Exploring Machine Intelligence

Week 3, Convolutional NNs

~ Vítek Růžička

Motivation for today

-1	-1	-1
-1	8	-1
-1	-1	-1

3x3 conv

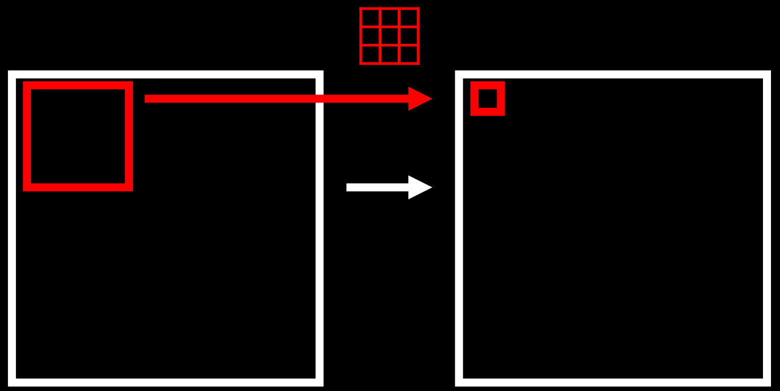
Manual filter: **edge detection**

>>> setosa.io/ev/image-kernels/ <<<

Today

Building a better machine:

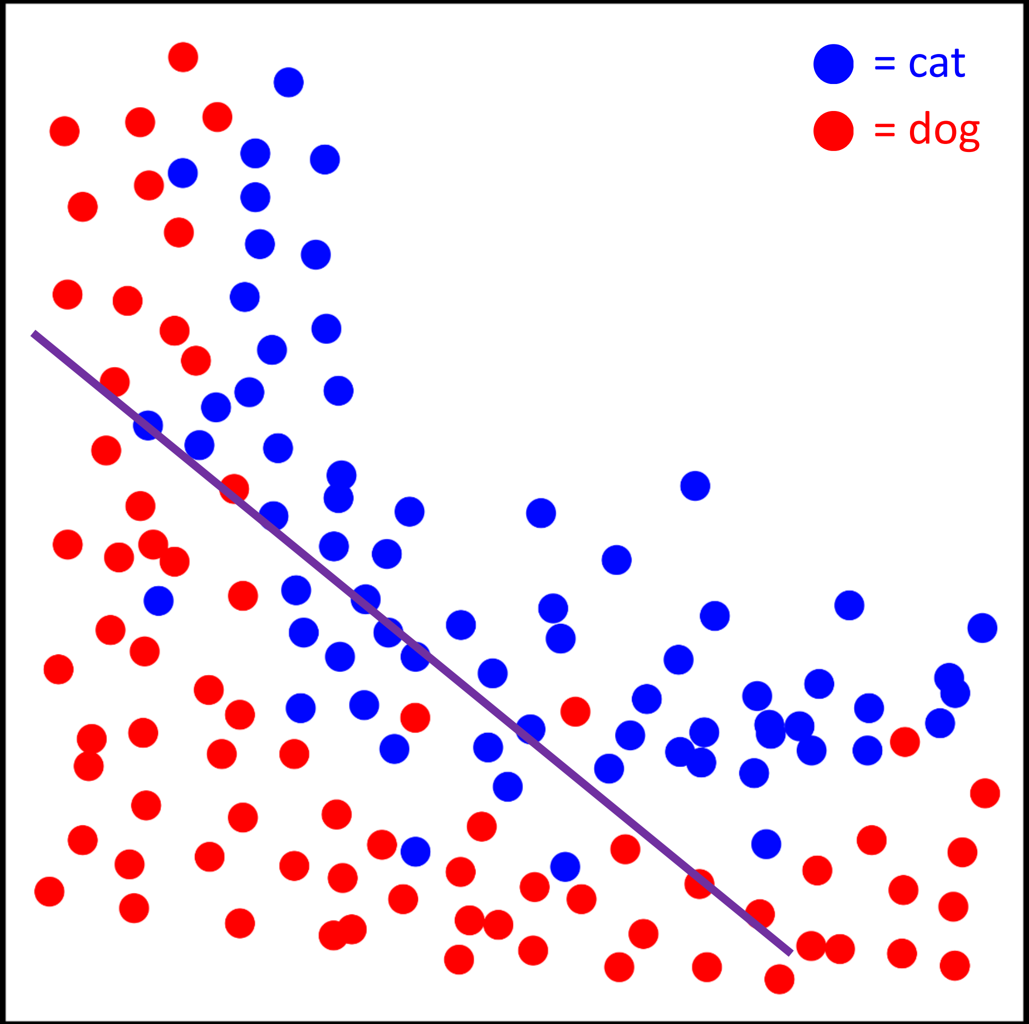
- Why should we split data?
- Convolutional operation
- **Convolutional NNs**
- Real world architecture: **AlexNet**
- General feature extractors



convolution

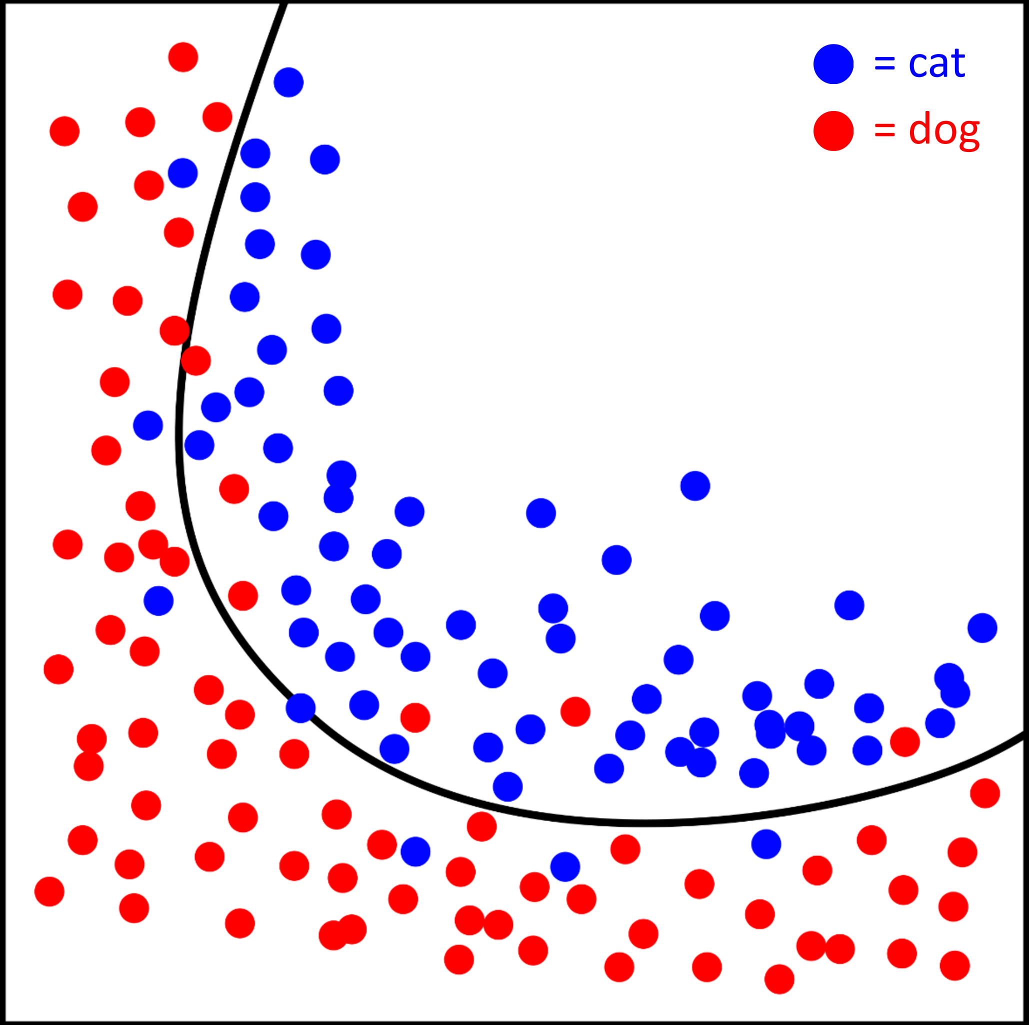
Data splitting and overfitting

- Let's illustrate **overfitting** on 2 classes example (dogs vs. cats)



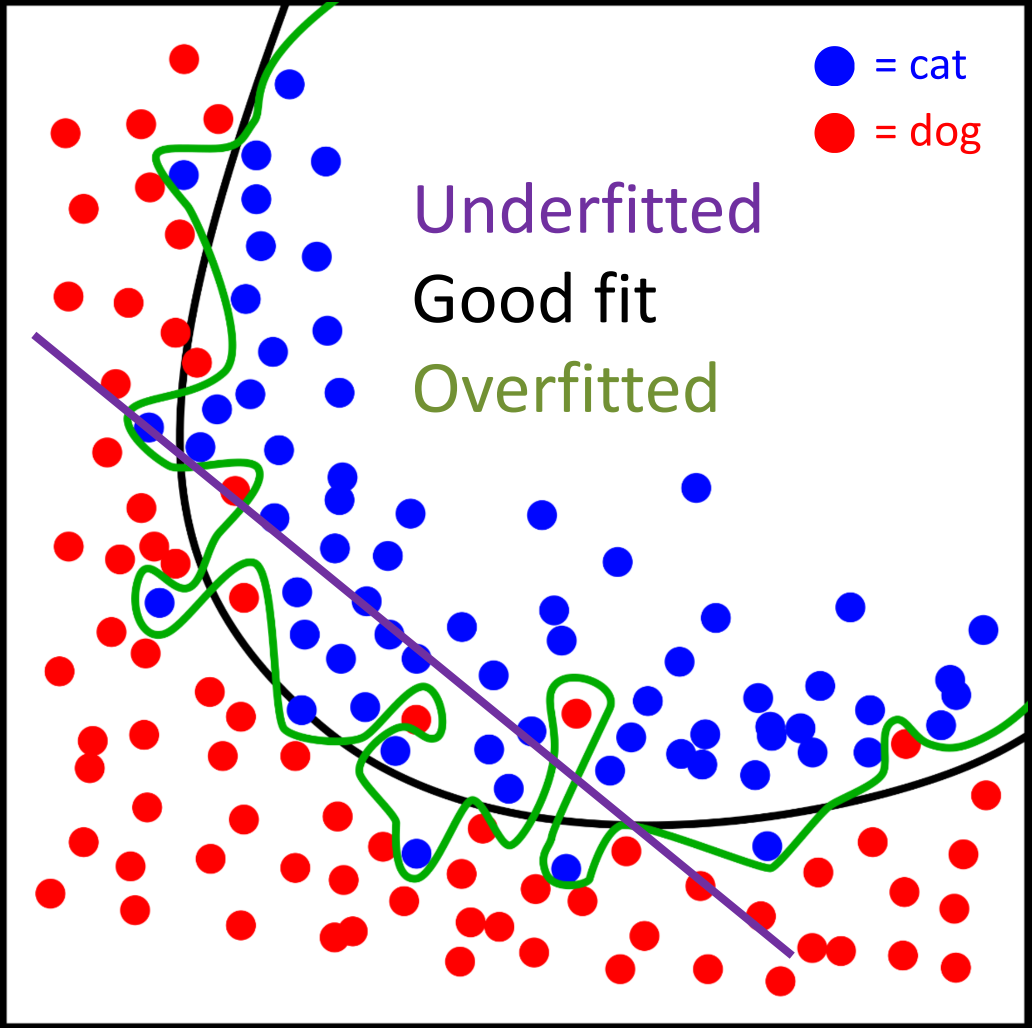
Data splitting and overfitting

- Let's illustrate **overfitting** on 2 classes example (dogs vs. cats)
- We want to split the data with a classifier – model a boundary between the two classes



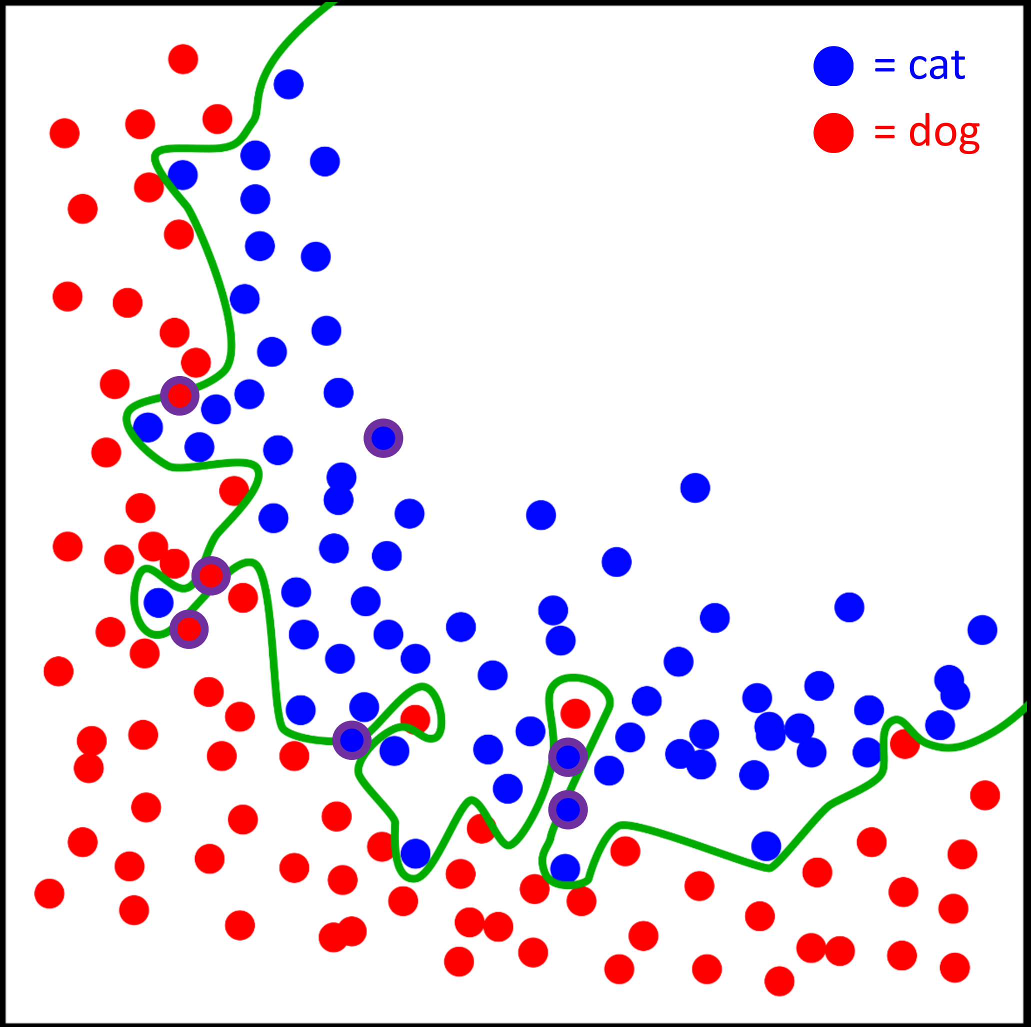
Data splitting and overfitting

- Let's illustrate **overfitting** on 2 classes example (dogs vs. cats)
- We want to split the data with a classifier – model a boundary between the two classes
- This **separation** can be *sensible* in terms of the boundary it creates … or it can be *overly complicated*

