padding



How many 3x3 patches are fully contained in a 5x5 map?

Details: padding

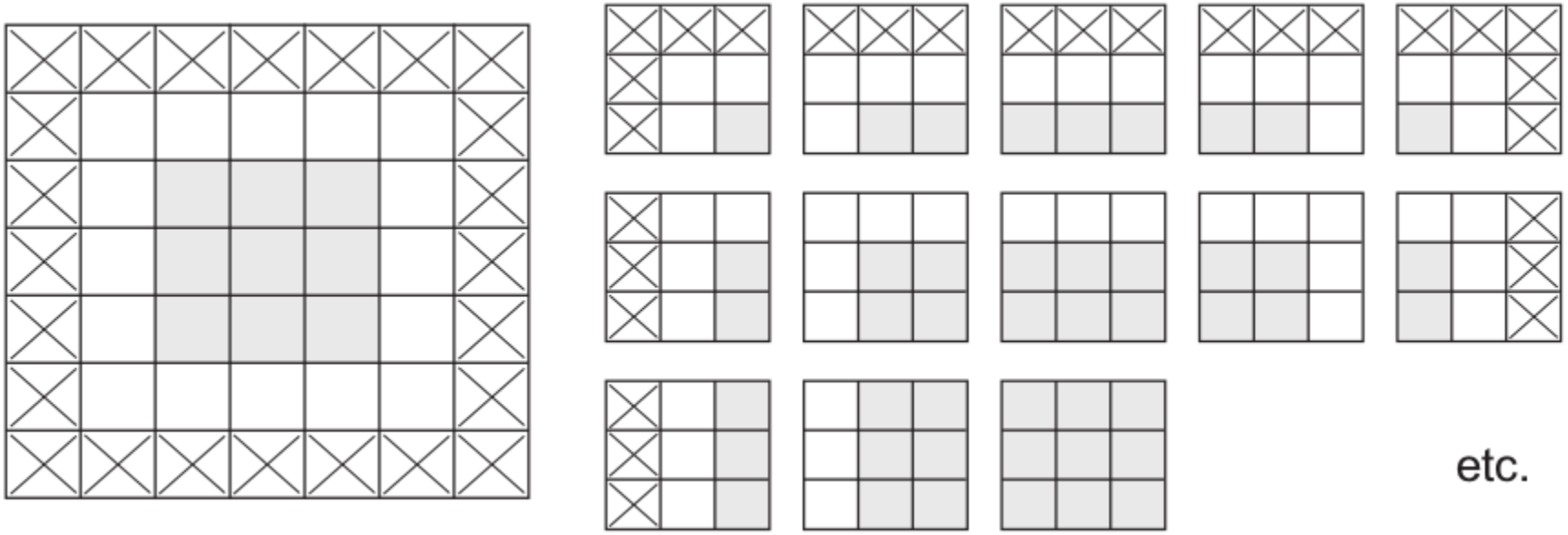




9: the output map is 3x3

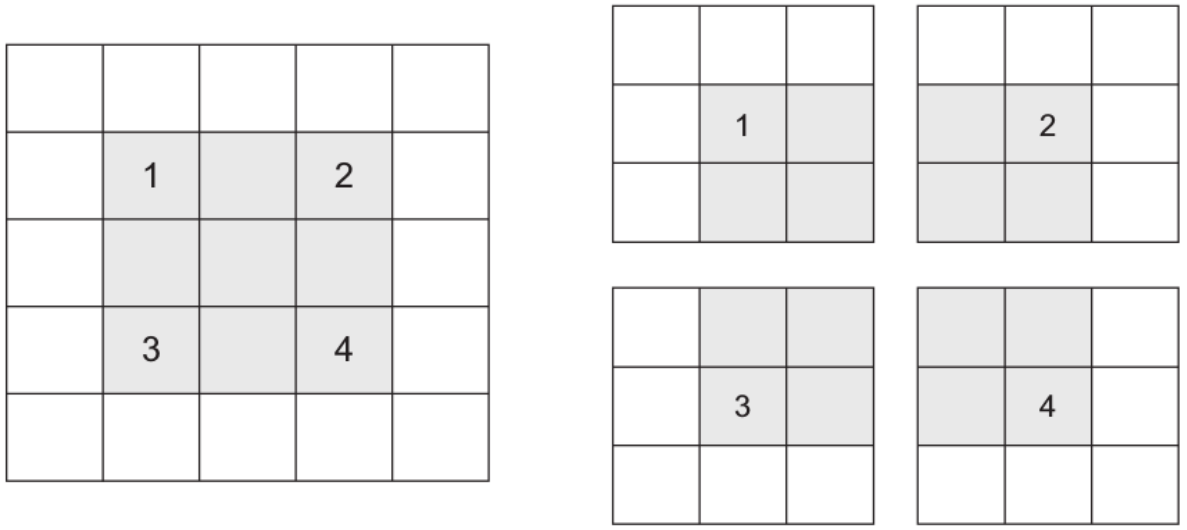
Details: padding

- This is known as “valid” padding mode (default)
- An alternative pads the input map with zeros to yield a same-sized map



Details: striding

- Stride 1x1 is most frequently used: shift 1 pixel at a time → patches are heavily overlapping
- Stride 2x2 skips one patch horizontally and vertically

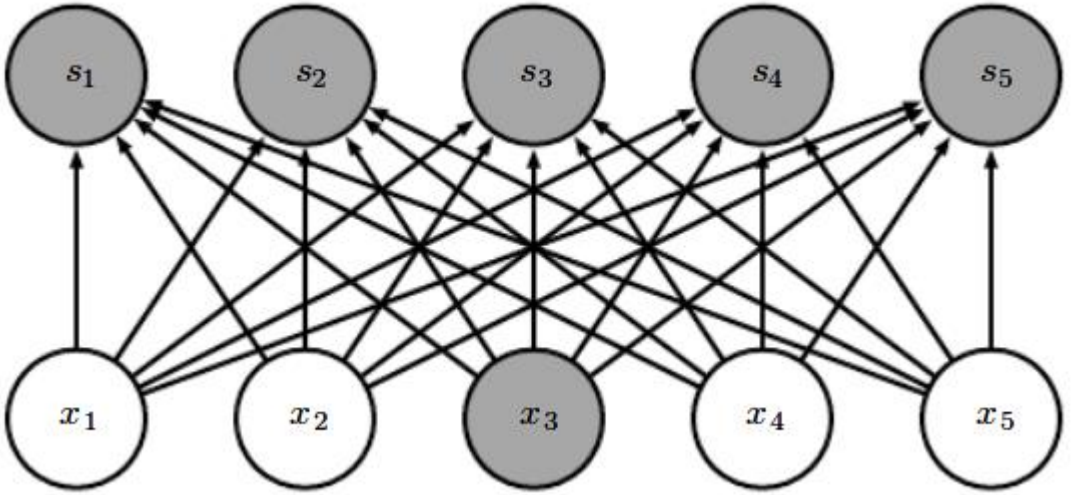


Why convolutional layers?

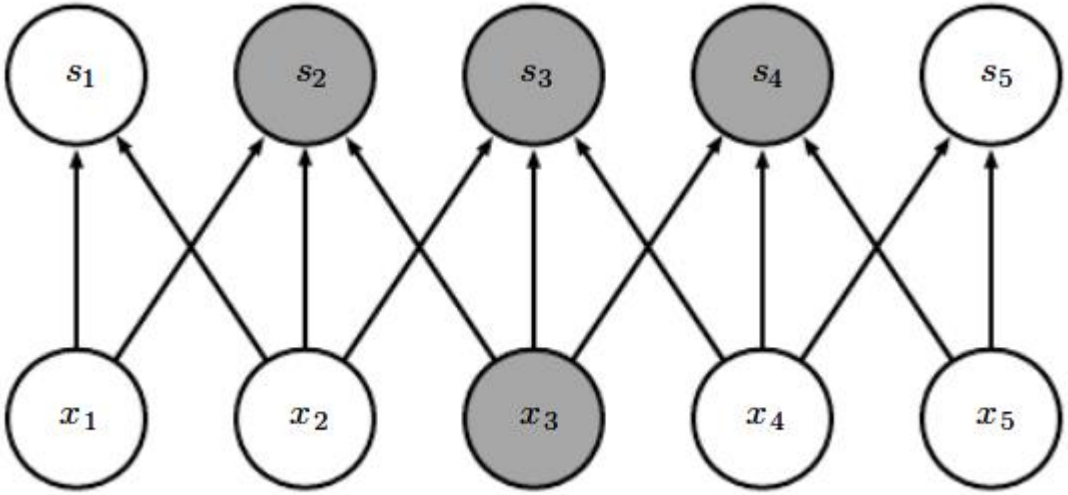
- Sparse connectivity
- Parameter sharing
- Translation invariance

