LESSON 21

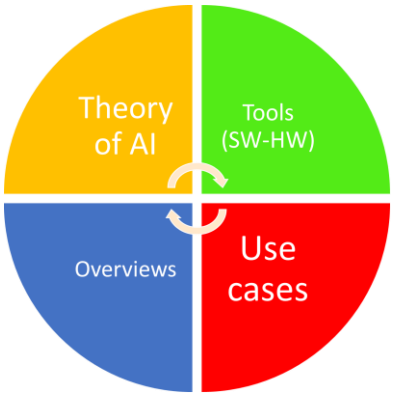
Models and Methods for Deep Learning
Convolutional Neural Networks (CNNs)
Coding CNNs in Keras
Examples of CNNs



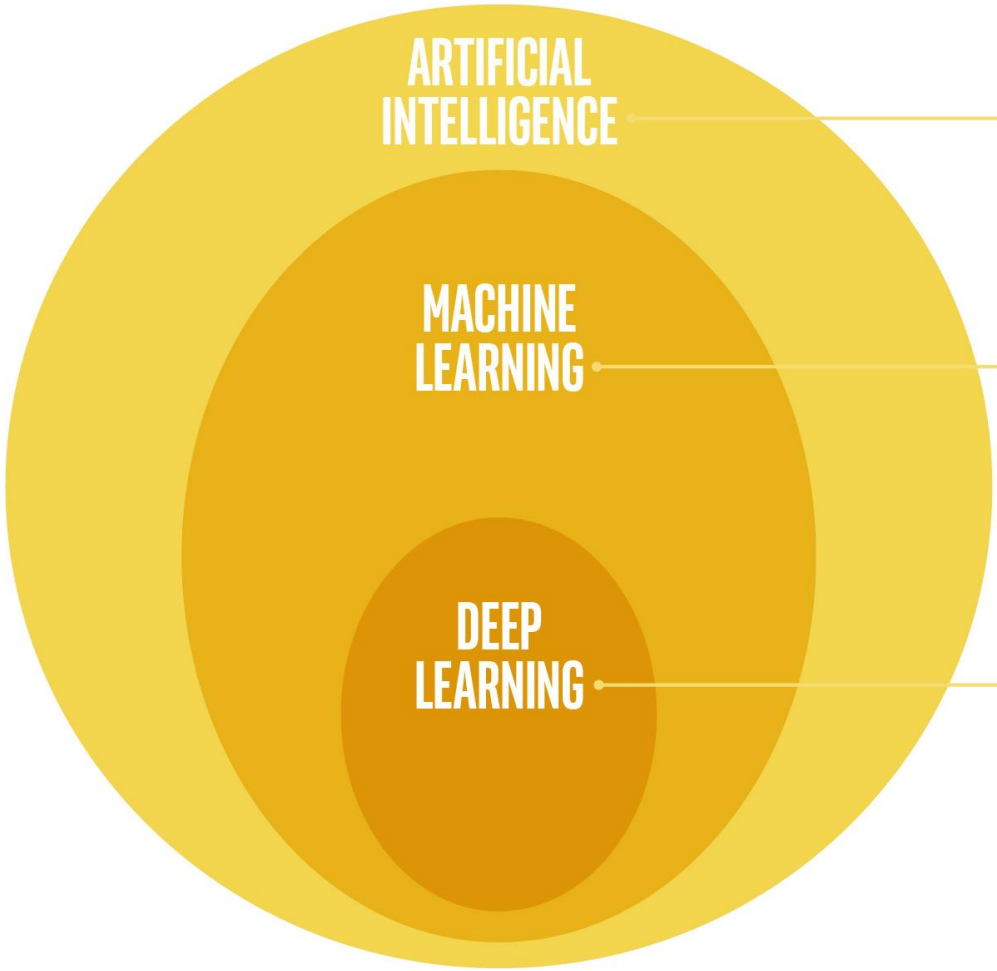

Outline

1. Deep learning VS classical neural networks
2. Convolutional Neural Networks
    1. Configuring
    2. Training
    3. Testing
3. Coding in Keras
4. Examples of large and public CNNs for applications

THEORY
Deep learning

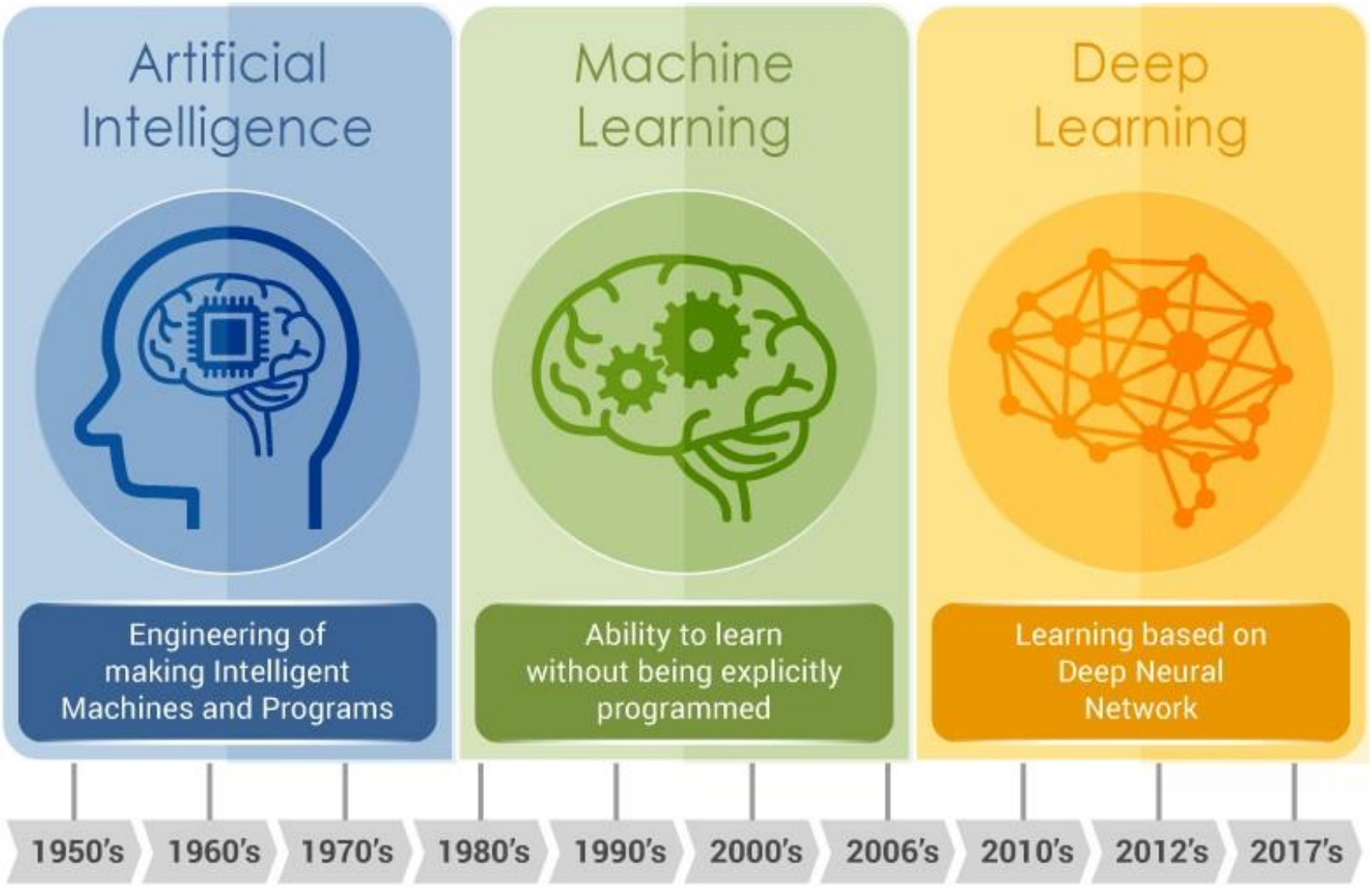


DL < ML < AI

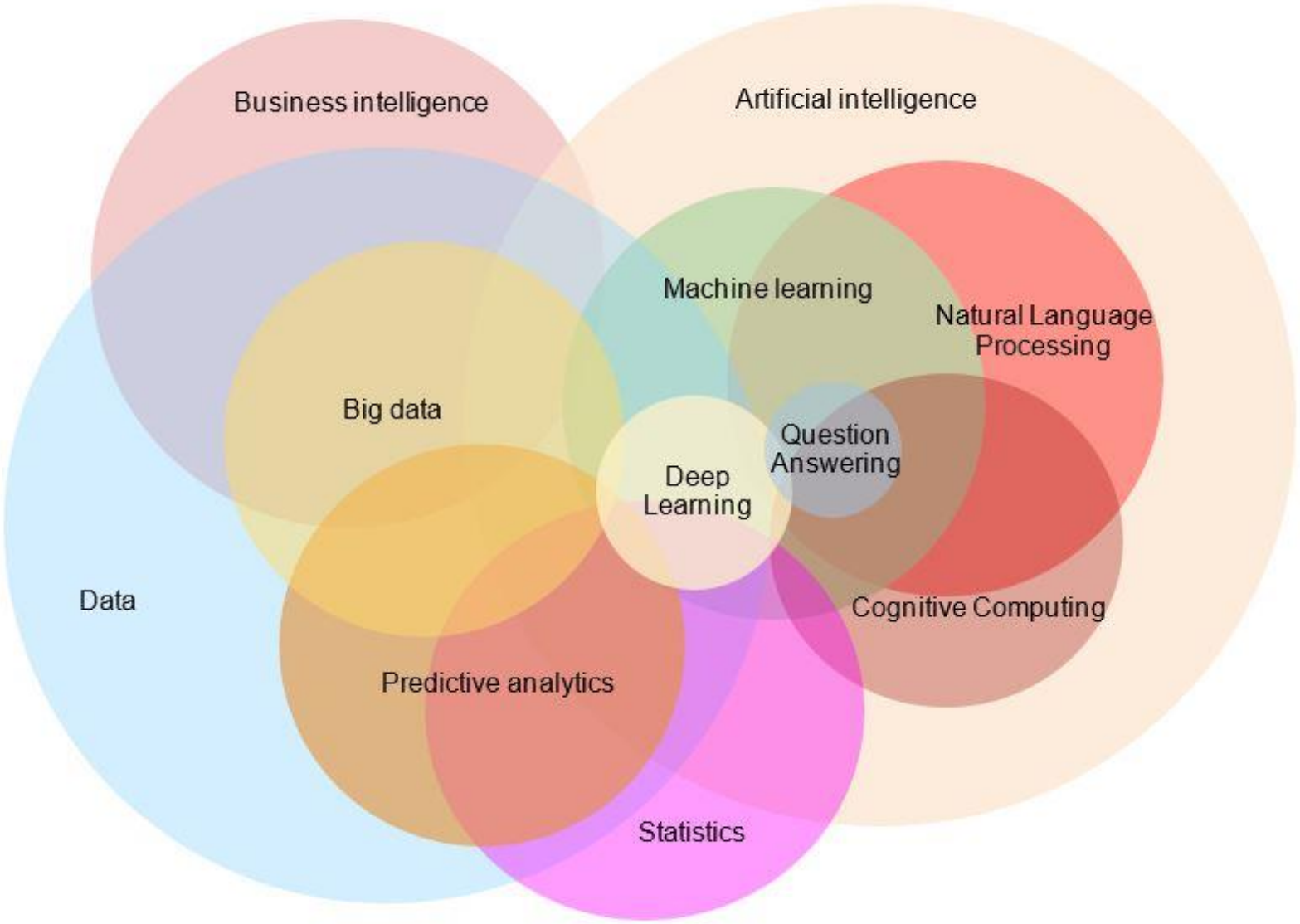


- **AI** is an umbrella term for machines capable of perception, logic, and learning.
- **Machine learning** employs algorithms that learn from data to make predictions or decisions, and whose performance improves when exposed to more data over time.
- **Deep learning** uses many-layered neural networks to build algorithms that find the best way to perform tasks on their own, based on vast sets of data.

«Deep» timeline

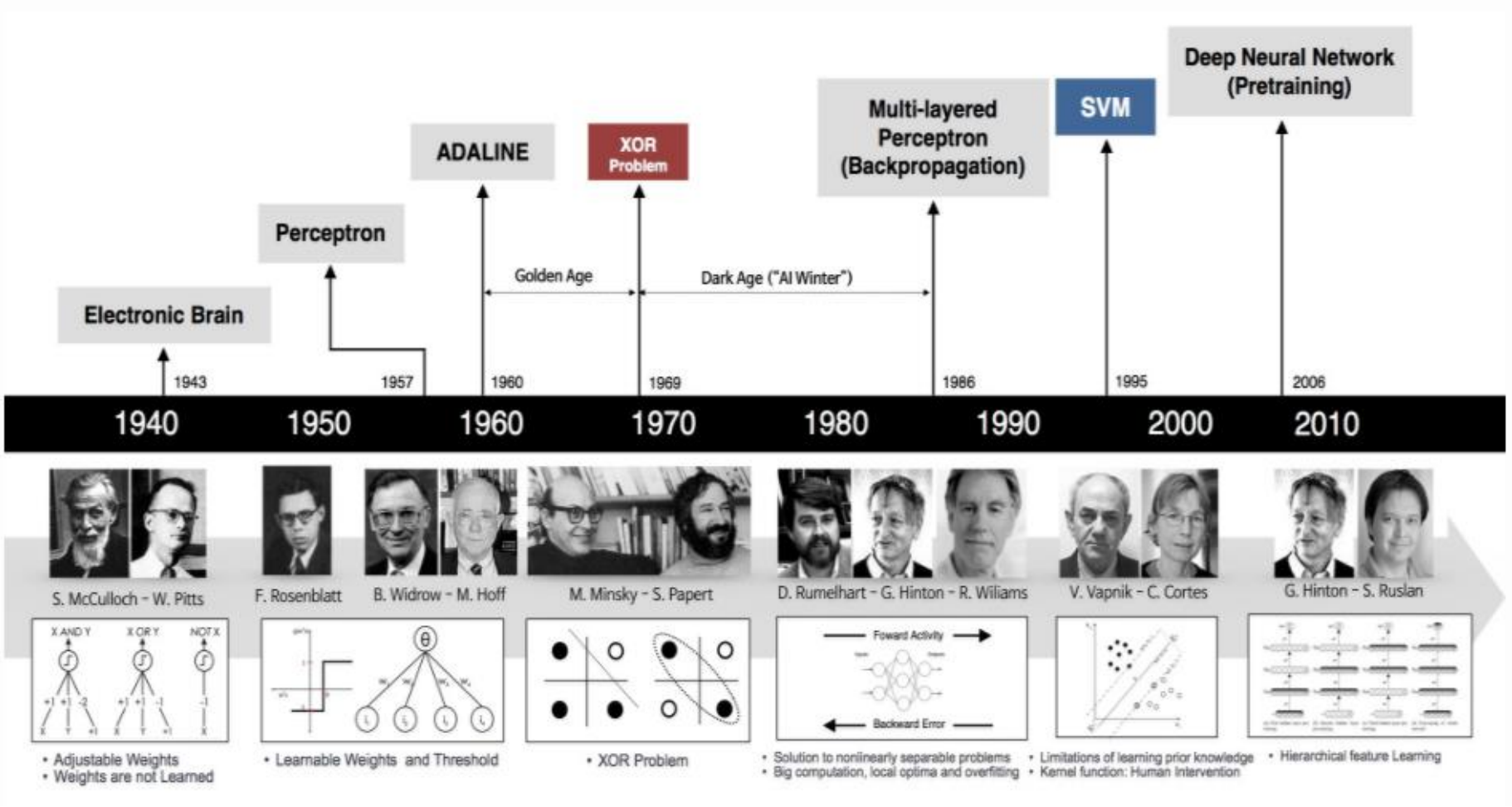


A «deeper» discussion



The road map

Between 2006 and 2012 the research group led by **Geoffrey Hinton** (University of Toronto) parallelized the ANNs algorithms to parallel architectures.
Increased number of layers, neurons, and model parameters in general (even over than 10 million) allowing to compute massive amounts of data through the system to train it.
→ 2012 Alexnet



Deep learning is..

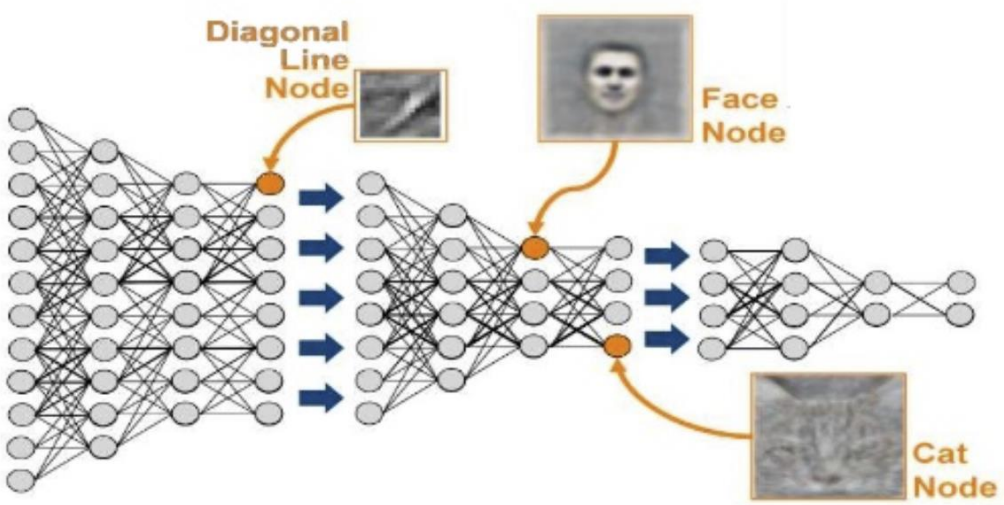
A class of algorithms

- Using different non linear herarchical layers with
  - *Feature extraction*
  - *Transformation*
- Algorithm can be
  - **Supervised** (classifiers, regressors)
  - Or and **unsupervised** (pattern analysis)

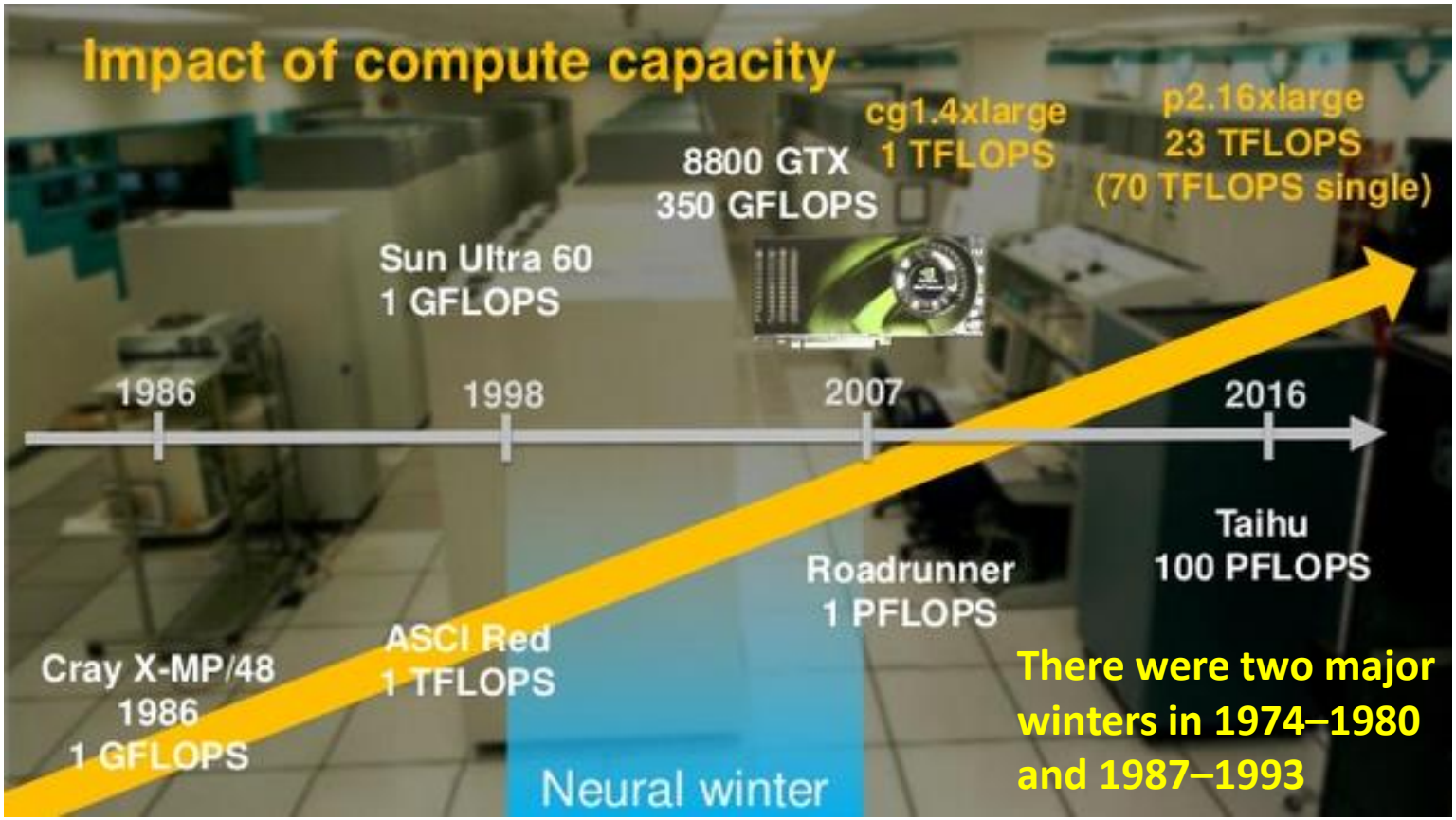
Deep learning is about

Functions in the hierarchical layers

- **are automatically created by the learning method**
- **high level features are derived/created from the low-level features creating an** hierarchical description of the input



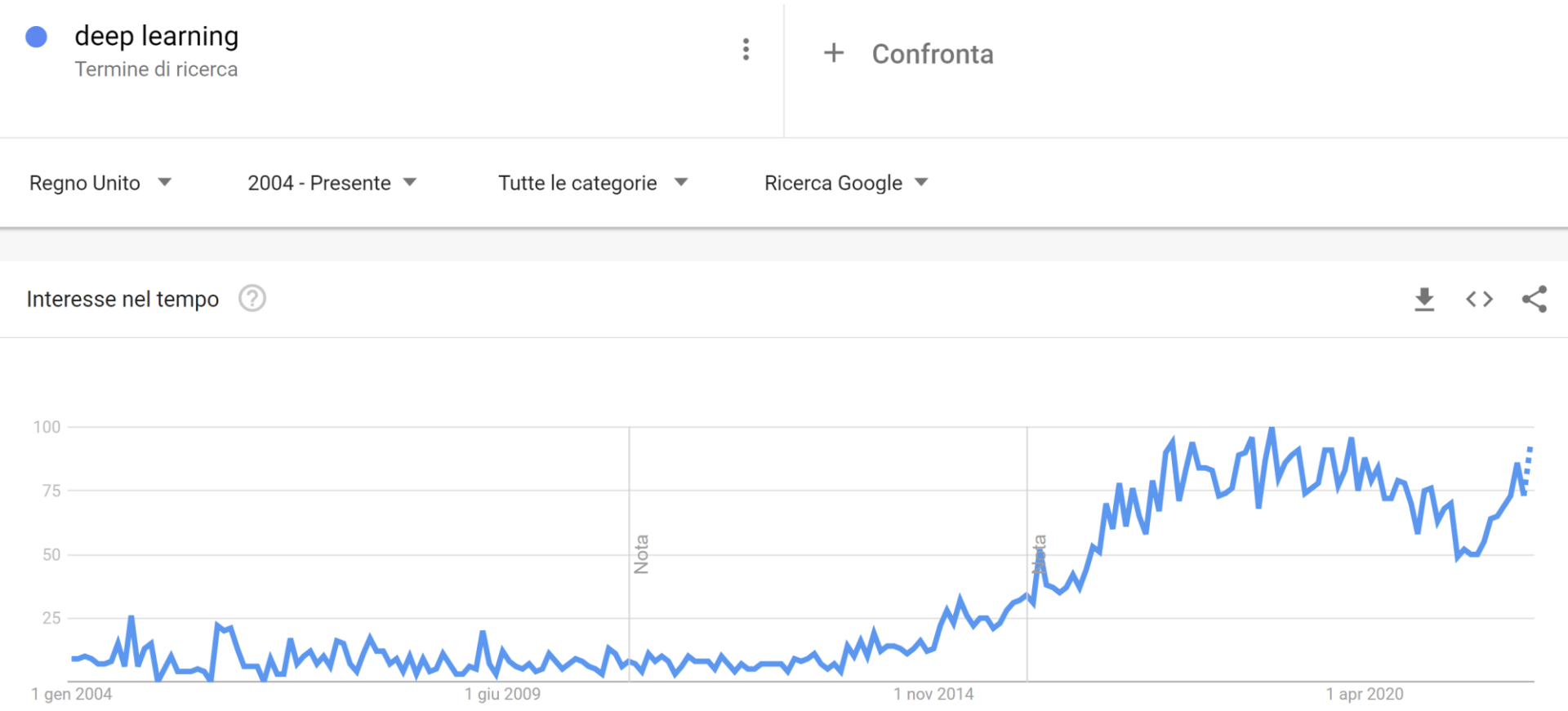
Deeplearn needs resources..