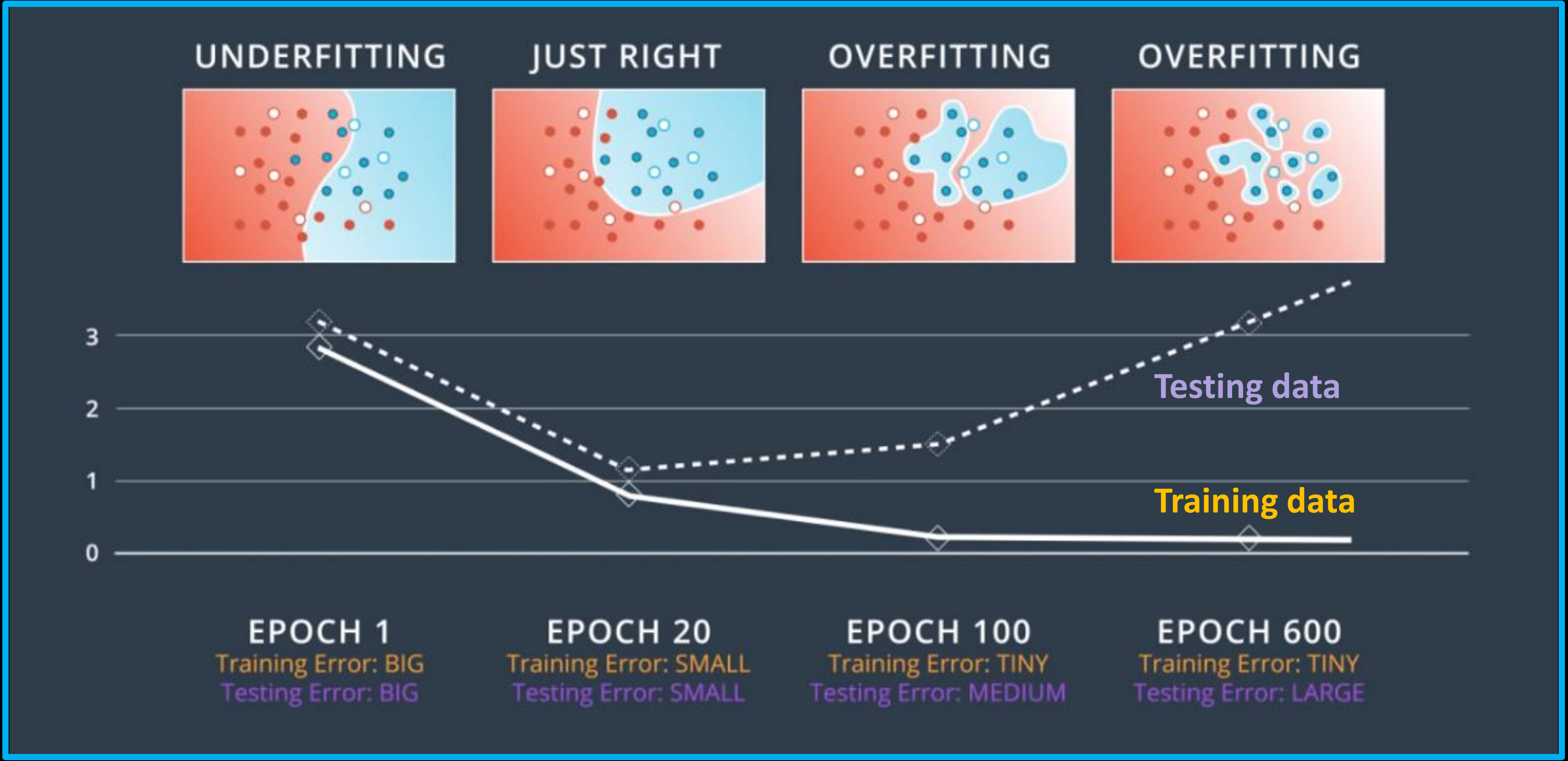


Data splitting and overfitting

- The problem with having the overly **complex boundary** is as follows:
  - It **perfectly separates all the data** we gave it **in the training set**! (100% accuracy on training set)
  - Imagine we have a new samples: With the boundary being so specialized on this particular training set, the **new samples will fail to be classified correctly**.



- Can we **detect overfitting**? Yes:

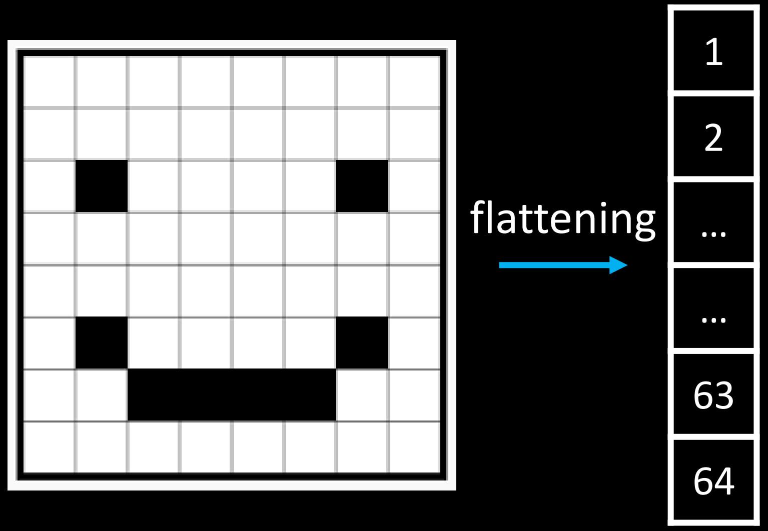


- Can we **prevent overfitting** from happening?
  - Smaller models, early stops in training, various tricks …

Plot from: [here]
More intuition: [video]

Building a better machine

- To work with images, we need to do better! While something like handwritten digits are reasonably easy to learn with even simple fully connected NNs (around 1998) … We will need more powerful tools to handle larger datasets (from around 2012).



- One **simplification we made previously**, was that we simply **flattened** the **whole image** and looked at **all pixels independently**.

Images have locality

^^^ *Let's imagine any image in here*

Images have locality

^^^ *Let's imagine any image in here*

These two pixels will likely contain something relevant to each other.

While these two pixels won't likely have that much in common.

Images have locality

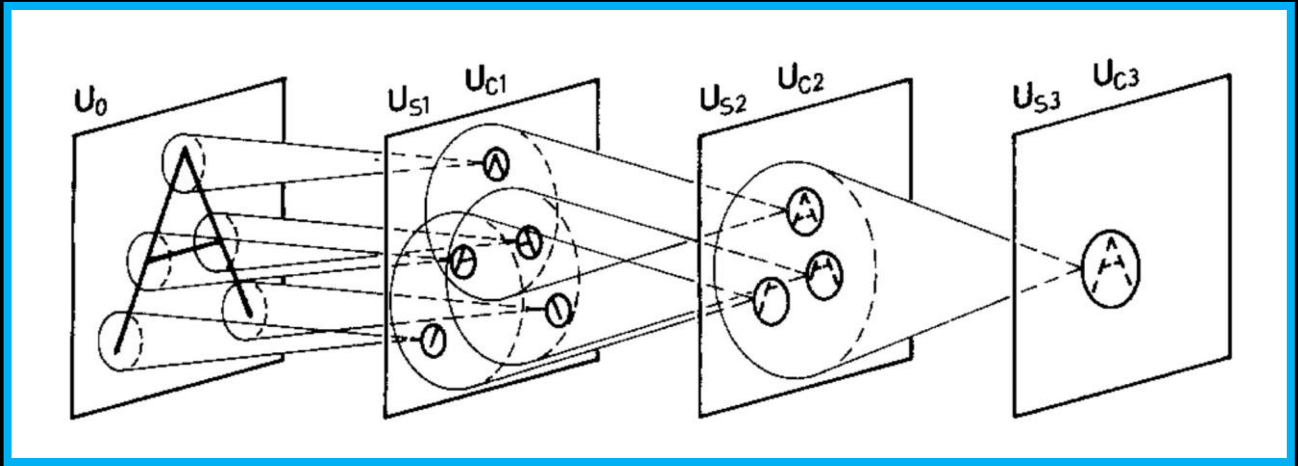
^^^ *Let's imagine any image in here*

These two pixels will likely contain something relevant to each other.

While these two pixels won't likely have that much in common.

- In 1962 a study by *Hubel and Wiesel* explored human visual system and discovered two types of cells – **localized cells** processing details and **complex cells** covering larger areas.

Neocognitron

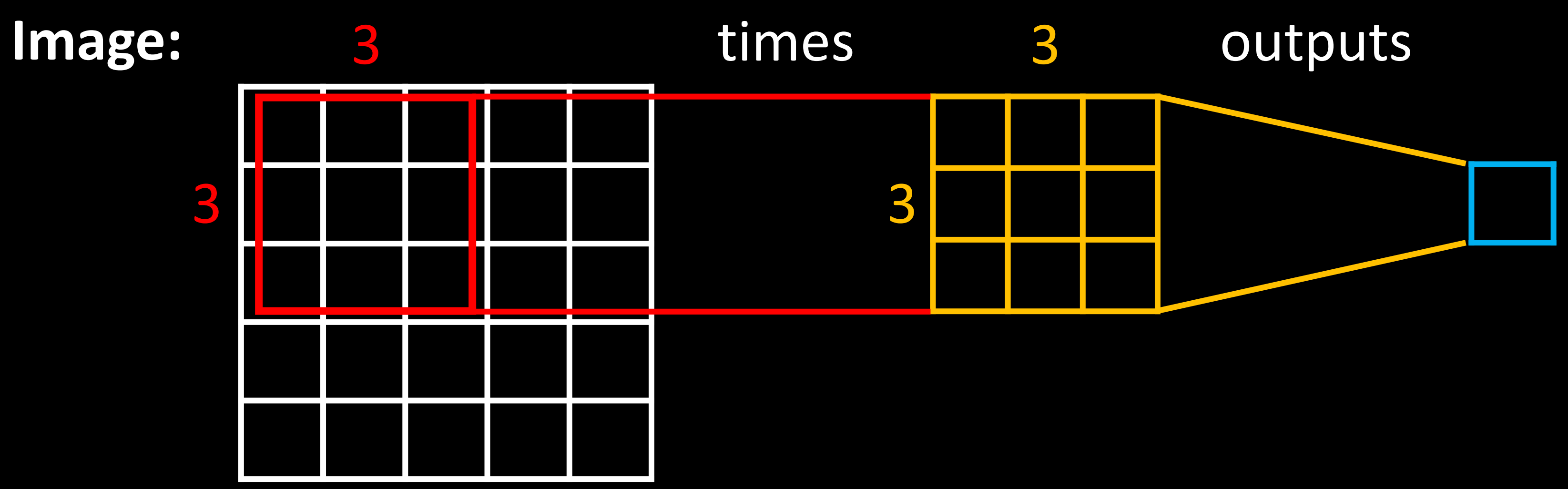


<- Predecessor to Convolutional Neural Networks

- In the 1982 work by Kunihiko Fukushima, they attempted to build an **artificial visual system** which had two types of units:
  - *"s-cells"* focusing on single image features
  - *"c-cells"* combining information from an area

Bonus: [video]()

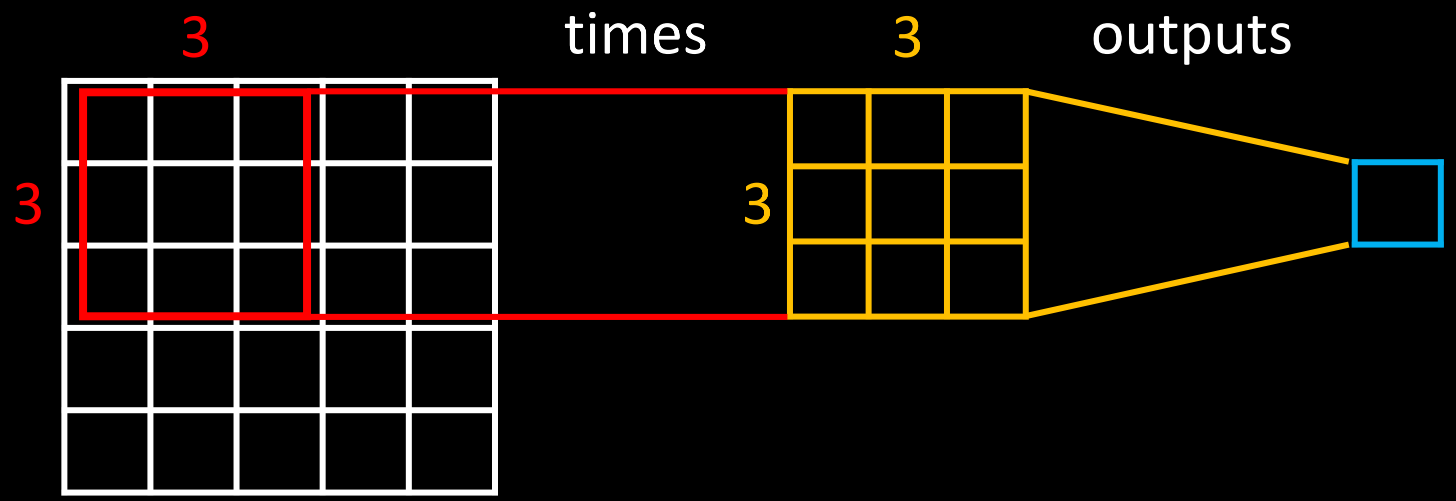
Convolution



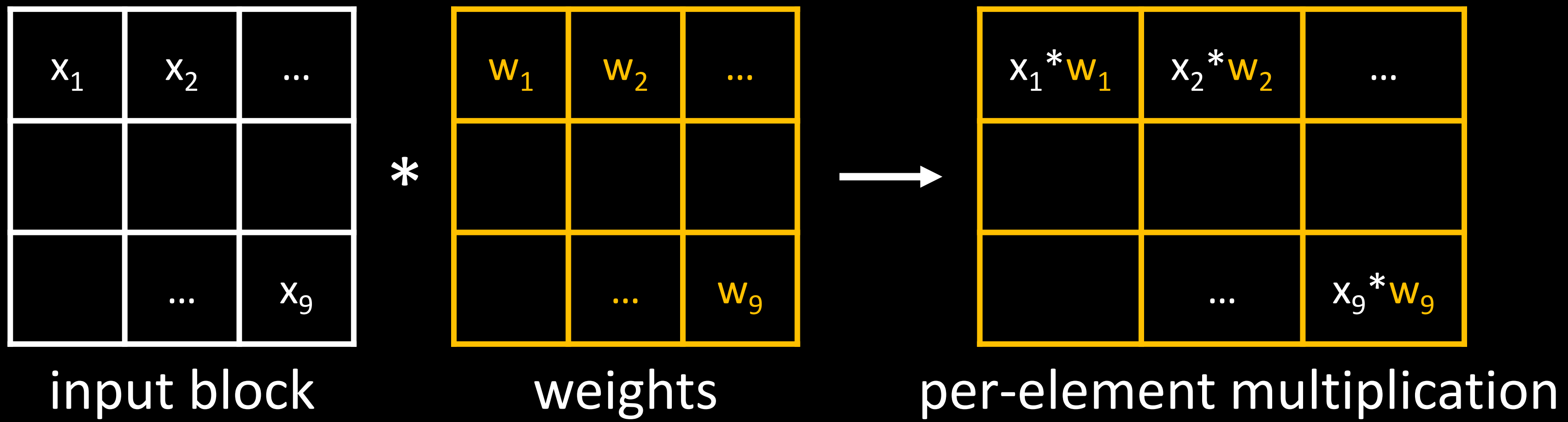
- **Convolution** is an operation applied on an image …

Convolution

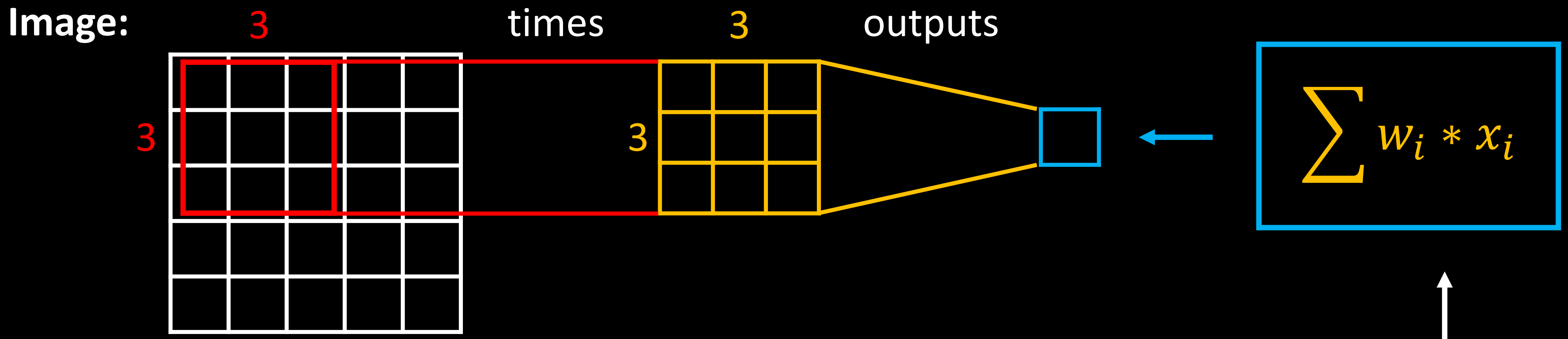
Image:



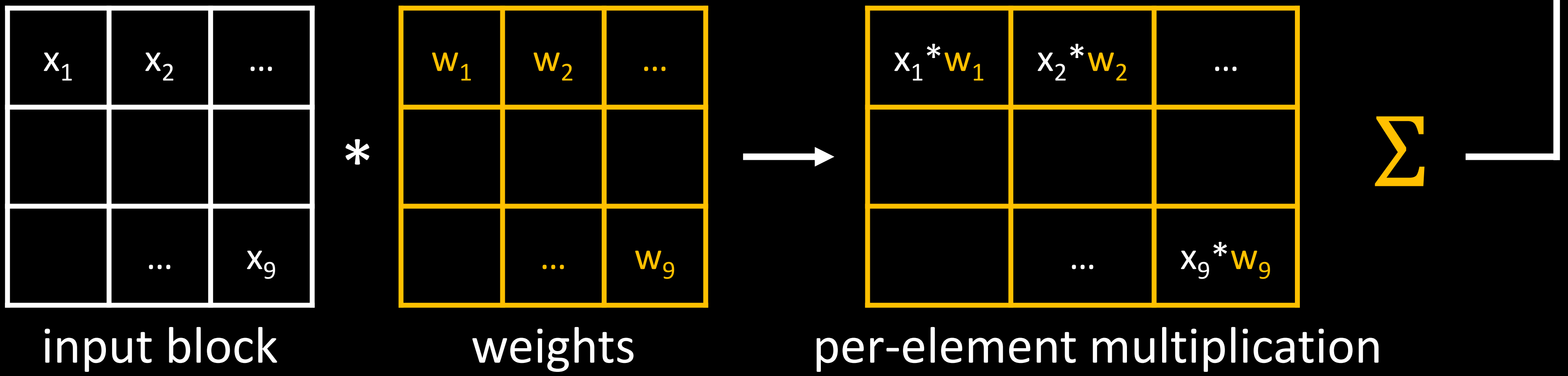
- **Convolution** is an operation applied on an image …



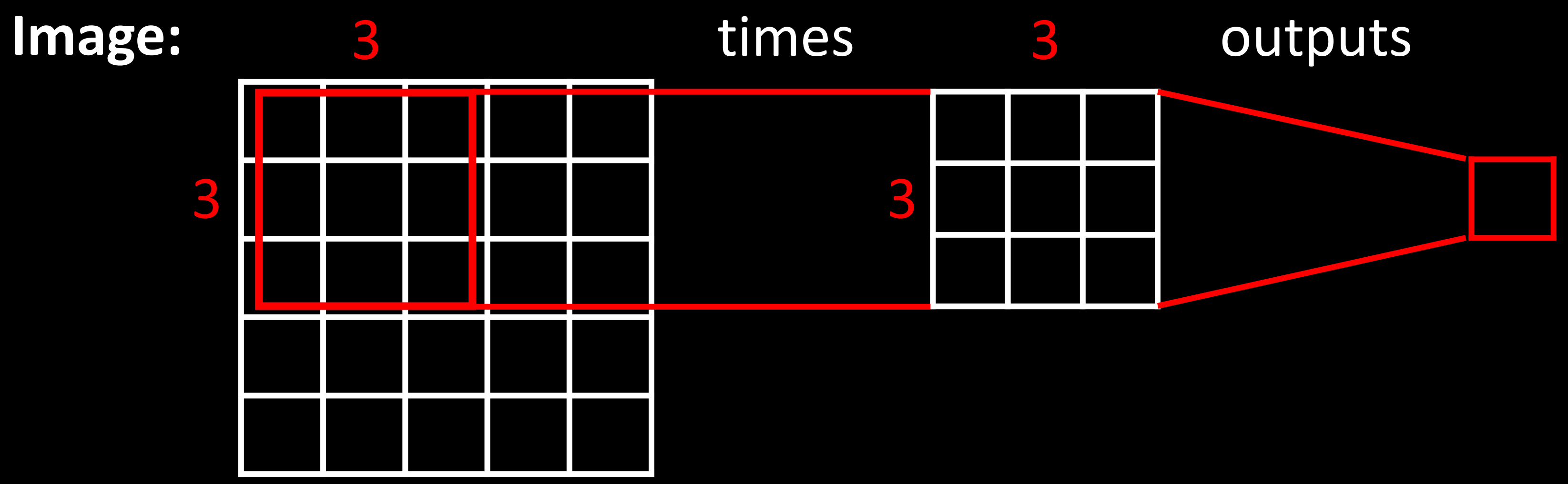
Convolution



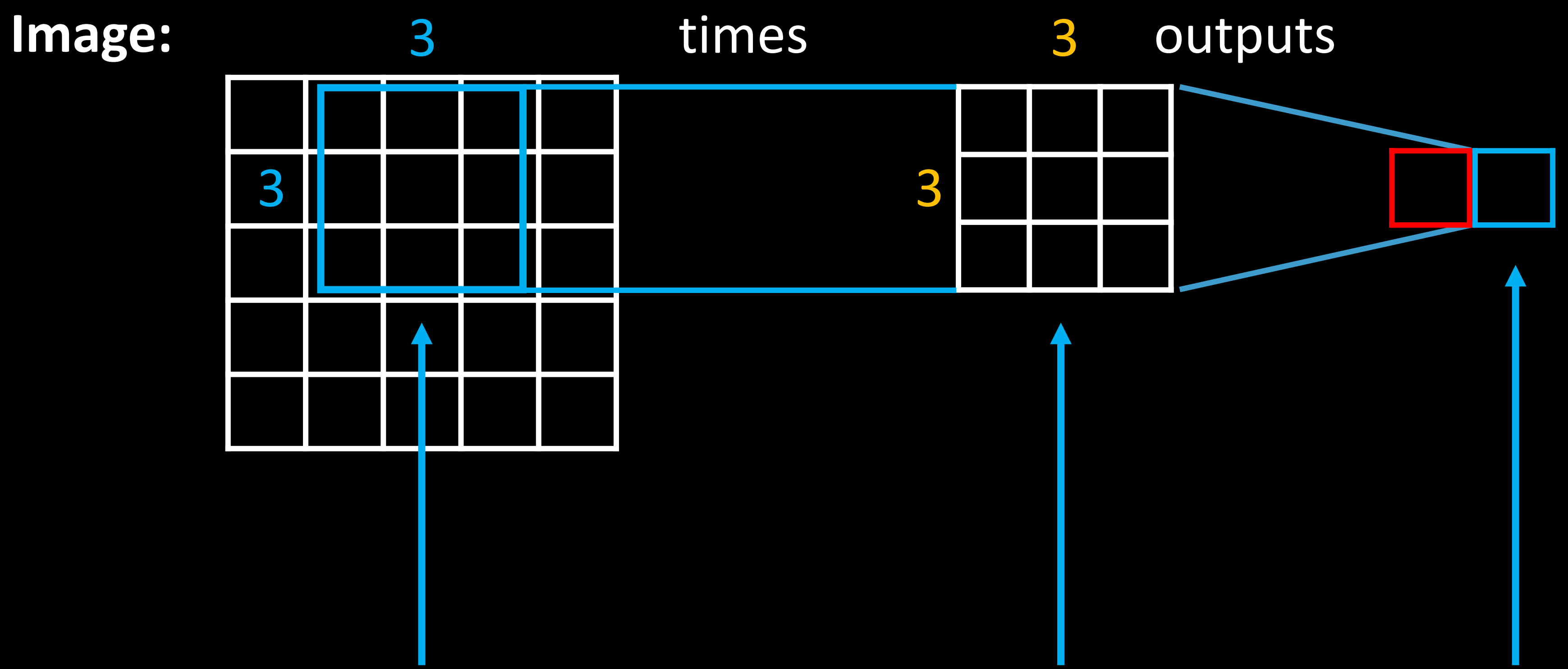
- **Convolution** is an operation applied on an image …



Convolution



Convolution

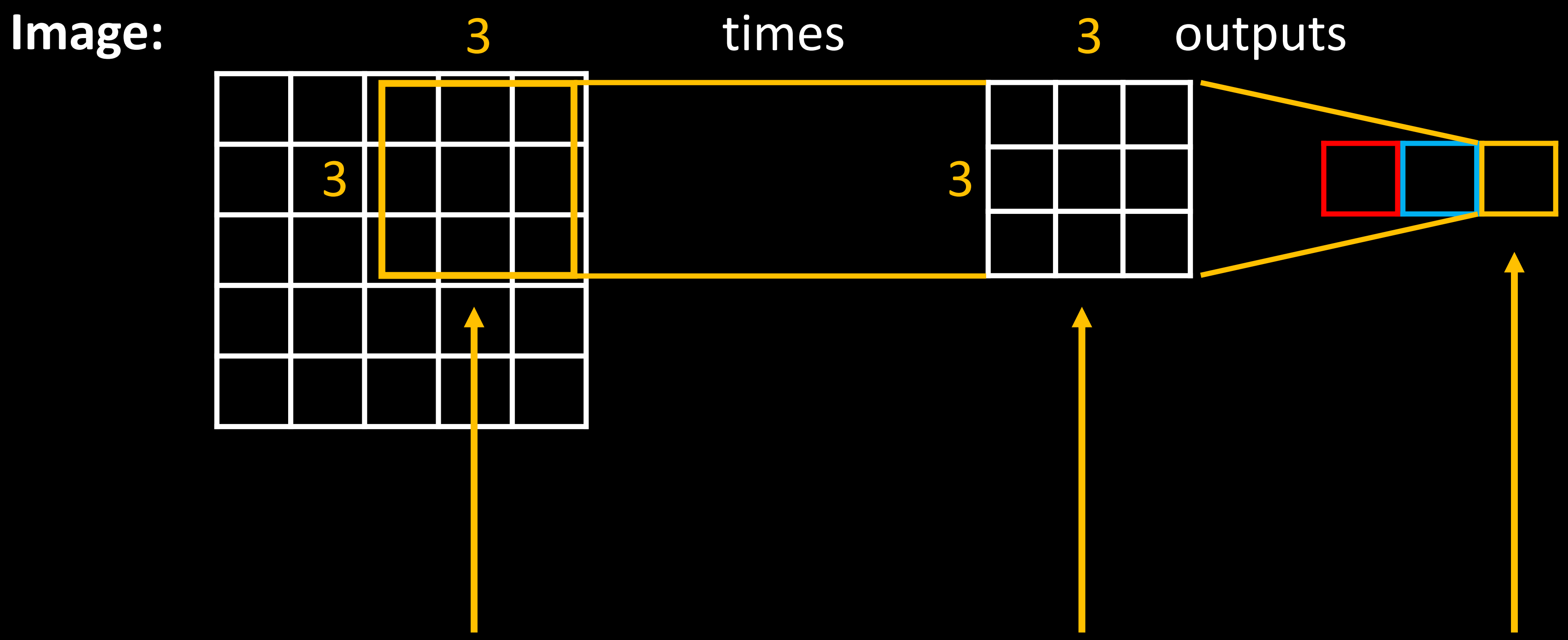


Jump by one space in input data

The same 9 weights!

Also jump by one space in the outputs

Convolution



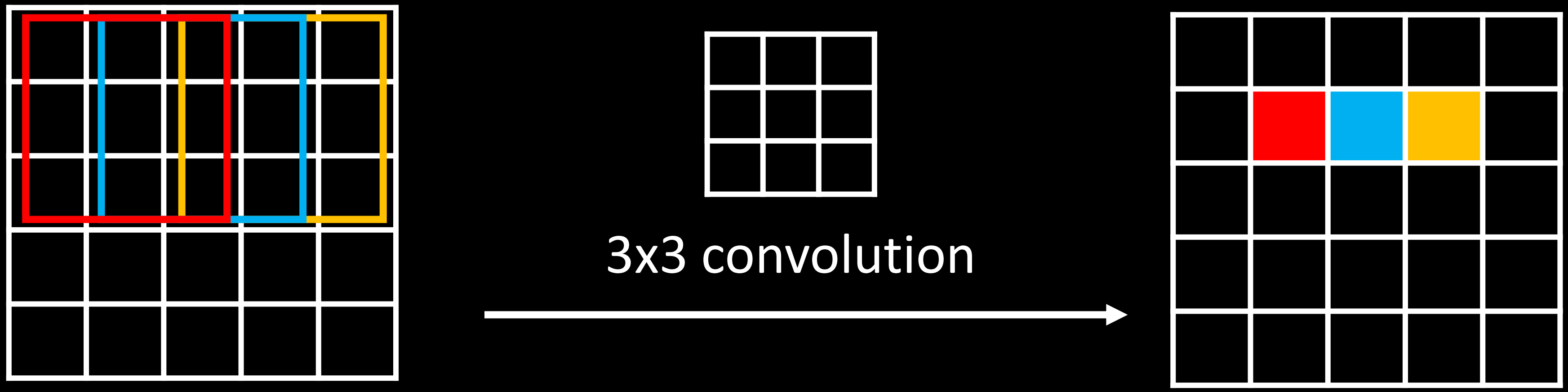
Jump by one space in input data

The same 9 weights!

Also jump by one space in the outputs

Convolution

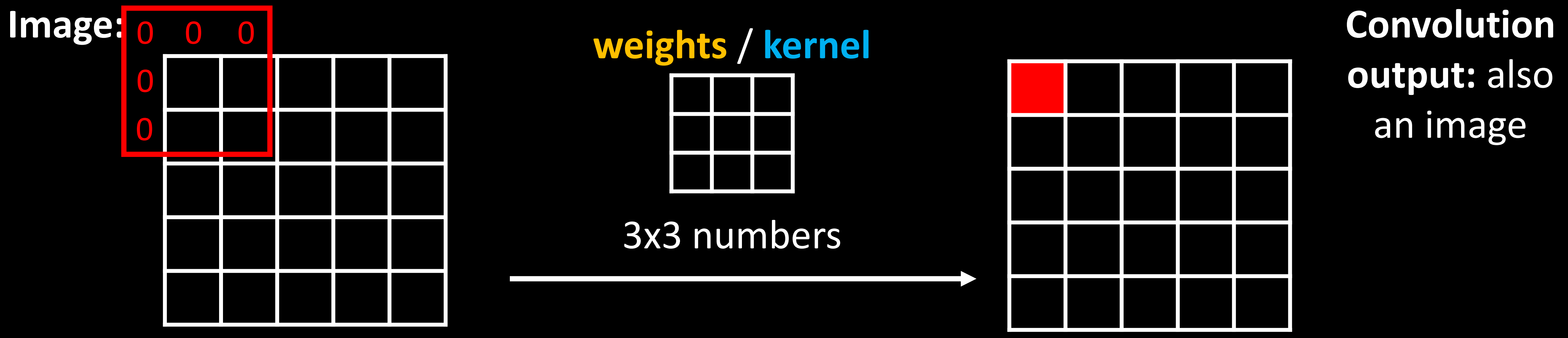
Image:



Convolution output: also an image

- The **same weights** are applied as a **filter**, jumping over the whole image … effectively producing an image on the output!
- Convolutions were used before neural networks as transformation functions to process images.

