Introduction to computer vision : from OpenCV to YOLO

Damien Meur

Proxyclick

dmeur@proxyclick.com

March 4, 2021

Overview

Required tools

- Python 3.6+
- OpenCV 4.X+
- TensorFlow 2.X
- Jupyter 1.X+

Links

- Anaconda : `docs.anaconda.com/anaconda/install/`
- GitHub repository : `github.com/damioune123/computer_vision_crash_course`

Installation with anaconda

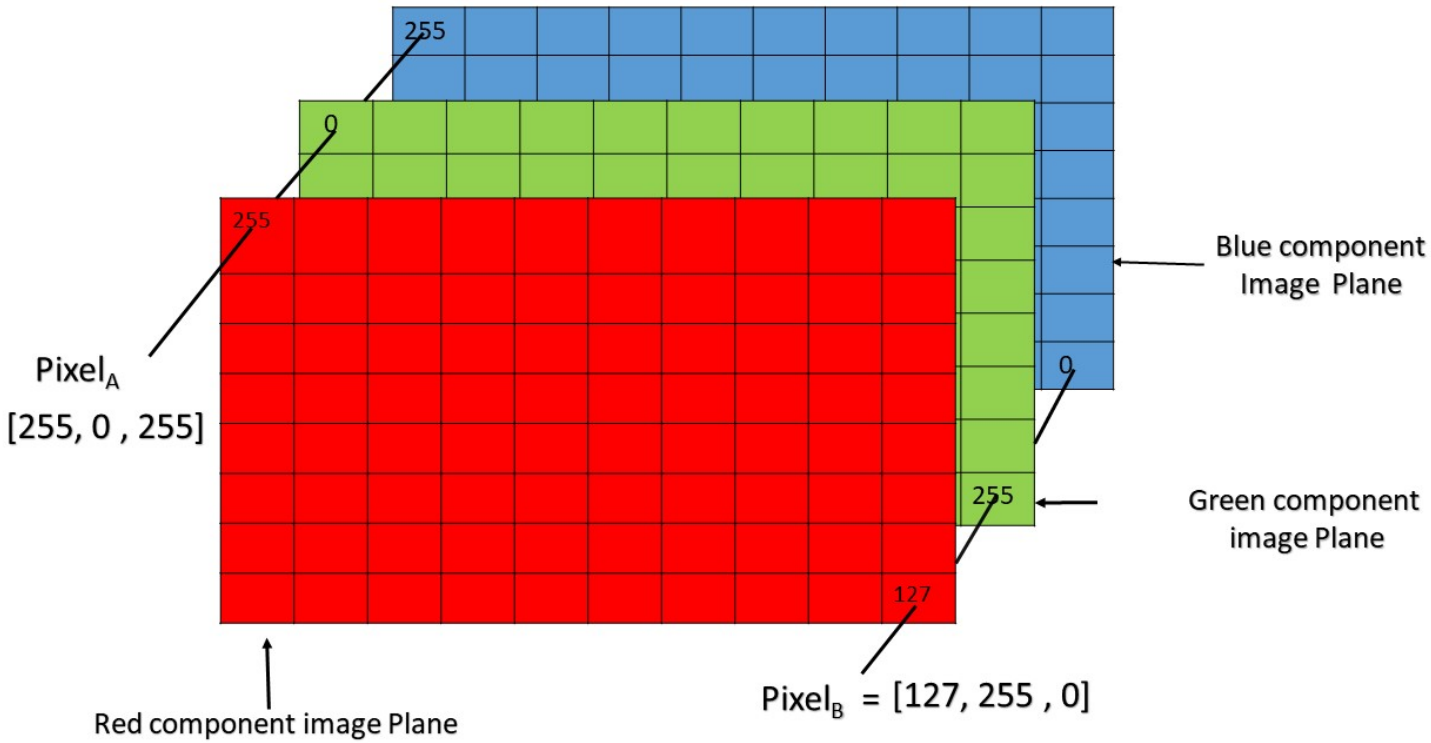
```
#installing conda environment
conda create -n cv --clone base
conda activate cv
pip install --upgrade pip
pip install opencv-python
pip install --upgrade tensorflow
pip install jupyter

#setting up our Jupyter notebook for this course
git clone https://github.com/damioune123/computer_visi
cd computer_vision_crash_course
jupyter notebook
```

What is computer vision ?

- The human brain is amazingly good at **detecting and classifying objects**, a large part of our brain is dedicated to vision (occipital lobe).
- **CV** is a collection of machine learning techniques that makes computer mimic these two specific tasks.
- How a computer would do this task that is so easy to us though ? First let's dig into how a computer "sees".

Numeric image representation



Pixel of an RGB image are formed from the corresponding pixel of the three component images

What are images ?

Basically a matrix of pixels, each pixel is represented by 3 color (RGB) values (0-255)

Numeric image representation

Is it possible to use these raw values directly to detect or classify objects ?

No

Let's take an average image (3.1 Megapixels resolution) :

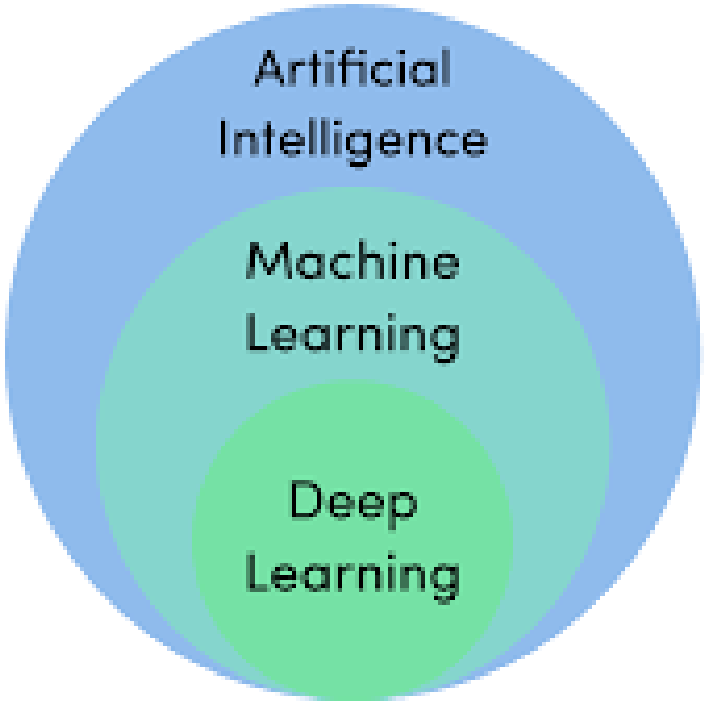
- 2048 X 1536 pixels => 3,145,728 pixels.
- X 3 color channels => 9,437,184 parameters !