In the history of artificial intelligence, an AI winter is a period of reduced funding and interest in artificial intelligence research

https://trends.google.com/trends/explore?date=all&geo=GB&q=deep%20learning

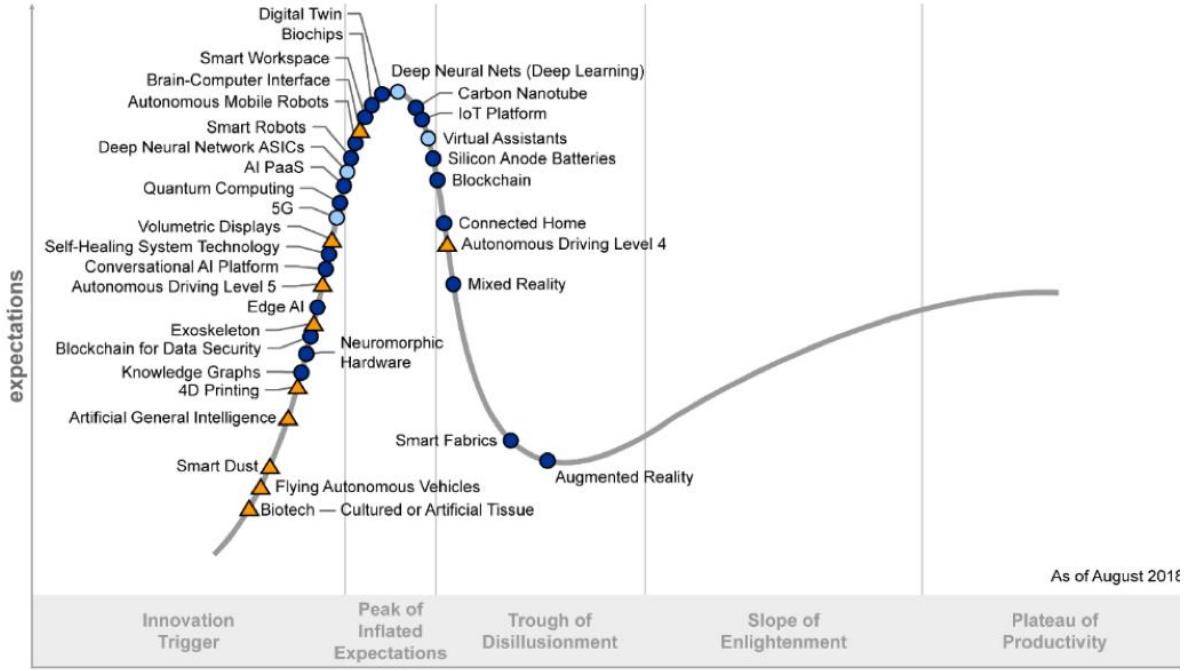
Deep learning: trend



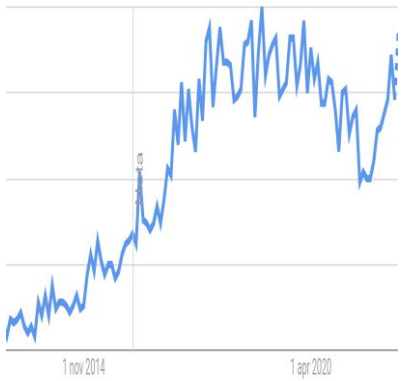
The Google search index since 2004 Picture source: Google Trends

https://trends.google.com/trends/explore?date=all&geo=GB&q=deep%20learning

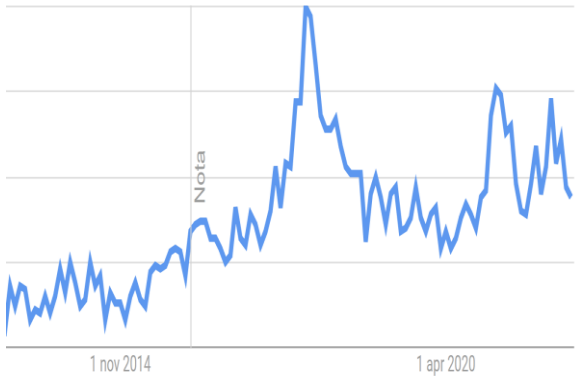
Deep learning: trend



Deep Learning

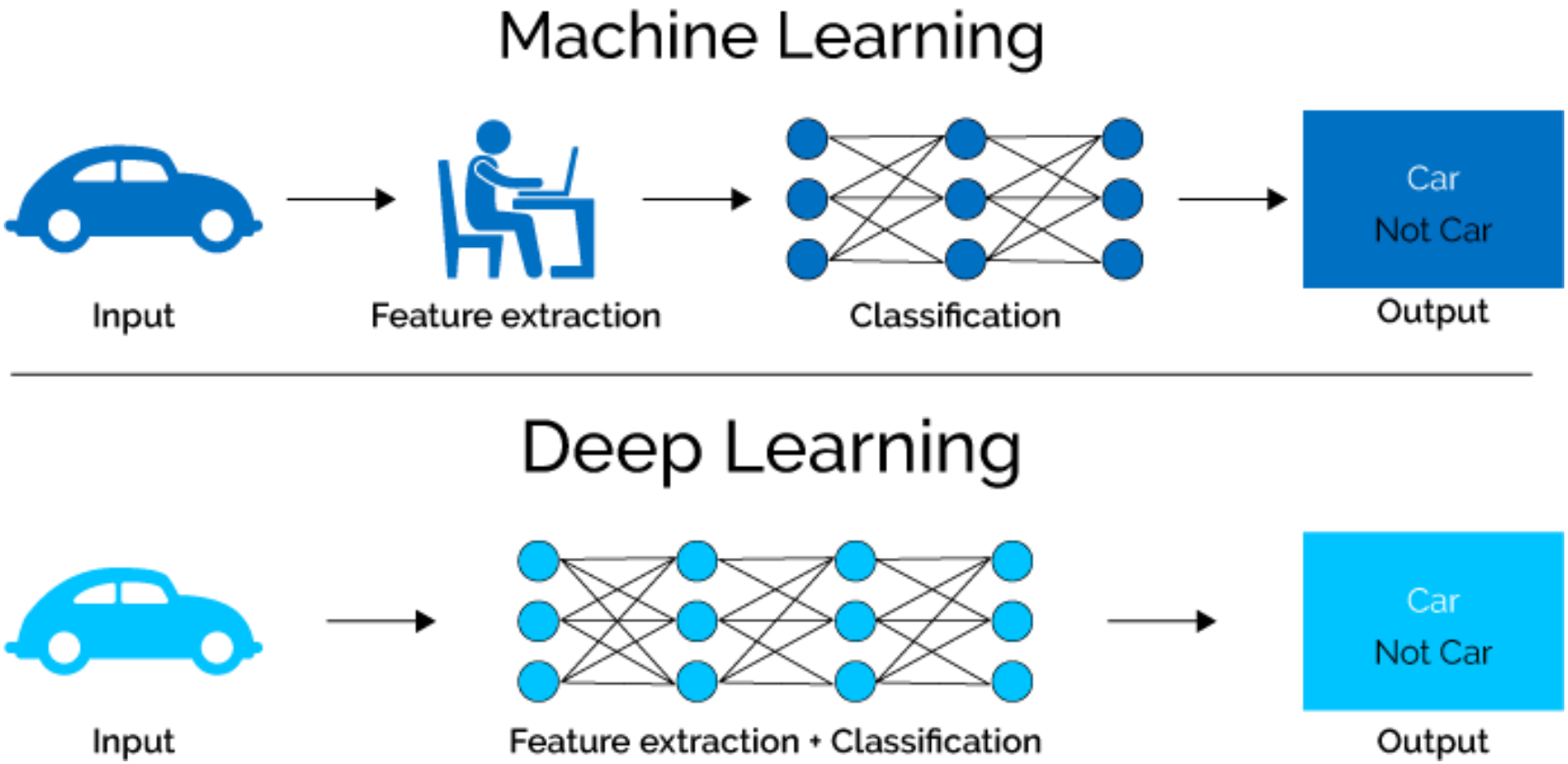


Blockchain

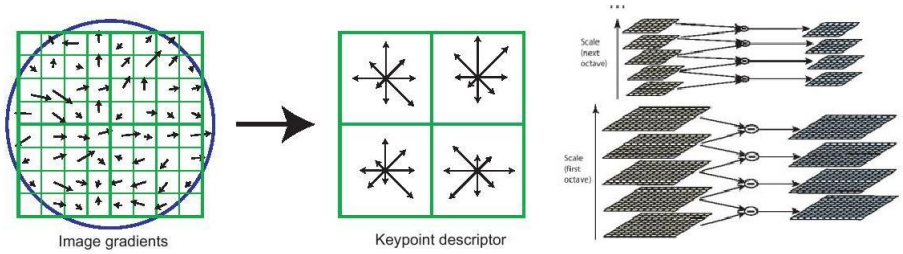


The Google search index since 2011 VS Gartner Hype plot….

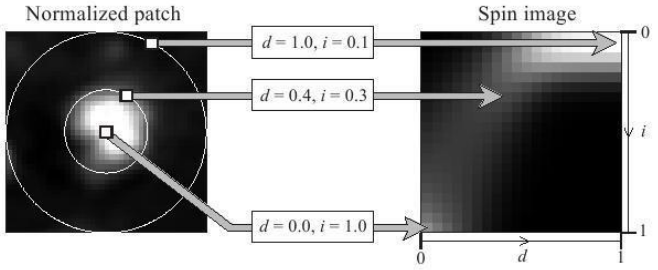
ML vs Deep Learning



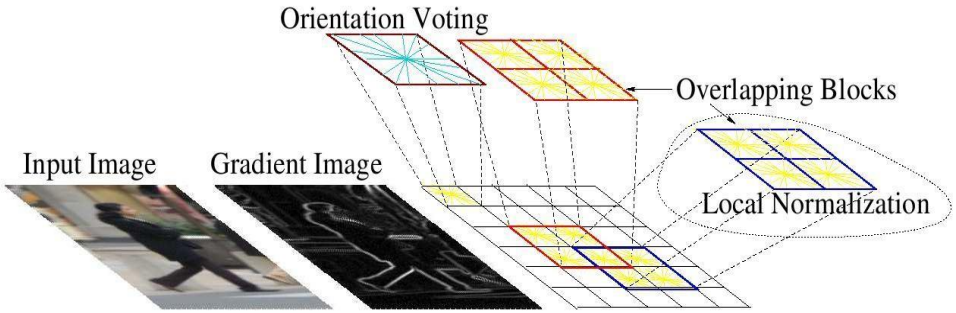
Designer Feature extraction



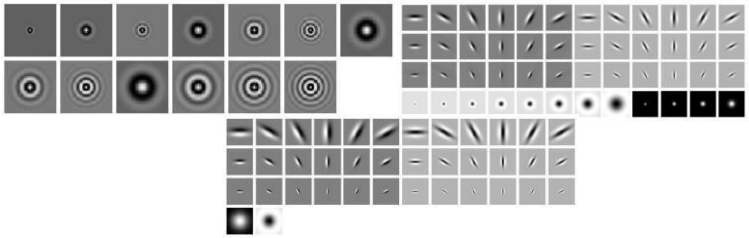
SIFT



Spin image



HoG



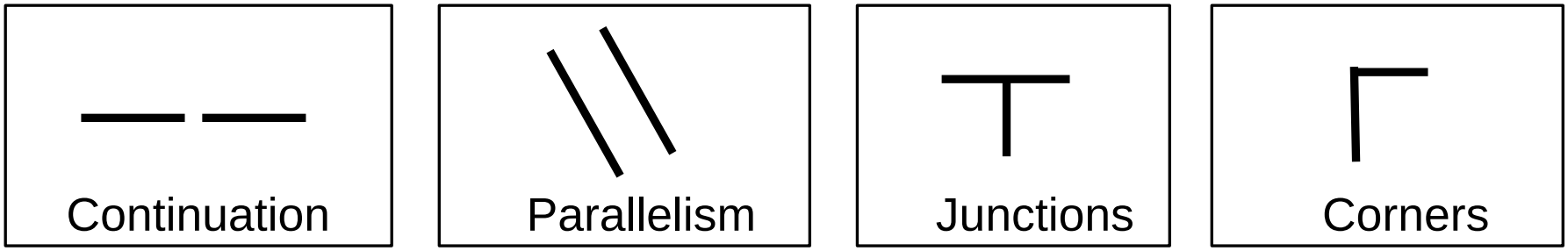
Textons

and many others:

SURF, MSER, LBP, Color-SIFT, Color histogram, GLOH, …..

Designer
Mid level representations

- Mid-level cues



"Tokens" from Vision by D.Marr:

- Object parts:

- Difficult to hand-engineer→ What about learning them?

Learning Feature Hierarchy (1/2)

- Learn hierarchy
- All the way from pixels → classifier
- One layer extracts features from output of previous layer

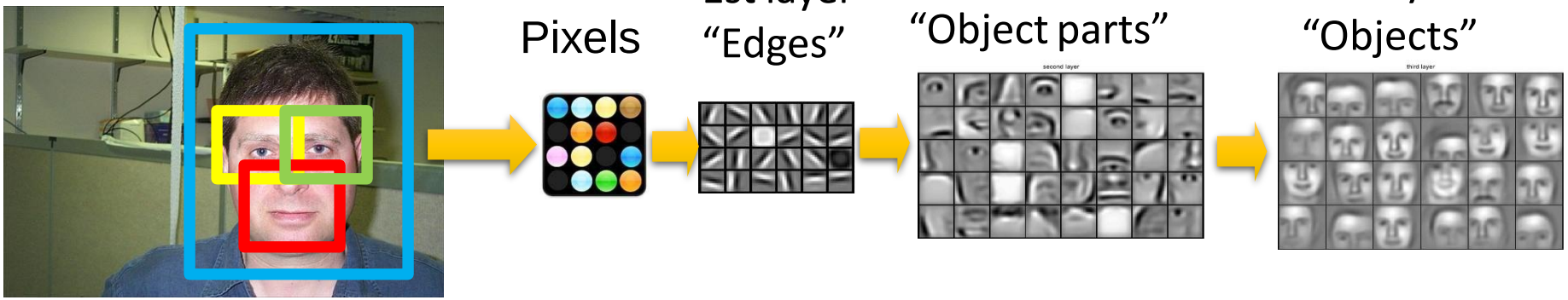
- Train all layers jointly

Learning Feature Hierarchy (2/2)

Learn **useful higher-level features** from images

Feature representation

Input data

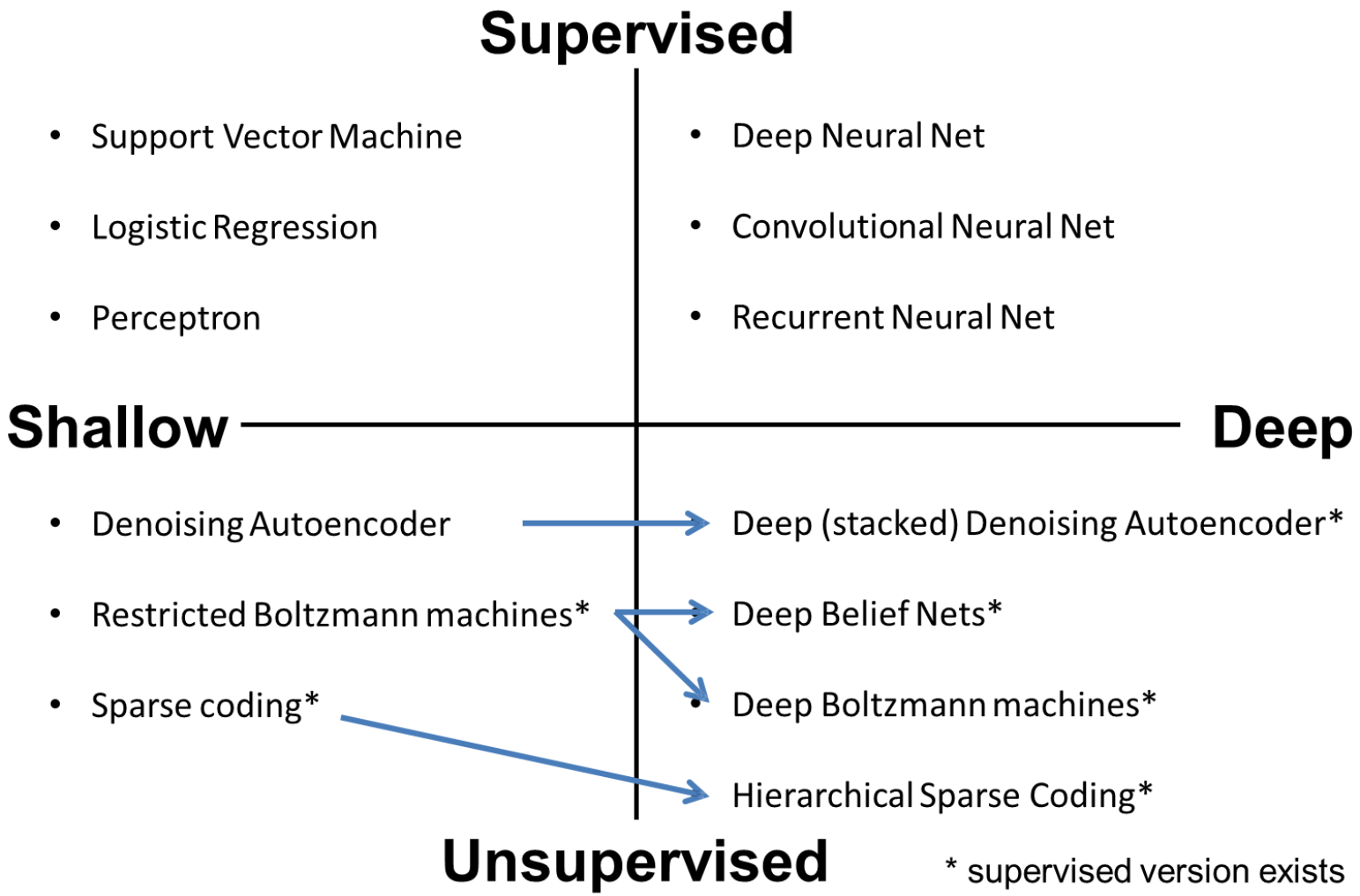


Face Detector

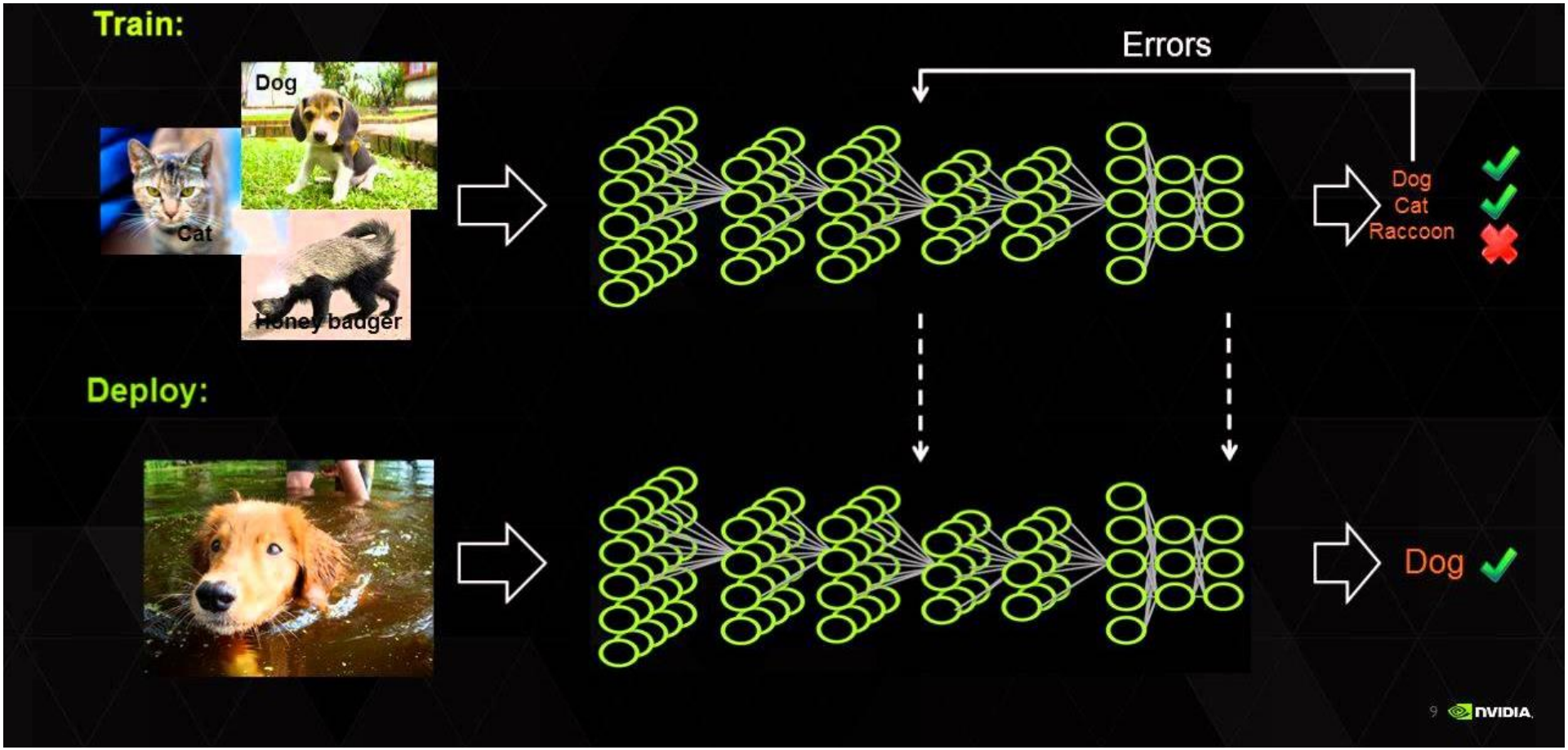
=

«Is there a face in my input»?