Convolution



- **Terminology**: we called these 3x3 numbers **"weights"**, but with convolutions they are also referred to as **kernel**
- **Detail**: To keep the same size of the output image, we need to extend the original image by 1 pixel – we pretend that it includes zeros (also called **"zero padding"**)

Manual convolution filters

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

3x3 conv

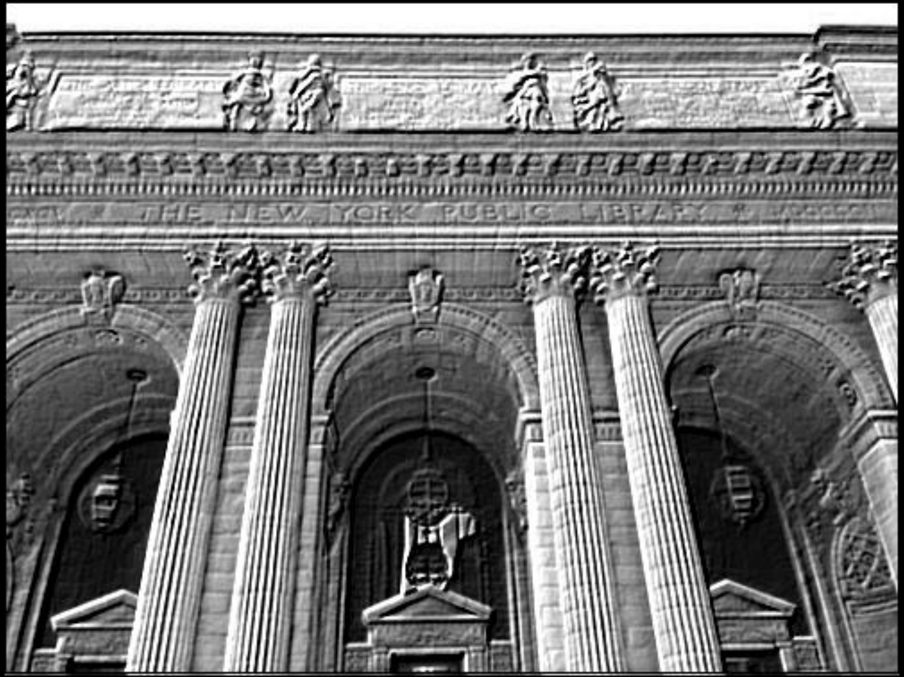
blur

>>> setosa.io/ev/image-kernels/ <<<

Manual convolution filters

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

3x3 conv



emboss

>>> setosa.io/ev/image-kernels/ <<<

Manual convolution filters

-1	-1	-1
-1	8	-1
-1	-1	-1

3x3 conv

outline

>>> setosa.io/ev/image-kernels/ <<<

Automatic convolution filters?

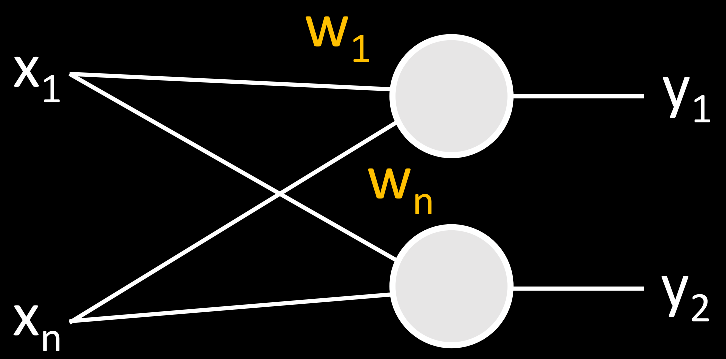
- Similarly as in the previous cases (and your homework), you can manually find values for weights in Neural Network models to process the given dataset in some way (minimizing error).

- Or you can try to get these **parameters of convolution kernels automatically** (like any other parameters of the model) **during training**.

Neural Networks with Convolution

- In addition to the **fully connected layers** of neurons:

Fully connected / Dense layer

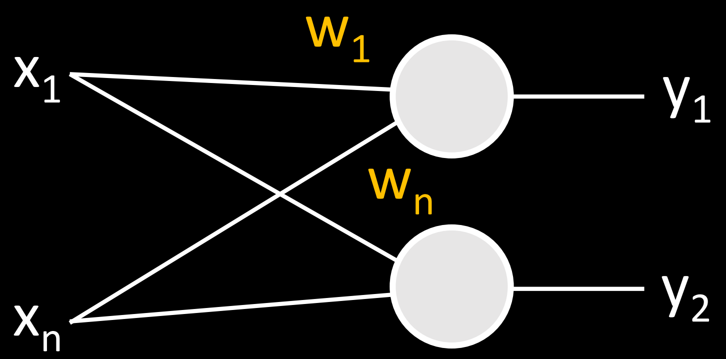


Parameters in connections

Neural Networks with Convolution

- In addition to the **fully connected layers** of neurons:

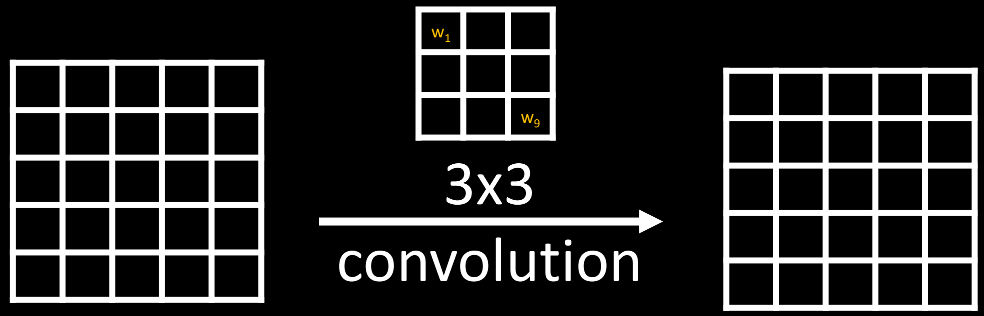
Fully connected / Dense layer



Parameters in connections

- We can also use the **convolutional layers**:

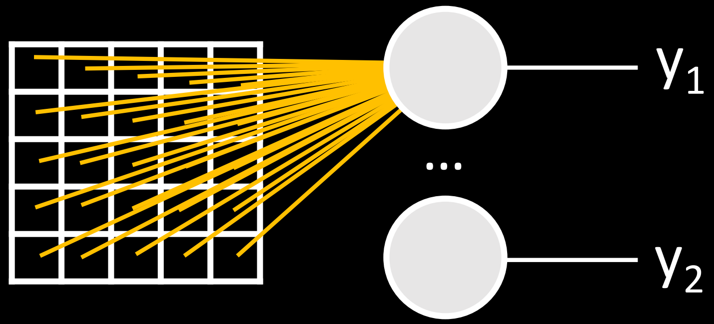
Convolutional layer



Parameters in the kernel of the convolution operation

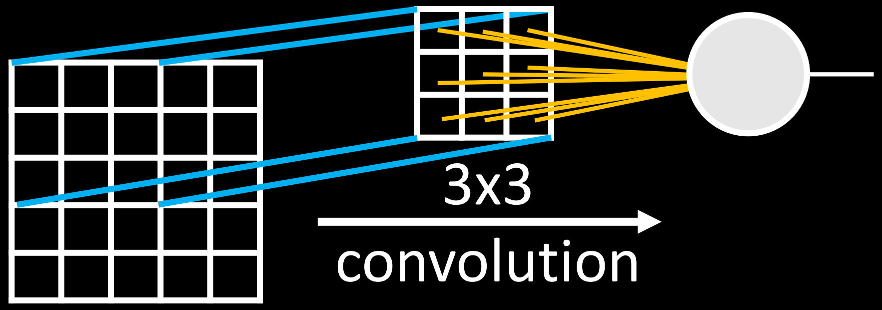
Difference in processing images

Fully connected / Dense layer



# of Weights = # of Pixels Multiplied by number of neurons in layer.
… a lot!

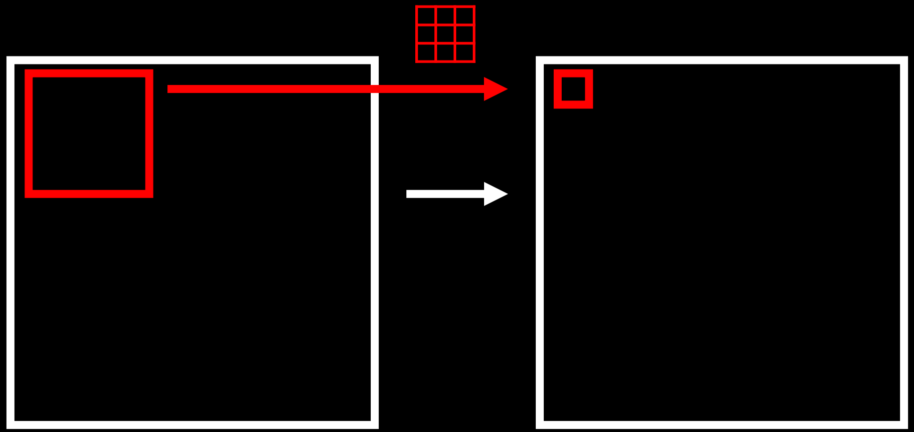
Convolutional layer



# of Weights = size of kernel

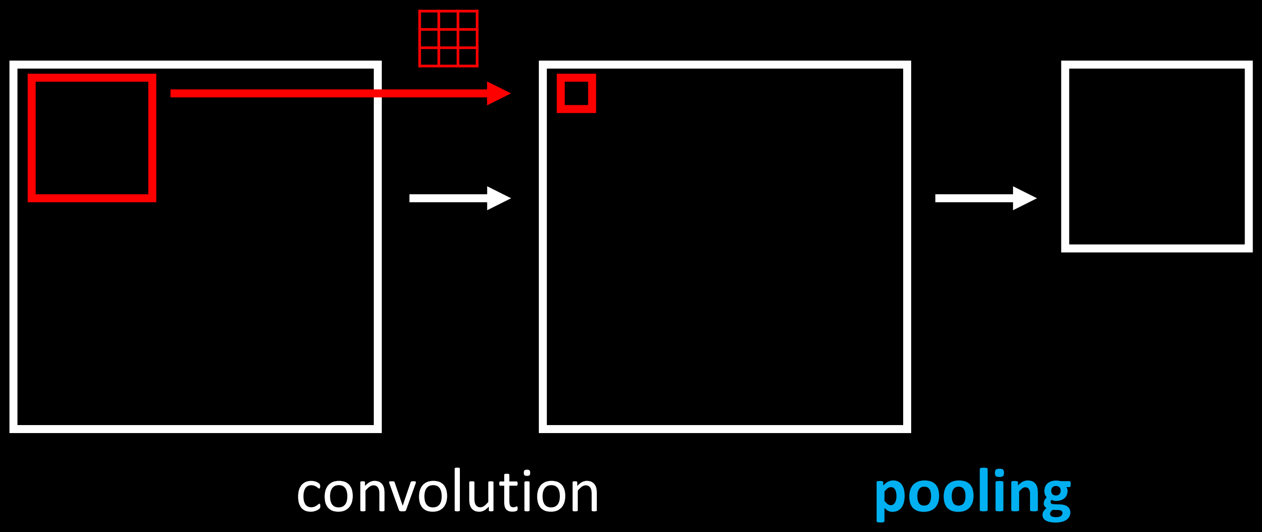
… fixed for any input size!

Convolutions and Pooling



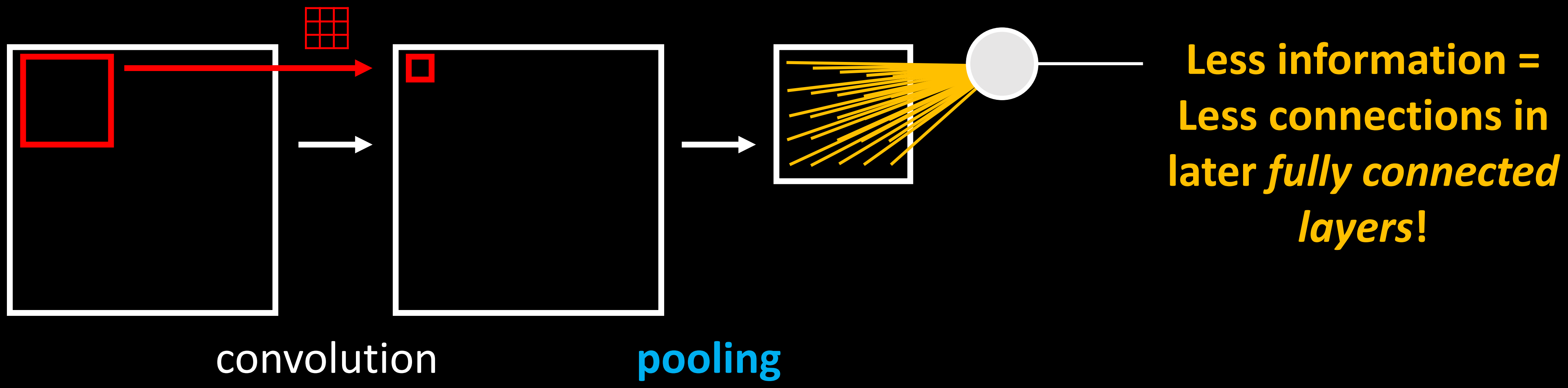
convolution

Convolutions and Pooling



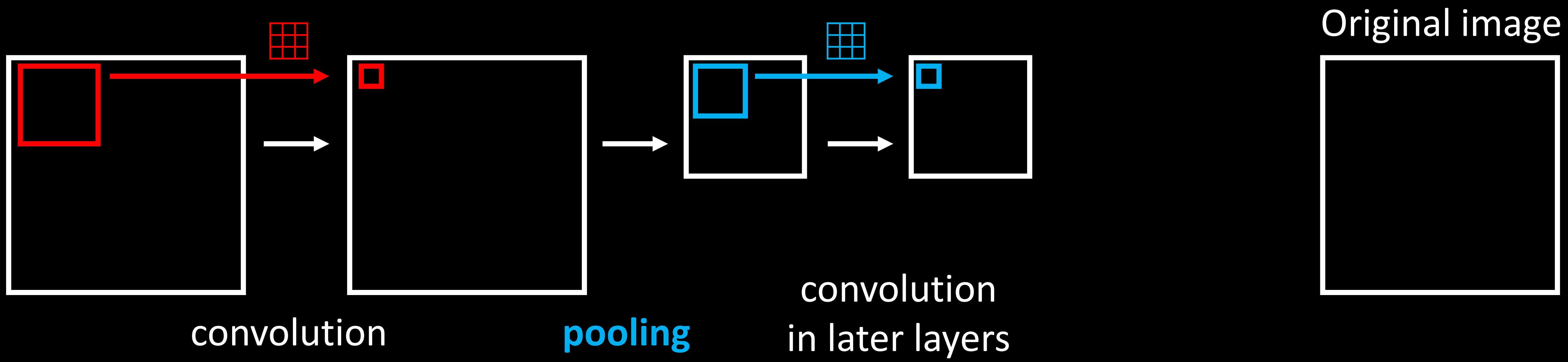
- Convolutional layers are usually followed by **downsampling layers (pooling)** which rescale the image (less information)