Sparse connectivity

Fully connected

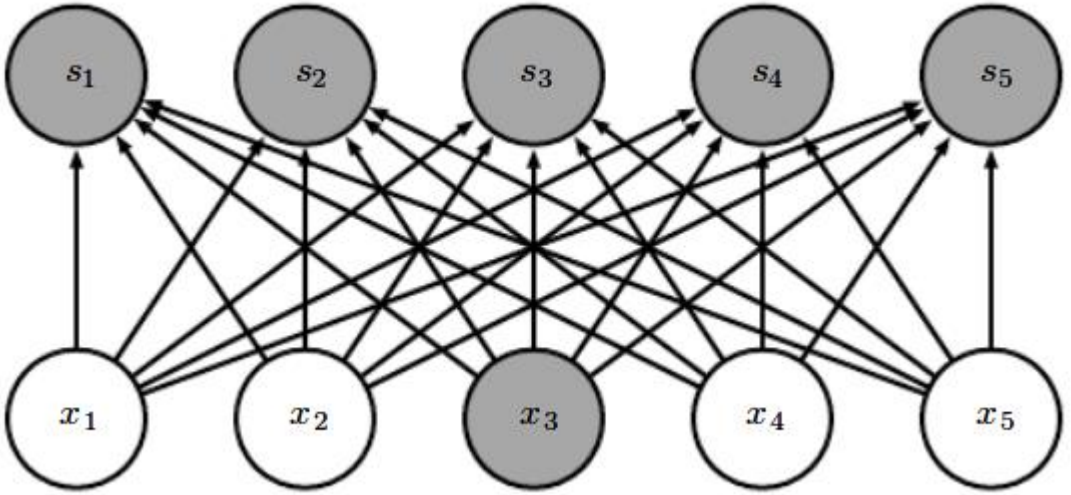


3x1 convolutional

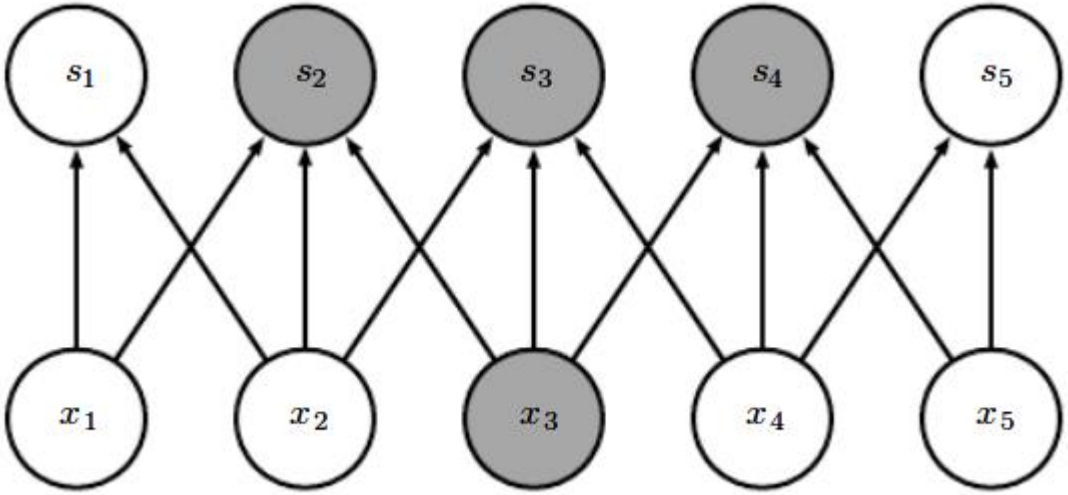


Sparse connectivity

Fully connected

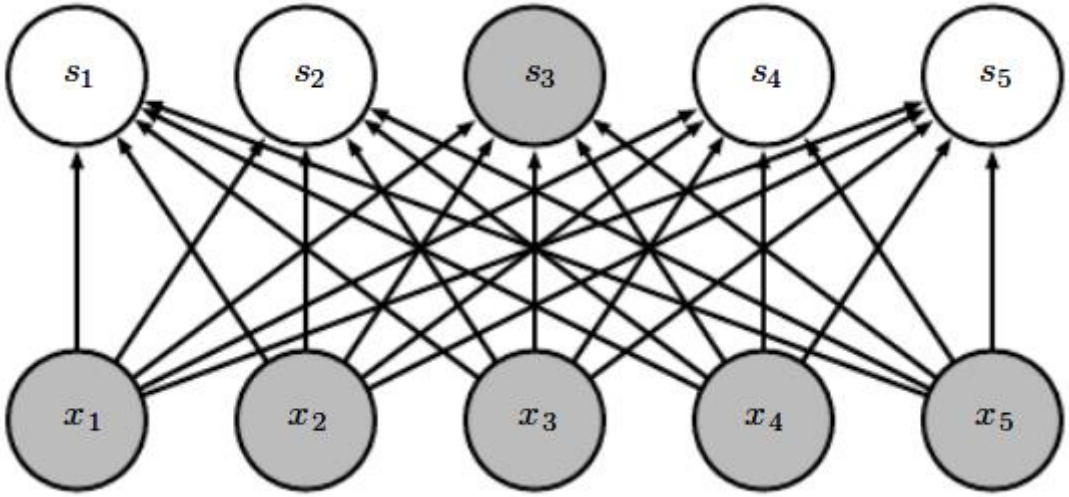


3x1 convolutional

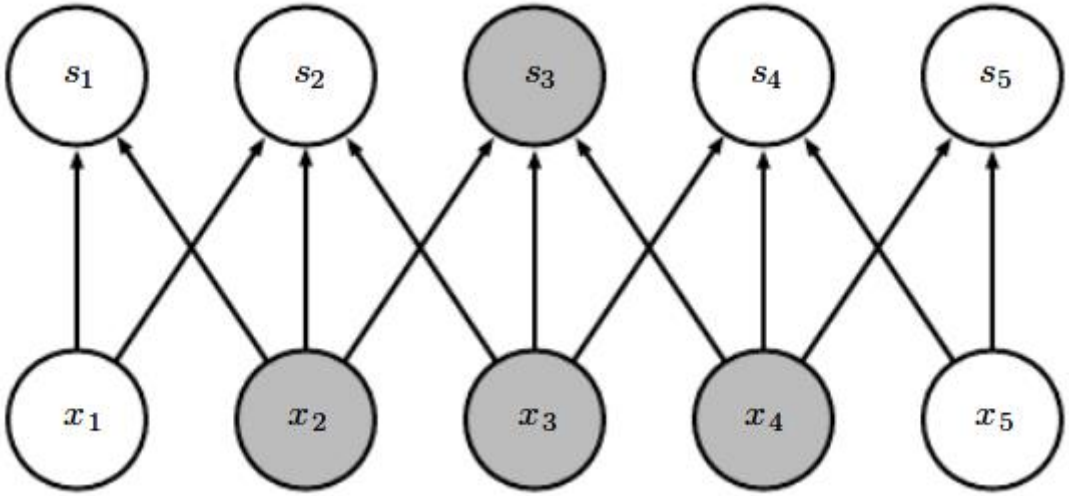


Receptive fields

Fully connected



3x1 convolutional

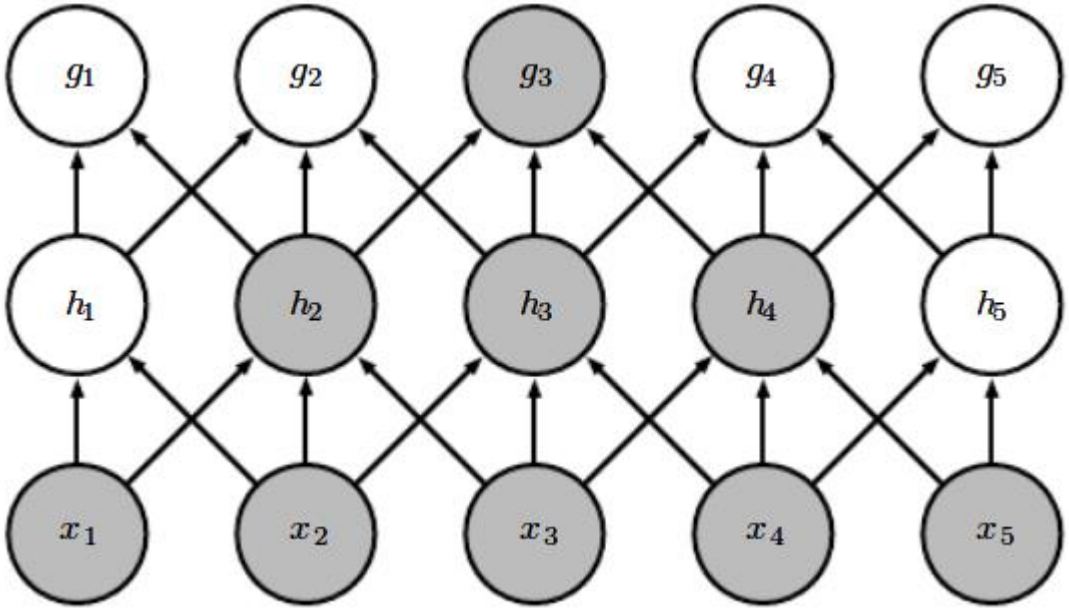


Receptive fields

Deeper neurons depend on wider patches of the input