Could we simplify the problem ?

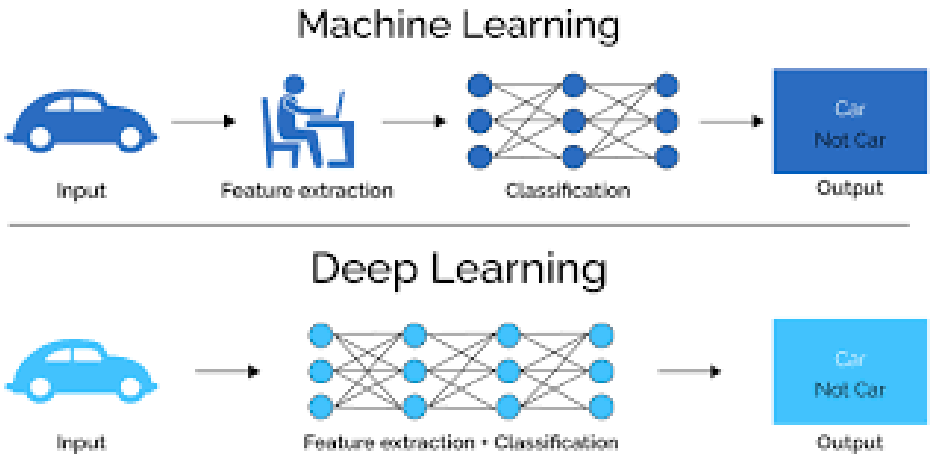
Yes

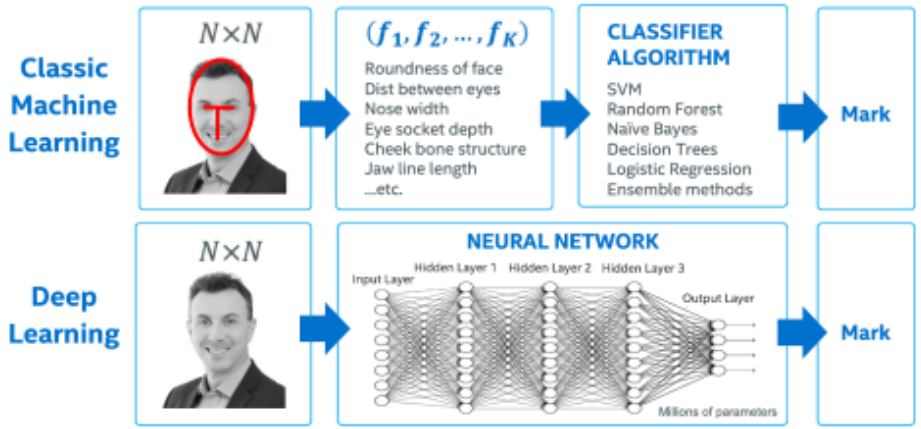
By keeping only the relevant information (**feature extraction**). Bearing in mind there is a trade-off acccuracy vs performance.

Different technical approaches : old machine learning vs new deep learning techniques



Different technical approaches : old machine learning vs new deep learning techniques





Different technical approaches : old machine learning vs new deep learning techniques

	HAAR-like features cascade classifiers	YOLO deep network
Single-class object detection accuracy	Good	Very good
Single-class object detection performance	Excellent	Very good
Multi-class object detection accuracy	N/A	Very good
Multi-class object detection performance	N/A	Very good
Object classification accuracy	N/A	Very good
Object classification performance	N/A	Very good

1 2

[1] https://baseapp.com/opencv-vs-yolo-face-detector

[2] https://towardsdatascience.com/diff-between-yolo-and-haar

Different technical approaches : old machine learning vs new deep learning techniques

It seems HAAR cascade is obsolete...

`https://www.youtube.com/watch?v=XkfSqOvJRIw`

...But

- It's also easier to understand/implement
- Requires less resource (still widely used on embedded system such as cameras) [a]

[a] `https://www.youtube.com/watch?v=uEJ71VlUmMQ`

I choose to present you 3 techniques in this order :

- HAAR cascade : face and eye detection program
- Custom convolutional network : Face mask detector
- YOLO : Use the pre-trained network to detect any common objects

Hands on OpenCV

OpenCV

Open Source Computer Vision Library is a library of programming functions mainly aimed at real-time computer vision.

- Built with C++
- Python/Java/Octave/Matlab official interfaces
- Contains a lot of tools to do basic/advanced image processing
- Has built-in machine learning object detection algorithms such as HAAR cascade