Convolutions and Pooling



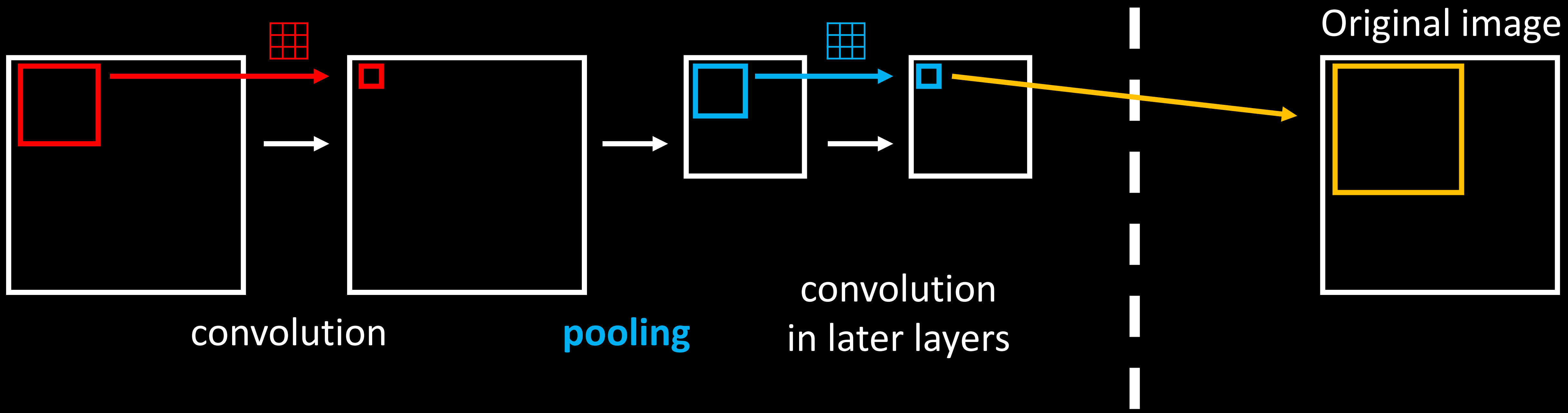
- Convolutional layers are usually followed by **downsampling layers (pooling)** which rescale the image (**less information**)

Convolutions and Pooling



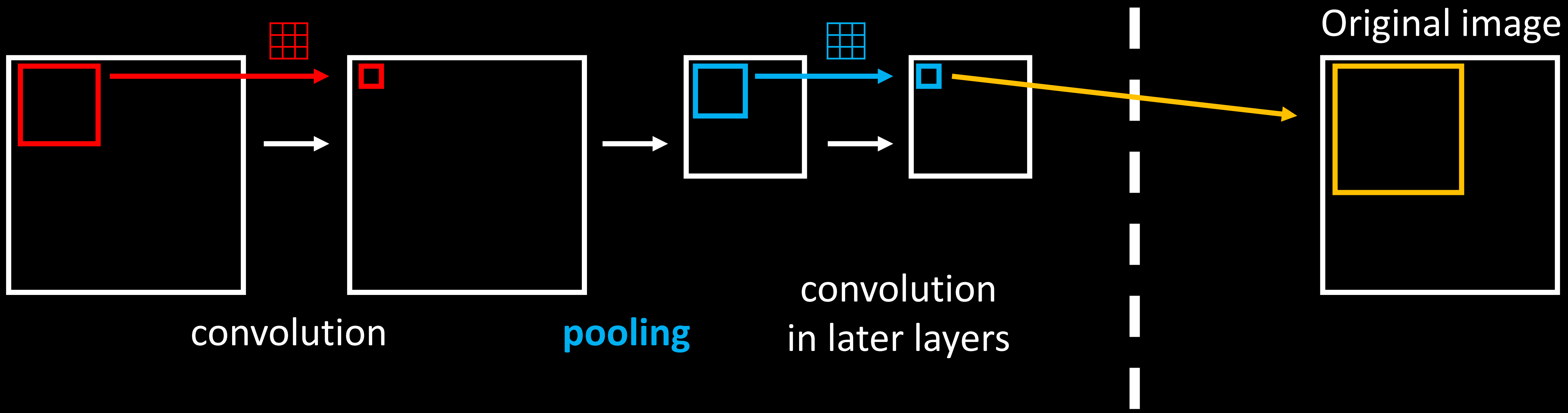
- Convolutional layers are usually followed by **downsampling layers (pooling)** which rescale the image (less information)
- "Later" convolutions are also looking at much **larger region in the original image** with their kernel

Convolutions and Pooling



- Convolutional layers are usually followed by **downsampling layers (pooling)** which rescale the image (less information)
- "Later" convolutions are also looking at much **larger region in the original image** with their kernel

Convolutions and Pooling

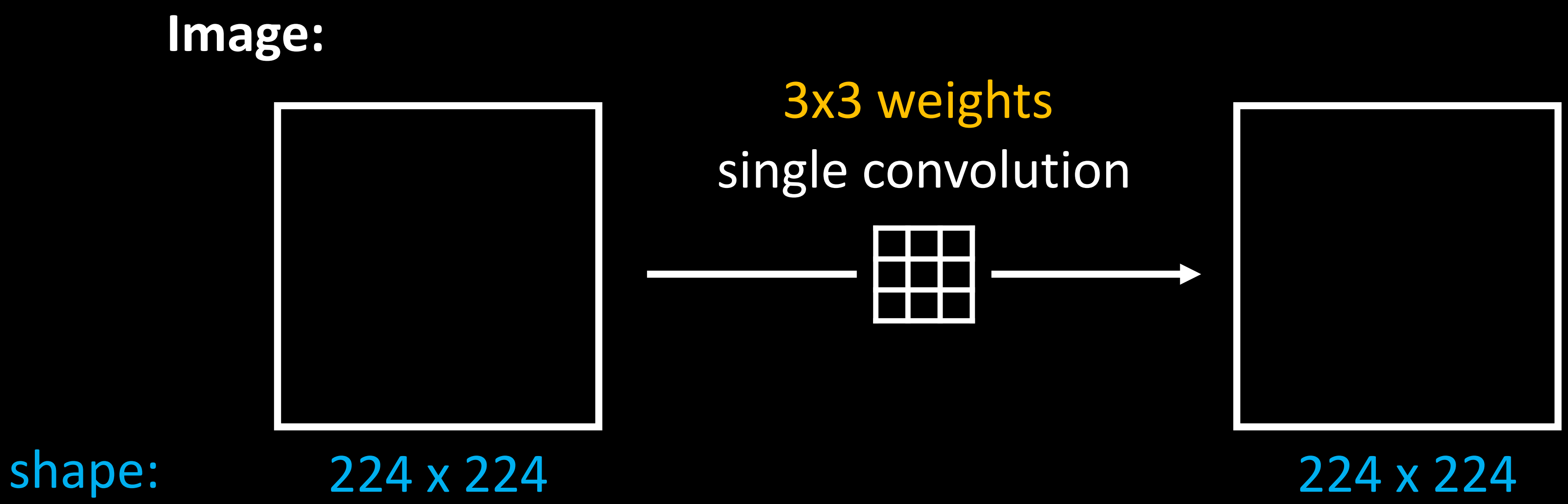


- This means, that the later convolutional layers can **specialize on processing different scales of the image** (This is huge!)

details   larger areas   … We will see that this means visually!

One more detail before the pause

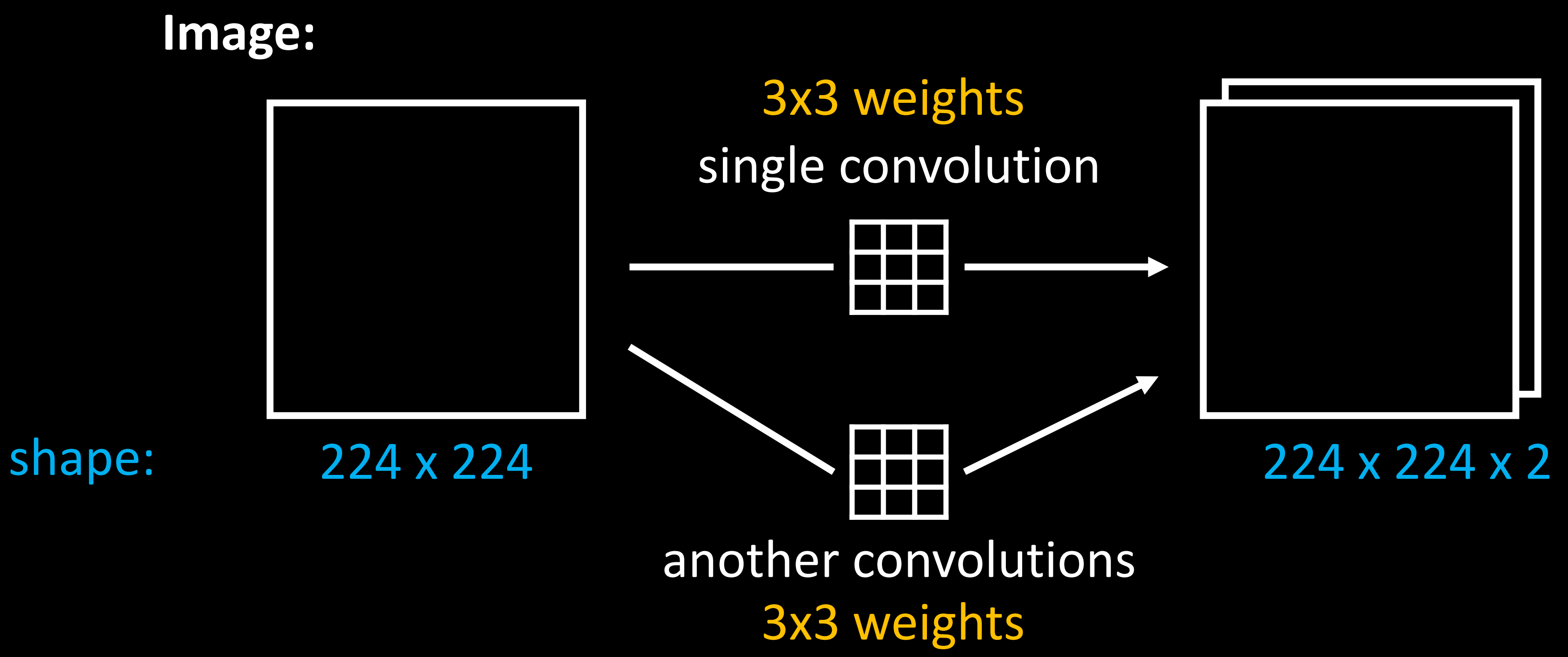
Shapes and weights!



- It can be slightly confusing to read what happens in NN architecture graphs – this should help ^^

One more detail before the pause

Shapes and weights!

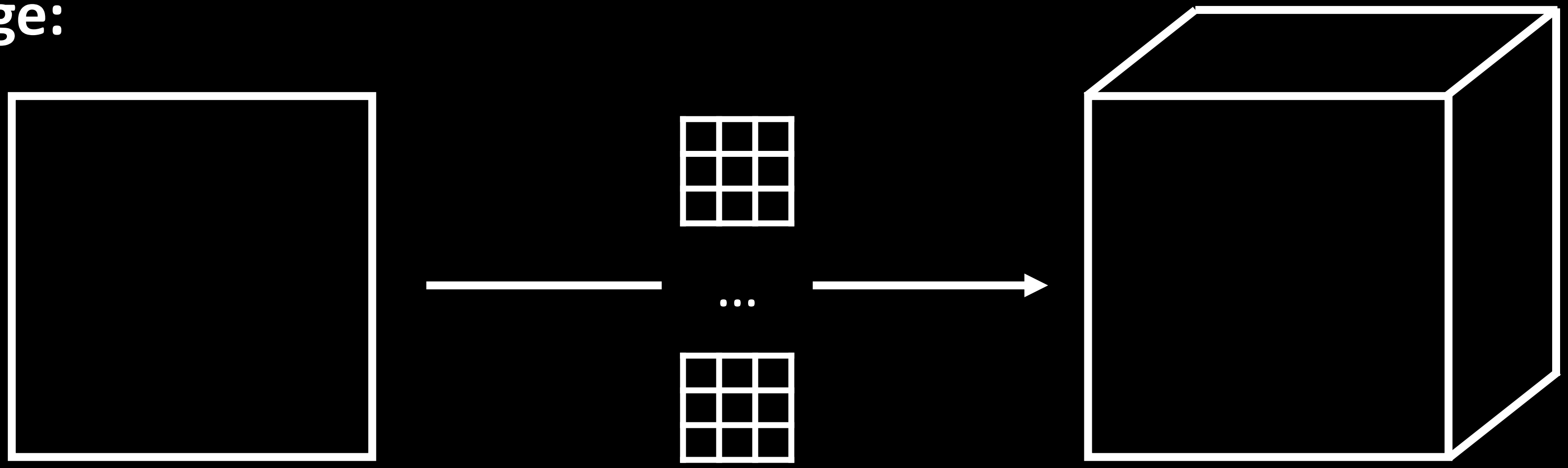


- It can be slightly confusing to read what happens in NN architecture graphs – this should help ^^

One more detail before the pause

Shapes and weights!

Image:



shape: 224 x 224 224 x 224 x M

M convolutions

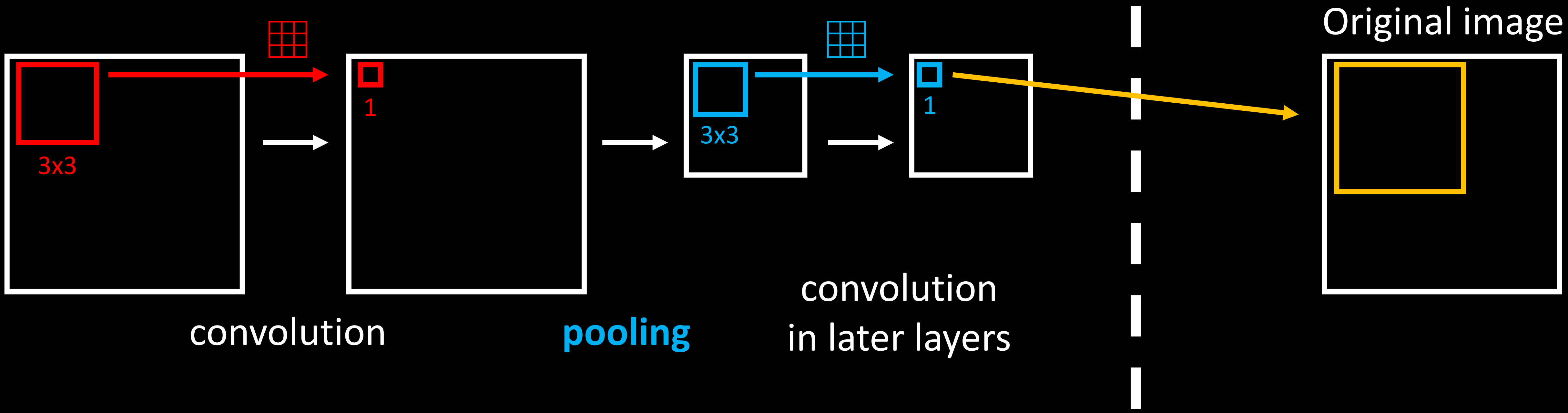
M x (3x3) weights

- It can be slightly confusing to read what happens in NN architecture graphs – this should help ^^

Pause 1

Specialized filters

- The (**Convolution -> Pooling**)$^{\text{REPEAT}}$ **combo**:



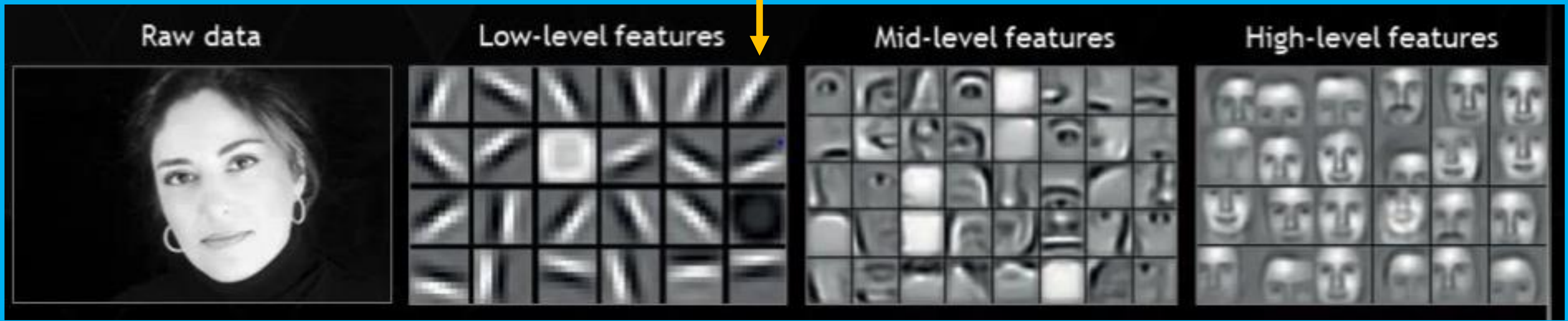
- Allows convolutional layers to **specialize on different scales**!