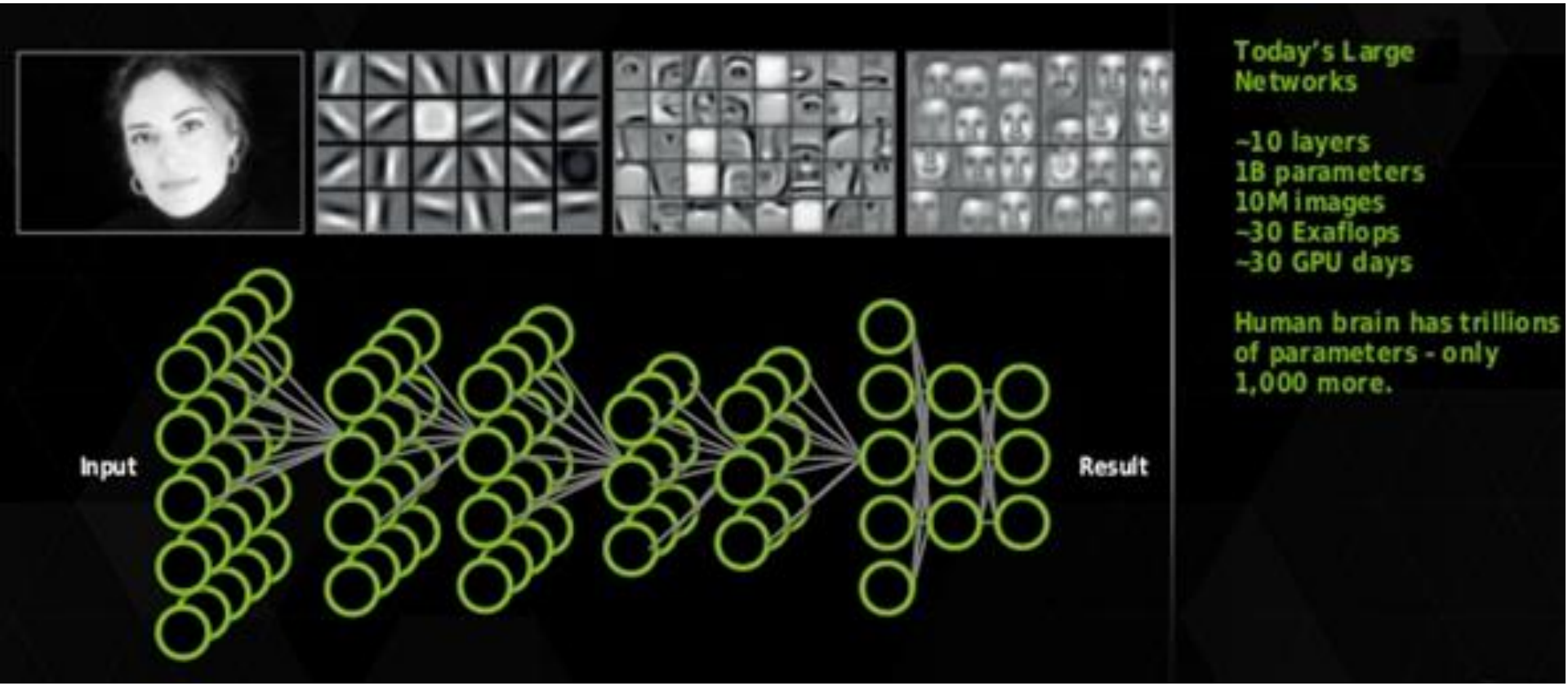
Taxonomy


Supervised
Support Vector Machine
Logistic Regression
Perceptron
Deep Neural Net
Convolutional Neural Net
Recurrent Neural Net
Shallow
Deep
Denoising Autoencoder
Restricted Boltzmann machines*
Sparse coding*
Deep (stacked) Denoising Autoencoder*
Deep Belief Nets*
Deep Boltzmann machines*
Hierarchical Sparse Coding*
Unsupervised
* supervised version exists

Deep Learning approach

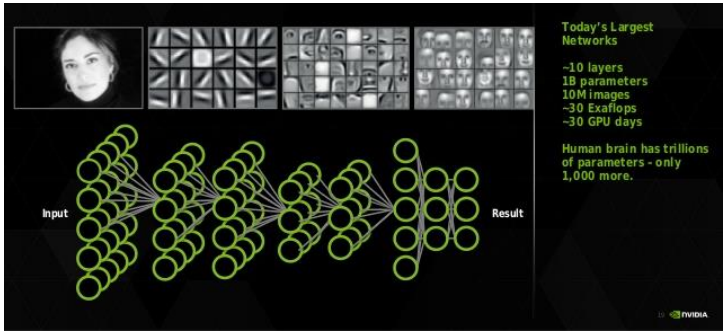


What makes Deep Learning deep?

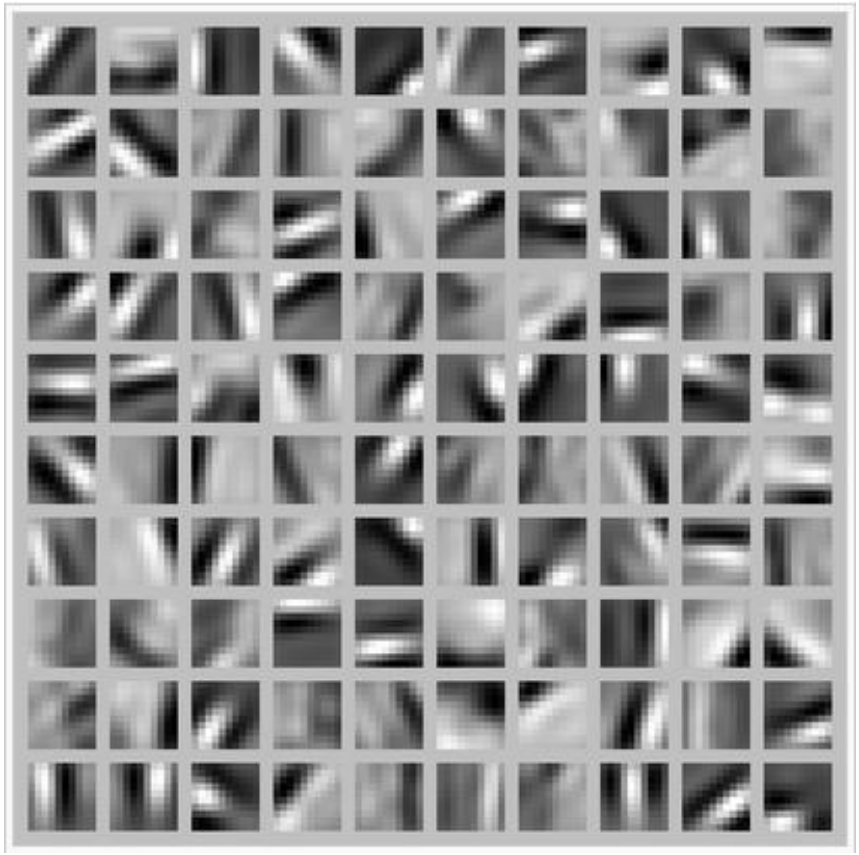


Deep Learning tasks

Explainability



- Example:
  - "view" of a learned vision feature layer
  - each square in the figure shows the input image that maximally activates one of the 100 unit



Why Deep Learning

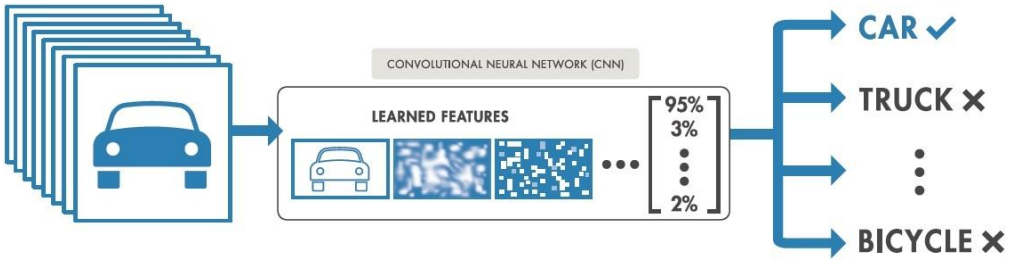
- Biological Plausibility – e.g. Visual Cortex
- Hastad proof - Problems which can be represented with a polynomial number of nodes with *k* layers, may require an exponential number of nodes with *k*-1 layers (e.g. parity) → many layers is good!
- Highly varying functions can be **efficiently** represented with deep architectures
  - Less weights/parameters to update than a less efficient shallow representation
- Sub-features created in deep architecture can potentially be shared between multiple tasks
  - Type of Transfer/Multi-task learning

Early Work

- Fukushima (1980) – Neo-Cognitron
- LeCun (1998) – Convolutional Neural Networks (CNN)
  - Similarities to Neo-Cognitron
- Many layered MLP with backpropagation
  - Tried early but without much success
    - Very slow
    - Diffusion of gradient
  - Very recent work has shown significant accuracy improvements by "patiently" training deeper MLPs with BP using fast machines (GPUs)

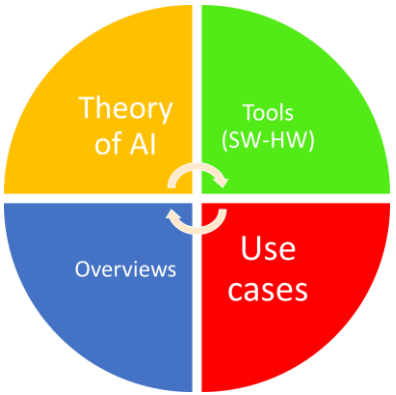
    → may be best most general approach



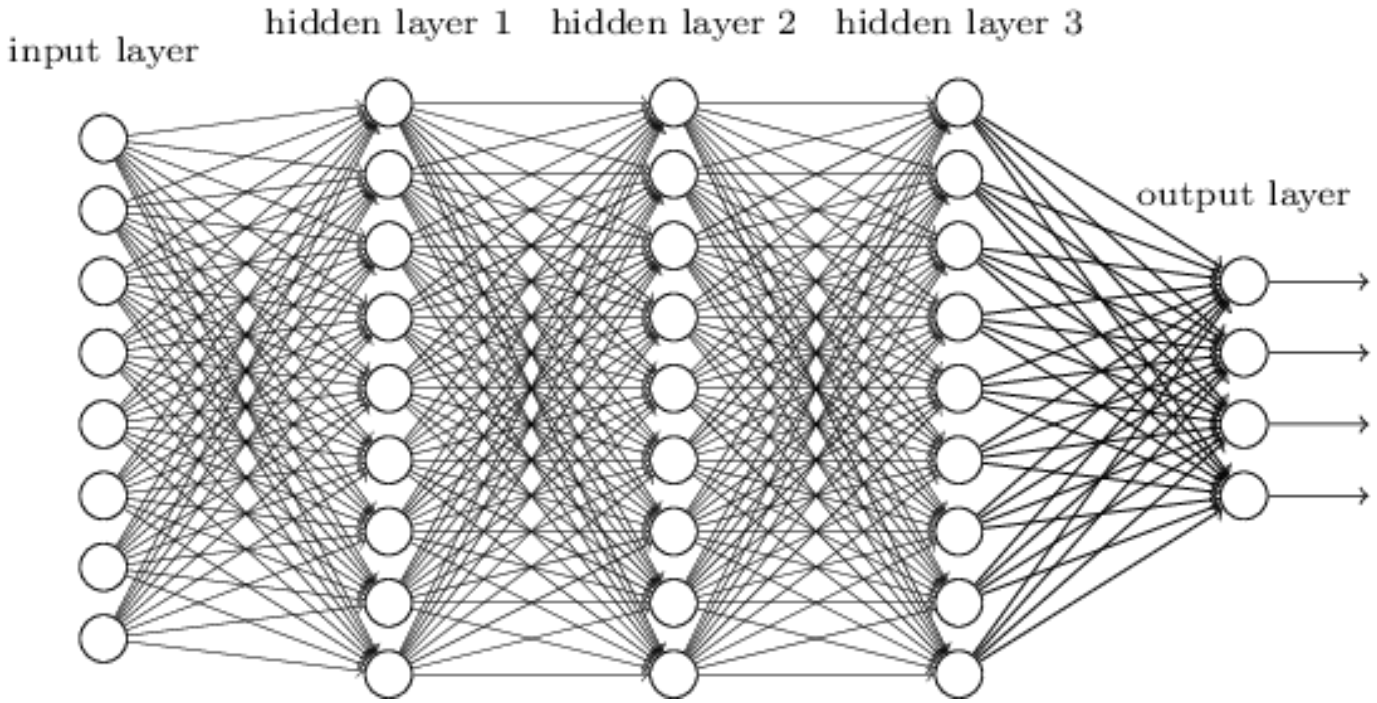
THEORY

Convolutional Neural Networks (CNNs)



Starting from feedforward neural networks

- We know it is good to learn a small model
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?

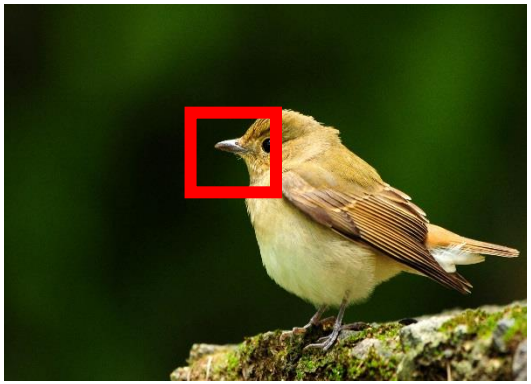


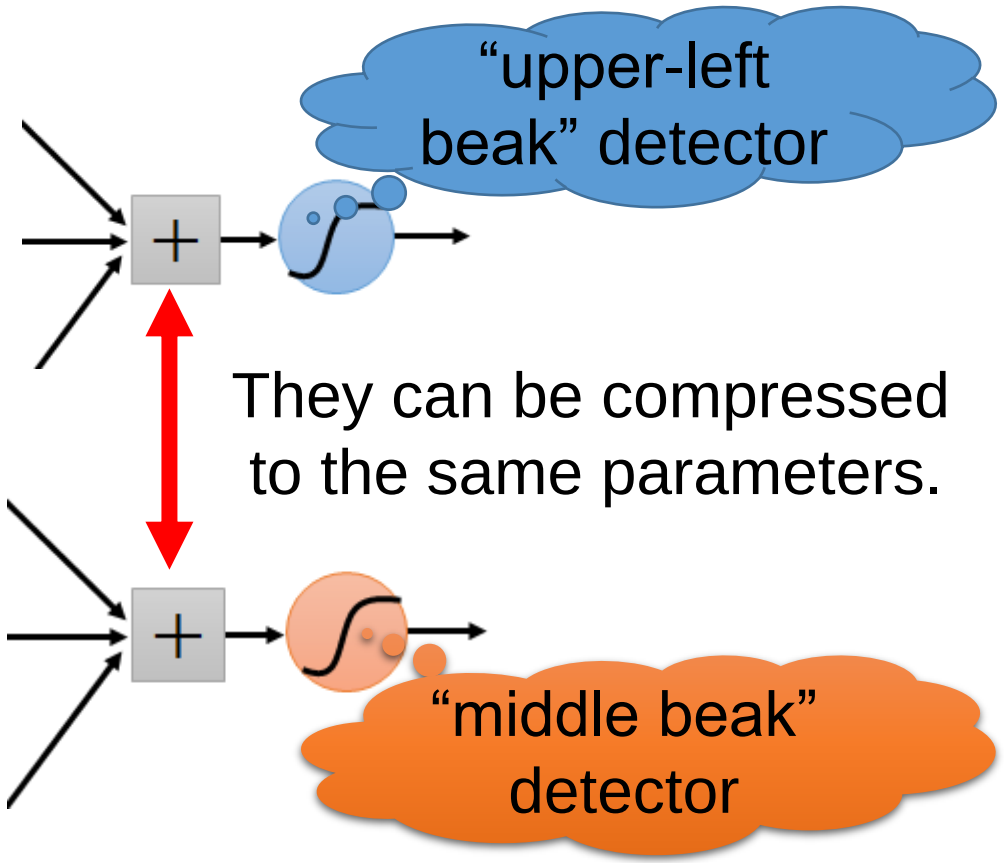
Consider learning an image:

- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters

Same pattern appears in different places:
They can be compressed!
What about training a lot of such “small” detectors and each detector must “move around”.

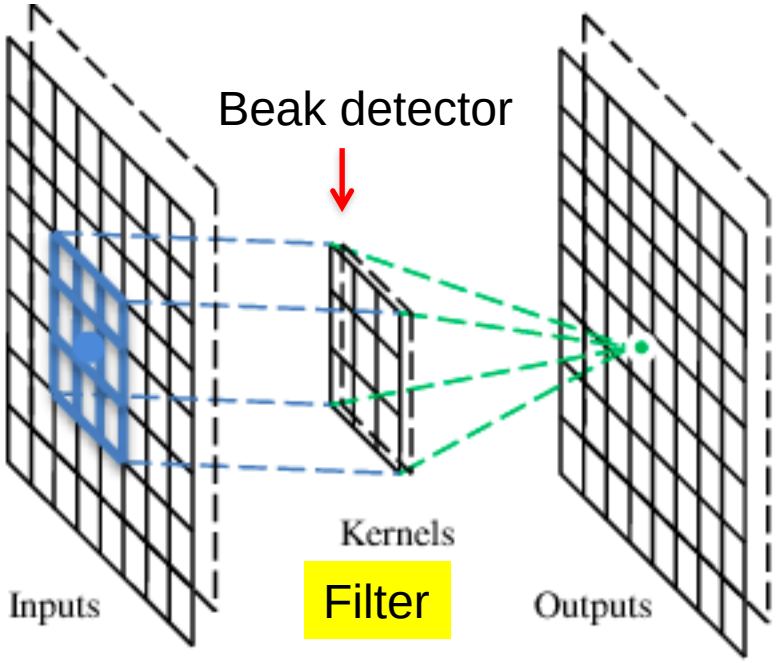




The idea to “move around” →

A convolutional layer

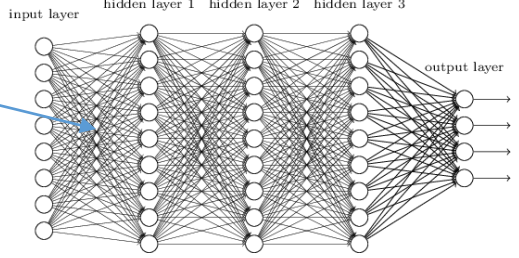
A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



The convolution output is high where the input image is similar to the kernel pattern

Convolution kernels

In trad. NN we have only weights



These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 input image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

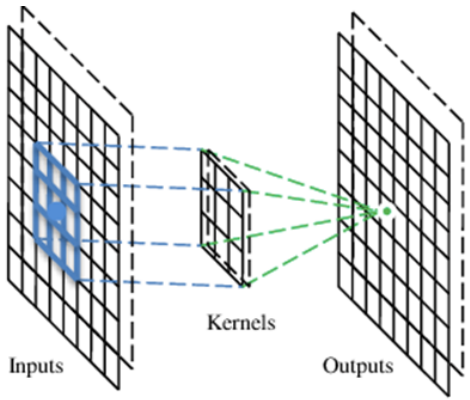
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product → 3 -1

6 x 6 input image

Let's focus on this step



Convolution (2)

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

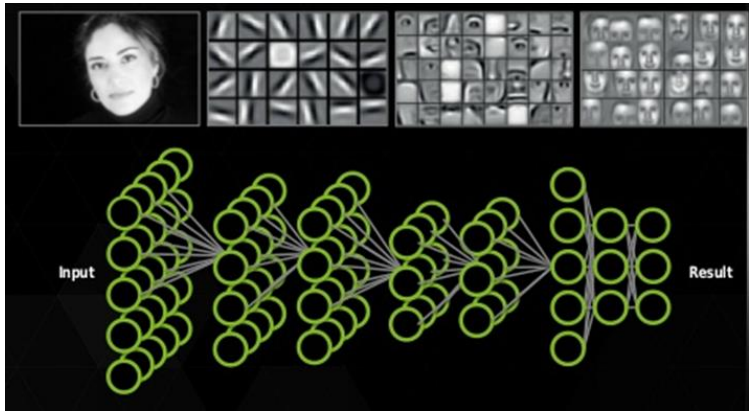
If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