Basic image manipulations

Let's do some image manipulations on OpenCV to better understand this library

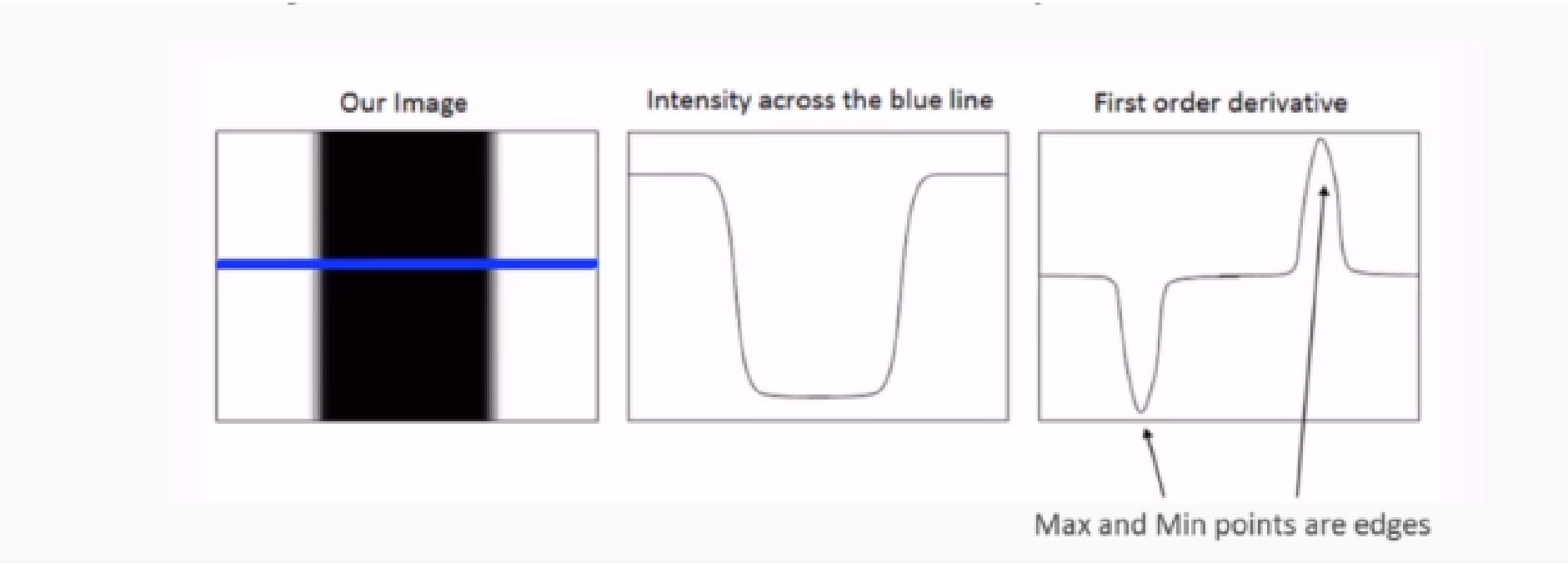
Edge detections

Edges

Edges are sudden discontinuities in an image. They hold a lot of information about what's in the image.

Edge detection

Edge detection is a very important in computer vision : helps with finding contours.



Edge detection

They are lots of different type of edge detection (all available in OpenCV)

- Sobel : vertical or horizontal
- Laplacian : all orientations
- Canny (created by J.F Canny in 1986): optimal => low error rate, accurate (few noise detected as edge)

Edge detection

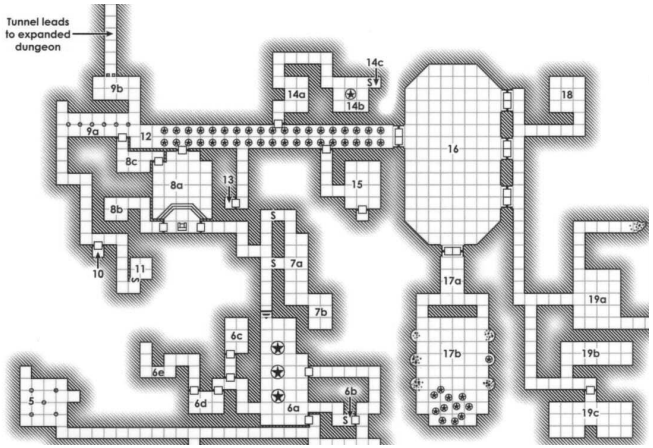


Figure: Original

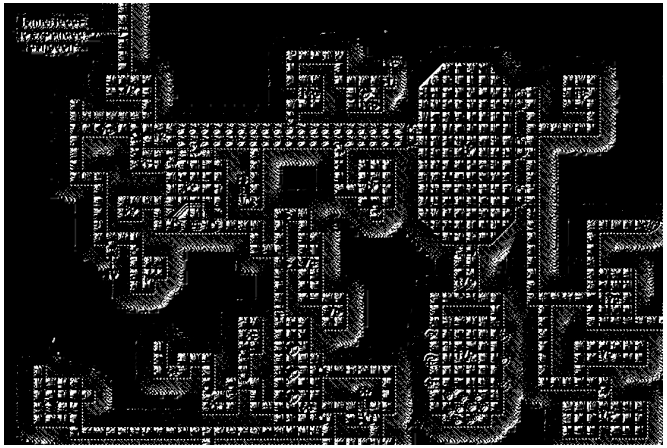


Figure: Sobel (X + Y merged)

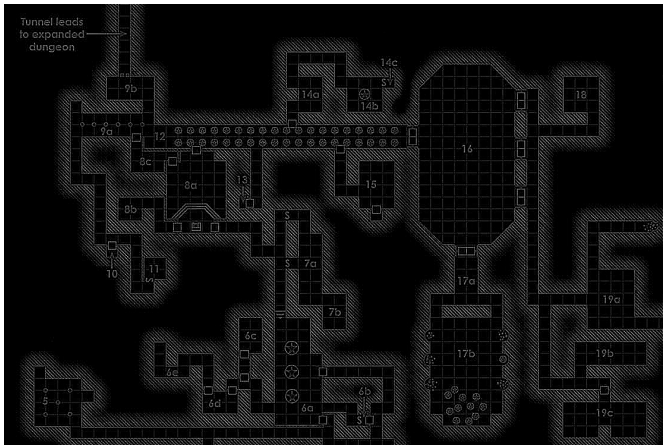


Figure: Laplacian

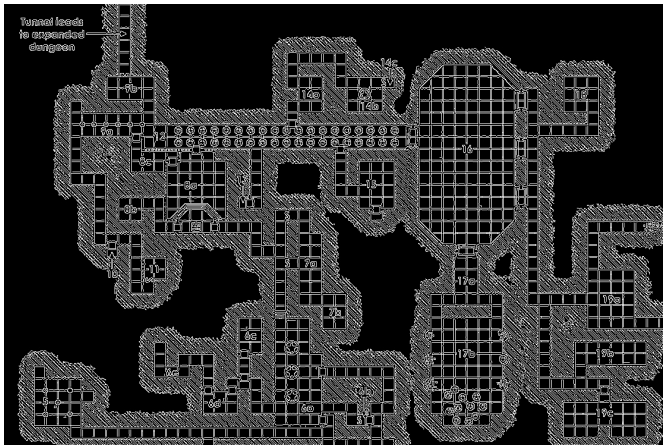


Figure: Canny

Edge detection

```
import cv2
image = cv2.imread('images/dungeon.png')

# Grayscale
image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

sobel_x = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)
sobel_y = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
sobel_OR = cv2.bitwise_or(sobel_x, sobel_y)

laplacian = cv2.Laplacian(image, cv2.CV_64F)

canny = cv2.Canny(image, 30, 200)
```

Edge detection

Combine edge detection with a few OpenCV operations like bitwise operator, intensity threshold, opening (not detailed in this presentation) and you get ...[3]