3x1 convolutional

3x1 convolutional

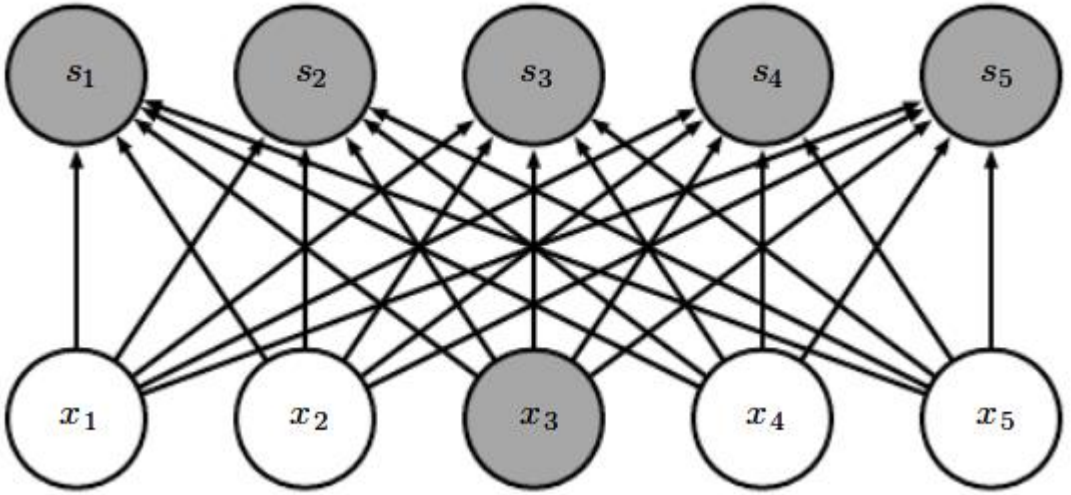


Parameter sharing

Fully connected

5x5 = 25 weights

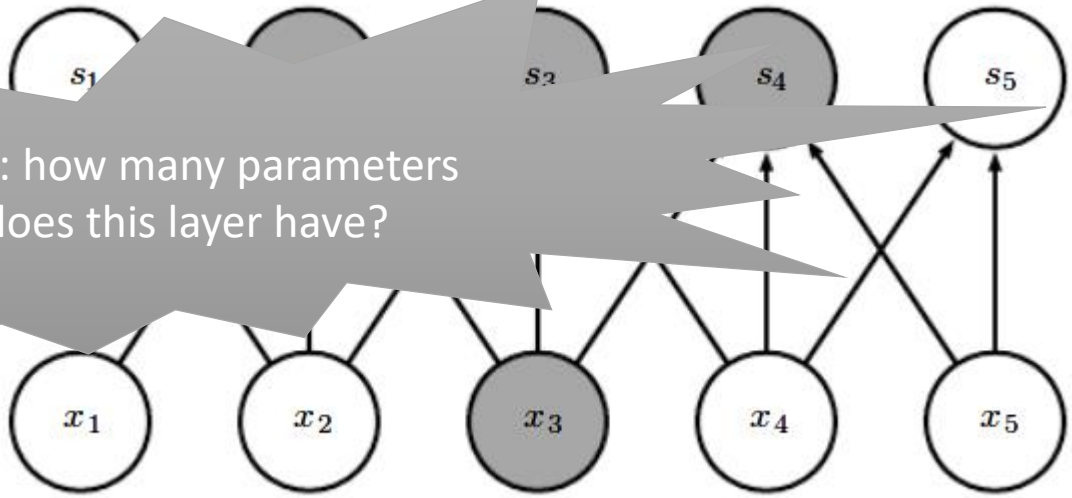
(+ 5 bias)



3x1 convol

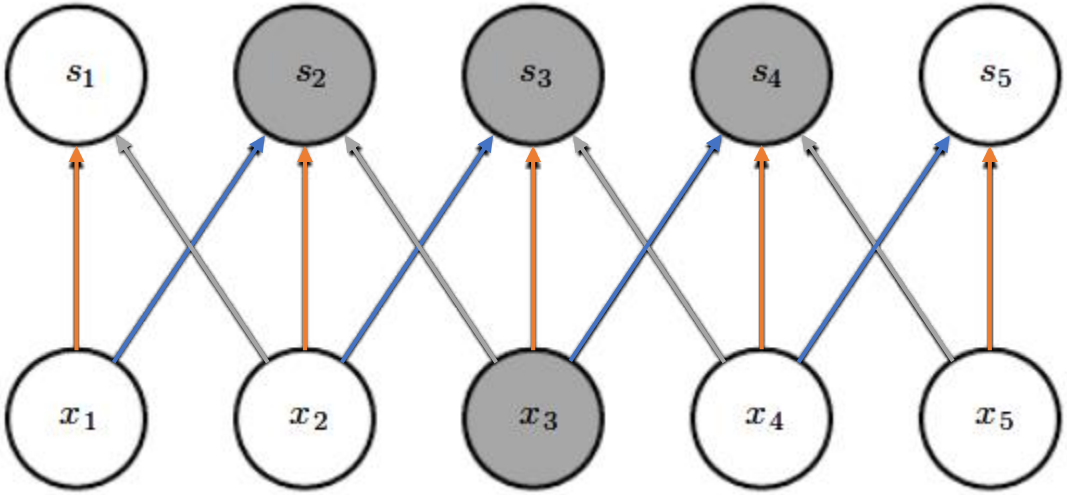
3 weights!

(+ 1 bias)



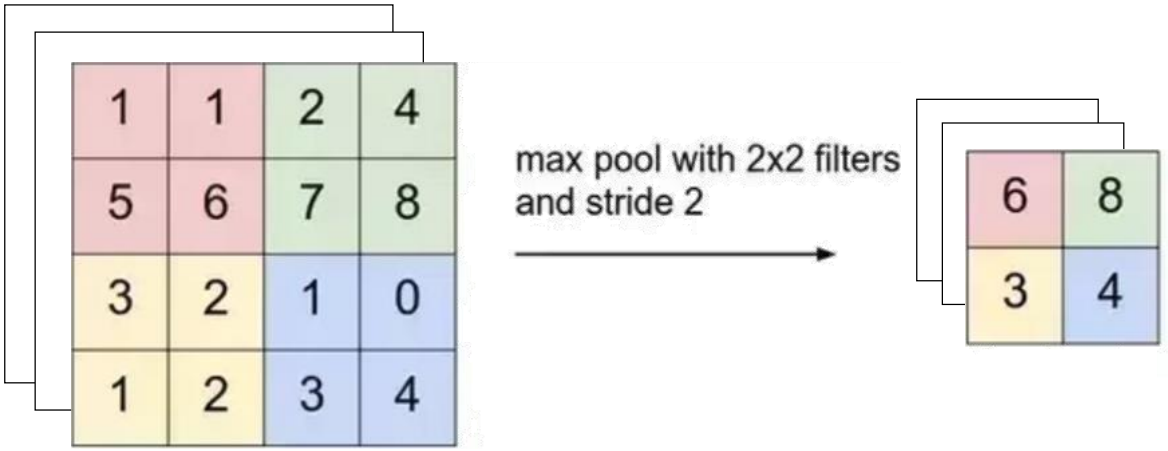
Quiz: how many parameters does this layer have?

Translational invariance



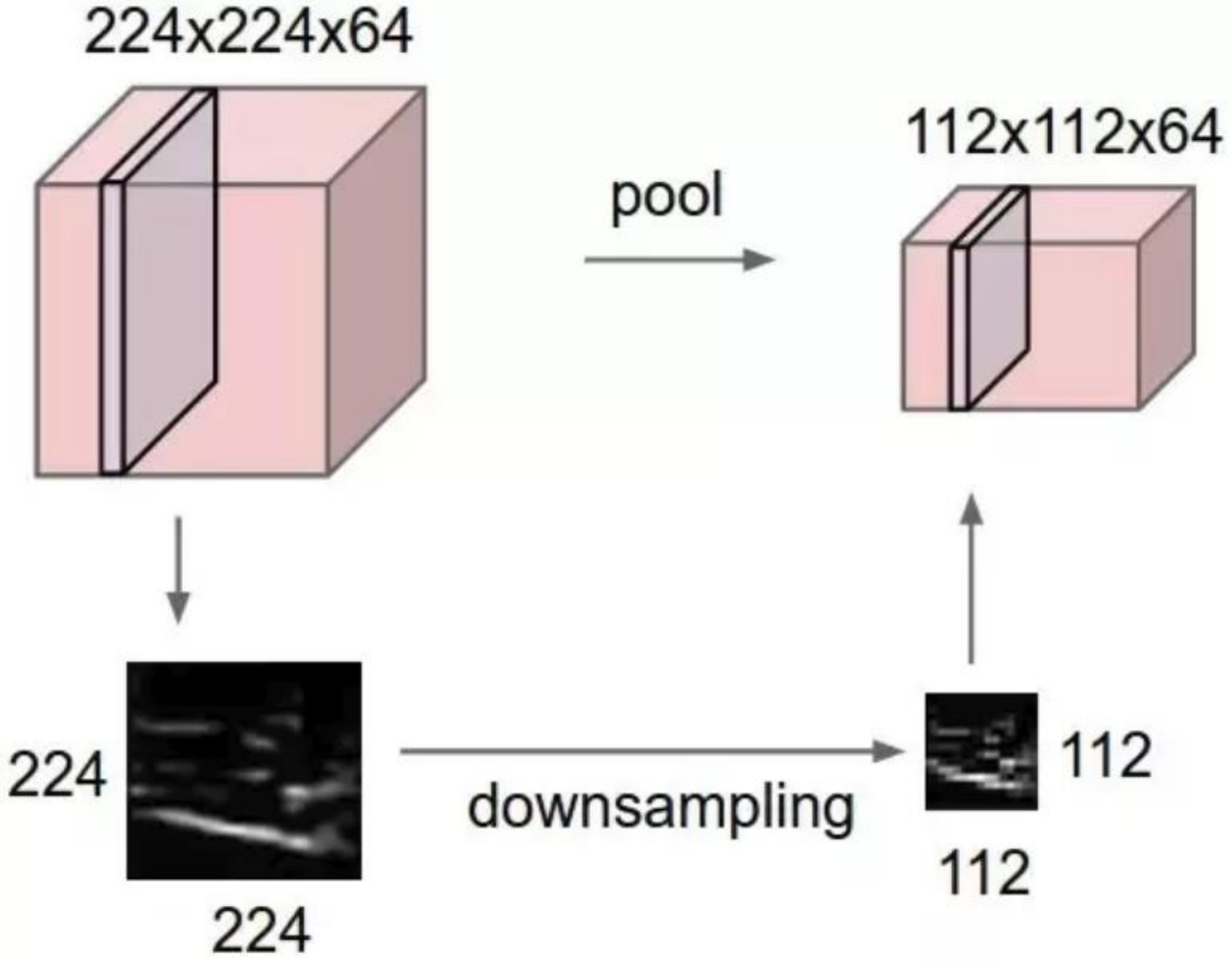
Max pooling layers

... on many maps?