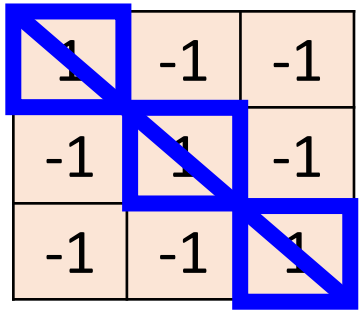
6 x 6 input image

3 -3

This can be an example of one convolutional layer of a large CNN

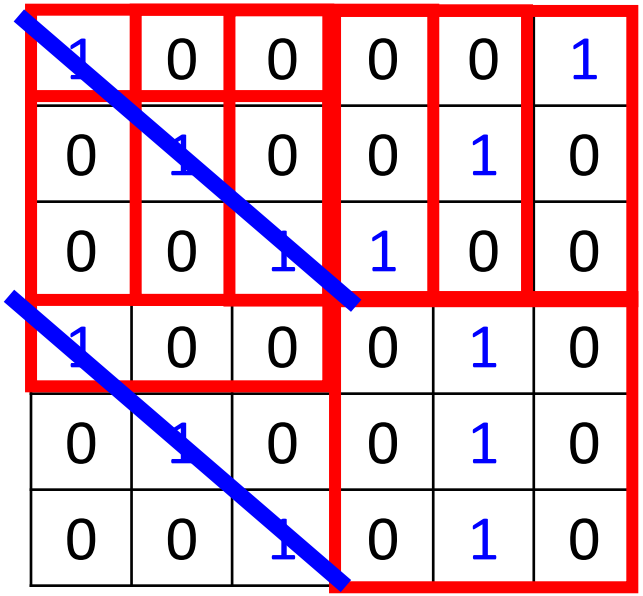


Convolution

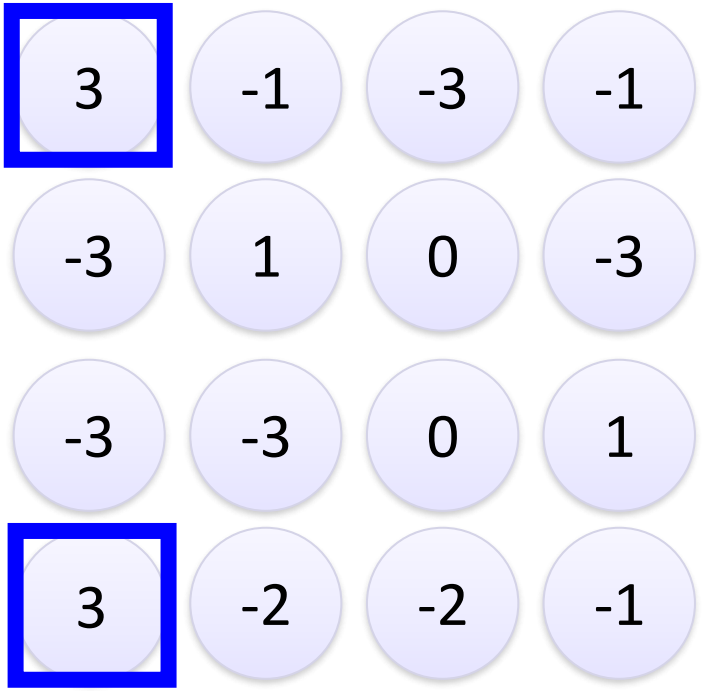


Filter 1

stride=1



6 x 6 image



CNN layer

-1	1	-1
-1	1	-1
-1	1	-1

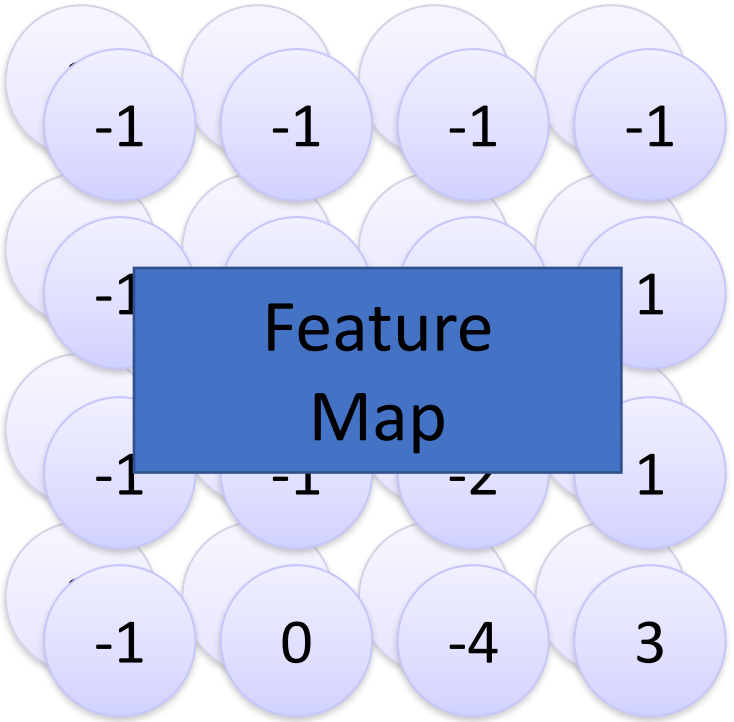
Filter **2**

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

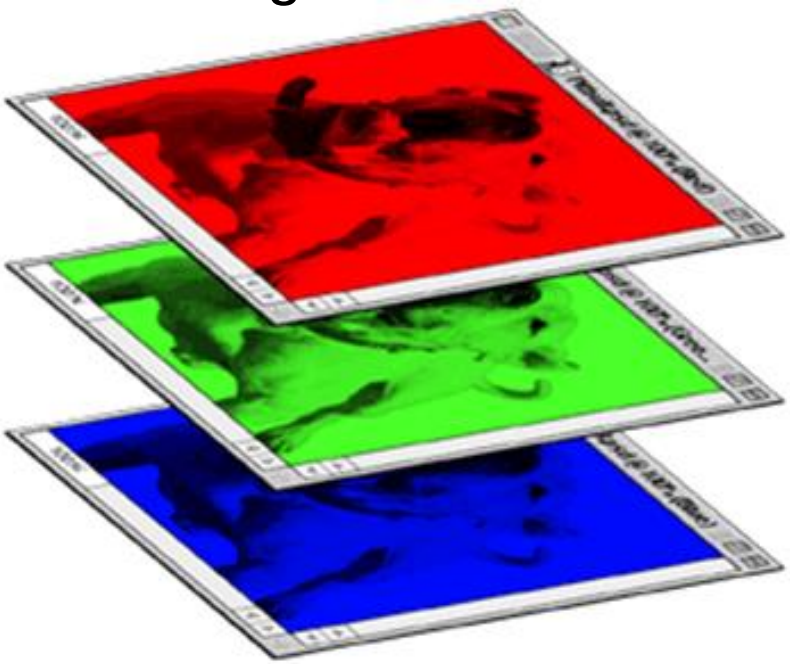
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

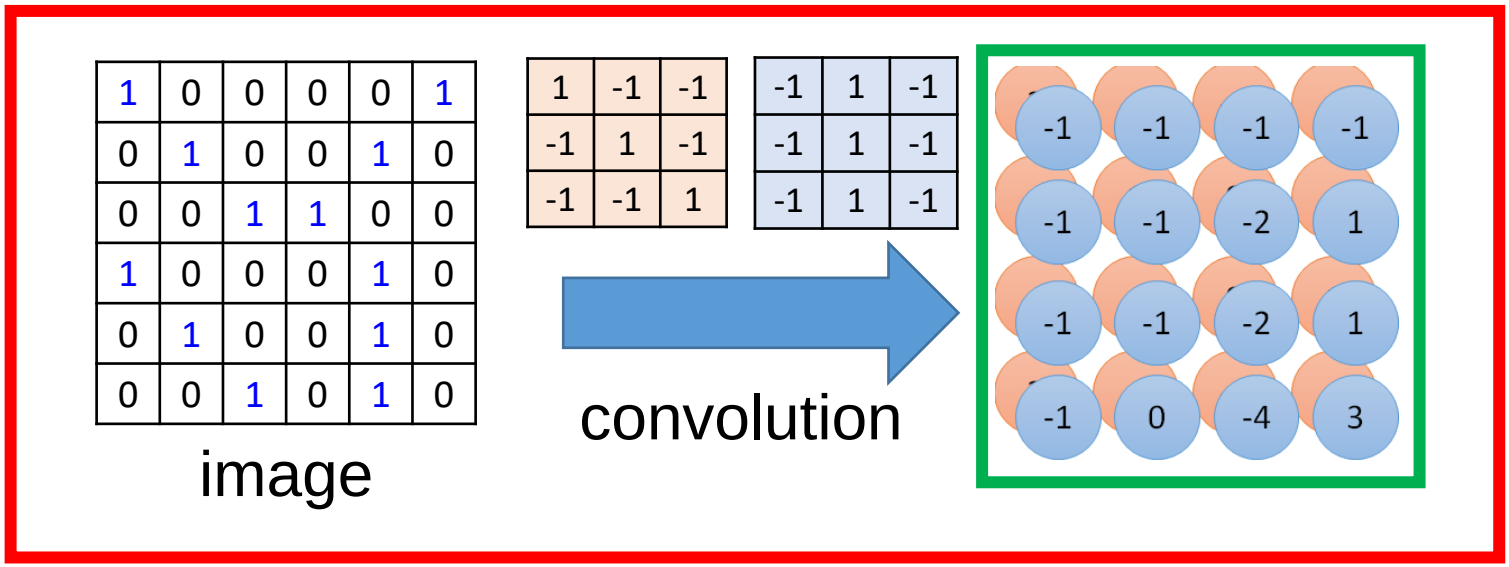
Filter 2

Color image

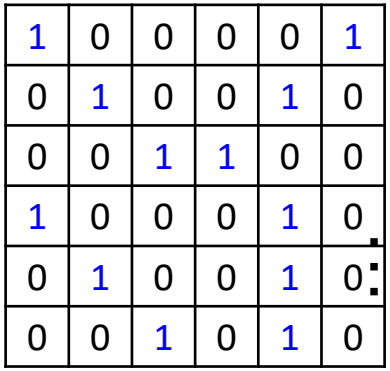


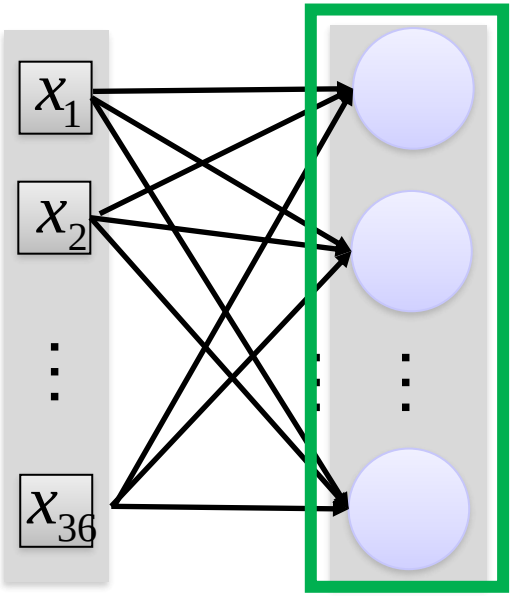
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

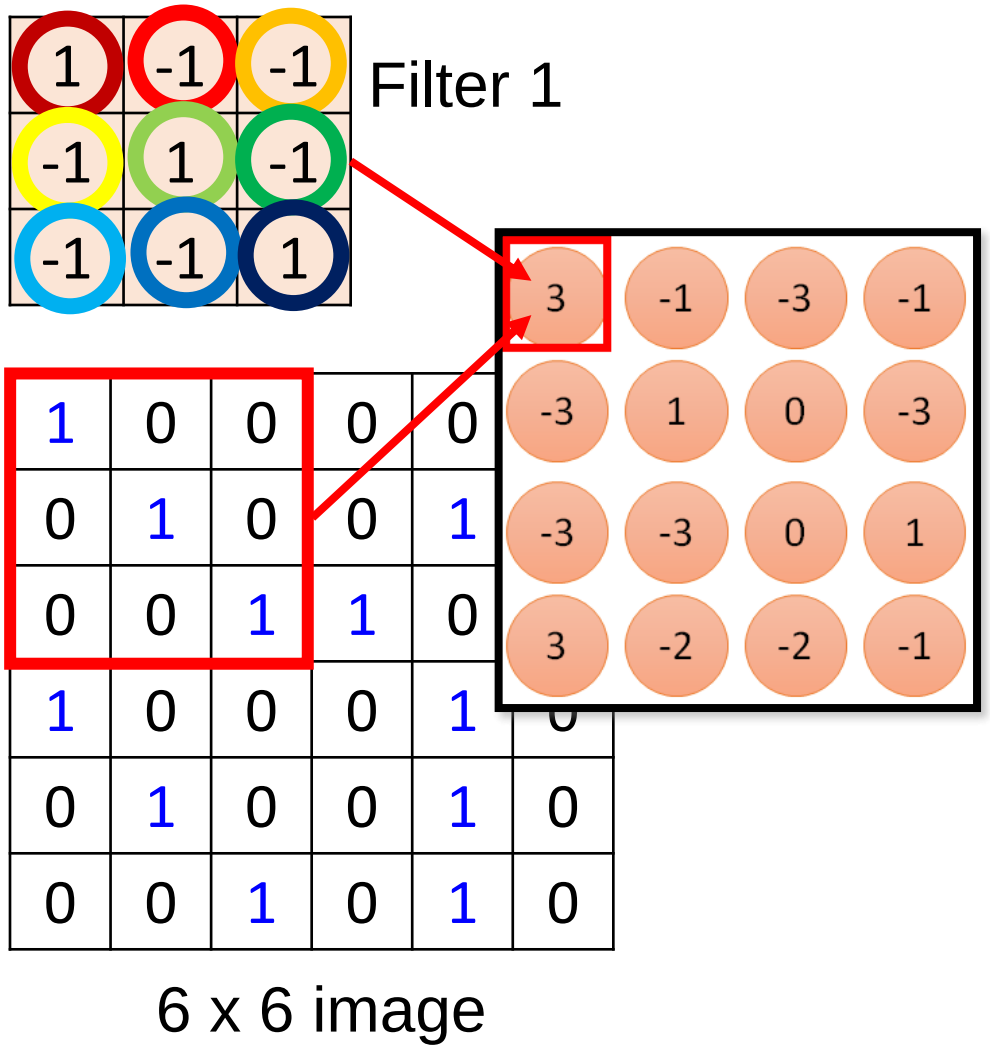
Convolution v.s. Fully Connected (1st layer)

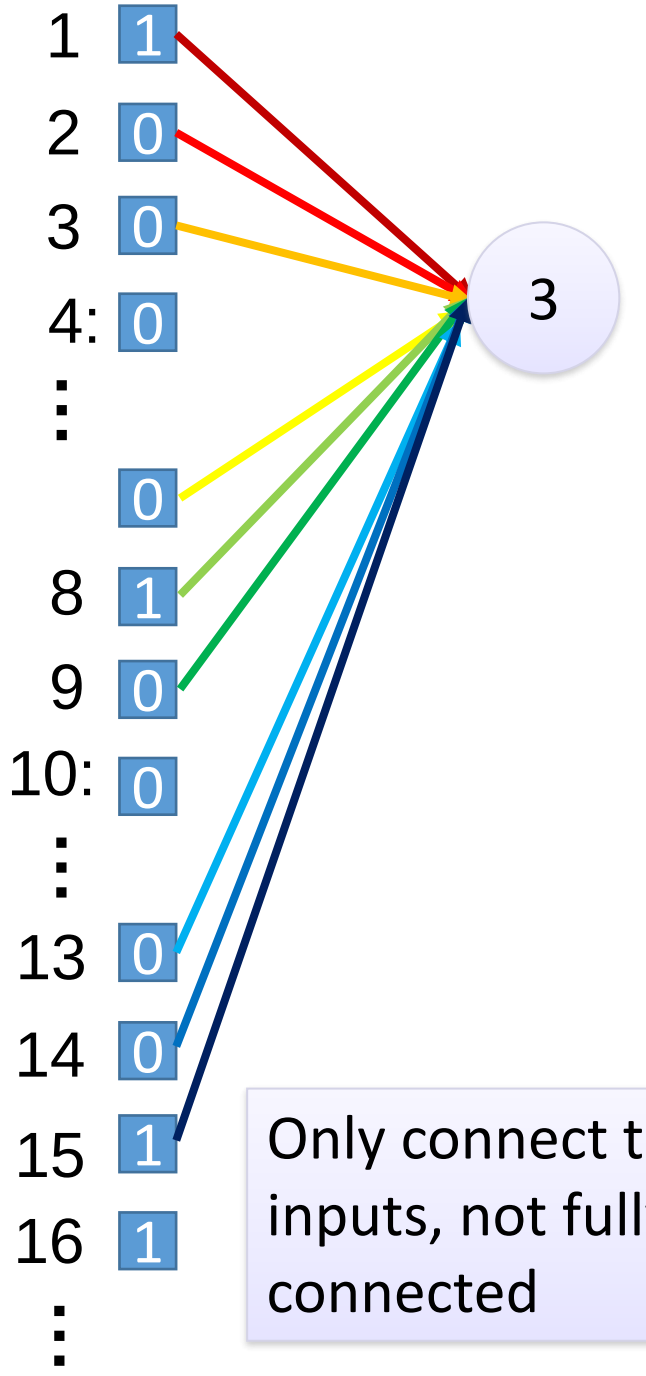


Fully-connected



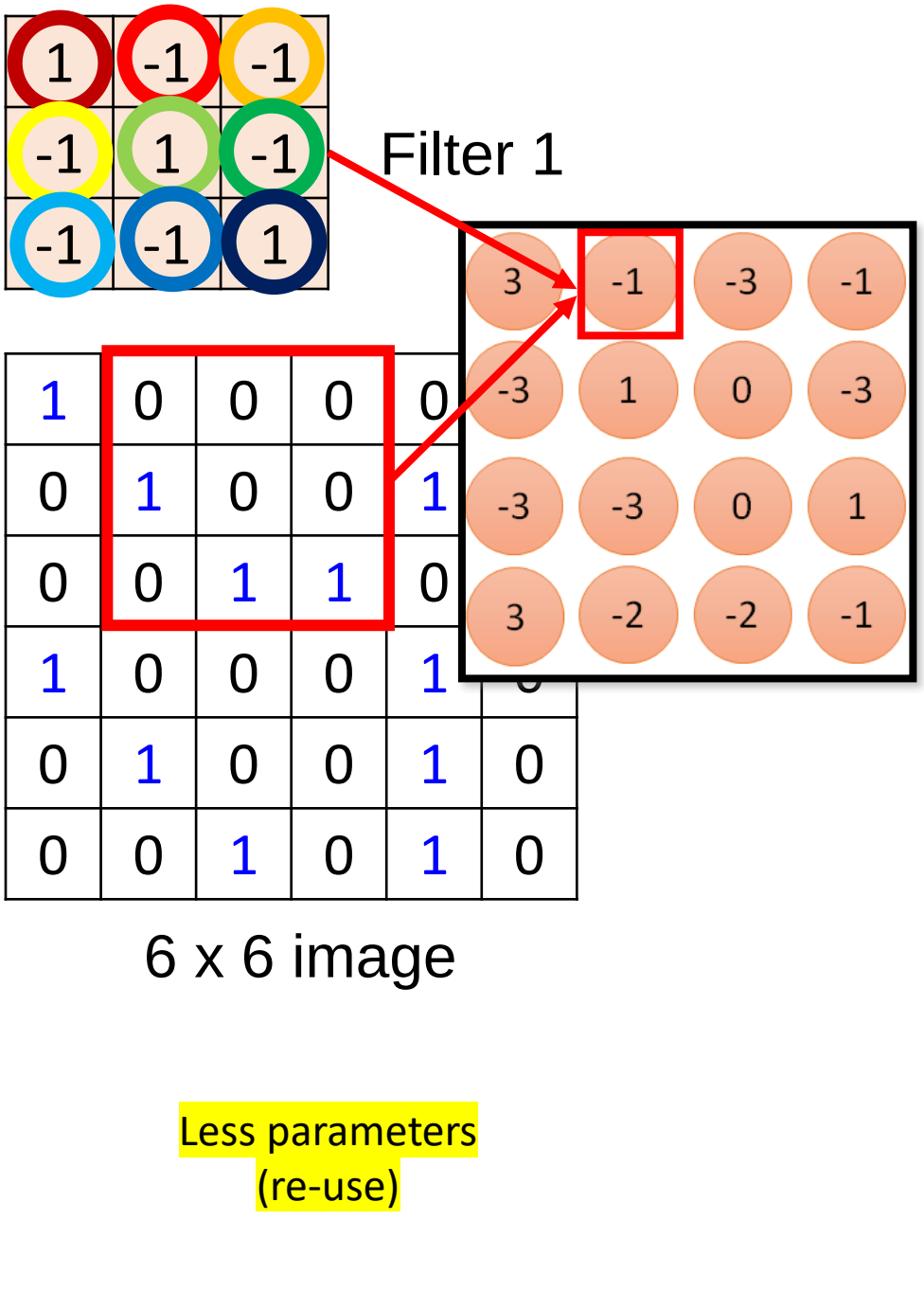






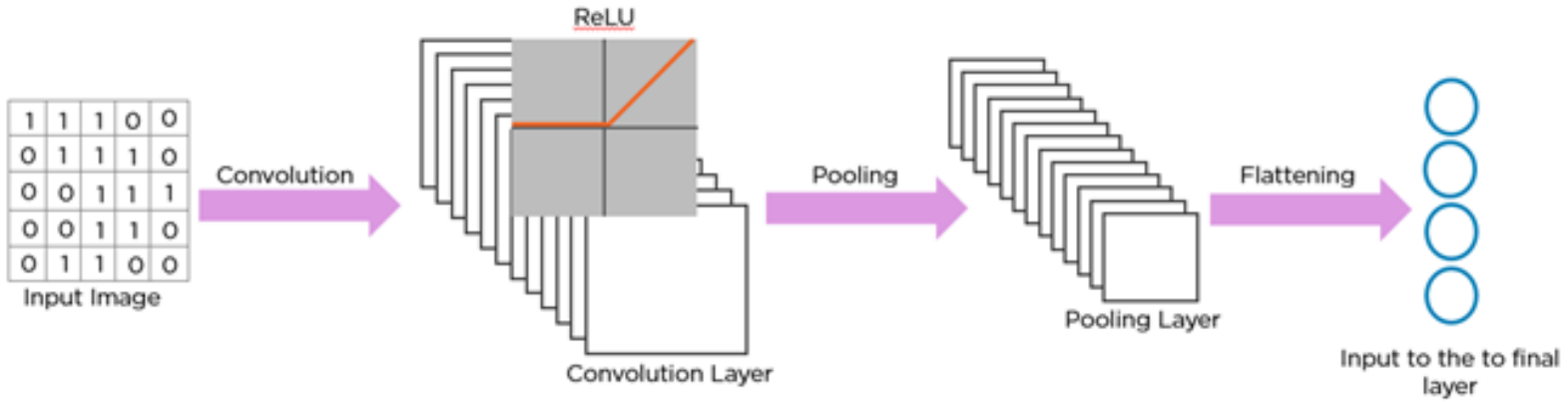
Reducing the number of parameters!!

Only connect to 9 inputs, not fully connected


1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Less parameters
(re-use)
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
-1
Shared weights

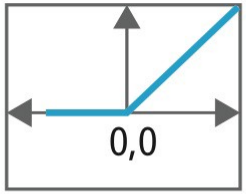
Grad = 0 in the optimization function
→ Learining is simplified

ReLu
(Rectified Linear Unit)

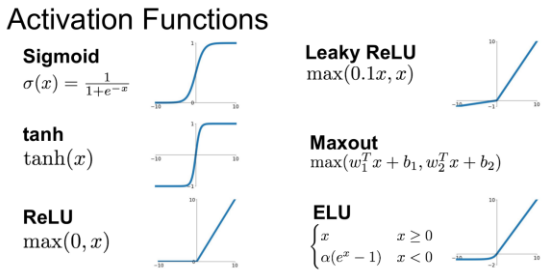


Transfer Function

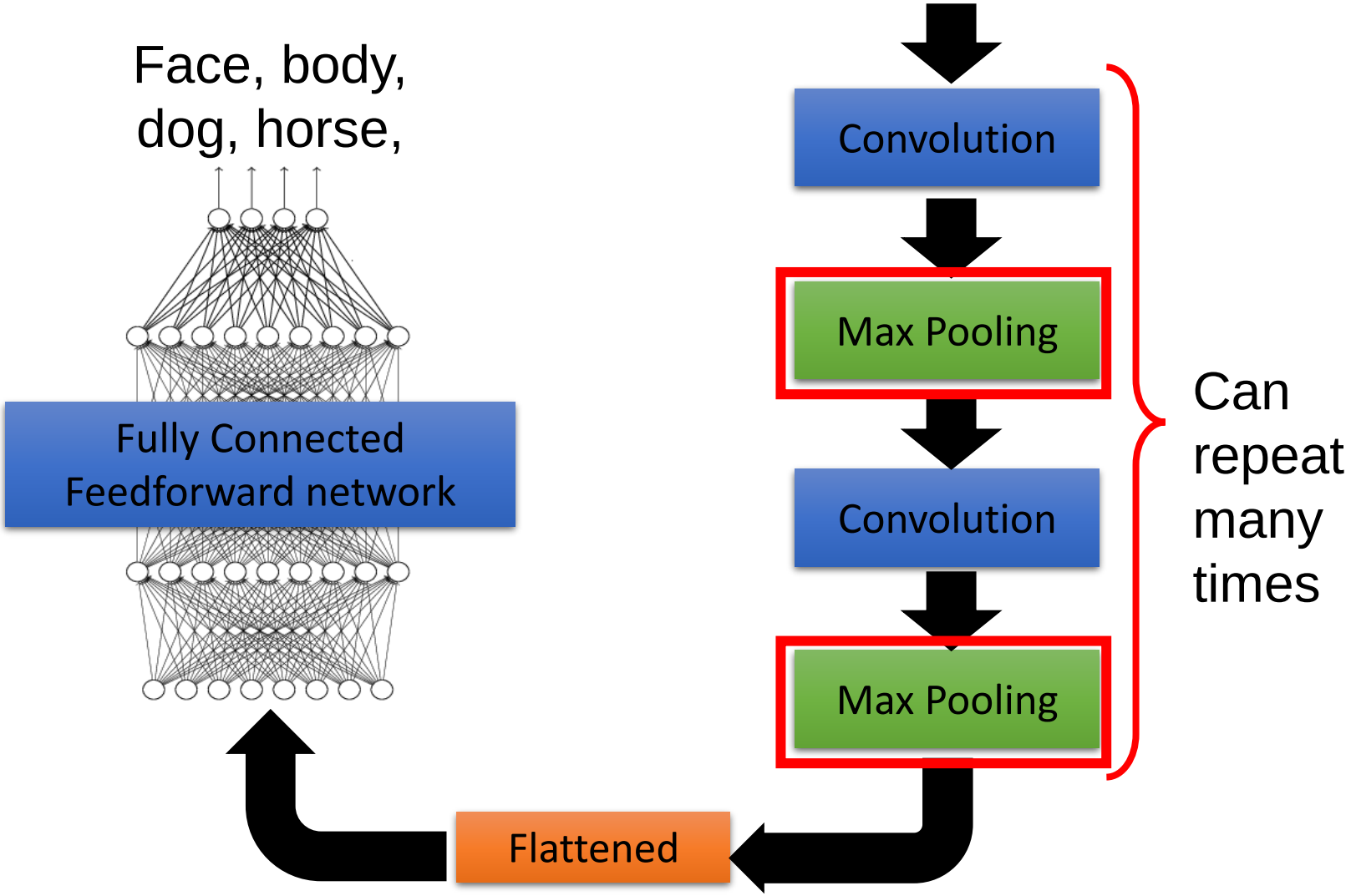
15	20	-10	35
18	-110	25	100
20	-15	25	-10
101	75	18	23