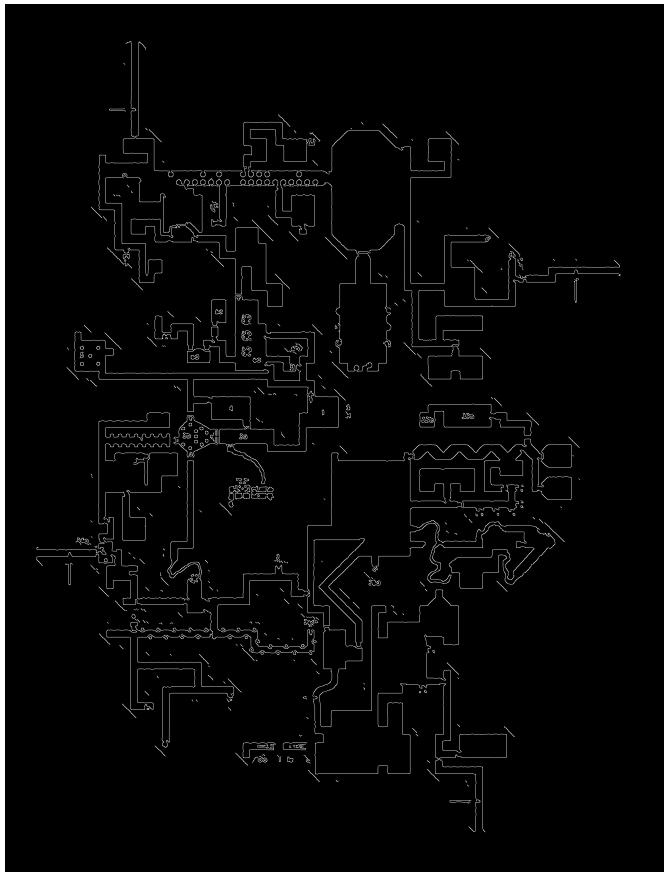


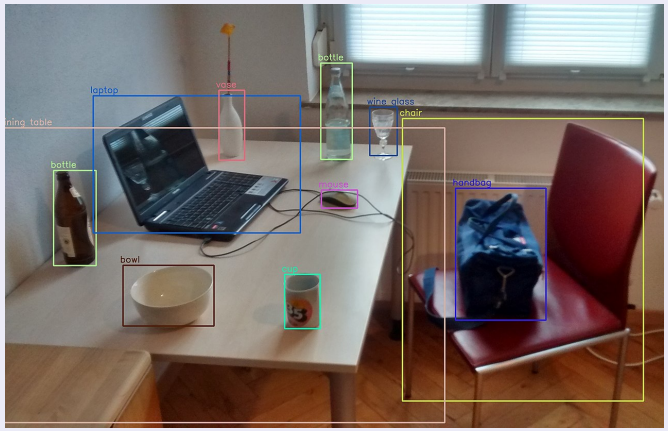
Figure: Dungeon map border extraction example

[3] Code available in the jupyter notebook though

Object detection with HAAR cascade classifiers

What is object detection ?

The ability to detect individual objects within an image.



It is very hard because :

- Objects vary within a class (think of the different dog breeds)
- Different angles of view
- Ambiguity in identification (even for a human : sword, katana, knife, dagger, ...)
- Light/clarity of the scene
- ...

What is HAAR cascade classifiers ?

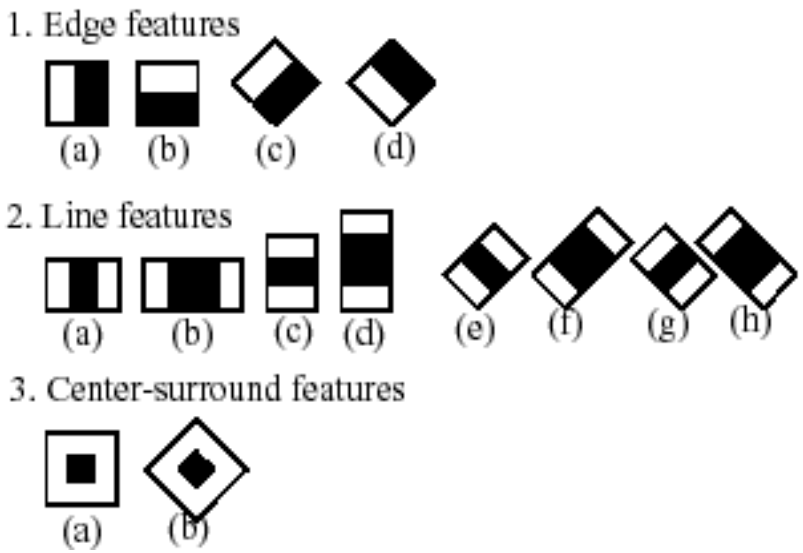
As we saw in traditional machine learning algorithms, **features can be extracted** from an image and **injected into classifiers** to identify objects.

HAAR cascade classifiers

- Created by Paul Viola and Michael Jones in 2001 (also known as the Viola-Jones object detection framework)
- Cascade function is trained with positive/negative images
- Works only for a single class of object but can be used in parallel
- Is composed in 4 steps

First step : HAAR feature selection

- Objects are classified using very simple feature
- Haar features are sequence of rescaled square shape functions proposed by Alfred Haar in 1909
- Each feature is defined by the difference of each pixel intensity within a defined pixel kernel.



First step : HAAR feature selection

Figure: Original

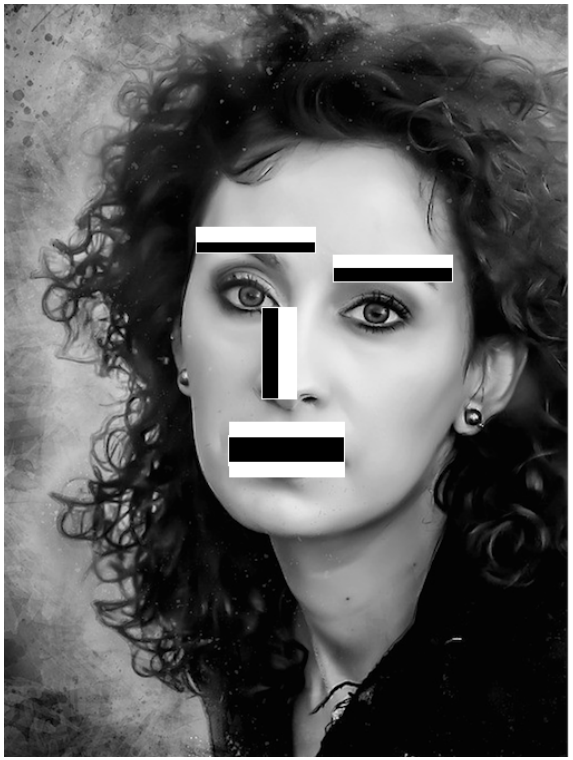


Figure: HAAR features on face

Second step : Integral Image Representation

- The Value of any point is the sum of all the pixels above and left of that point.
- For 100X100 image, with a 9X9 window (slides 100 times) : 800 vs 356 operations