Quiz: how many parameters does this layer have?

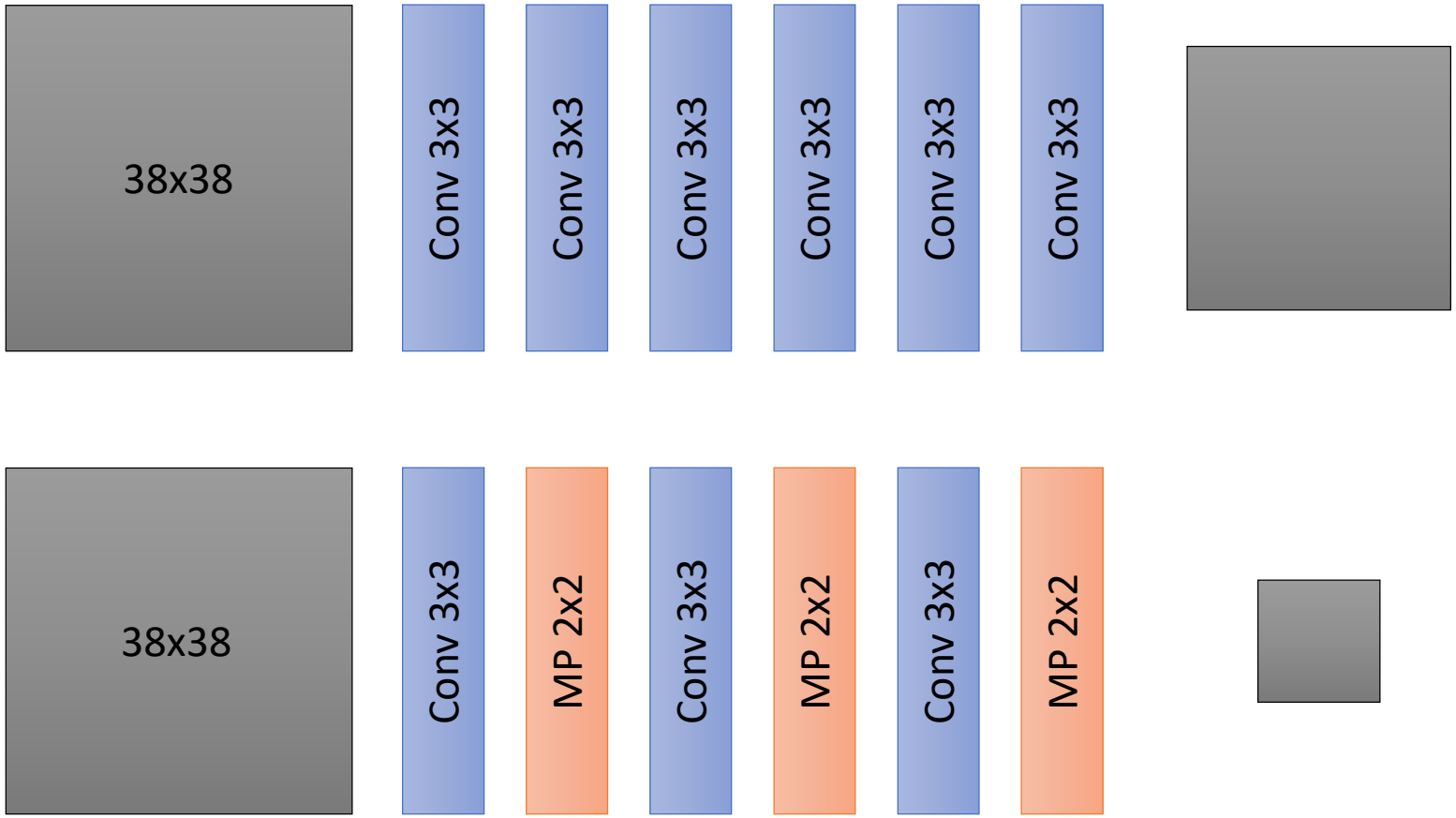
Max pooling downsamples activation maps



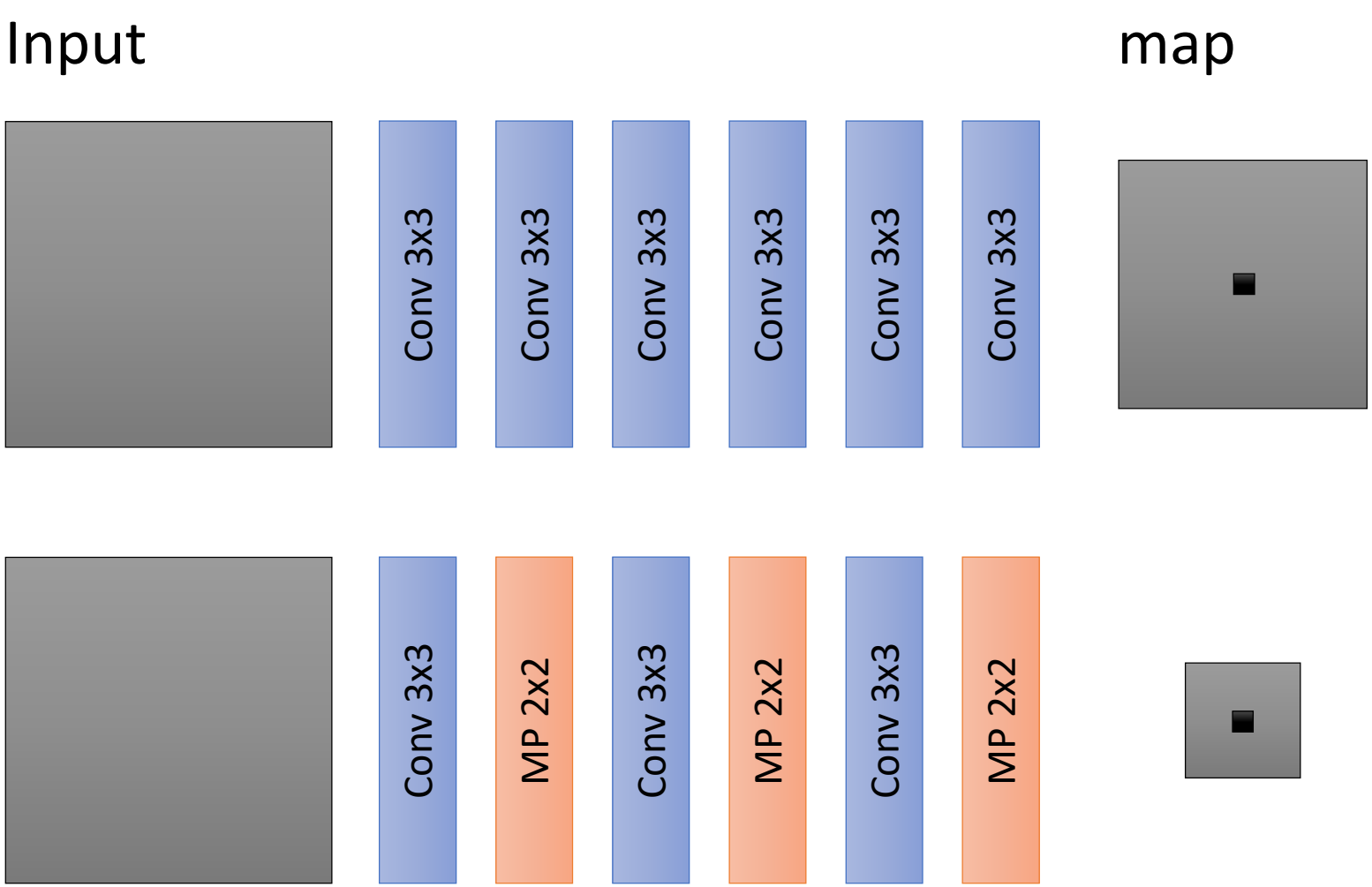
Exercise

Input:

map

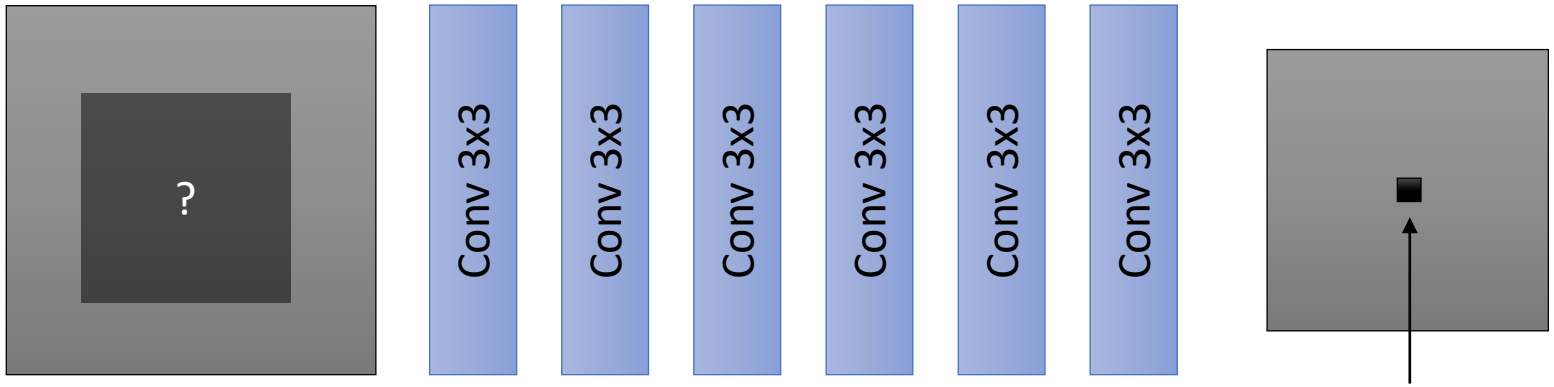


Receptive fields

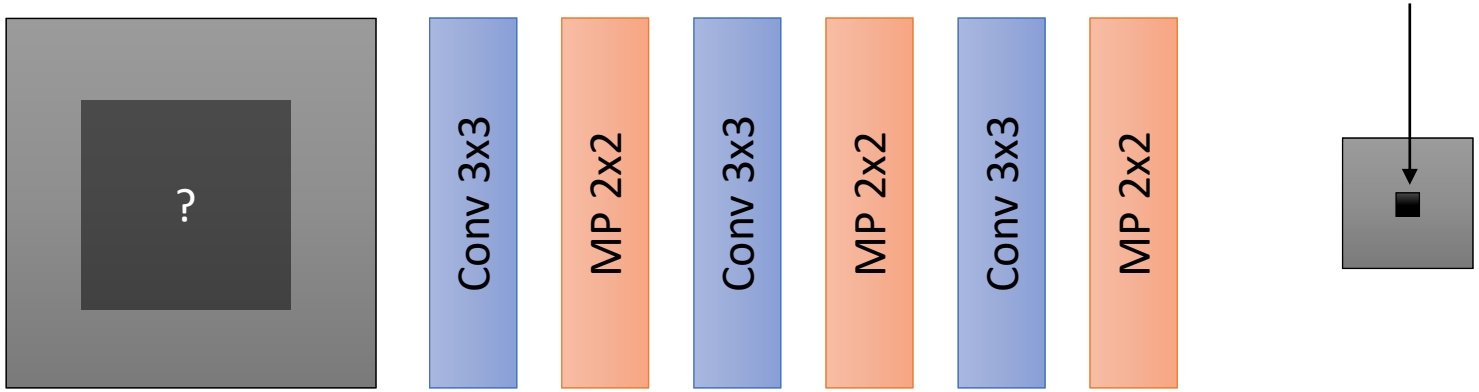


Receptive fields

Input map

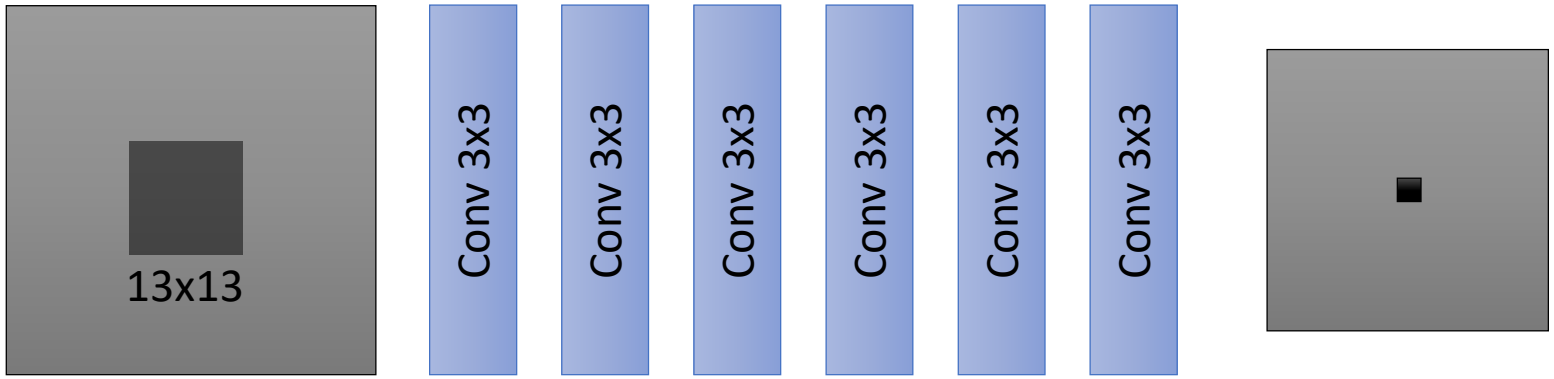


How large is the receptive field of the black neuron?

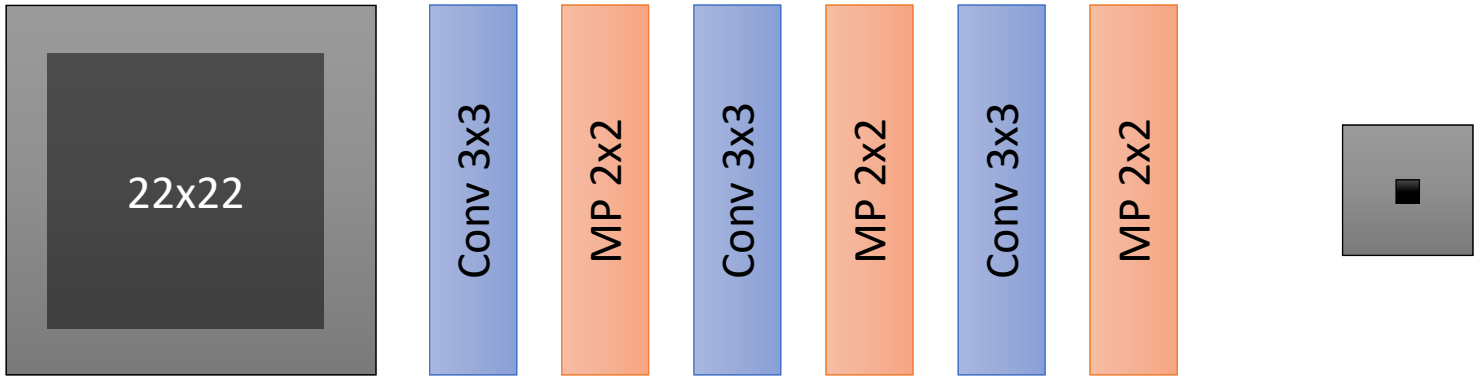


Receptive fields

Input map

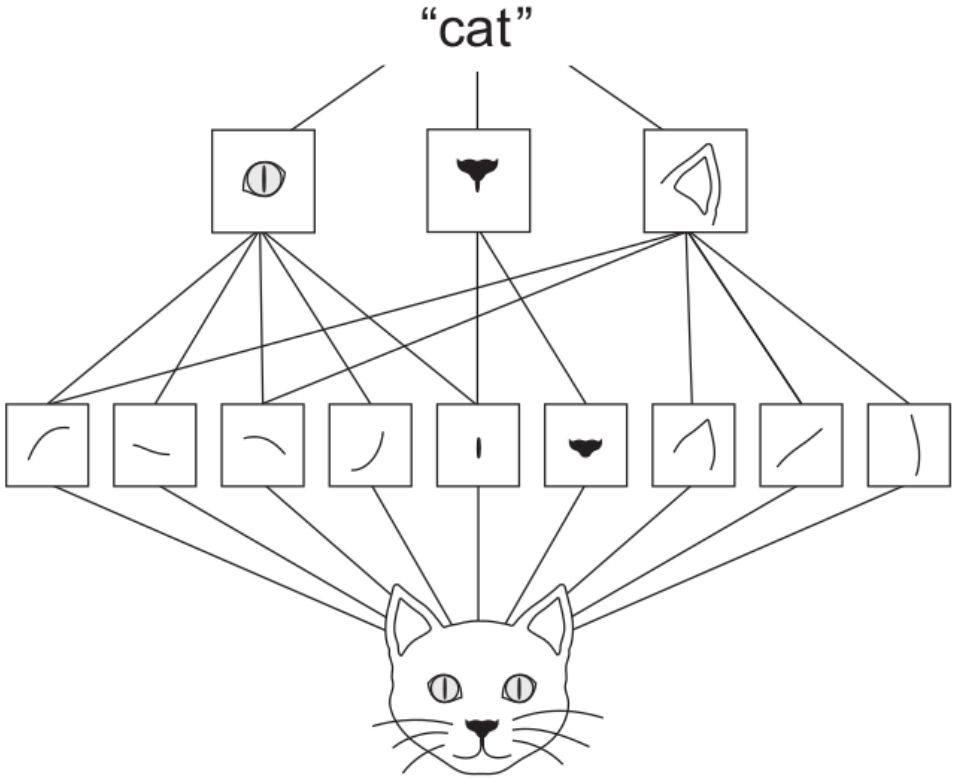


How large is the receptive field of the black neuron?