Specialized filters

- Local features
- Shapes, colors, edges, …

kernel visualizations

- Higher level features, more general
- Objects, concepts …



depth

Earlier layers

Later layers

Intuition for classification

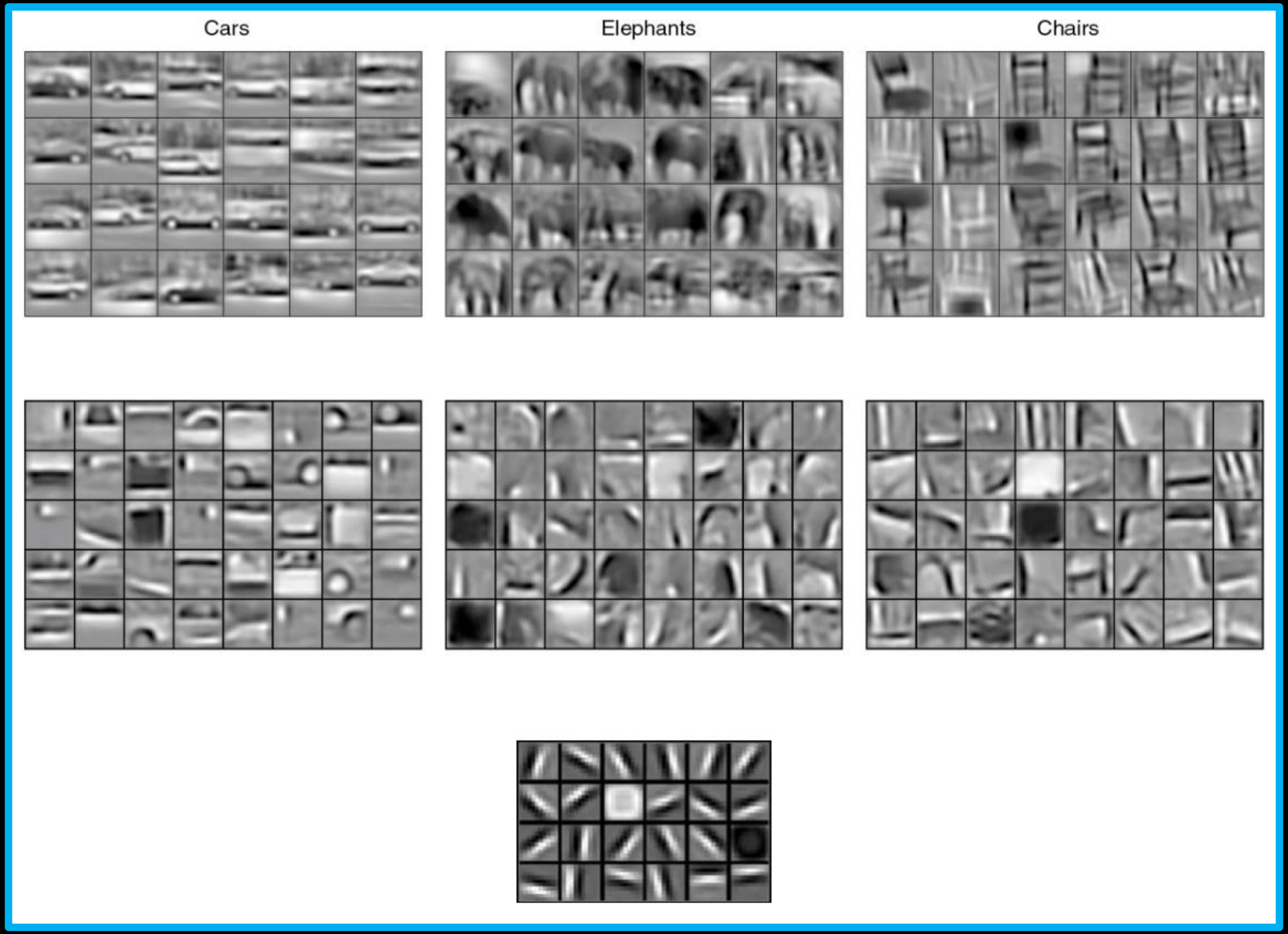
Classifications

High level features

Mid level features

Low level features

Inputs



Bonus: [novel methods of visualizations]

Intuition for generation

Let’s say we can hack into Convolutional layers of generative models.

- If we target convolutions of **high level features**:

large concepts

Intuition for generation

Let's say we can hack into Convolutional layers of generative models.

- If we target convolutions of **high level features**: ***large concepts***

- If we target convolutions of **low level features**: ***details, textures***

General Convolutional NN architecture:

- Most of CNN architectures follow this simple layout:

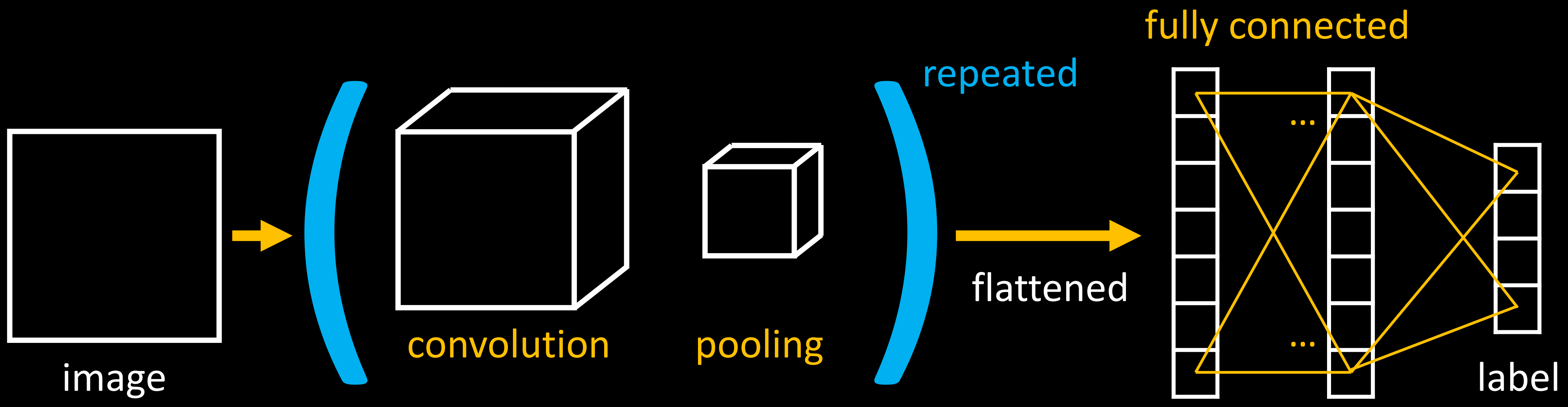


Image → (conv. – pool.)$^{\text{repeated}}$ → (fully connected)$^{\text{repeated}}$ → **label**

General Convolutional NN architecture:

Image → (conv. – pool.)$^{\text{repeated}}$ → (fully connected)$^{\text{repeated}}$ → **label**

$\vec{v}$

General Convolutional NN architecture:

Image → (conv. – pool.)$^{\text{repeated}}$ → (fully connected)$^{\text{repeated}}$ → **label**

$\vec{v}$

Feature extractor

- This part of the model is responsible for **extracting the relevant information** from the image and compressing it into a single vector (the flattened $\vec{v}$)

General Convolutional NN architecture:

Image $\rightarrow$ (conv. – pool.)$^{\text{repeated}}$ $\rightarrow$ (fully connected)$^{\text{repeated}}$ $\rightarrow$ **label**

$\vec{v}$

Feature extractor

- This part of the model is responsible for **extracting the relevant information** from the image and compressing it into a single vector (the flattened $\vec{v}$)

Classification

- This part is responsible for **classifying** the vector representation of the image and assigning it a label.

General Convolutional NN architecture:

Image $\rightarrow$ (conv. – pool.)$^{\text{repeated}}$ $\rightarrow$ (fully connected)$^{\text{repeated}}$ $\rightarrow$ **label**

Feature extractor

- This part of the model is responsible for **extracting the relevant information** from the image and compressing it into a single vector (the flattened $\vec{v}$)

$\vec{v}$

Classification

- This part is responsible for **classifying** the vector representation of the image and assigning it a label.

Feature vector

- A compressed representation of the original data.

General Convolutional NN architecture:

Image → (conv. – pool.)$^{\text{repeated}}$ → (fully connected)$^{\text{repeated}}$ → **label**

$\vec{v}$

Feature

Feature extractor

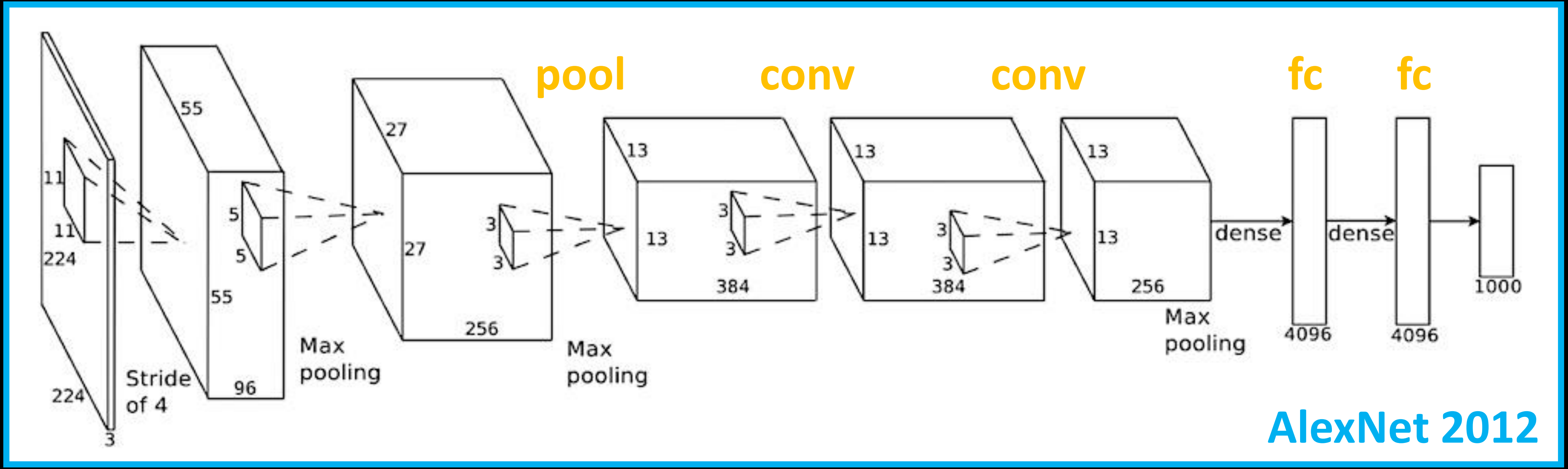
- This part of the model is responsible for **extracting the relevant information** from the image and compressing it into a single vector (the flattened $\vec{v}$)

Classification

- This part is responsible for **classifying** the vector representation of the image and assigning it a label.

This **differentiation of function emerges automatically** from the architecture design. It is not imposed manually by how we feed the data. (**Data driven** feature extraction and classification is cool!)

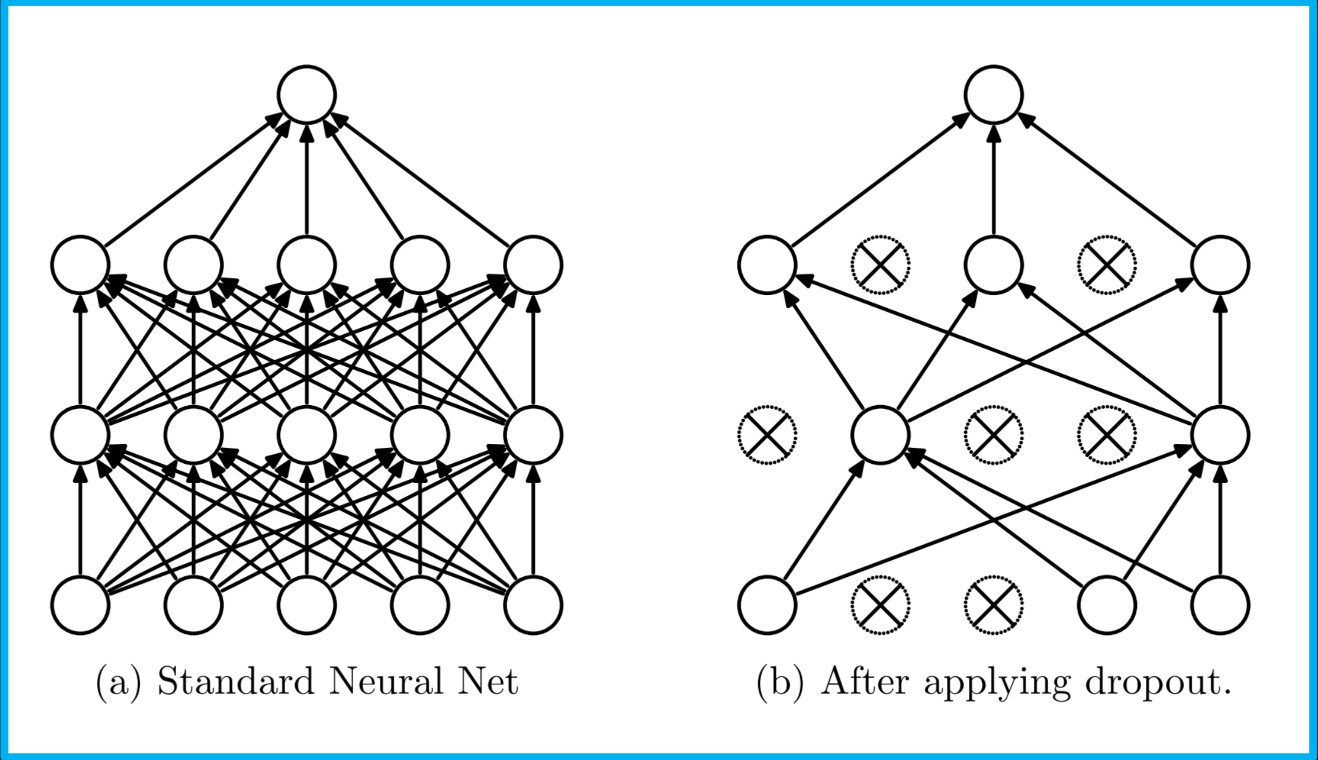
Real world example: AlexNet



By now, we know *most of* the layers used in this CNN architecture!

One more detail: the **Dropout layers**

Dropout layers



Dropout deletes connections (with chosen probability) during the training of a NN.

Why?