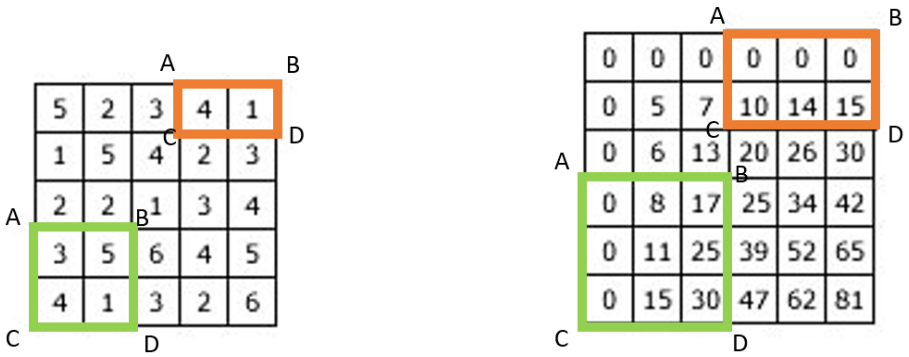


Figure: Original vs integral image representation

The sum of each pixel in the window is simplified by

$$\sum i(x, y) = I(D) + I(A) - I(B) - I(C) \quad (1)$$

Third step : Adaboost training

- All possible locations and size of each kernel is used to generate plenty features
- Using a 24X24 window size of the image will produce 160.000+ features

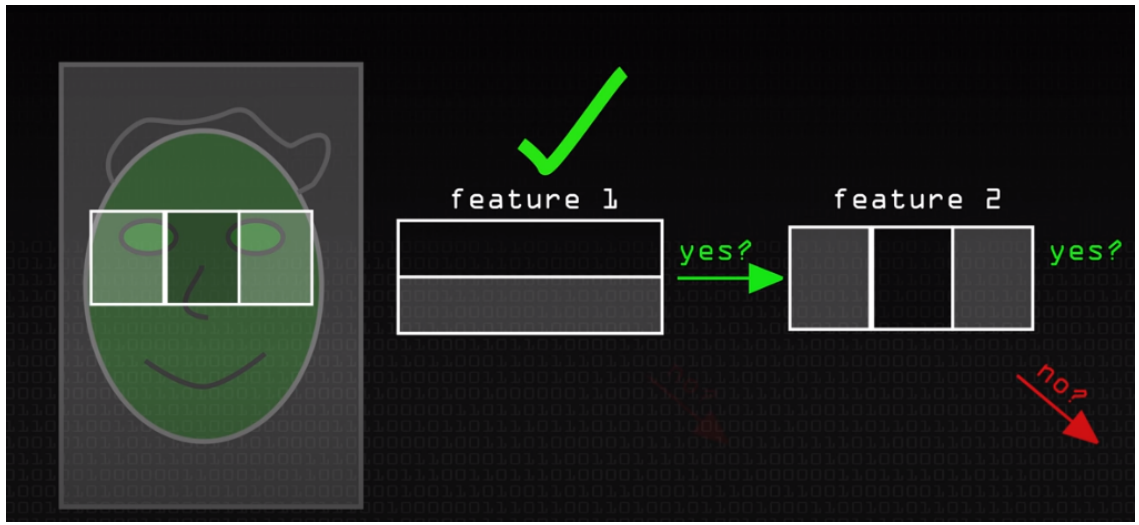
Wow that's a huge number, can we down size this ?

Yes

- Most of the features are irrelevant
- Adaboost machine learning algorithm selects only the relevant ones : 160.000+ => 6000 features

Fourth step : Cascade Classifier Architecture

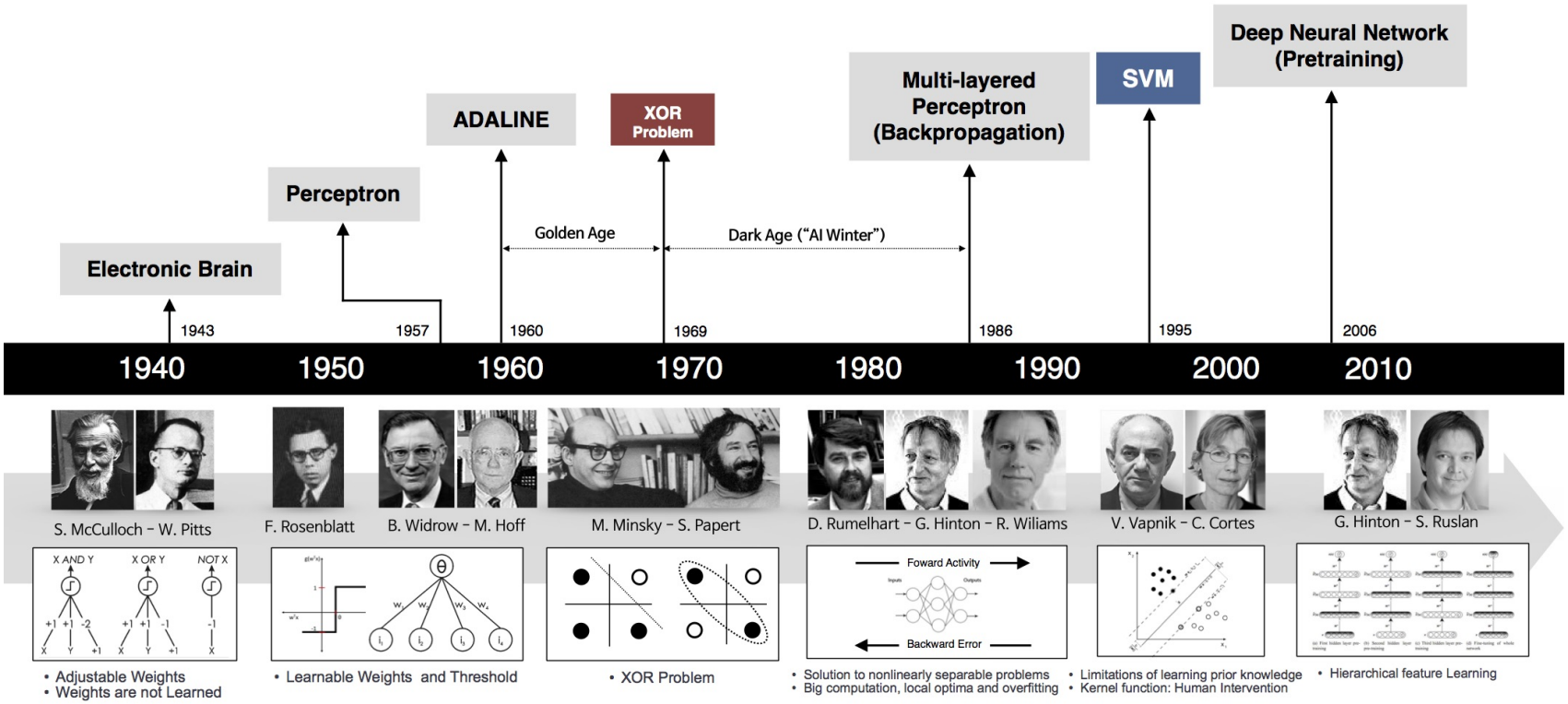
- Sliding a 24X24 window (generating 6000 features) over a 3.1 Megapixels image (120.000 times) would result into 72 million features.
- Binary tree with successive classifier (strong -> weak)
- The quicker we can say "No" the better
- On average only 3 evaluated features per window (72 M -> 360.000)
- Generally we scale down the image with a 2/3 factor (360.000 -> 240.000)



Face and eye detection

Let's use two pre-trained face and eye HAAR cascade classifiers in parallel with OpenCV.