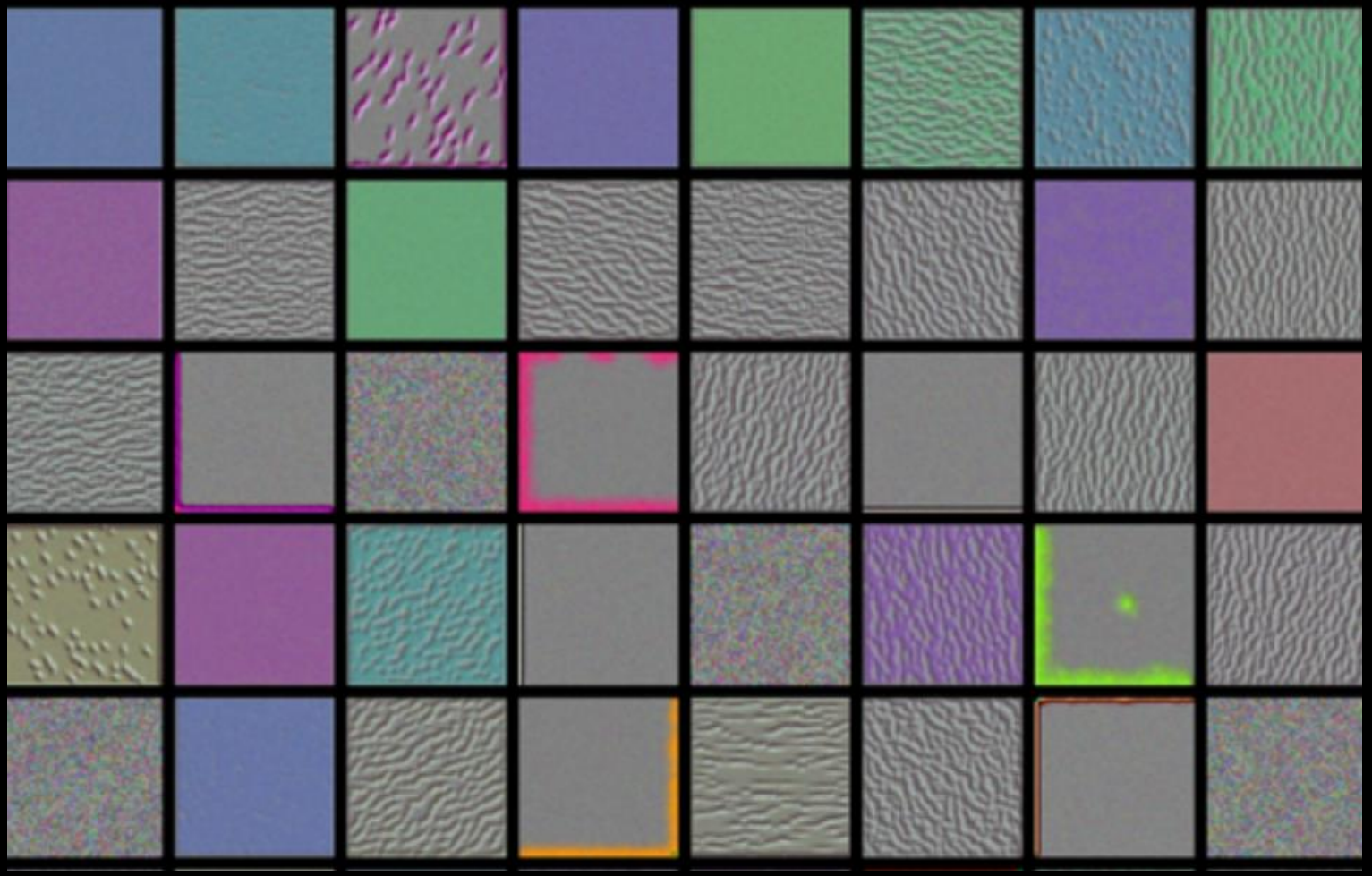
Why convnets work

Convnets learn a **hierarchy** of **translation-invariant** spatial pattern detectors

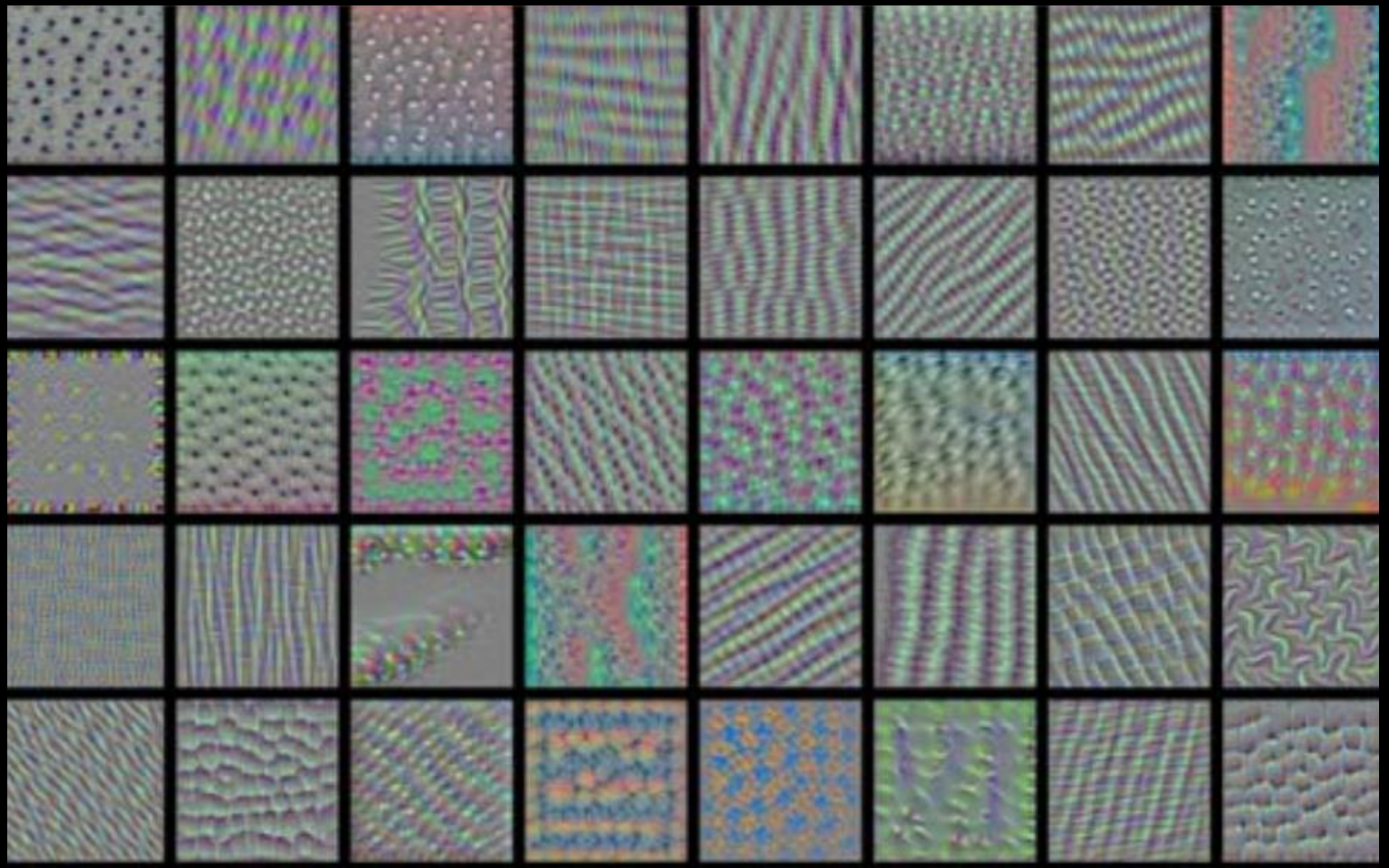


What are layers looking for? Data from a convnet trained on ImageNet

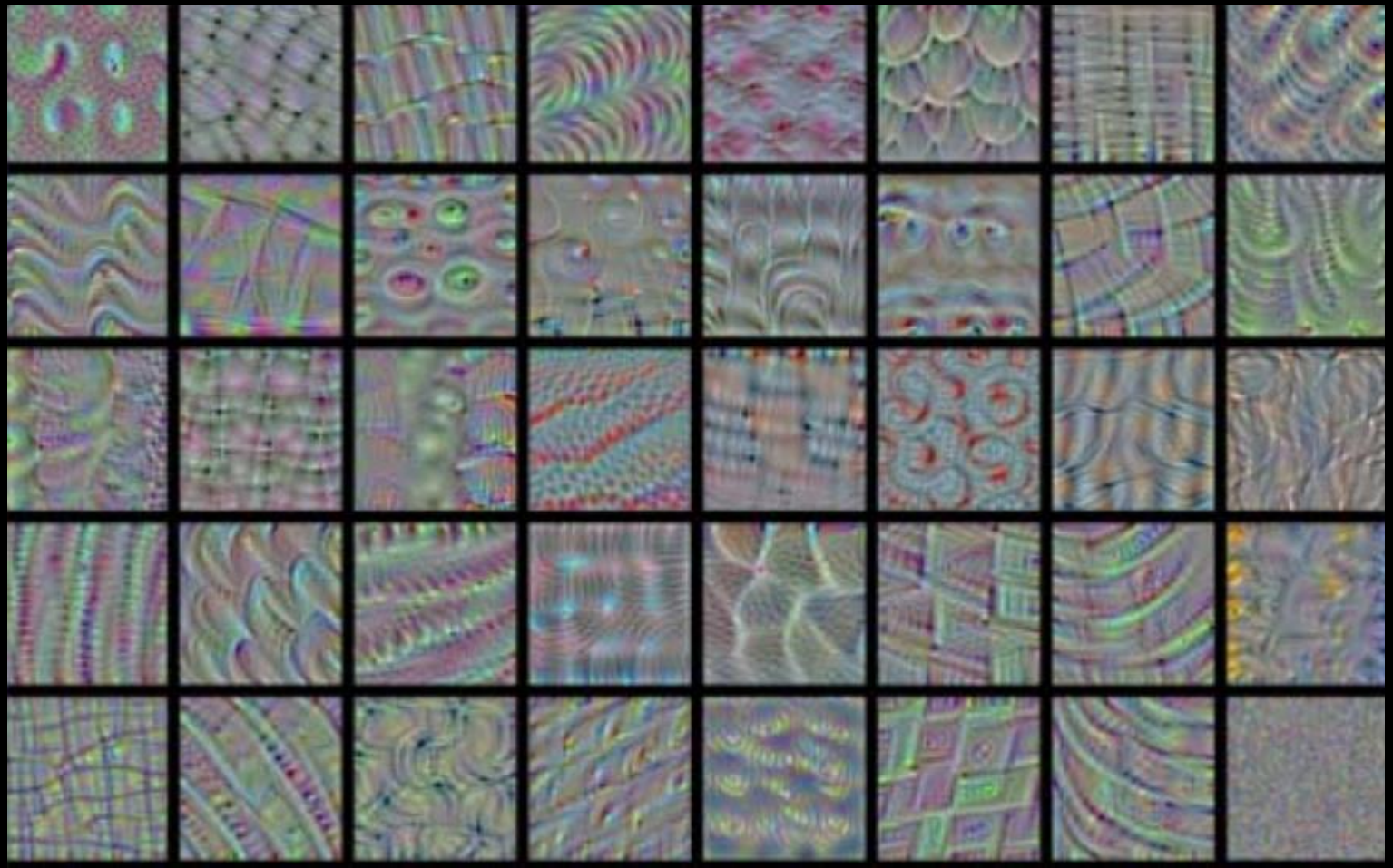
Shallow layers respond to fine, **low-level** patterns



Intermediate layers ...