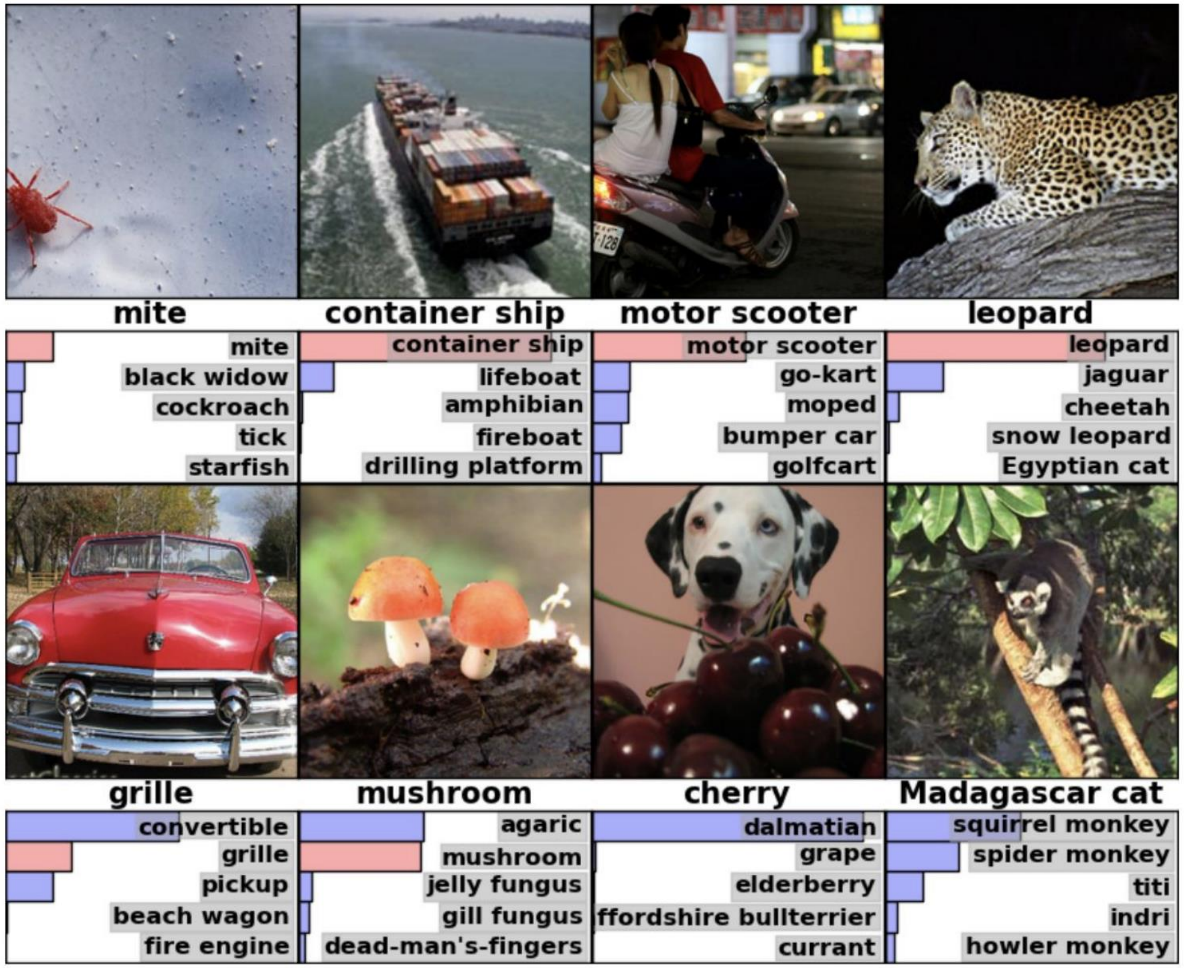
This prevents the network to create overly complicated mappings.

- Dropout is a widely used technique to **helps with overfitting** of NNs.

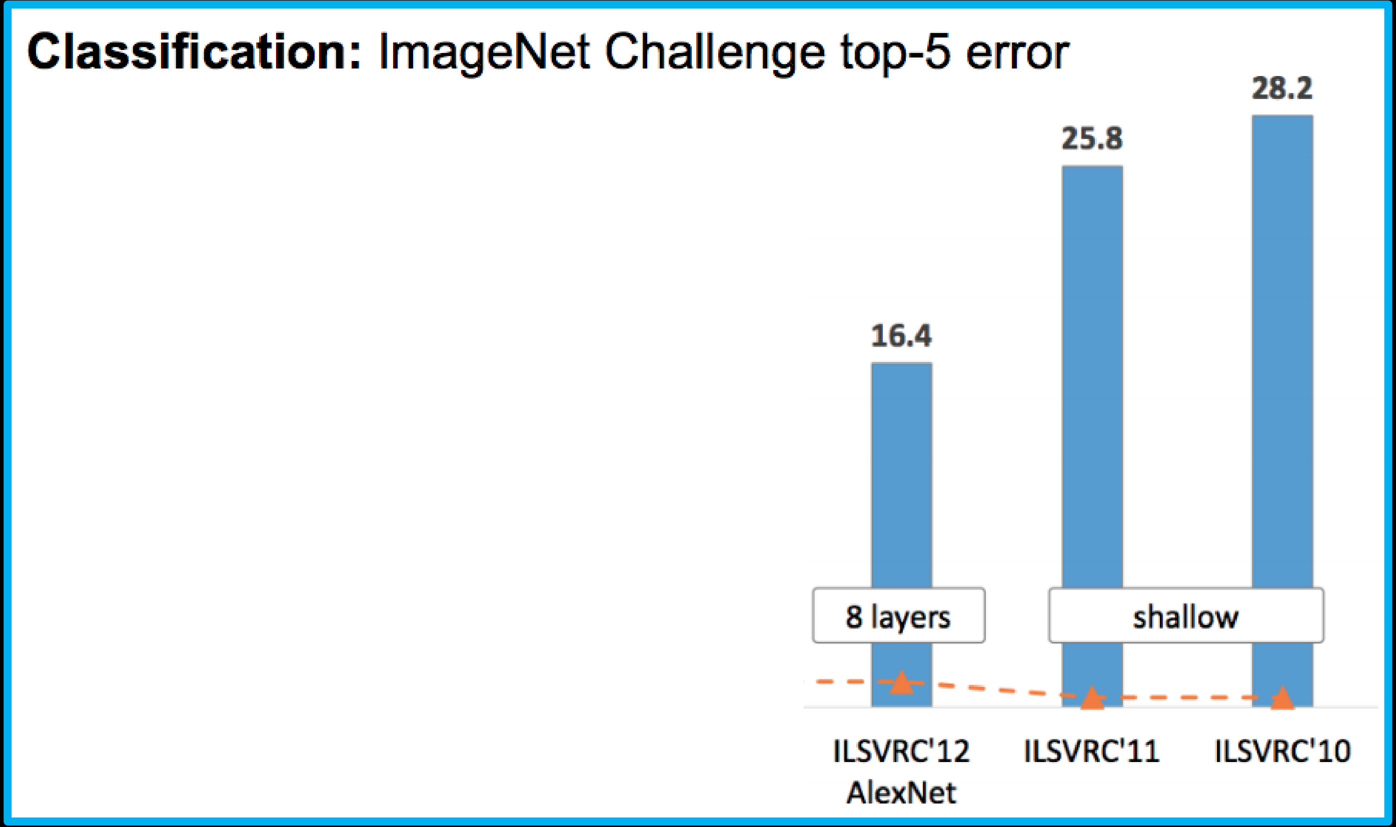
ImageNet competition

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



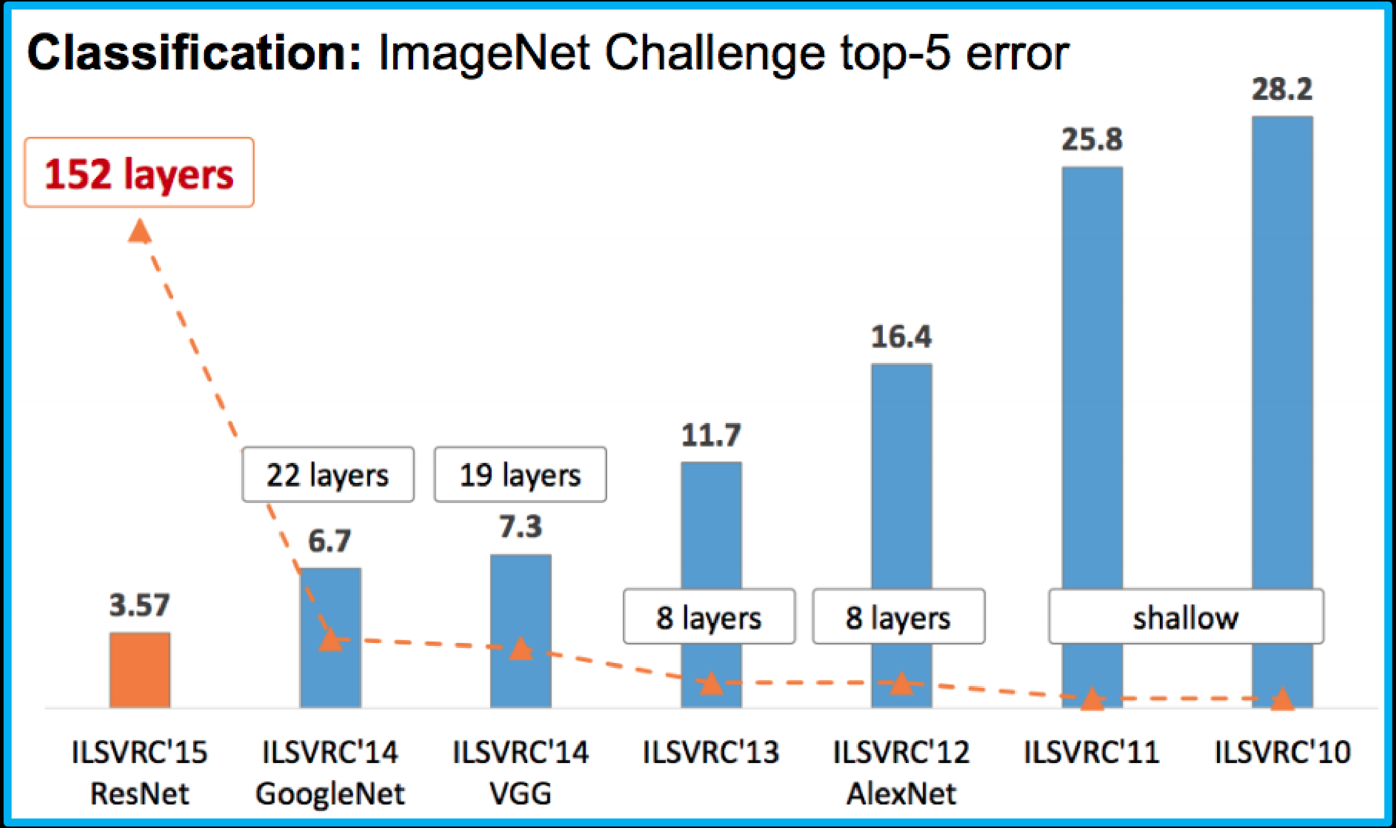
Context: the ImageNet competition



Top-5 error: each model gives top 5 most probable predictions. We count it if the correct class is at least in one of them.

BTW1: Human 5.0%

Context: the ImageNet competition



Top-5 error: each model gives top 5 most probable predictions. We count it if the correct class is at least in one of them.

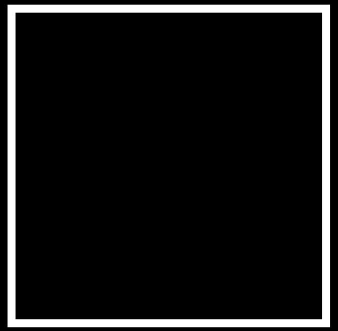
BTW1: Human 5.0%

BTW2: best today +- 2.3%

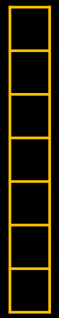
- Since **AlexNet**, all the following models used deep convolutional NNs.

General feature description

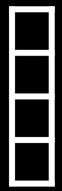
Image → Feature extractor → $\vec{v}$ → Classification → label



224x224x3 px



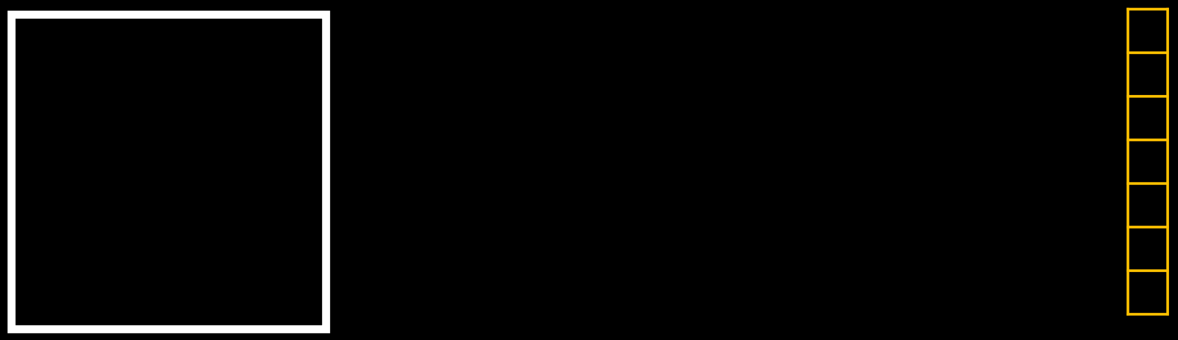
vector of 4096 real numbers



1000 classes

General feature description

Image → **Feature extractor** → $\vec{v}$



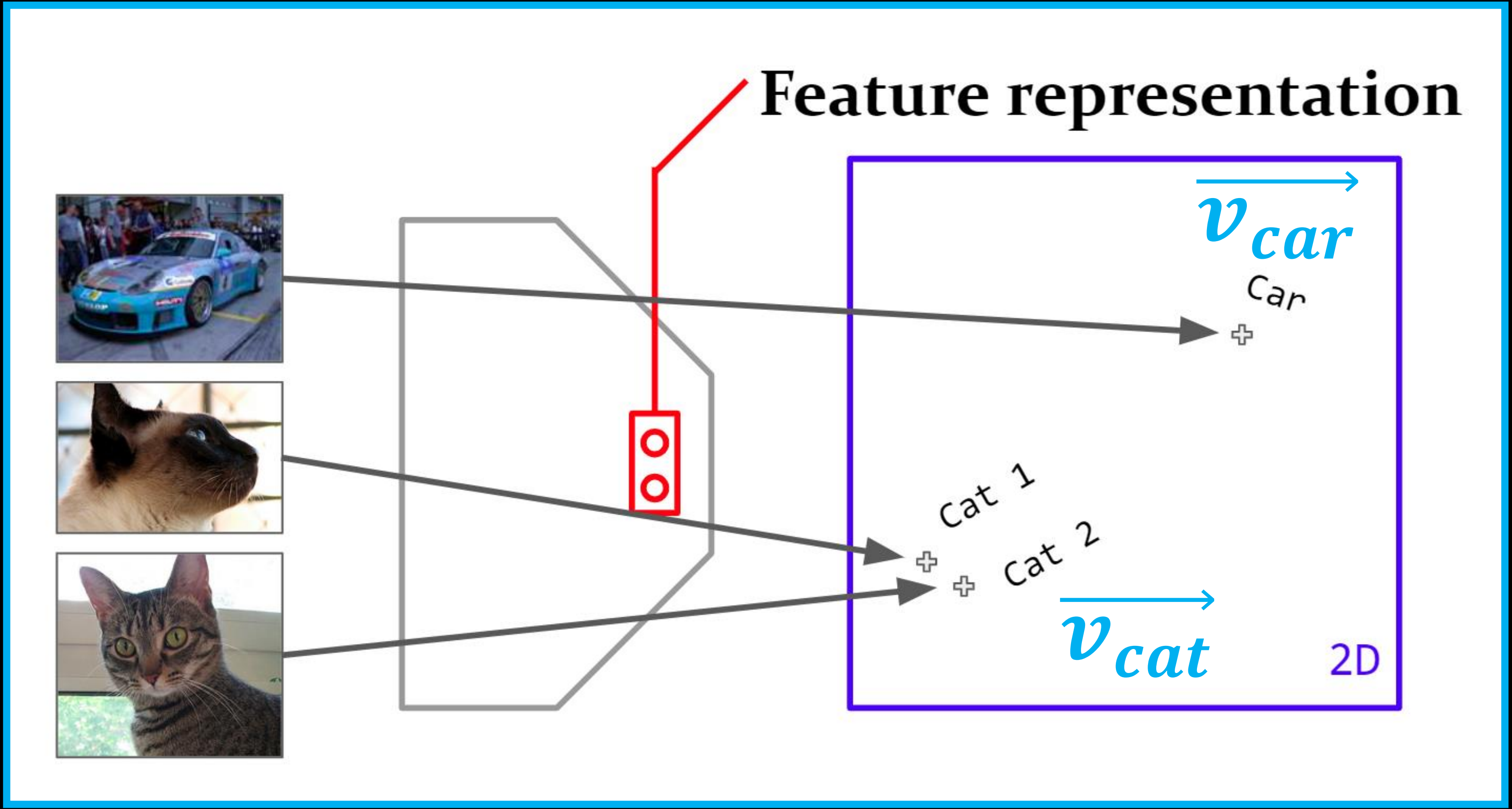
224x224x3 px

vector of 4096 real numbers

- This part of the model becomes useful as a general feature extractor (its learned **representation-ability is data driven** – depends on the dataset!)