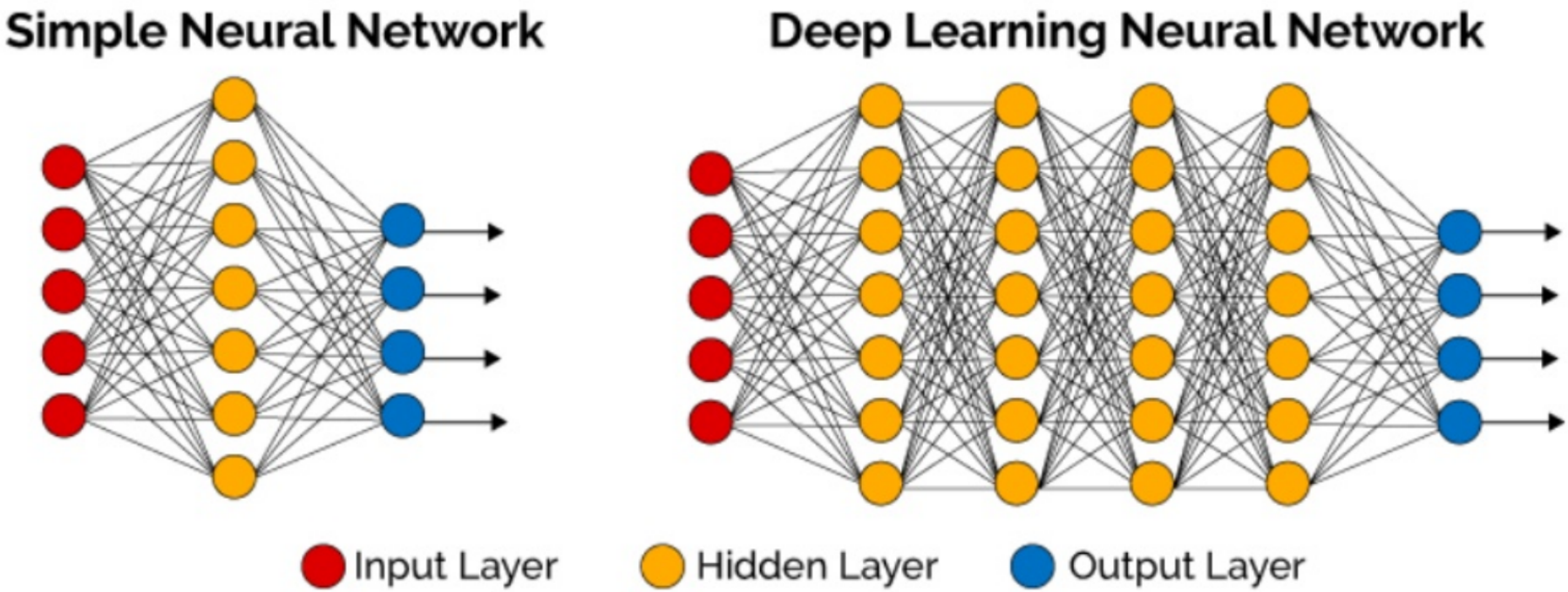
Brief history of neural networks



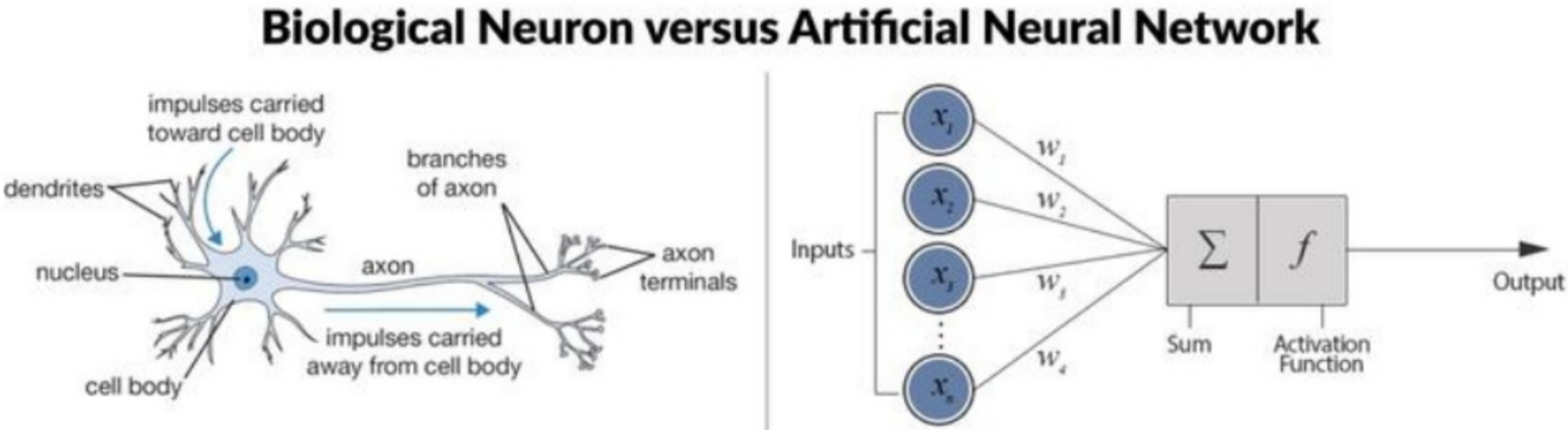
Deeplearning was not possible in the 90's :

- Not enough labeled data or no solution to store them
- Computers were too slow

Neural network vs deeplearning



What is a neuron ?