15	20	0	35
18	0	25	100
20	0	25	0
101	75	18	23



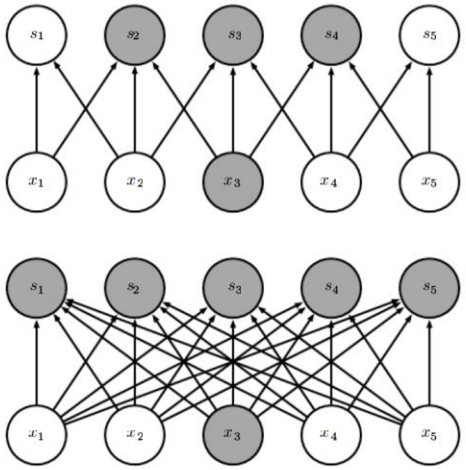
CNN architecture

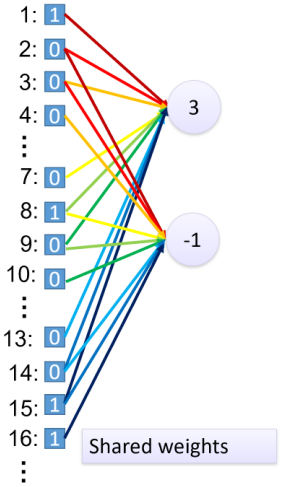


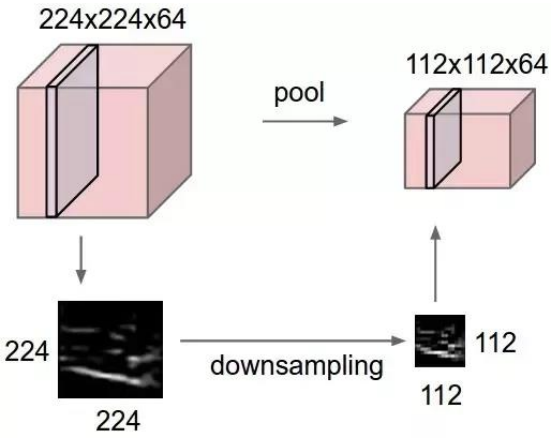
Convolution v.s. Fully Connected

CNNs and fully connected networks can be used to solve the same problems BUT

- CNNs reduce the number of connections
- CNNs can share the previous computations between the neurons
- CNNs can use pooling operations to reduce the computational complexity







Why Pooling

- Subsampling pixels will not change the object

bird

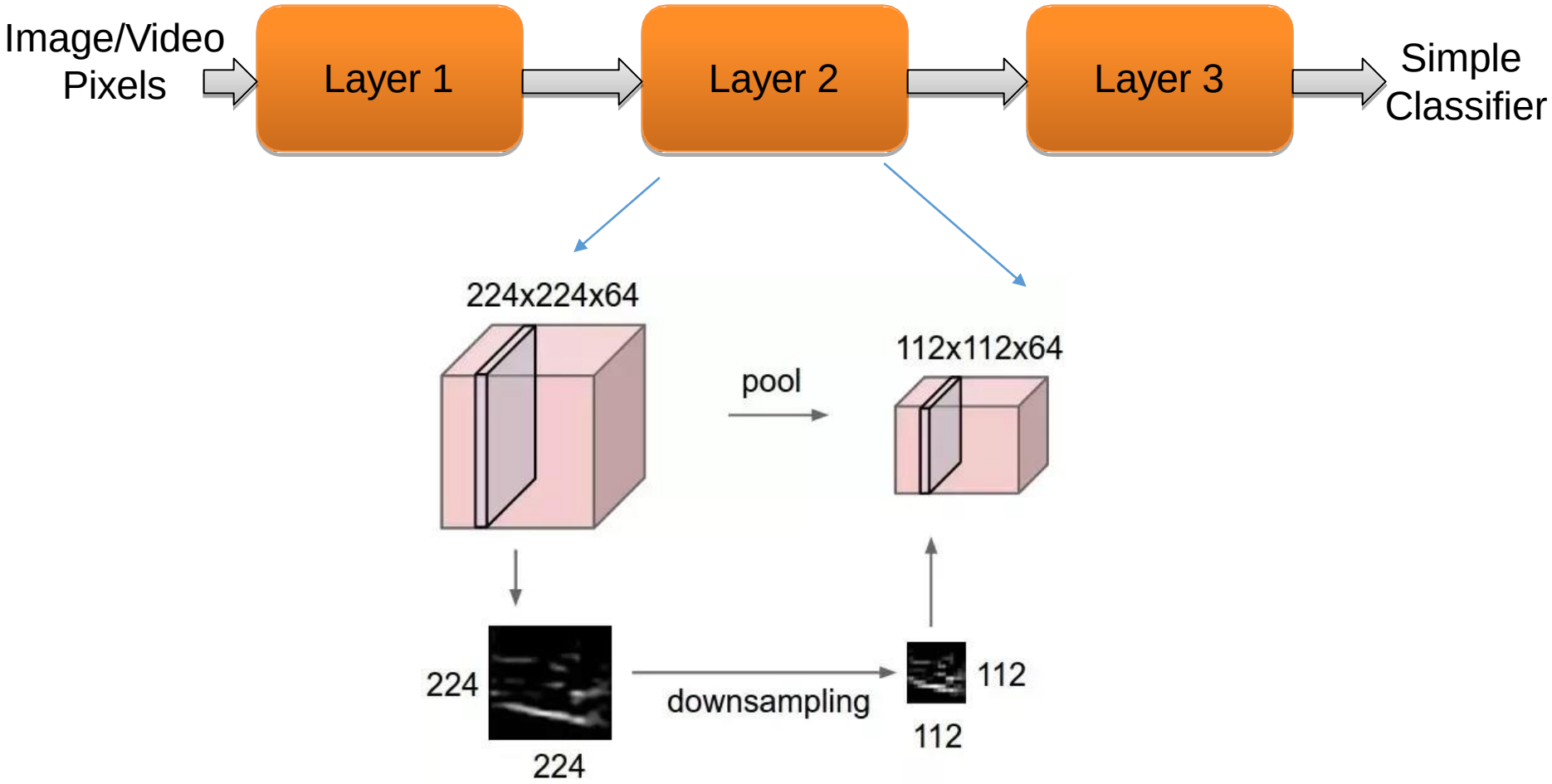
Subsampling

bird

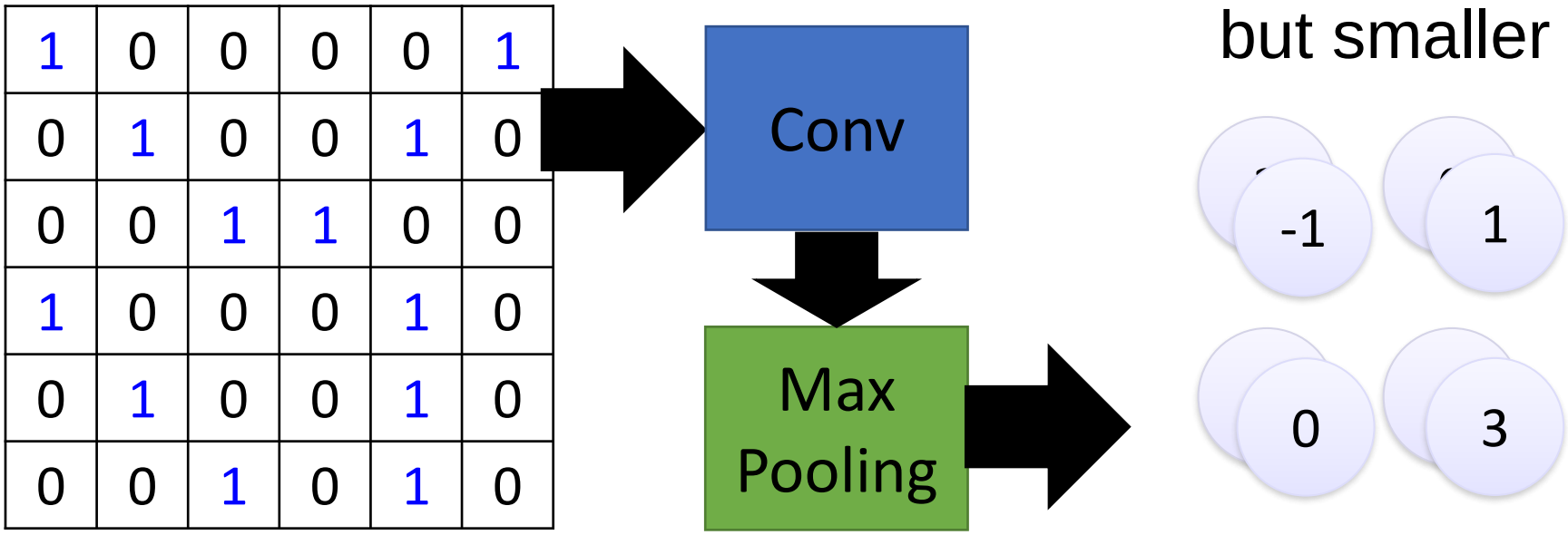
We can subsample the pixels to make image smaller

fewer parameters to characterize the image

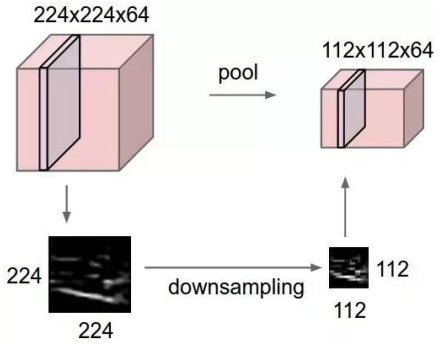
Pooling (cont.)



Max Pooling

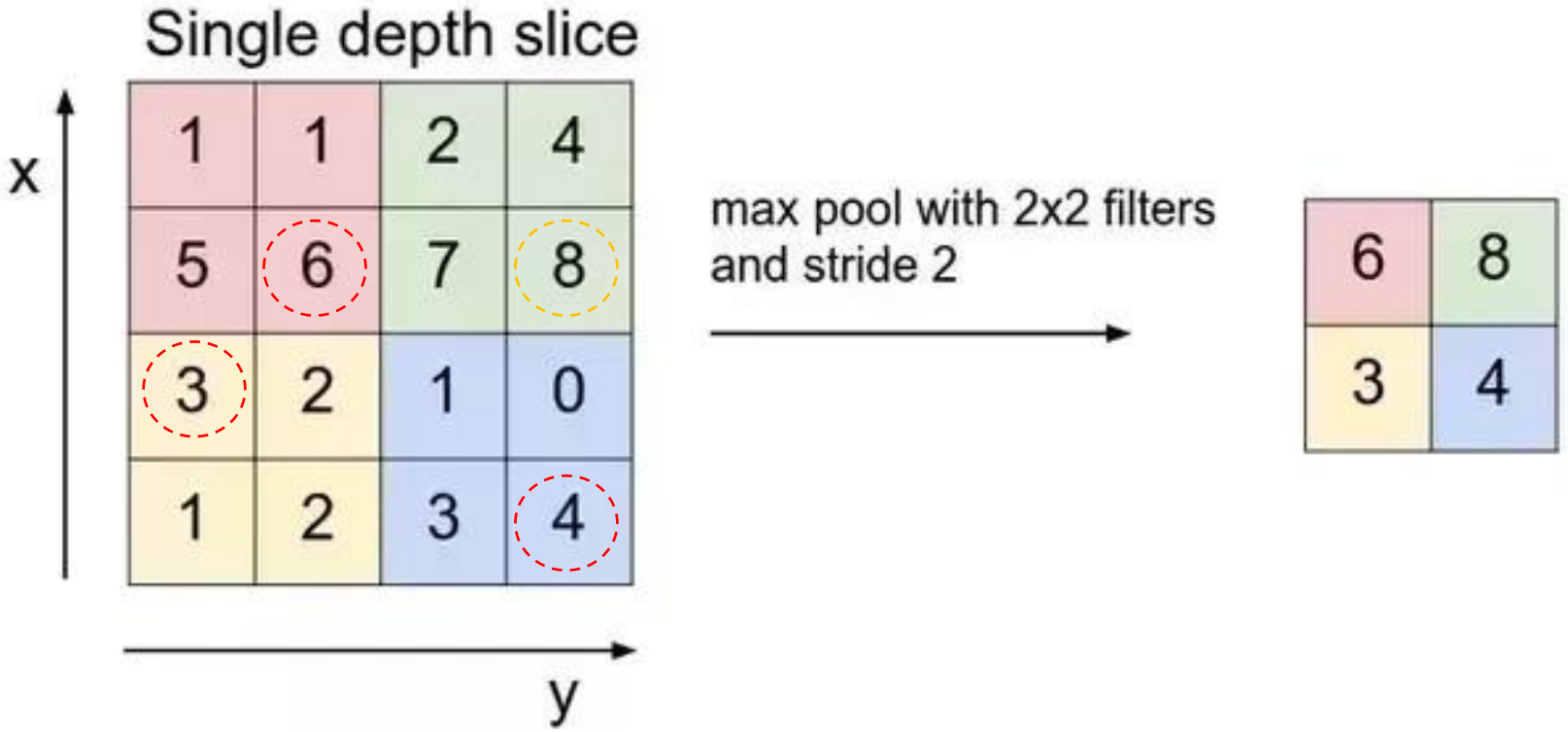

1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
Conv
Max Pooling
New image but smaller
-1
1
0
3
6 x 6 image
2 x 2 image
Each filter is a channel

Sub-Sampling (Pooling)

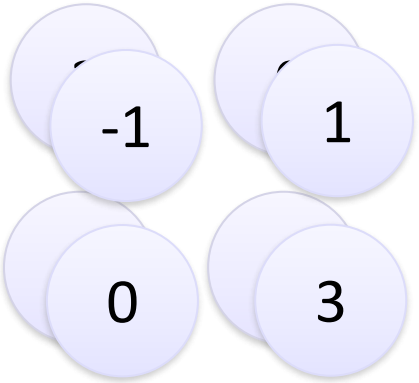


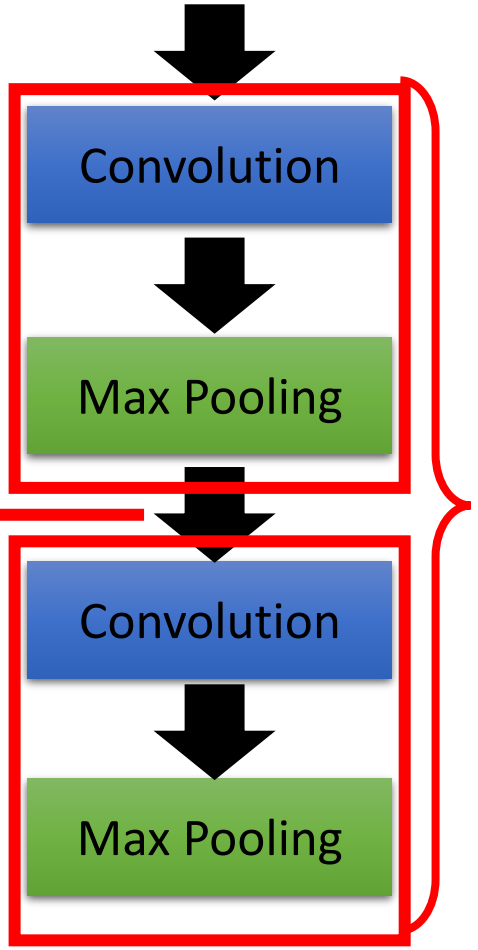
- Sub-sampling (Pooling) allows number of features to be diminished, non-overlapped
  - Reduces spatial resolution and thus naturally decreases importance of exactly where a feature was found, **just keeping the rough location**
  - **Averaging** or **Max-Pooling**
    - 2x2 pooling would do 4:1 compression, 3x3 9:1, etc.
  - Pooling smooths the data and **makes the data invariant to small translational changes**
  - Since after first layer, there are always multiple feature maps to connect to the next layer, it is a **pre-made human decision** as to which previous maps the current map receives inputs from

Example of max pooling



The whole CNN





Smaller than the original image

The number of channels is the number of filters

The whole CNN

cat dog ……

Convolution

Max Pooling

A new image

Convolution

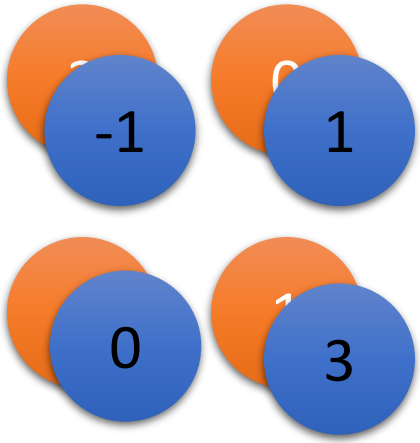
Max Pooling

A new image

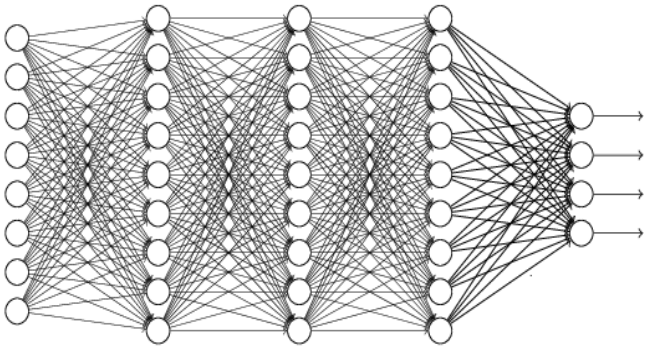
Flattened

Fully Connected
Feedforward network

Flattening

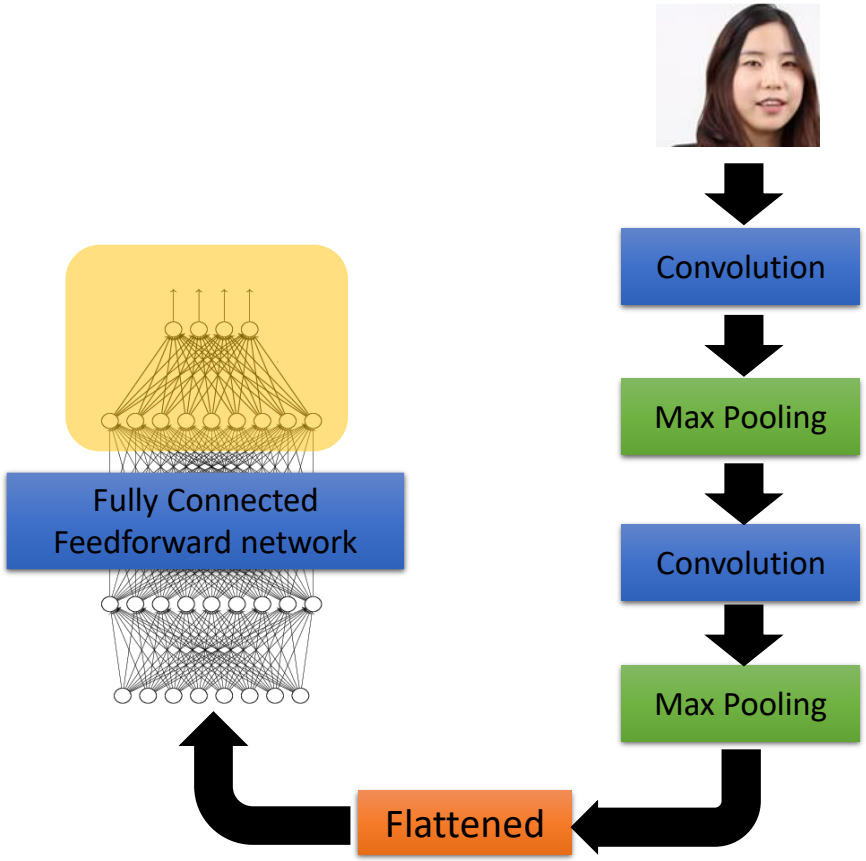


Flattened



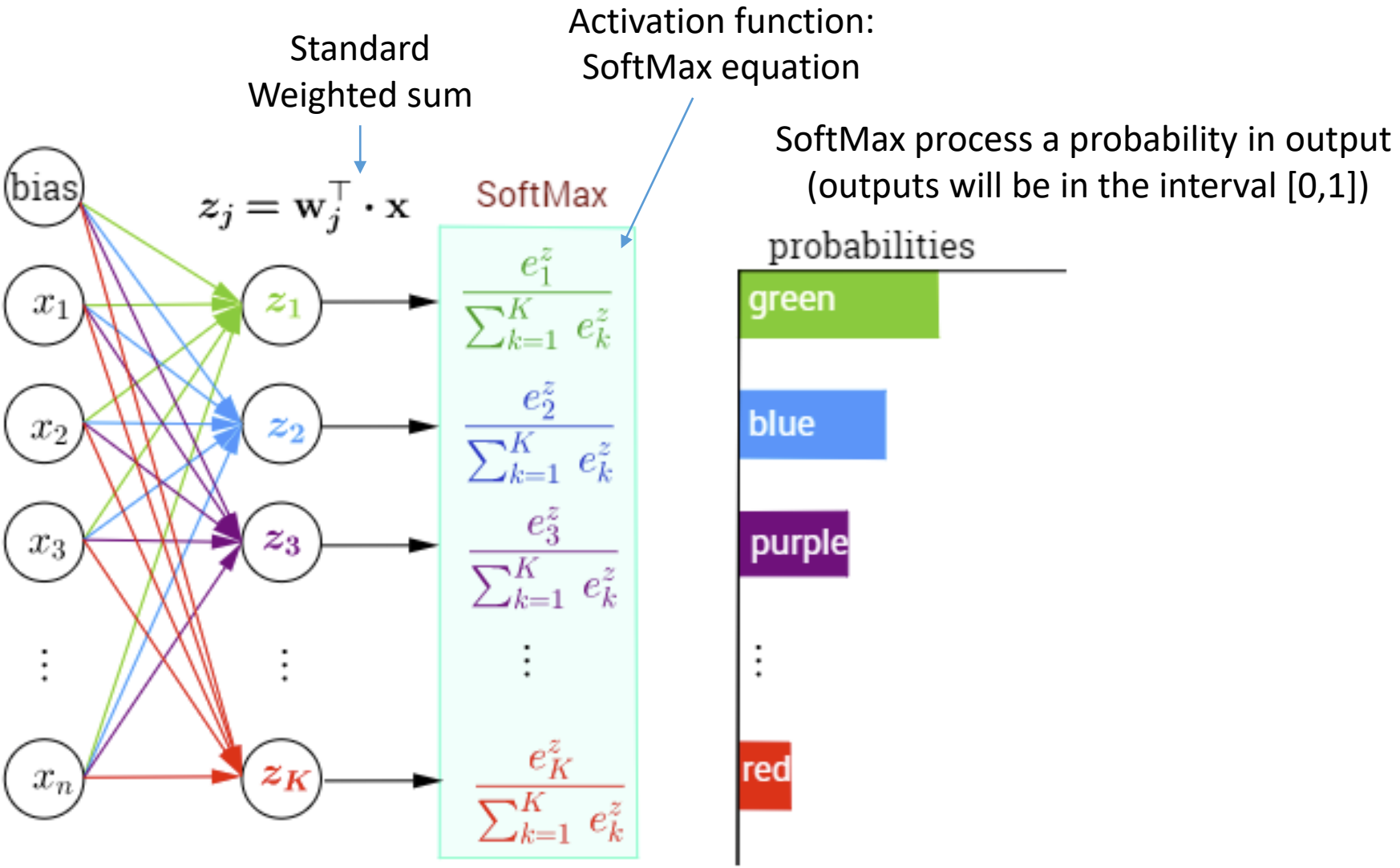
Fully Connected
Feedforward network
(with back prop. training)