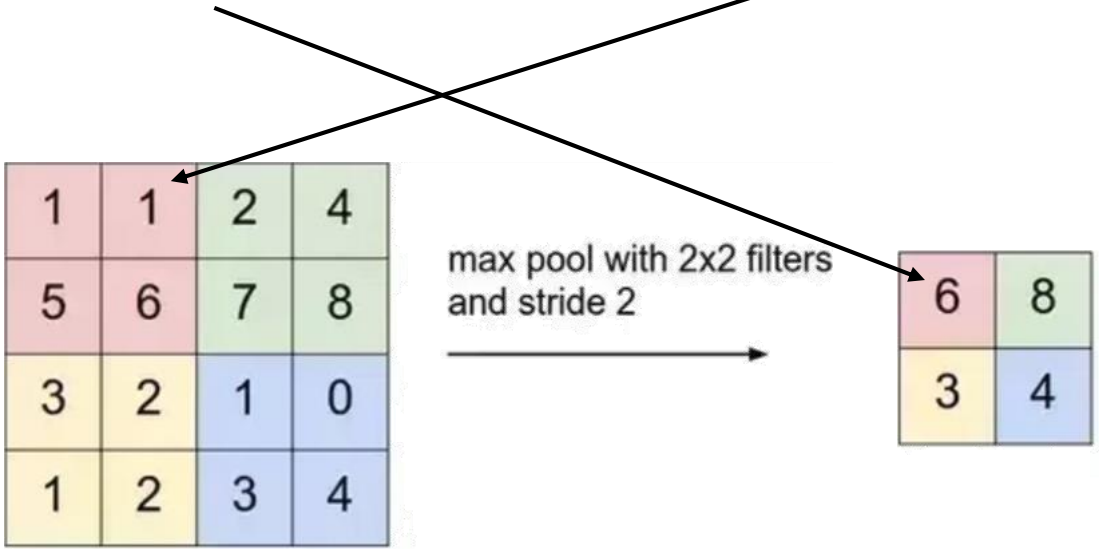
Deep layers respond to complex, **high-level** patterns



Detail: backprop with max pooling

The gradient is only routed through the input pixel that contributes to the output value; e.g.:

Gradient of • with respect to • = 0



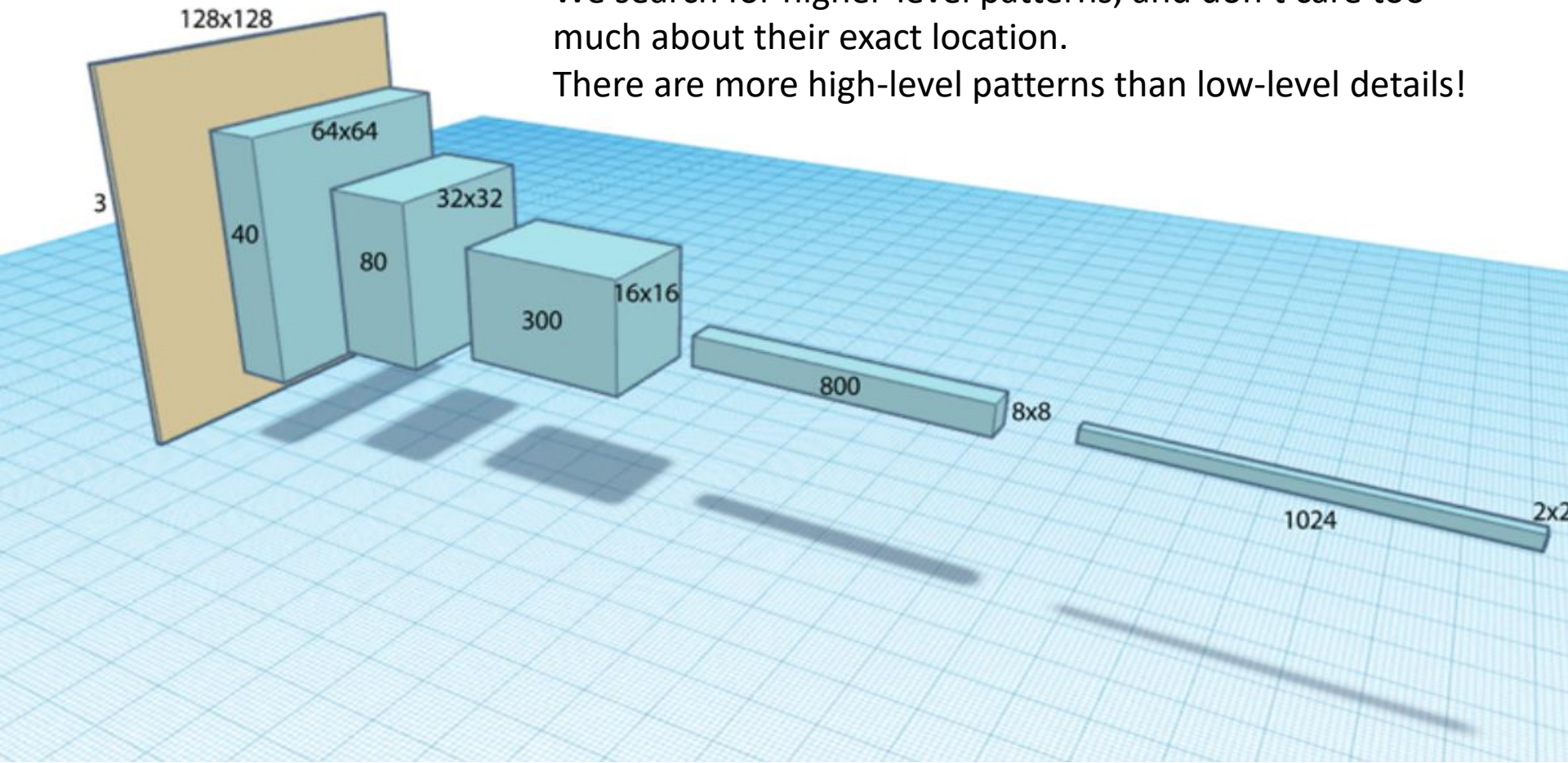
A typical architecture

As we move to deeper layers:

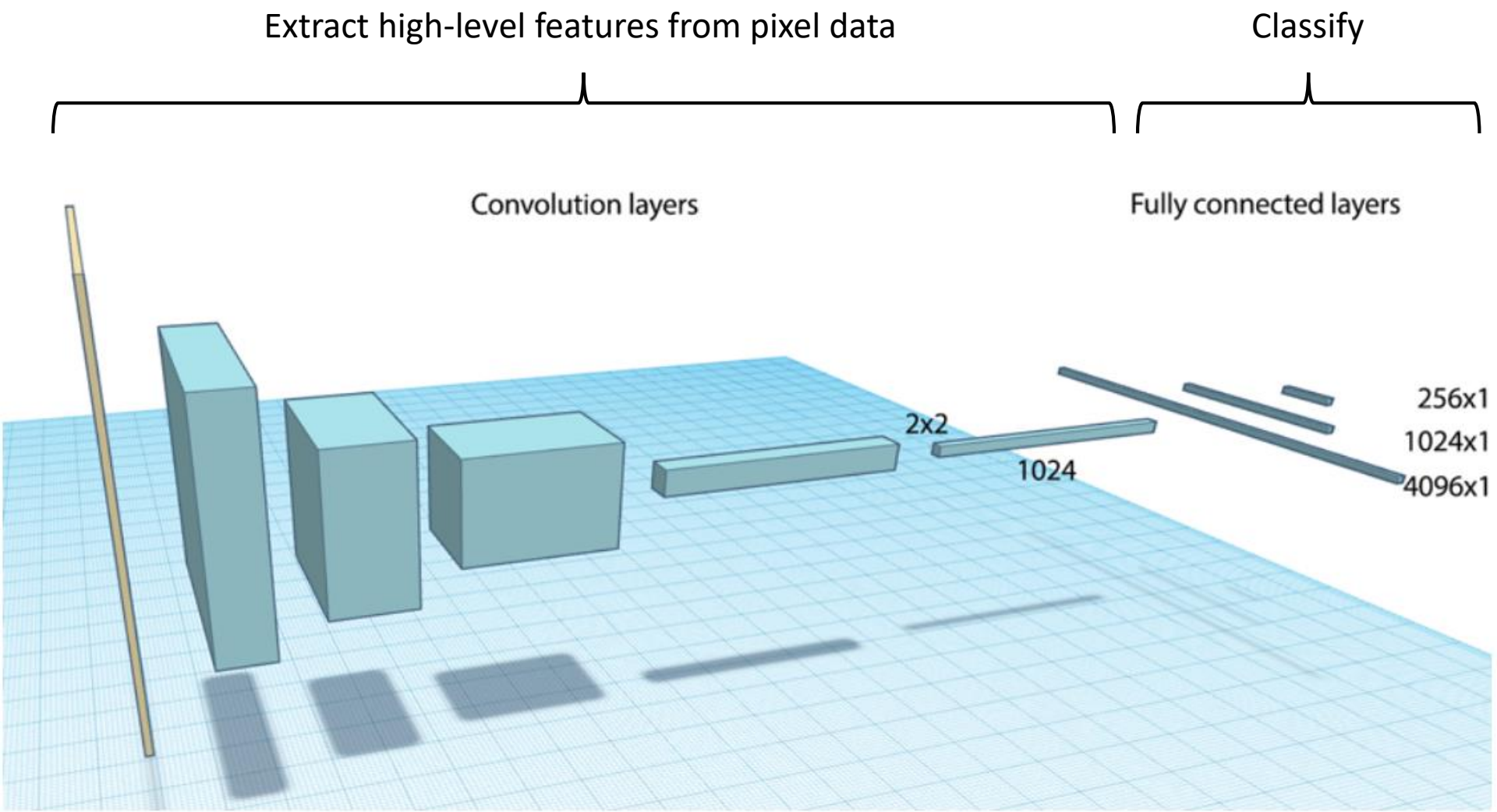
- spatial resolution is reduced
- the number of maps increases

We search for higher-level patterns, and don't care too much about their exact location.