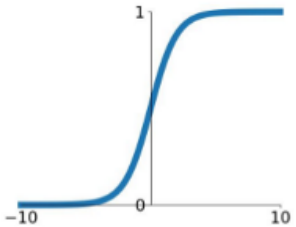


The different activation functions

Activation Functions

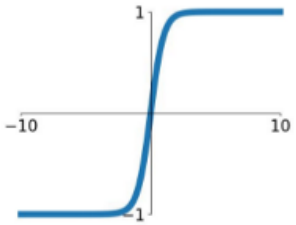
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



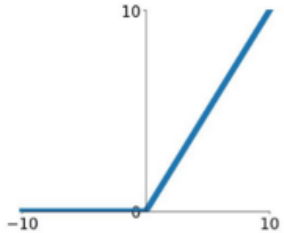
tanh

$$\tanh(x)$$



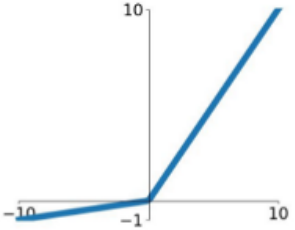
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

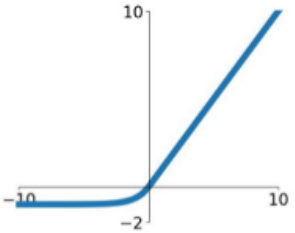


Maxout

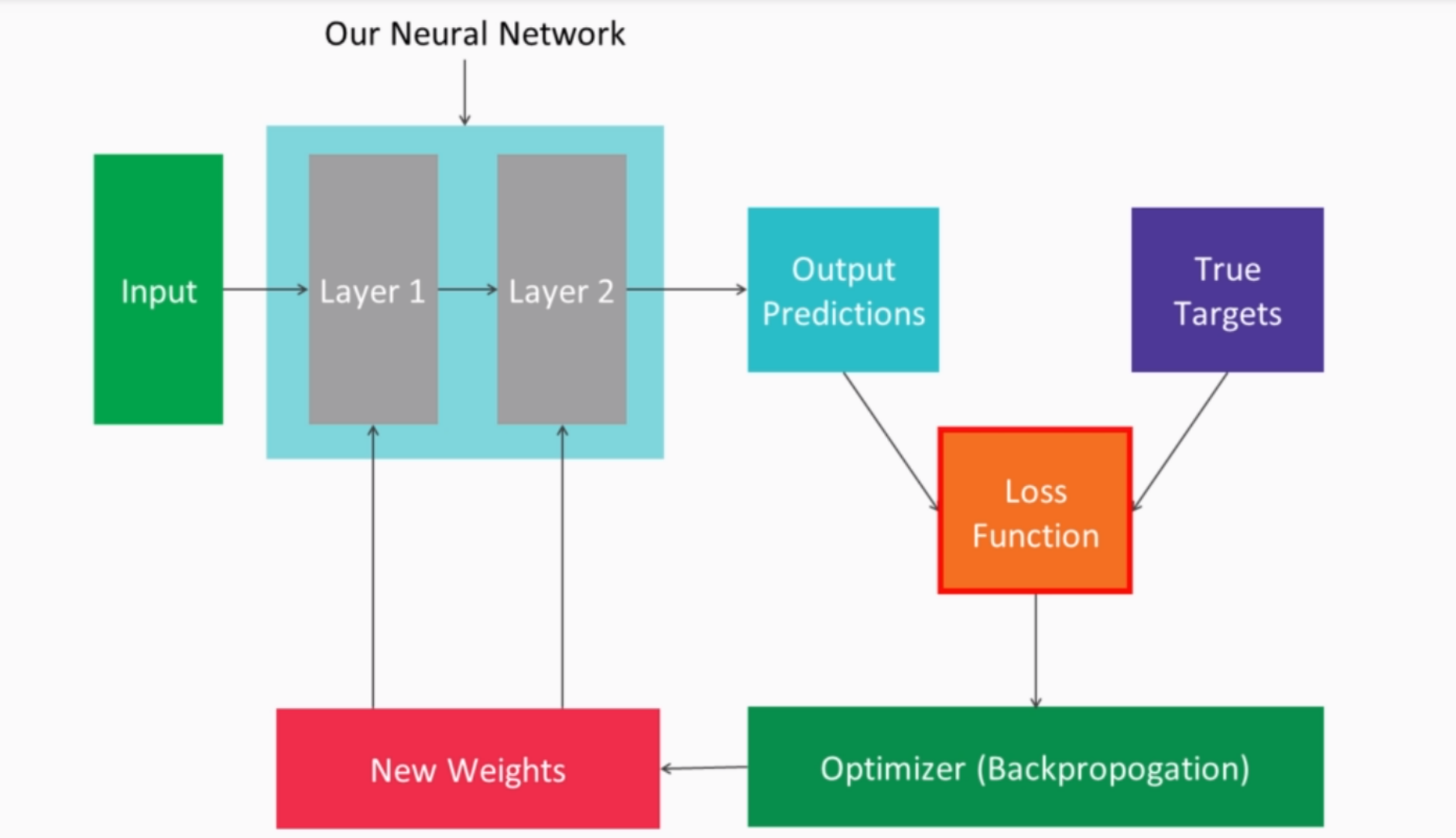
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