The typical last layer of CNNs: the Softmax output layer



Softmax:
Normalized exponential activation function

A typical final module of the CNN for classification

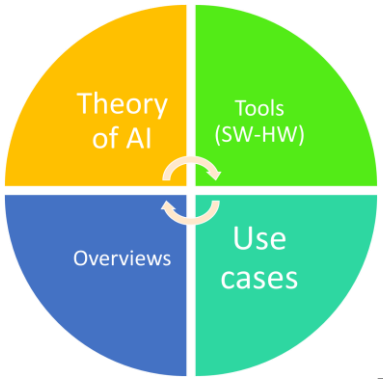





Toolboxes

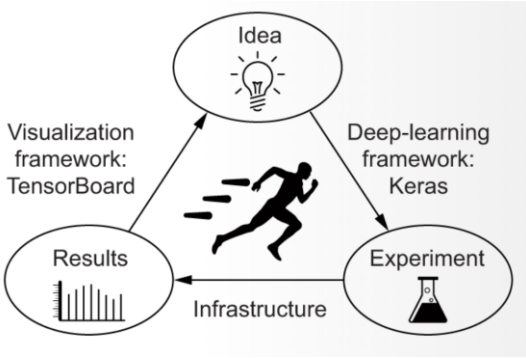
Deep learning in Keras

Keras is an open-source neural-network library written in Python



Keras

- Keras is an open-source neural-network library written in Python.
- Main features
  - User-friendly
  - Modular
  - Extensible
  - Fast experimentation
- Deployment
  - You can export Keras models to JavaScript to run directly in the browser, to TF Lite to run on iOS, Android, and embedded devices
- It is capable of running on top of
  - TensorFlow,
  - Microsoft Cognitive Toolkit,
  - R,
  - Theano,
  - PlaidML



Keras is used by CERN, NASA, and many more scientific organizations around the world

Keras

- Numerous implementations of
  - commonly used neural-network building blocks
    - layers
    - objectives
    - activation functions
    - optimizers
  - tools for
    - image
    - text data

Some Keras snipptes

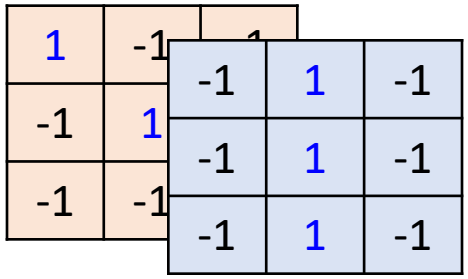
```
from tensorflow import keras
from tensorflow.keras import layers

# Instantiate a trained vision model
vision_model = keras.applications.ResNet50()

# This is our video.encoding branch using the trained vision_model
video_input = keras.Input(shape=(100, None, None, 3))
encoded_frame_sequence = layers.TimeDistributed(vision_model)(video_input)
encoded_video = layers.LSTM(256)(encoded_frame_sequence)

# This is our text-processing branch for the question input
question_input = keras.Input(shape=(100,), dtype='int32')
embedded_question = layers.Embedding(10000, 256)(question_input)
encoded_question = layers.LSTM(256)(embedded_question)

# And this is our video question answering model:
merged = keras.layers.concatenate([encoded_video, encoded_question])
output = keras.layers.Dense(1000, activation='softmax')(merged)
video_qa_model = keras.Model(inputs=[video_input, question_input],
                             outputs=output)
```

CNN in Keras

28 x 28 x 1 channel

input

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(28,28,1)) )
```



...

...

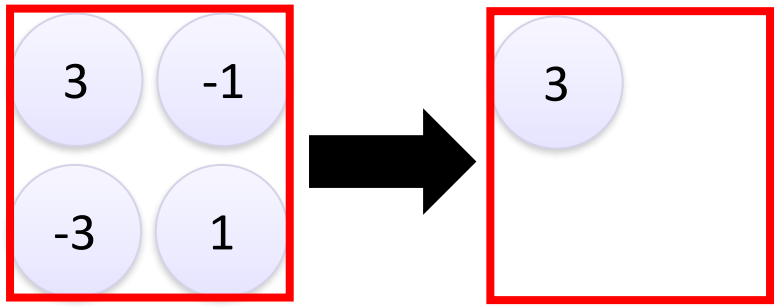
There are 25 3x3 filters.

Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: Gray, 3: RGB