There are more high-level patterns than low-level details!



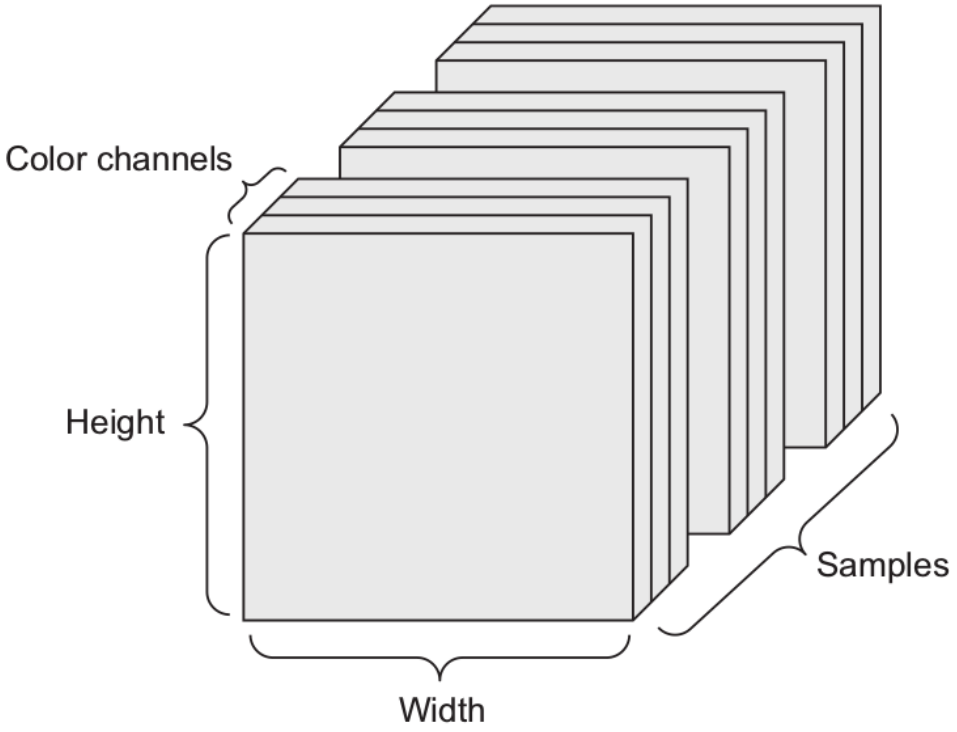
A typical architecture



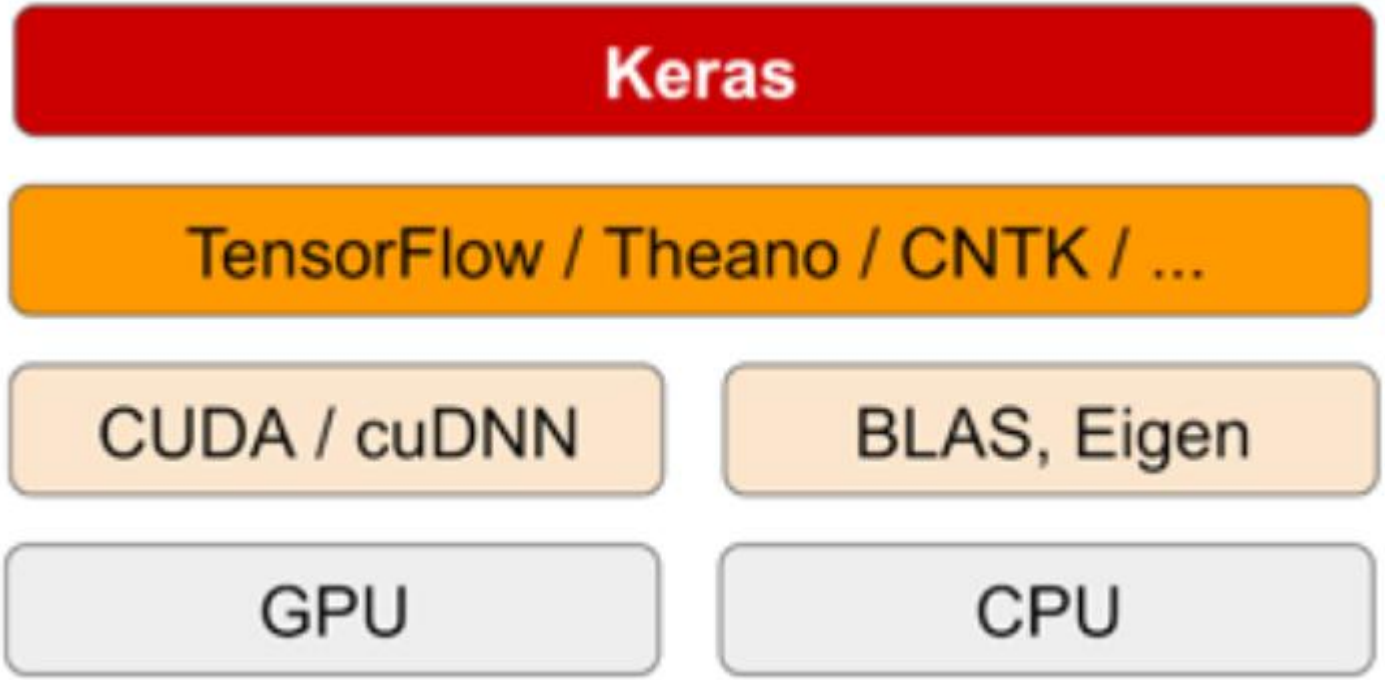
We will manipulate 4D tensors

Images are represented in 4D tensors:

Tensorflow convention:
(samples, height, width, channels)

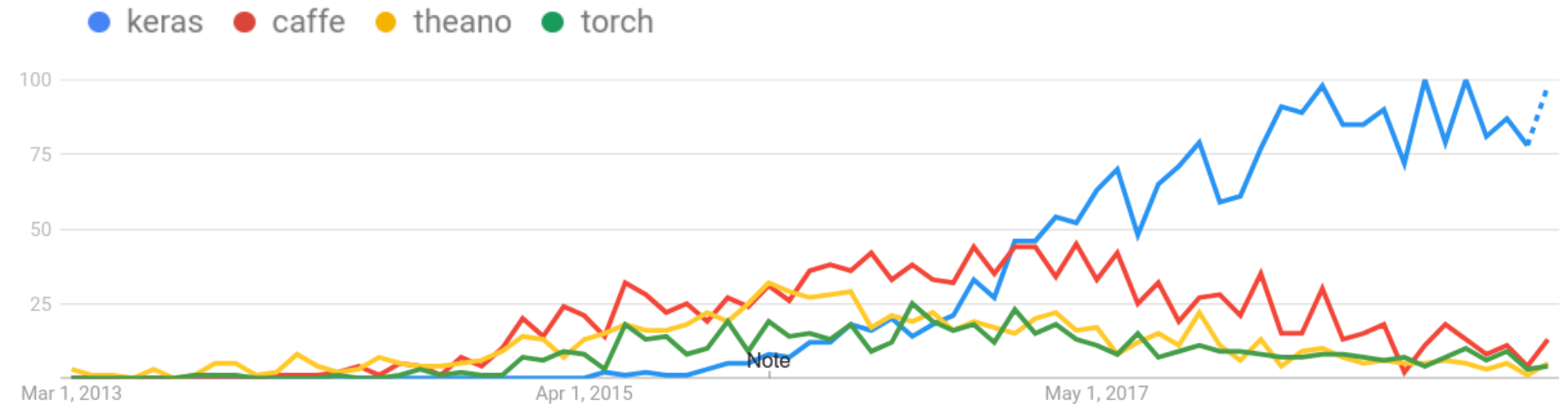


The software stack



What is Keras?

- A model-level library, providing high-level building blocks for developing deep-learning models.
- Doesn't handle low-level operations such as tensor manipulation and differentiation.
- Relies on backends (such as Tensorflow)
- Allows full access to the backend



Why Keras?

Pros:

- Higher level → fewer lines of code
- Modular backend → not tied to tensorflow
- Way to go if you focus on applications

Cons:

- Not as flexible
- Need more flexibility? Access the backend directly!

Conclusions

Conclusions

We have seen three approaches to image classification:

- Using a nearest-neighbor classifier direcly on raw pixel data (a trivial approach)
- Computing handcrafted features, then training some classifier
- Training neural networks that use raw pixel data as inputs:
  - Densely connected, or
  - With convolutional and maxpooling layers (convnets)