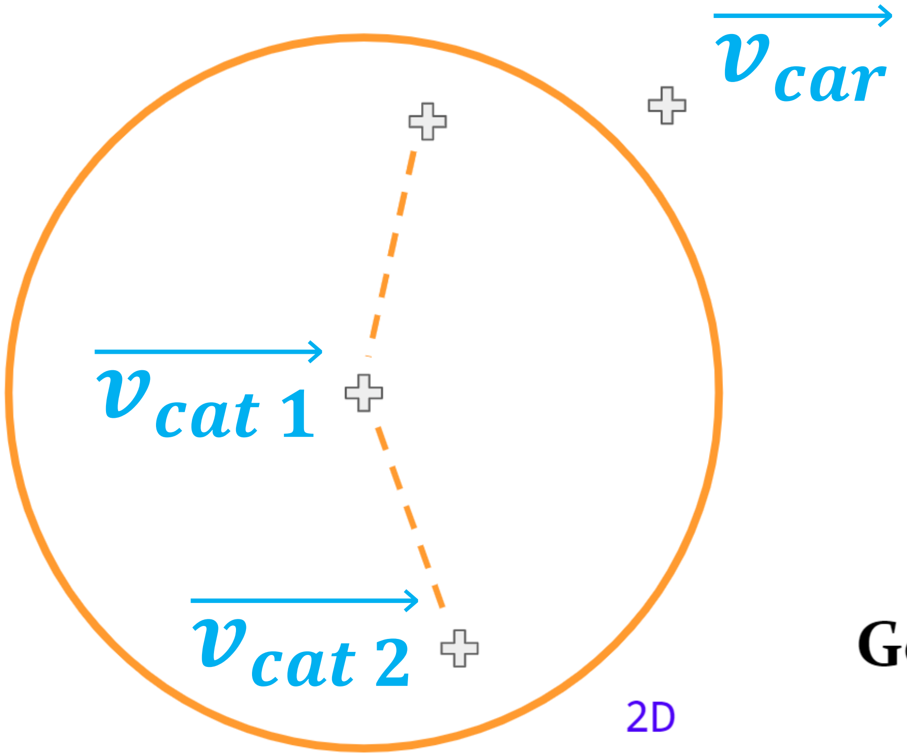
General feature description

Image → Feature extractor → $\vec{v}$ → Classification → label



General feature description

Image → Feature extractor → $\vec{v}$ → Classification → label

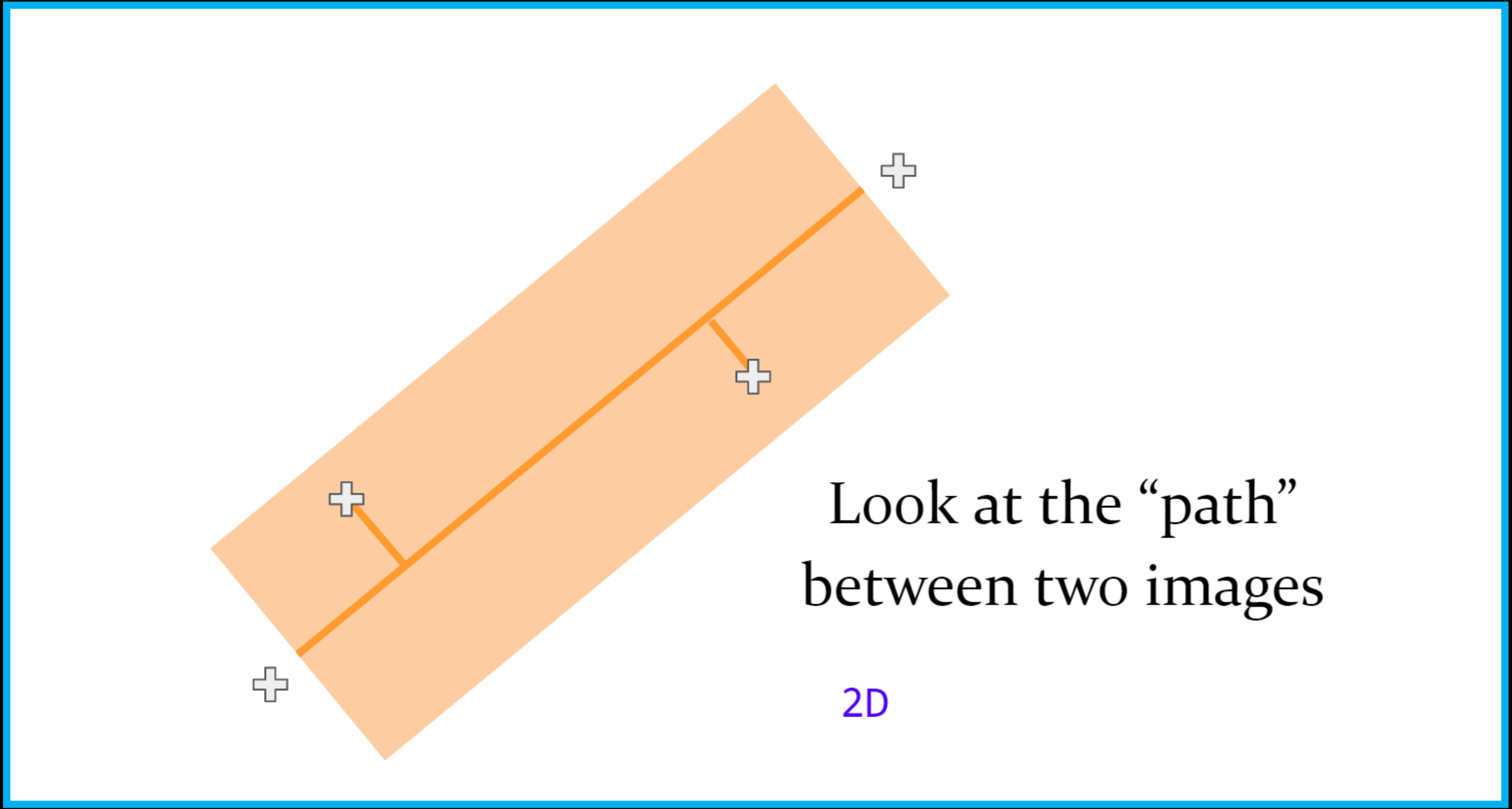


Search for
"nearby" images

Google Image search

General feature description

Image → Feature extractor → $\vec{v}$ → Classification → label



Mario Klingemann - X Degrees of Separation (2018)

Image 1 → Feature extractor → $\vec{v_1}$

Vectors in the path ← Feature extractor ← Other images

Image 2 → Feature extractor → $\vec{v_2}$

Mario Klingemann - X Degrees of Separation (2018)

Practicum: Convolutional NNs

Continue with code on our Github:

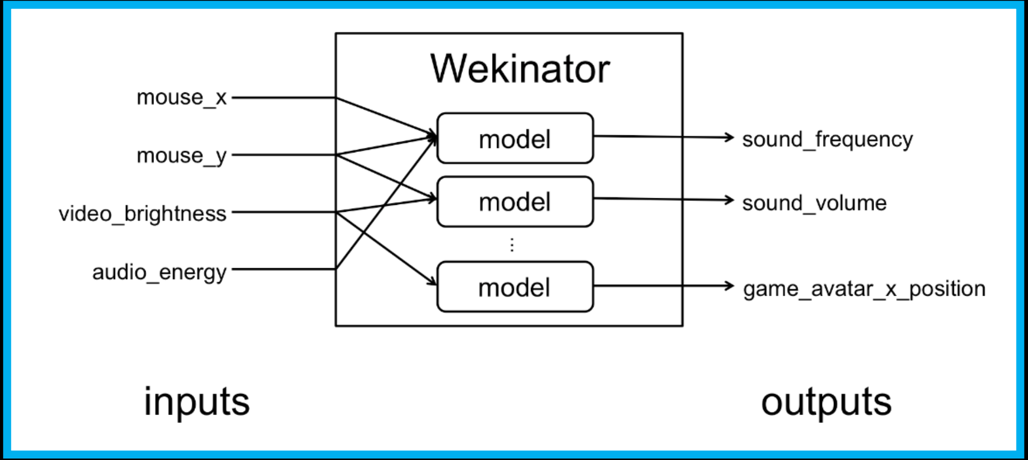
- github repo: github.com/previtus/cci_exploring_machine_intelligence
- notebook: ml03_cnn_image_search.ipynb

Next class

8.5. Bank holiday = no class!

15.5. Invited guest lecture by Rebecca Fiebrink

- Interaction with Machine Learning models using Wekinator
- Focus on Interaction Design!



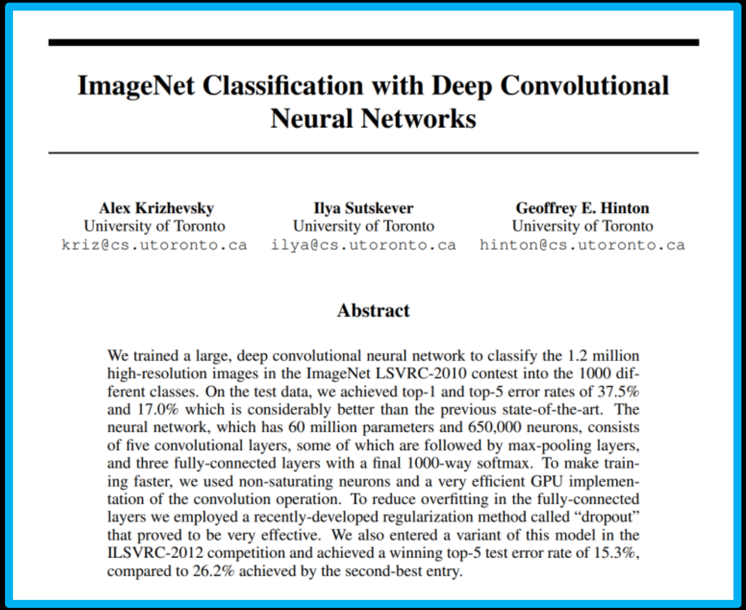
Homework reading:

Deadline 8.5.

<u>Please read</u>:

Everything, however, you can briefly skim through:

3.2 Training on Multiple GPUs
3.3 Local Response Normalization
3.4 Overlapping Pooling



ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

<u>Tasks</u>:

- In your words, **summarize the paper**.
- **Answer few** prepared **questions**.

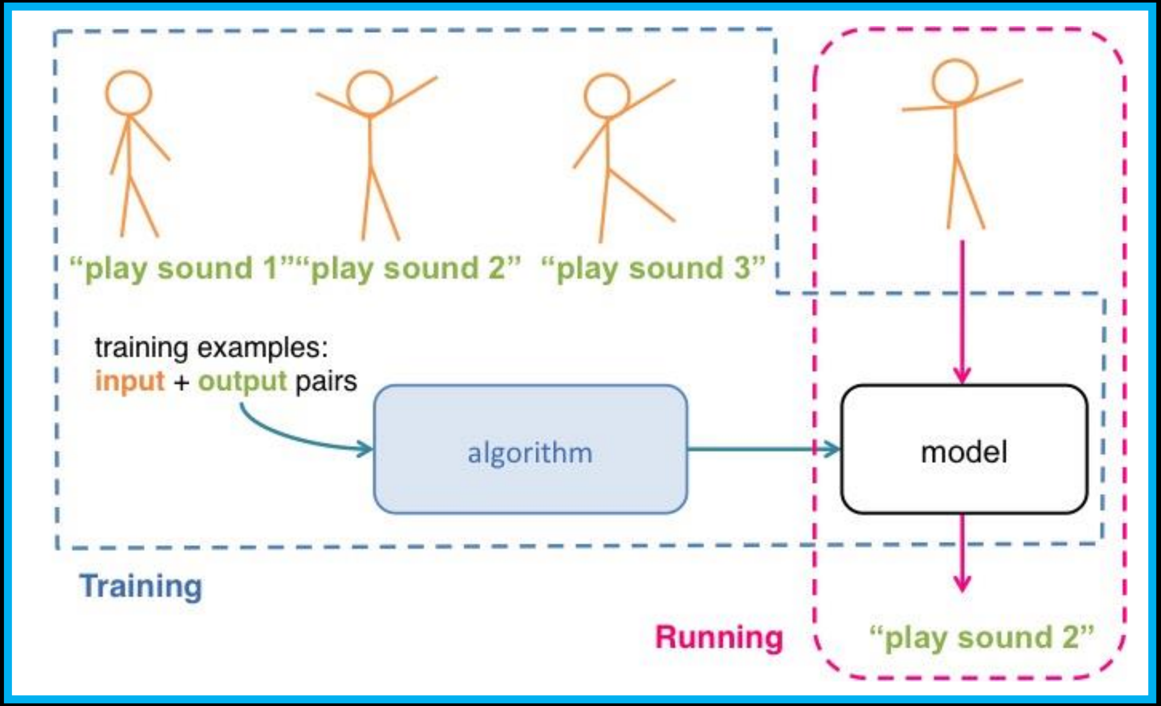
ImageNet Classification with Deep Convolutional Neural Networks – **A. Krizhevsky**, (2012)

PDF link: papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Pre-class preparation:

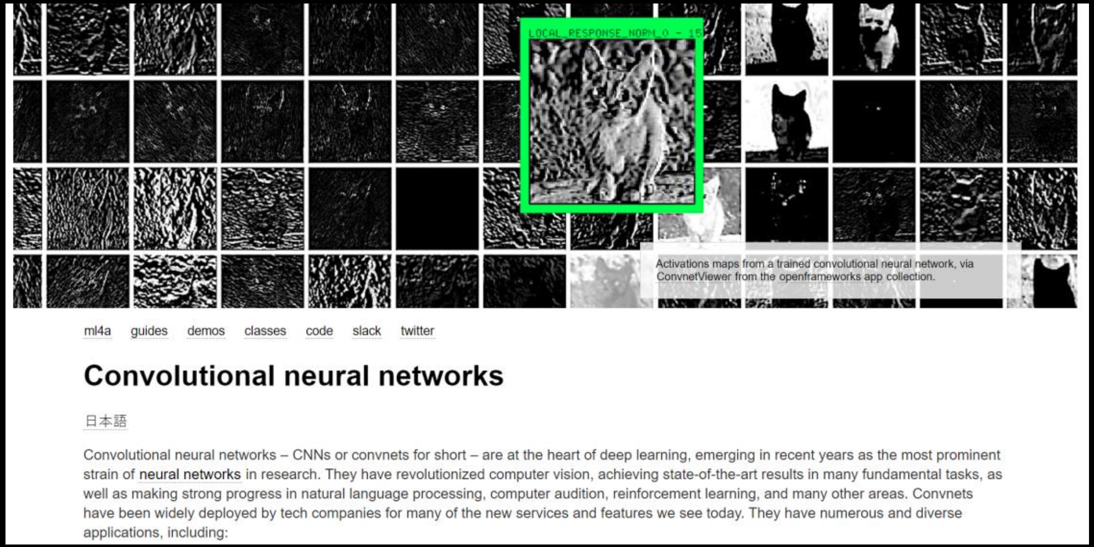
Deadline 15.5.

Get to know the basics of using **Wekinator**:



Start with the **walkthrough** at: www.wekinator.org/walkthrough/

Bonus links for further reading:



Convnets on ML4A: ml4a.github.io/ml4a/convnets/

Handy notebooks: ml4a.github.io/guides/

The end