```
model2.add(MaxPooling2D((2,2)))
```



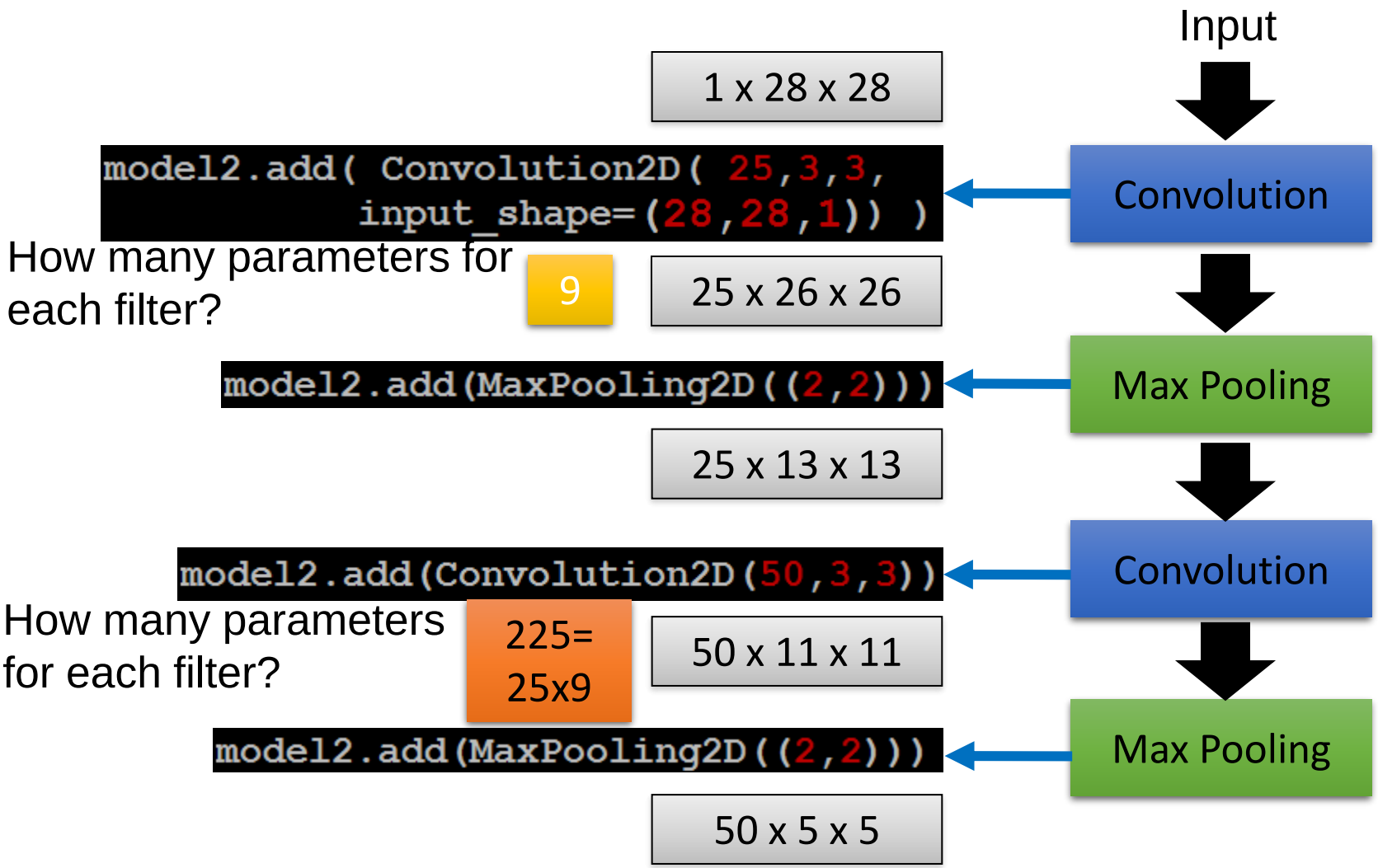
Convolution

Max Pooling

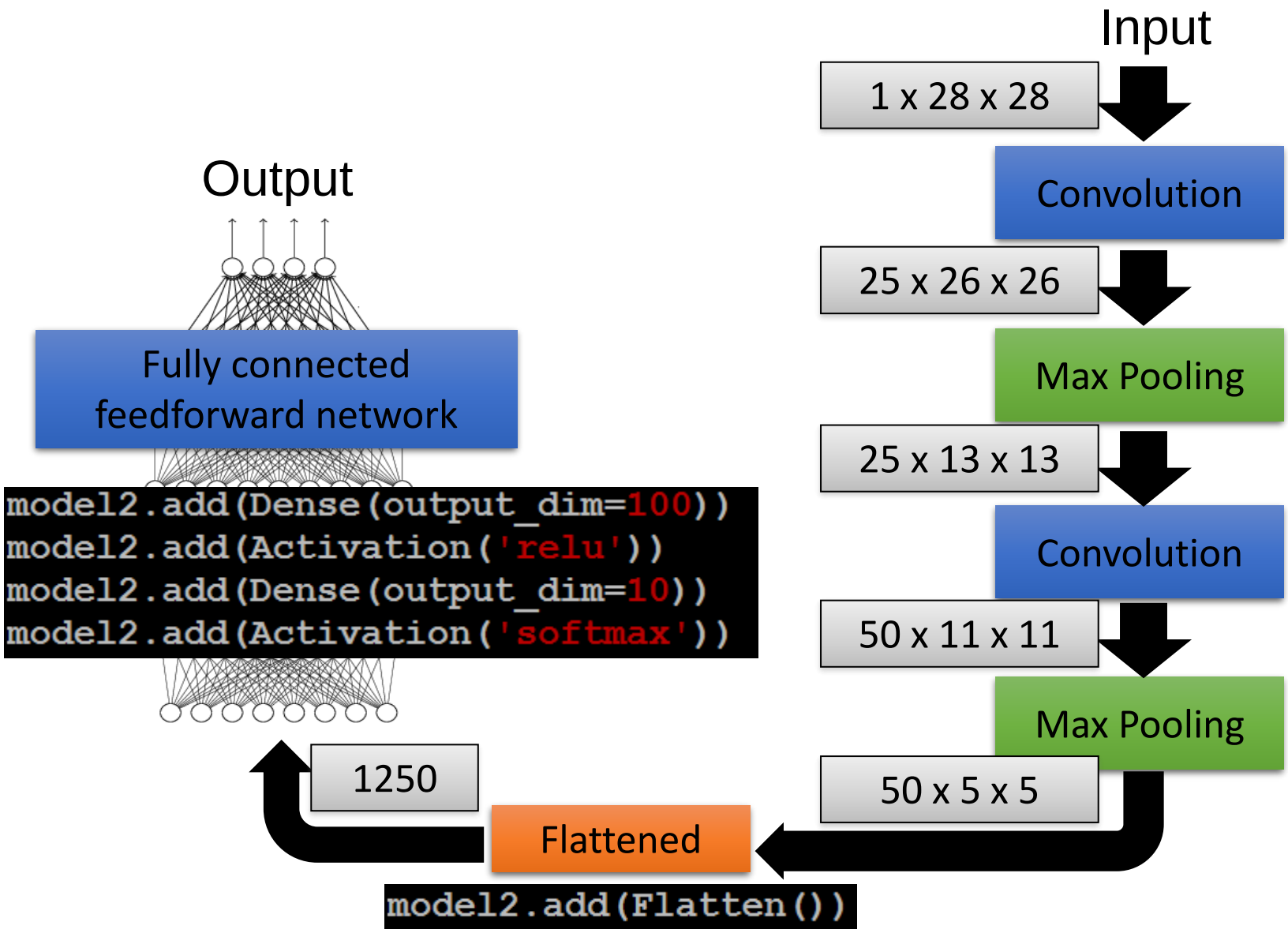
Convolution

Max Pooling

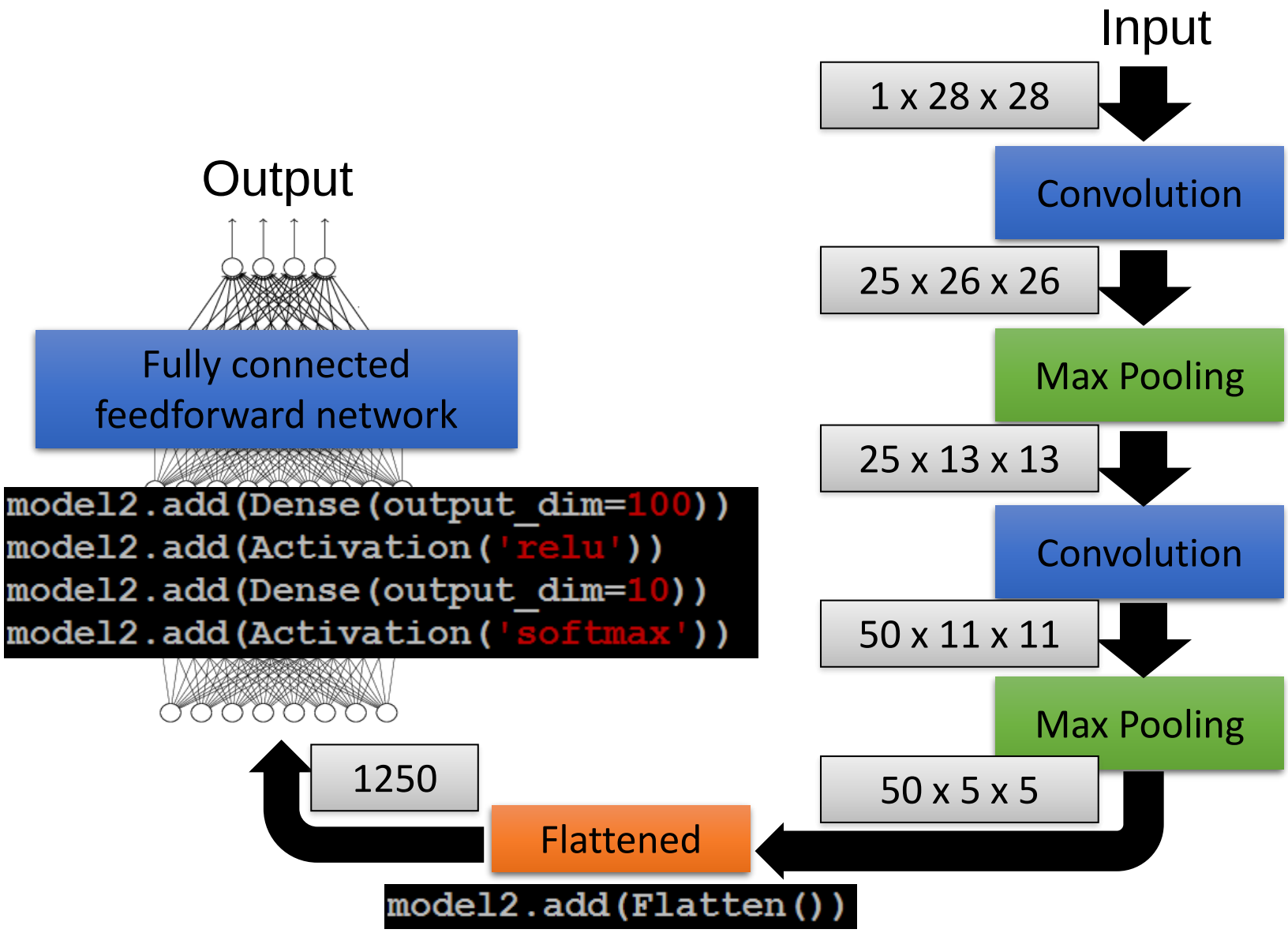
CNN in Keras


Input
1 x 28 x 28
model2.add(Convolution2D(25,3,3, input_shape=(28,28,1)))
Convolution
How many parameters for each filter?
9
25 x 26 x 26
model2.add(MaxPooling2D((2,2)))
Max Pooling
25 x 13 x 13
model2.add(Convolution2D(50,3,3))
Convolution
How many parameters for each filter?
225= 25x9
50 x 11 x 11
model2.add(MaxPooling2D((2,2)))
Max Pooling
50 x 5 x 5

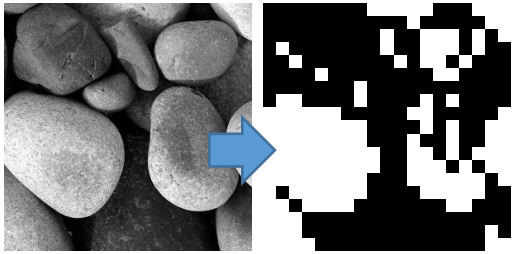
CNN in Keras


Input
1 x 28 x 28
Convolution
25 x 26 x 26
Max Pooling
25 x 13 x 13
Convolution
50 x 11 x 11
Max Pooling
50 x 5 x 5
Flattened
model2.add(Flatten())
1250
Fully connected
feedforward network
Output
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))

CNN in Keras


Input
1 x 28 x 28
Convolution
25 x 26 x 26
Max Pooling
25 x 13 x 13
Convolution
50 x 11 x 11
Max Pooling
50 x 5 x 5
Flattened
model2.add(Flatten())
1250
Fully connected
feedforward network
Output
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))

AlphaGo



300x300 pixel
8 bit gray level
image

19x19 pixel
Black/White
image

19 x 19 matrix

Neural Network

Next move

(19 x 19 positions)

Black: 1

white: -1

none: 0

Fully-connected feedforward network can be used

But CNN performs much better

AlphaGo's policy network

The following is quotation from their Nature article:

Note: AlphaGo does not use Max Pooling.

Neural network architecture. The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

CNN - Designing and Training:

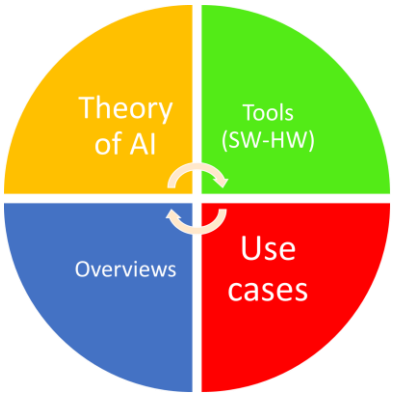
- Most weights are trained with **Back Propagartion (BP)**

- The structure of the CNN is currently usually **hand crafted with trial and error:**
  - Number of total layers,
  - number of receptive fields,
  - size of receptive fields,
  - size of sub-sampling (pooling) fields,
  - which fields of the previous layer to connect to

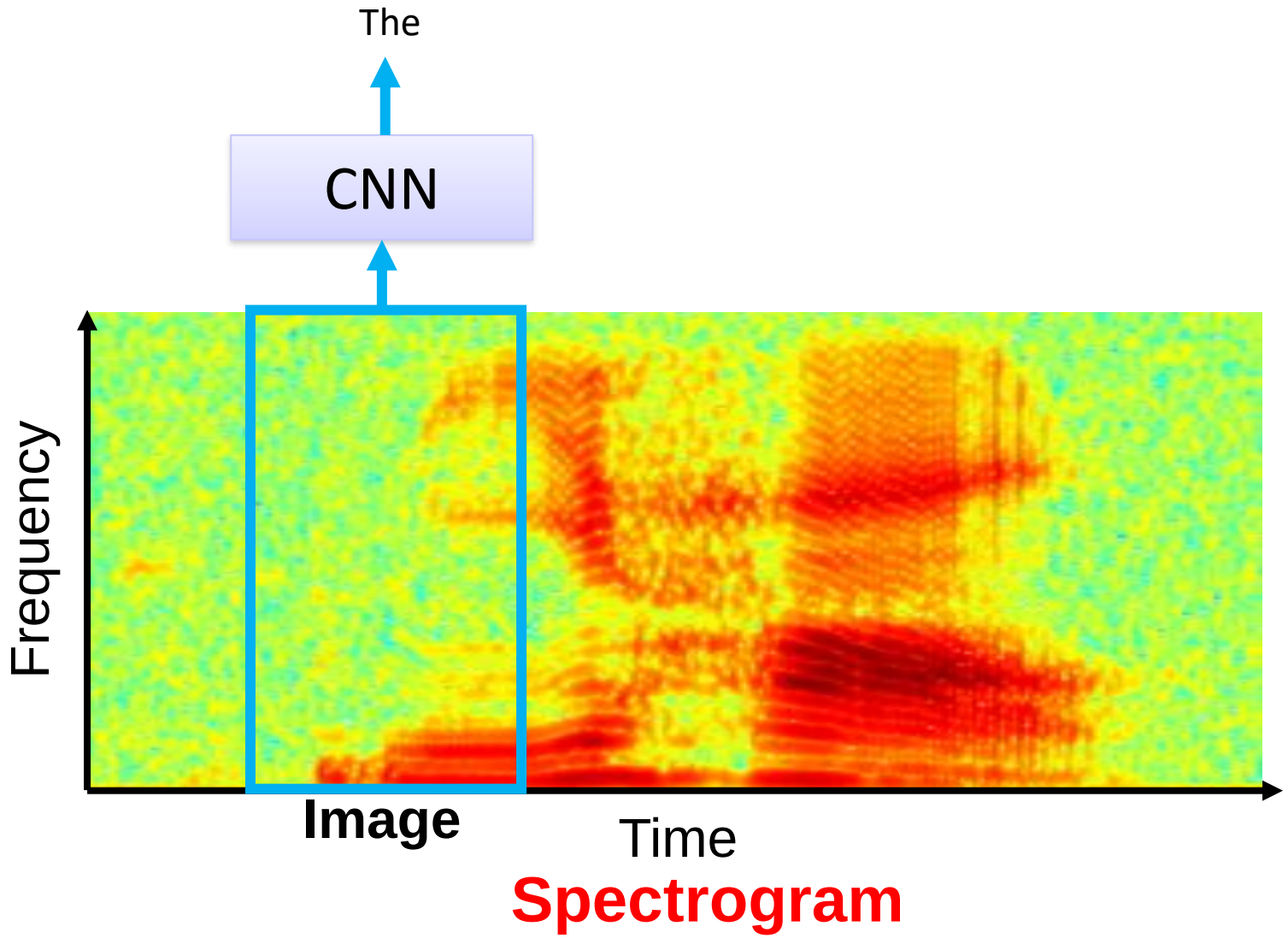
- Typically decrease size of feature maps and increase number of feature maps for later layers

THEORY

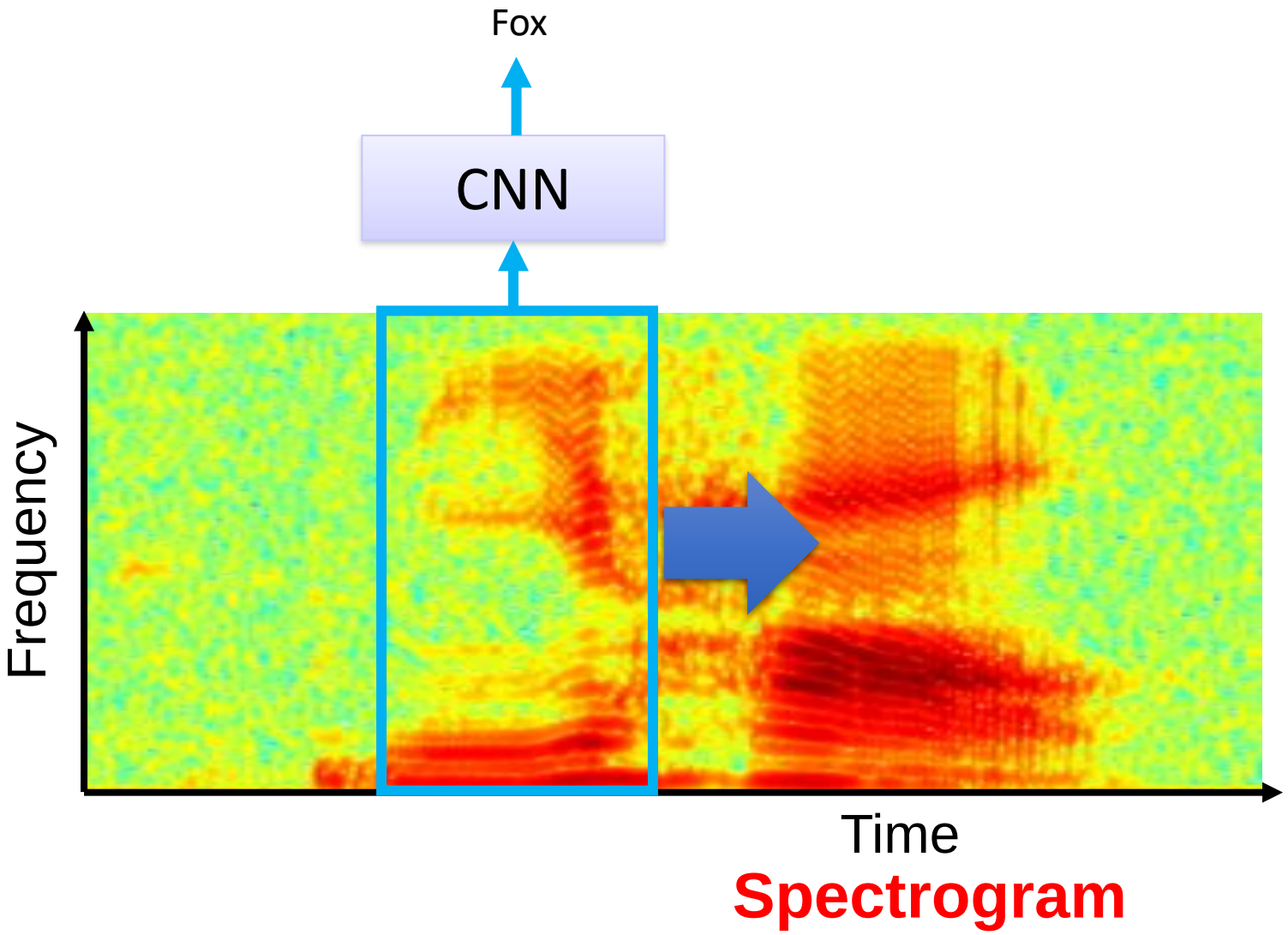
Using CNNs for signals and texts



CNN in speech recognition



CNN in speech recognition



Word embedding

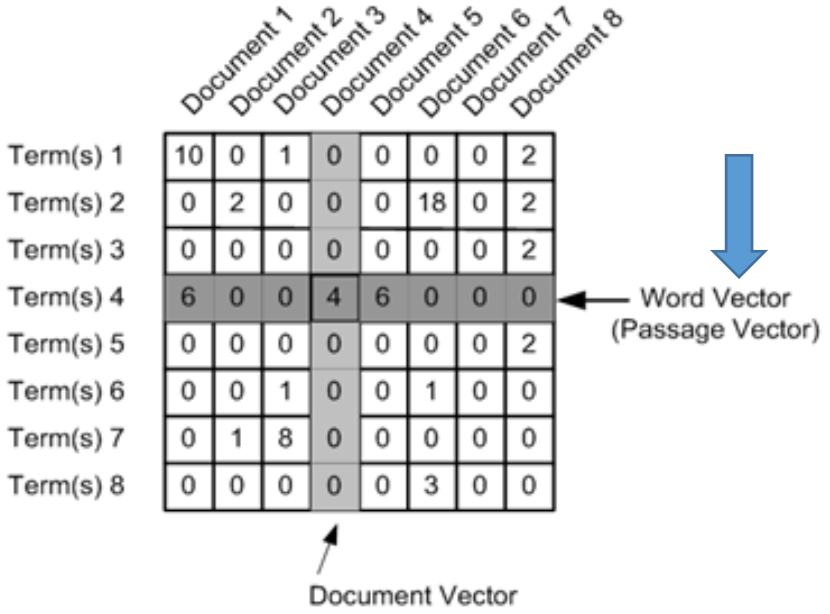
Word Embeddings are Words converted into numbers

1) **Create the Corpus of Documents**

- D1: He is a lazy boy. She is also lazy.
- D2: Rob is a lazy person.

2) **Create the Dictionary**,
for example, a list of unique tokens (words) in the corpus (drop 'is' to simplify)

=['He','She','lazy','boy','Rob','person']

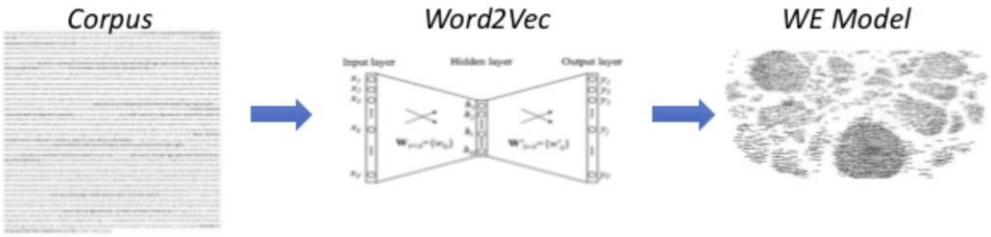


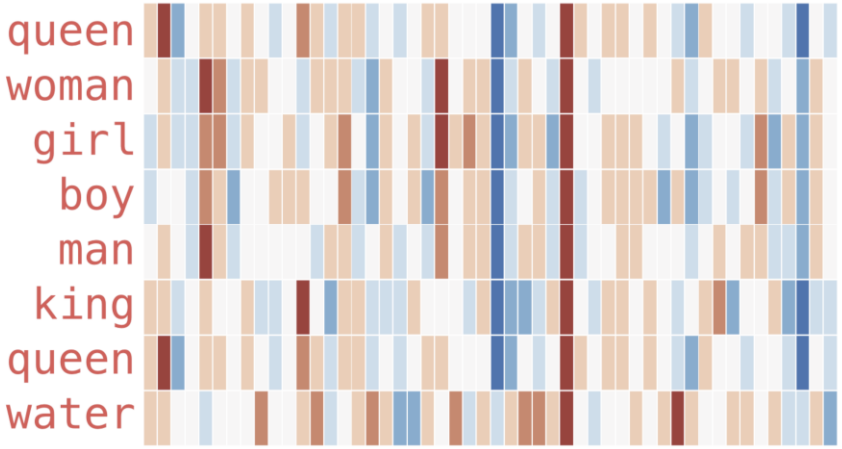
	He	She	lazy	boy	Rob	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

The are more complex way based on co-occurance matrix

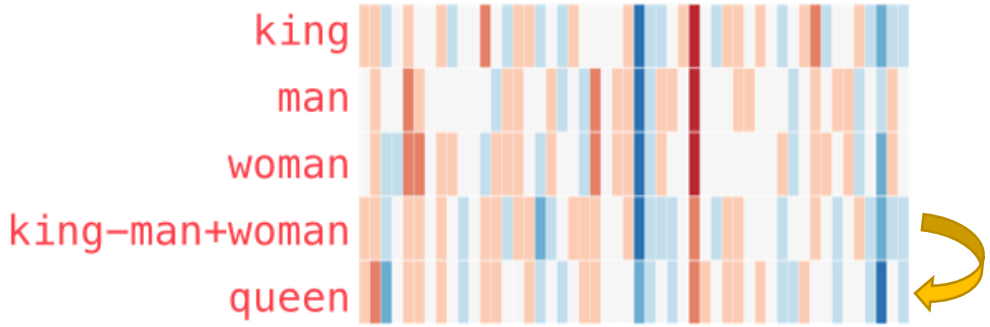
Word2Vec
Embedding Neural Architectures

The Word2Vec technique is based on a feed-forward, fully connected architecture to create vectors from words
A neural FEATURE EXTRACTOR!





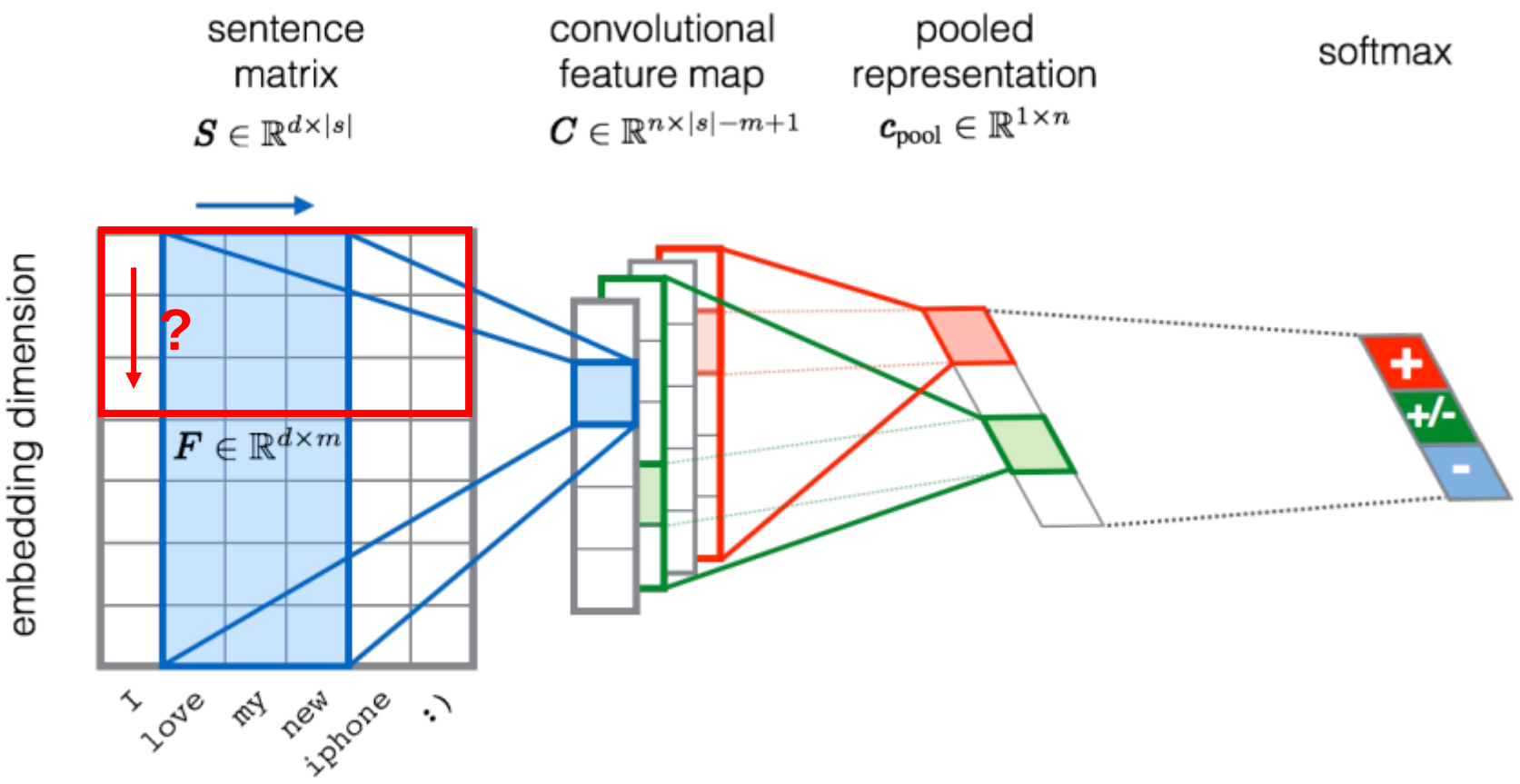
king − man + woman ~= queen



Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

CNN in text classification

Twitter Sentiment Classification



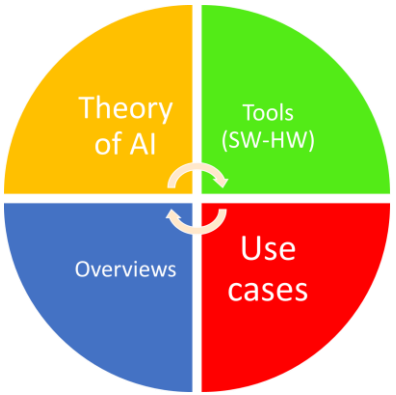
well-known




THEORY

“Well-known” CNNs

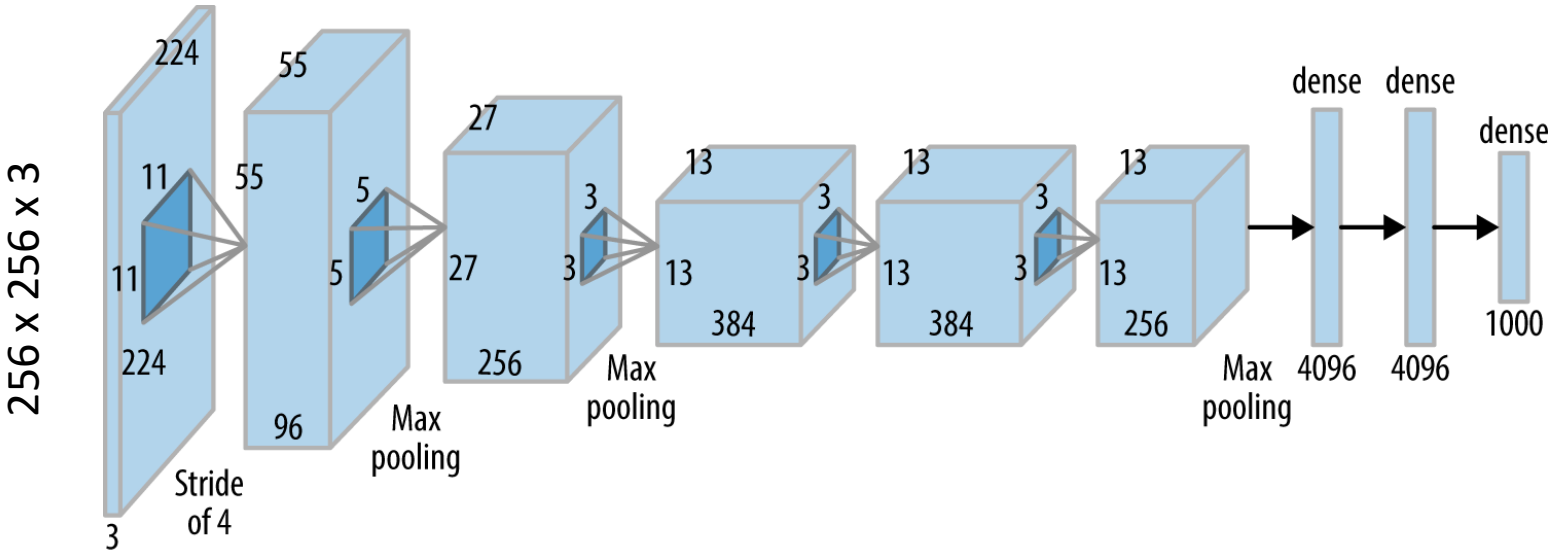
Alex Net, LeNet, Inception-v3



AlexNet

Output: 1000x1 probability vector
one corresponding to each class

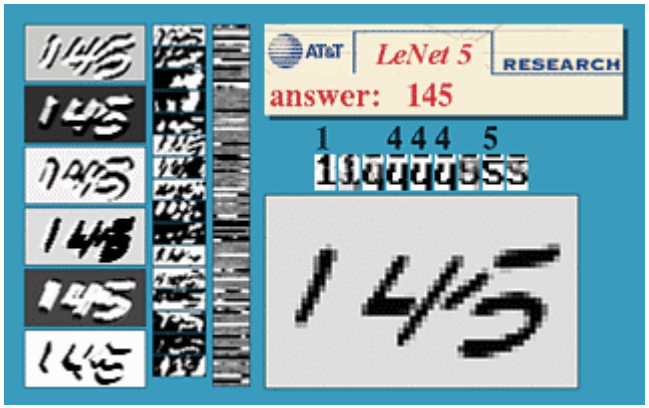
Tested on the ImageNet DB
8 layers (5 conv. Layers + 3 fully conn.)



	AlexNet Network - Structural Details												
	Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
1	227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
	55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
2	27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
	27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
3	13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
4	13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
5	13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
	13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
							fc6			1	1	9216	4096	37752832
							fc7			1	1	4096	4096	16781312
							fc8			1	1	4096	1000	4097000
	Total													**62,378,344**

LeNet 5 (by Yann LeCun)

5 layers (3 conv. Layers + 2 fully conn.)



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

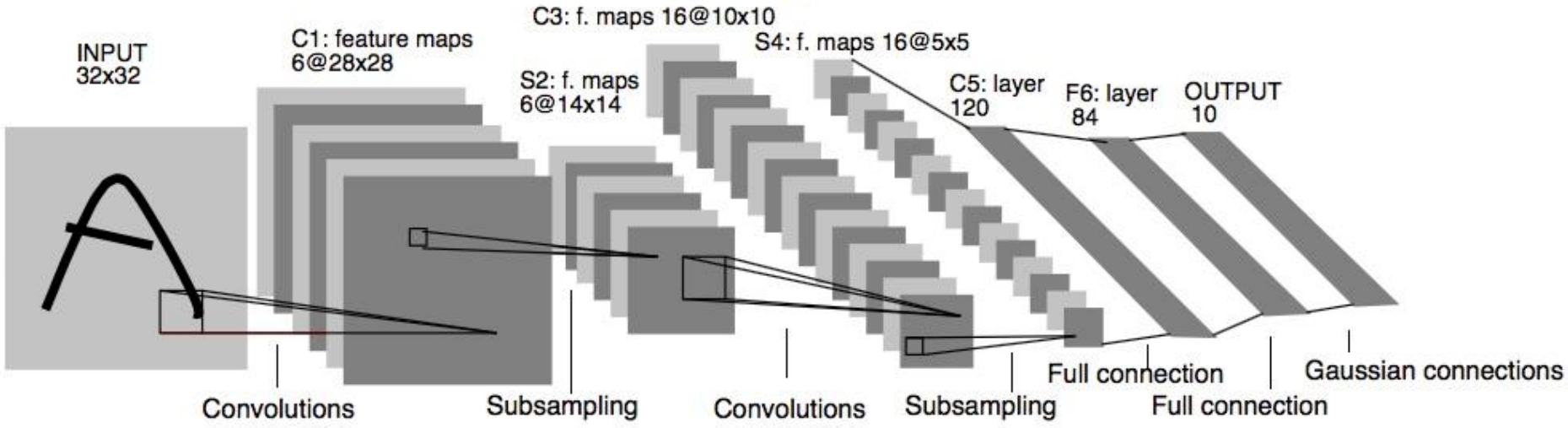
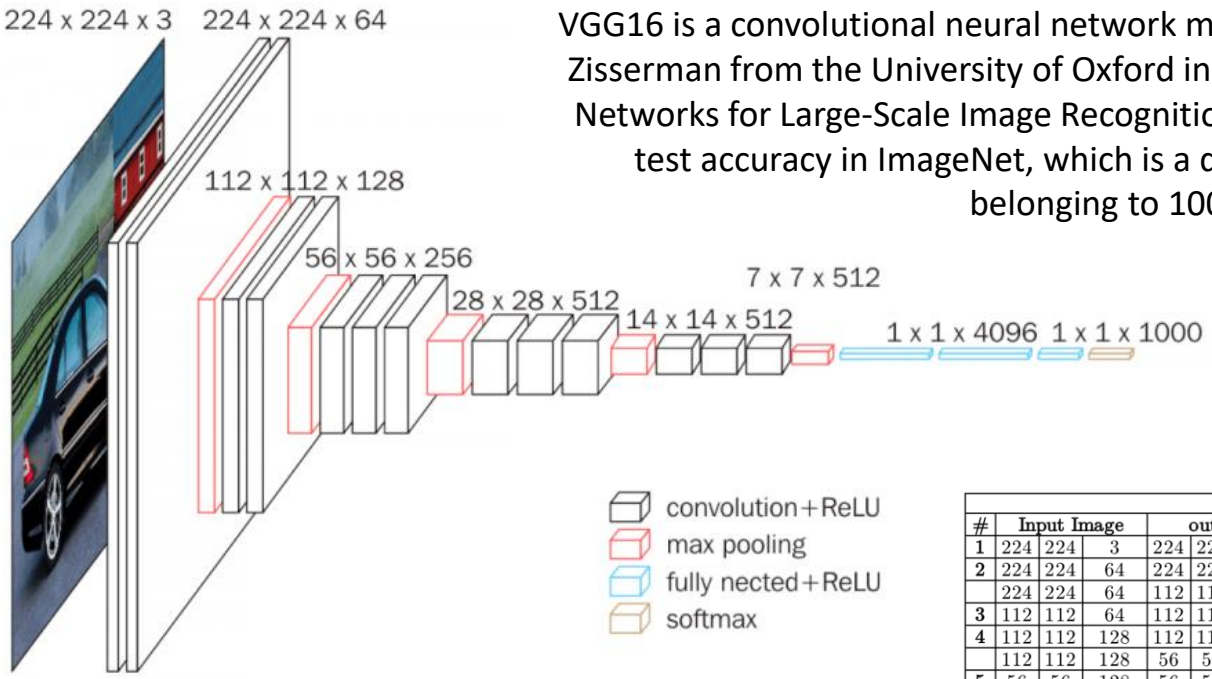


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

16 layers (13 conv. Layers + 3 fully conn.)

VGG16: Classification and Detection



VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total													138,423,208

Example: VGG-FACE-CNN + PCA

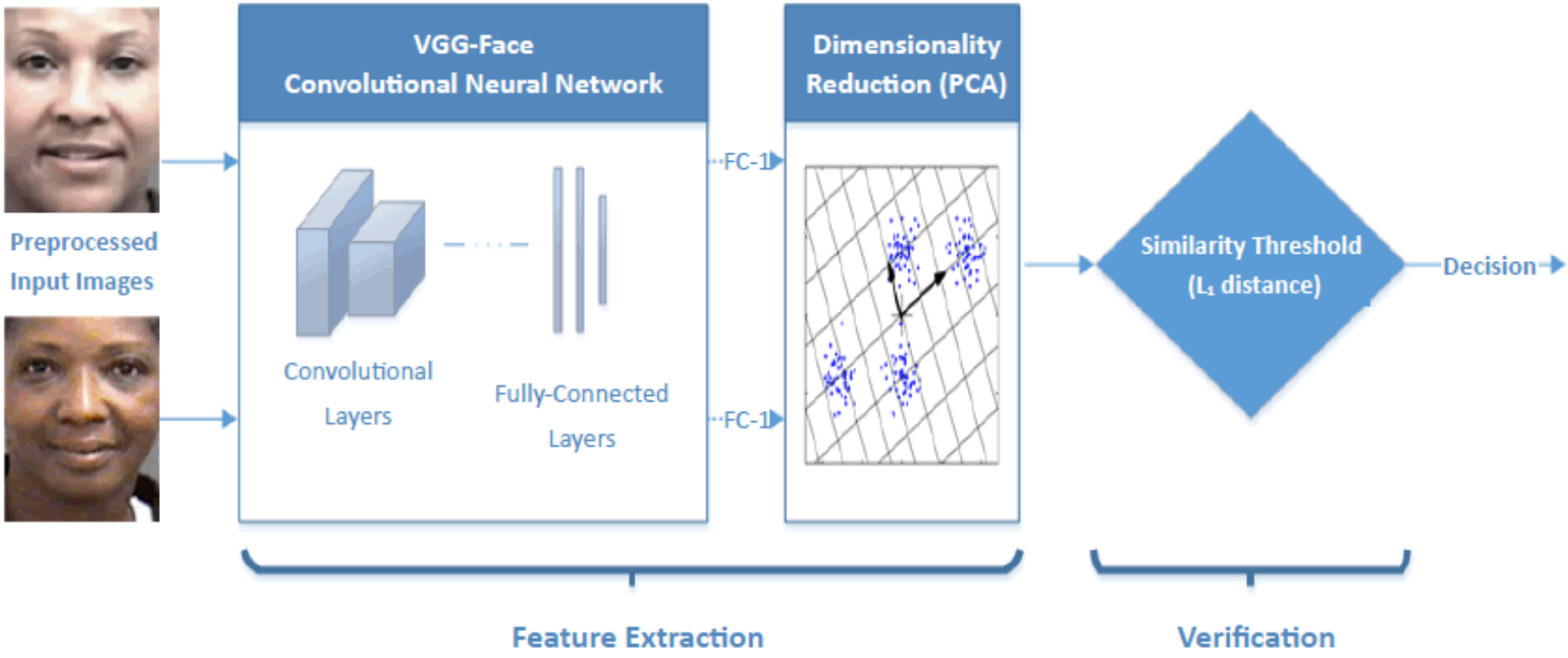


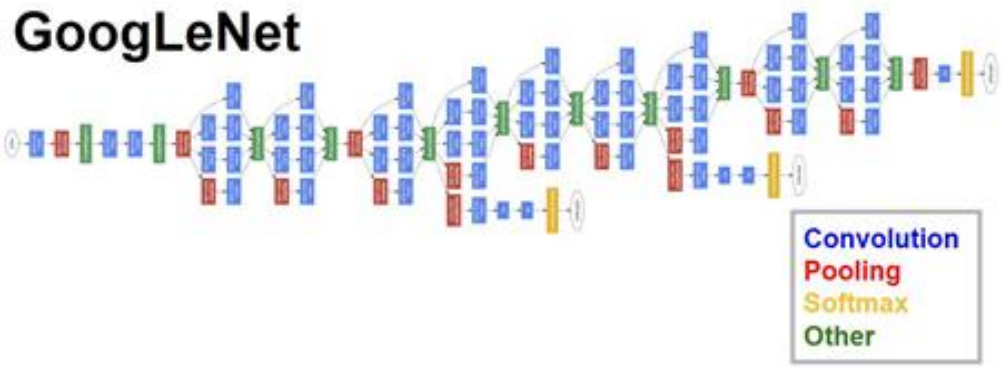
Figure 2: Authentication architecture using VGG-Face CNN feature descriptors.

https://www.robots.ox.ac.uk/~vgg/software/vgg_face/

Other interesting public deep models for classification / feature extraction /pre-training: *GoogLeNet*

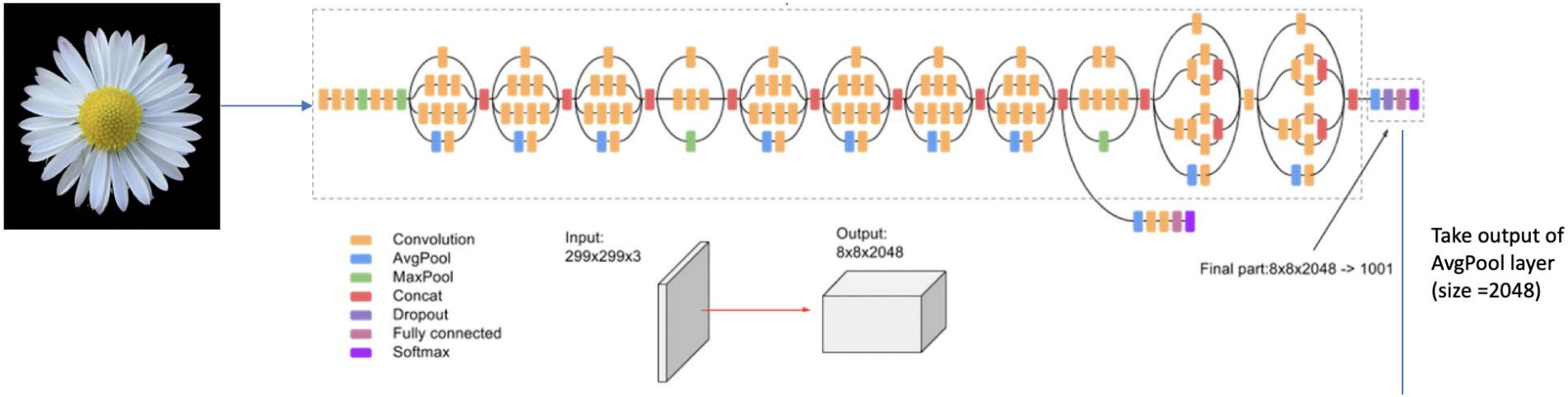
According to your applications, choose the **pretrained** network taking into account **net dimensions** and **type of training dataset**

- GoogLeNet is a convolutional neural network that is 22 layers deep.
- You can load a pretrained version of the network trained on different datasets
  - ImageNet (same as AlexNet)
    - The network trained on ImageNet classifies images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.
  - Places365 data sets.
    - The network trained on Places365 is similar to the network trained on ImageNet, but classifies images into 365 different place categories, such as field, park, runway, and lobby.



Other interesting public deep models for classification / feature extraction /pre-training: Inception-v3

- A convolutional neural network that is 48 layers deep.
- You can load a pretrained version of the network trained on more than a million images from the ImageNet database.

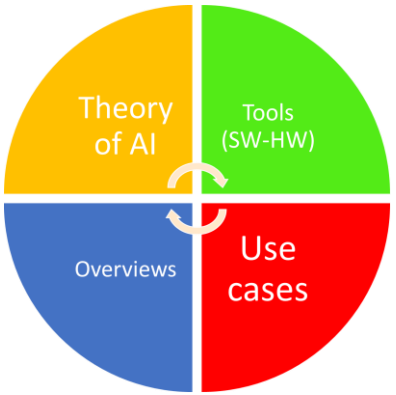


Note: **Dropout** refers to dropping out units (both hidden and visible) in a neural network.
It's regularization approach reducing interdependent learning amongst the neurons and overfitting.

CODING

Deep learning

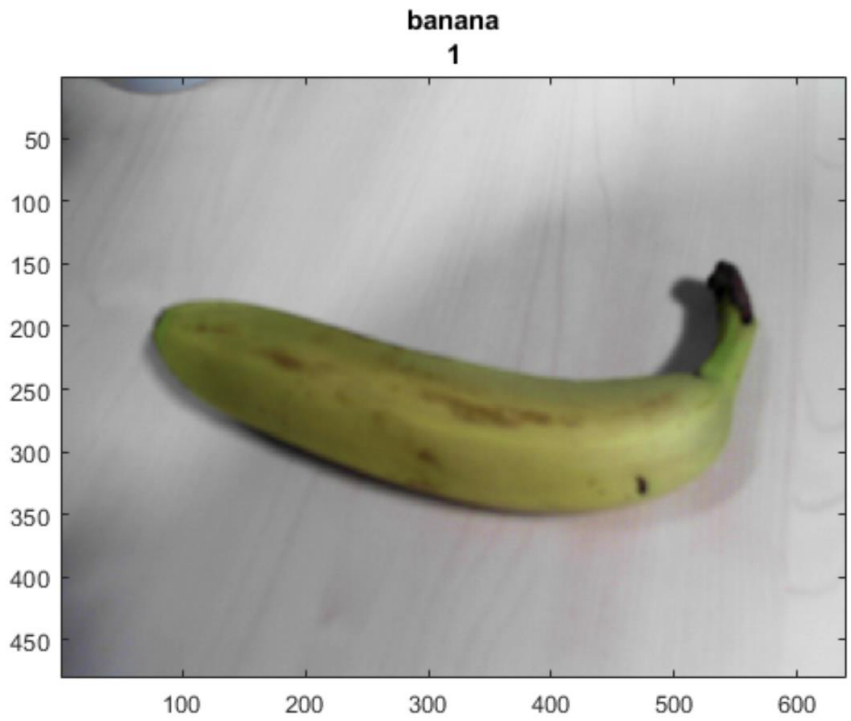
Using Google LeNet
for image classification in Matlab



Coding Image Classification with Matlab and *GoogLeNet*

```
% Note:
% in case open Add-On Explorer to install the Webcam
and GoogLeNet

% copy and paste this code in your matlab
camera = webcam;
net = googlenet;

% change size of the input image to network size
inputSize = net.Layers(1).InputSize(1:2)

figure
im = snapshot(camera);
image(im)
im = imresize(im,inputSize);
[label,score] = classify(net,im);

title( {char(label), num2str(max(score),2)} );
```

# of digits

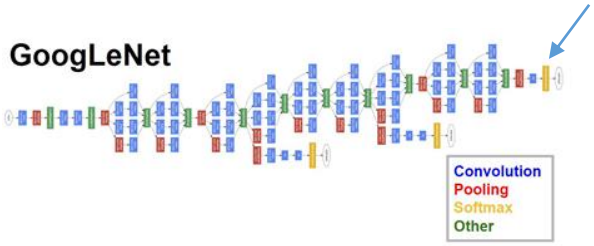


More code here: https://it.mathworks.com/help/deeplearning/ug/classify-images-from-webcam-using-deep-learning.html

More code here: https://it.mathworks.com/help/deeplearning/ug/classify-images-from-webcam-using-deep-learning.html

Output of GoogLeNet

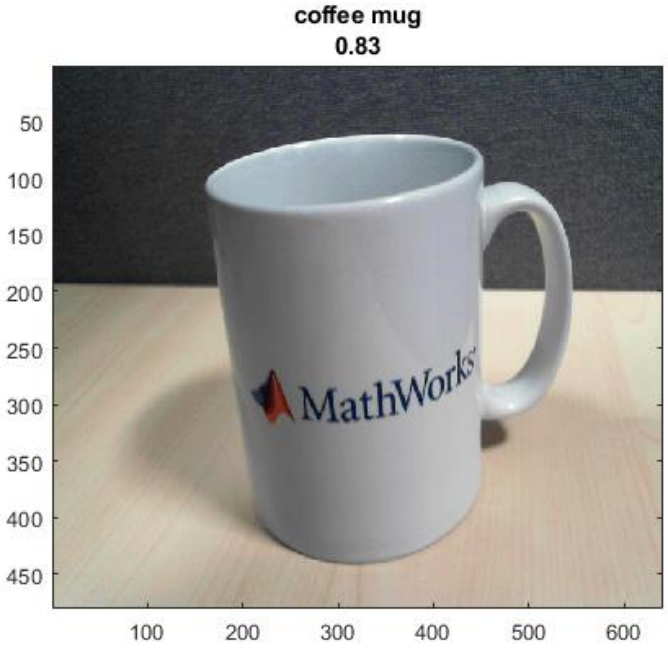
dealing with a softmax layer

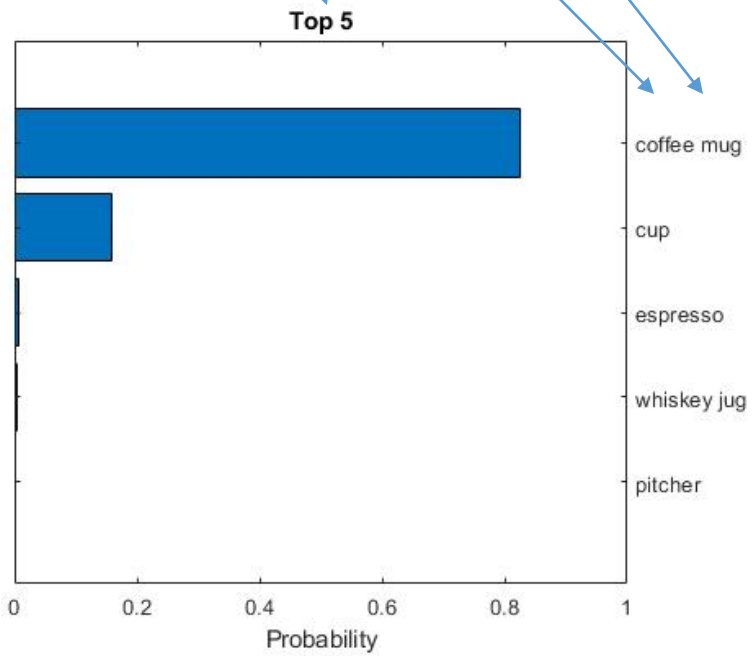



```
[label,score] = classify(net,im);
```

```
[~,idx] = sort(score,'descend');
idx = idx(5:-1:1);   %top 5 desceasing
classes = net.Layers(end).Classes;
classNamesTop = string(classes(idx));
scoreTop = score(idx);      %max score
```

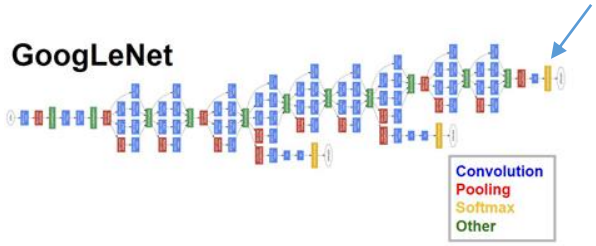
```
h = figure; h.Position(3) = 2*h.Position(3);
ax1 = subplot(1,2,1); ax2 = subplot(1,2,2);

barh(ax2,scoreTop)
xlim(ax2,[0 1])
xlabel(ax2,'Probability')
yticklabels(ax2,classNamesTop)
ax2.YAxisLocation = 'right';
title(ax2,'Top 5')
```



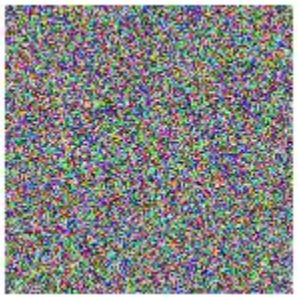


More code here: https://it.mathworks.com/help/deeplearning/ug/classify-images-from-webcam-using-deep-learning.html

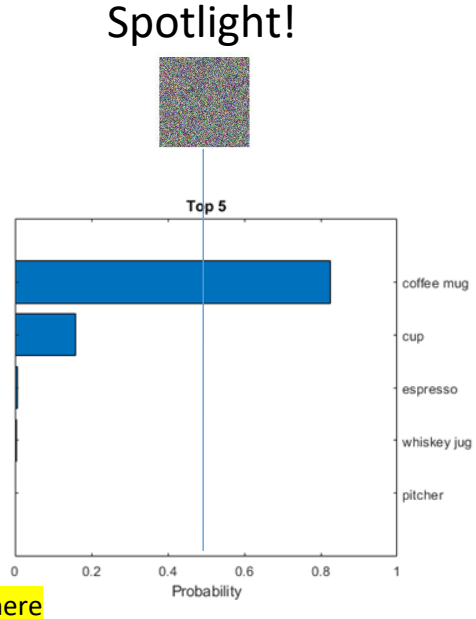
GoogLeNet
try a random input



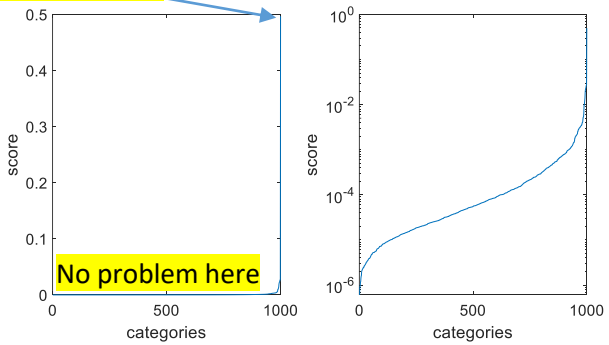

```
im = rand(224,224,3);
imshow(im)

tic;
[label,score] = classify(net,im);
Toc
>> 0.417492 seconds
>> label
label =
  categorical
     spotlight
```



max(score) → 0.4968



Only class Spotlight is here

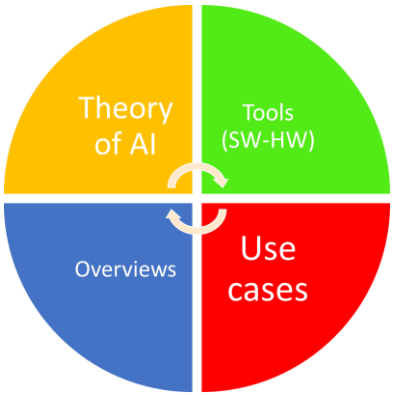



```
% score analysis
subplot(1,2,1); plot(sort(score));
xlabel('categories'); ylabel('score')
subplot(1,2,2); semilogy(sort(score));
xlabel('categories'); ylabel('score')
```

CODING

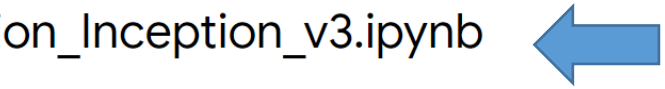
Deep learning

Using Inception-V3
for image classification in CoLab



Coding Image Classification with CoLab and Inception-v3

Load in your CoLab the following example

Image_classification_Inception_v3.ipynb 

Note. In Colab you can write shell commands preceded with a '!' within the notebook. It gives us control over the Virtual Machine (VM) to install other non default packages

```
[1]  !pip install -q keras
```

Any image recognition model available in Keras can be loaded with just two lines of code.

```
[2]  from keras.applications.inception_v3 import InceptionV3
     model = InceptionV3(weights='imagenet', include_top=True)
```

Load the network

Coding Image Classification with CoLab and Inception-v3

To start to make Image Classification we can define a fuction predict() with in input the input image and in output the decode predicted output class.

```
import numpy as np
from keras.preprocessing import image
from keras.applications.inception_v3 import preprocess_input, decode_predictions
from PIL import Image

# prediticton function only for TOP 5 matchings
def predict(model, img, top_n=5):

    # Network Inception_v3 requires resizedinput size, in this case target_size=(299, 299),
    # The NumPy array is correct for the use with deep learning models
    x = np.expand_dims(img, axis=0)   # required input preparation

    #  data normalization to zero center the image data using the mean channel values from the training dataset. This is an extremely important step!
    x = preprocess_input(x)  # The preprocess_input function is meant to adequate your image to the format the model requires

    # HERE'S THE CLASS PREDICTION  (ONLY TOP top_n)
    preds = model.predict(x)
    return decode_predictions(preds, top=top_n)[0]
```

Just some pre-Processing

The real prediction
+ selection of the best 5 matchings

Coding Image Classification with CoLab and Inception-v3

```
[4]  #  click on your own file in the dir e.g. Truck.jpg #
     from google.colab import files
     uploaded = files.upload()
     for fn in uploaded.keys():
      print('User uploaded file "{name}" with length {length} bytes'.format(name=fn, length=len(uploaded[fn])))

      print(fn)
```

Scegli file Truck.jpg

- ***Truck.jpg***(*image/jpeg*) - *593821 bytes, last modified: 24/5/2020 - 100% done*

```
Saving Truck.jpg to Truck (4).jpg
User uploaded file "Truck.jpg" with length 593821 bytes
Truck.jpg
```

Load an example

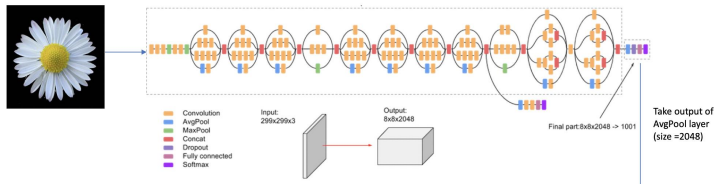
Coding Image Classification with CoLab and Inception-v3

Once you loaded the image, you have to resize to the input size of 299,299 pixels

```
[6]  import matplotlib.pyplot as plt
     import matplotlib.image as mpimg
     import cv2

     img = mpimg.imread(fn)
     img  = cv2.resize(img, dsize=(299, 299), interpolation=cv2.INTER_CUBIC)
     print(fn)
     plt.imshow(img)
```

```
Truck.jpg
<matplotlib.image.AxesImage at 0x7f6c11275890>
```

Resize to 299x299x3
to fit the input of the CNN



Coding Image Classification with CoLab and Inception-v3

Here we have the real predicition

```
[8]  print(model)
     print(fn)

     pred = predict(model, img)
     print(pred[0])
```

Execute the prediction

Just print the most probabile match

```
     <tensorflow.python.keras.engine.functional.Functional object at 0x7f6c26a390d0>
     Truck.jpg
     ('n03796401', 'moving_van', 0.43757406)
```

...

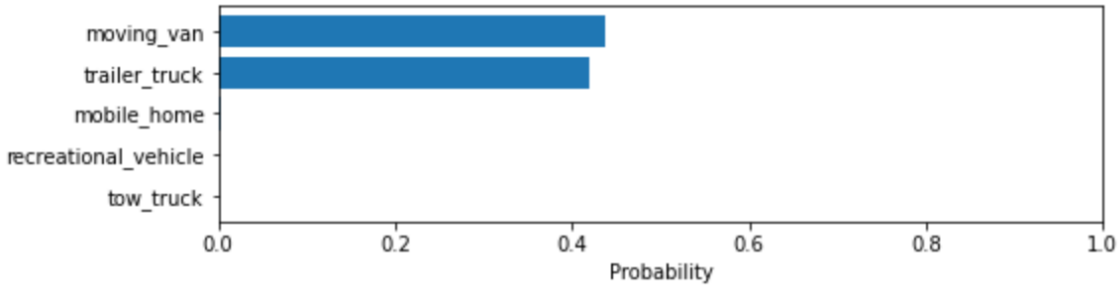
Coding Image Classification with CoLab and Inception-v3

Plot the probabilty in the output pred as an image

```
[9]  # just for visualization we will use it to plot the output classes as an histogram image
     # print as image the bar plot  of the probabilities
     plt.figure(figsize=(8, 2))
     classes = [c[1] for c in pred]
     probas = [c[2] for c in pred]
     y_pos = np.arange(len(classes))
     plt.barh(y_pos, probas, align='center')
     plt.yticks(y_pos, classes)
     plt.gca().invert_yaxis()
     plt.xlabel('Probability')
     plt.xlim(0, 1)
     print(classes)
```

Plotting the probabilities of the final softmax layer

```
['moving_van', 'trailer_truck', 'mobile_home', 'recreational_vehicle', 'tow_truck']
```



Main points

1. Deep learning VS classical neural networks
2. Convolutional neural networks
    1. Configuring
    2. Training
    3. Testing
3. Coding in Keras
4. Examples of large and public CNNs for applications
5. Image classification in
    - Matlab (GoogleLeNet)
    - Colab (Inception-V3)