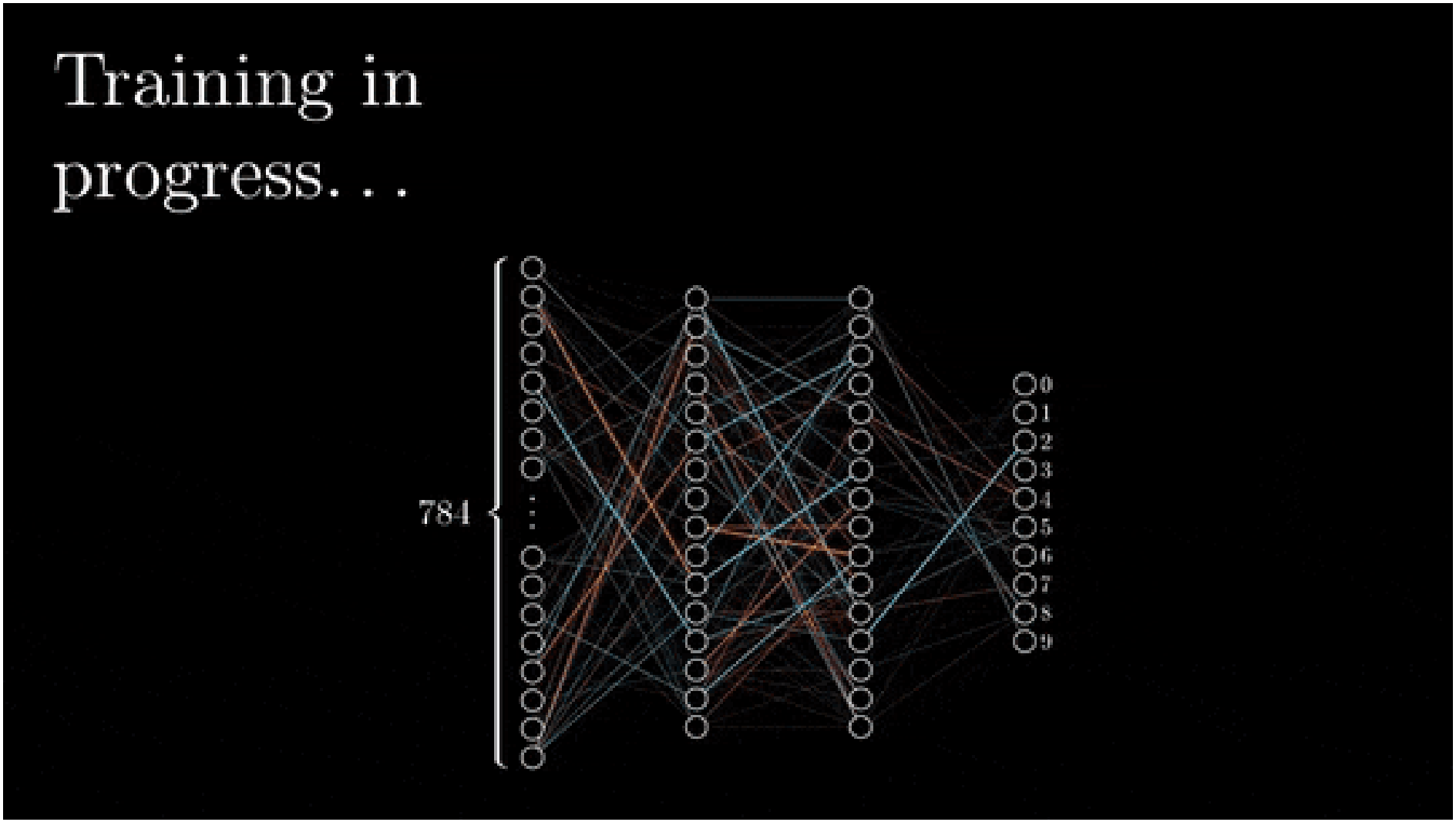
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



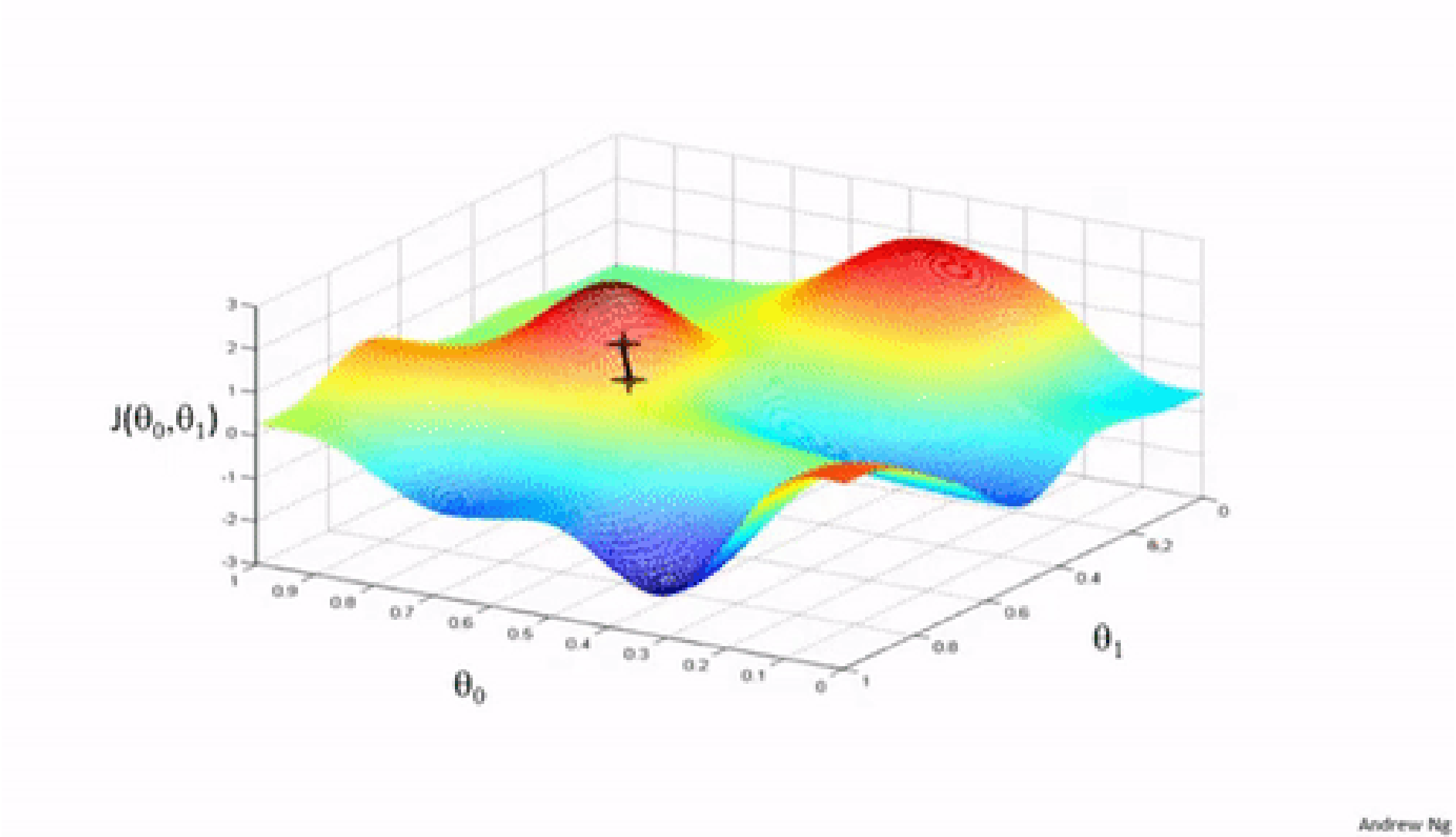
Neural network training



Neural network training : Backpropagation



Neural network training : Gradient descent



Convolutional Neural Network

- Take a colorized 680X480 image, the input of our NN is 680X480X3 = 979.200 weights !

Are all the pixels value unrelated ?

No :

- There are patterns through colors, edges, contours, ...
- All pixels are listed next to each other (spatially related)

Convolutional Neural Network

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

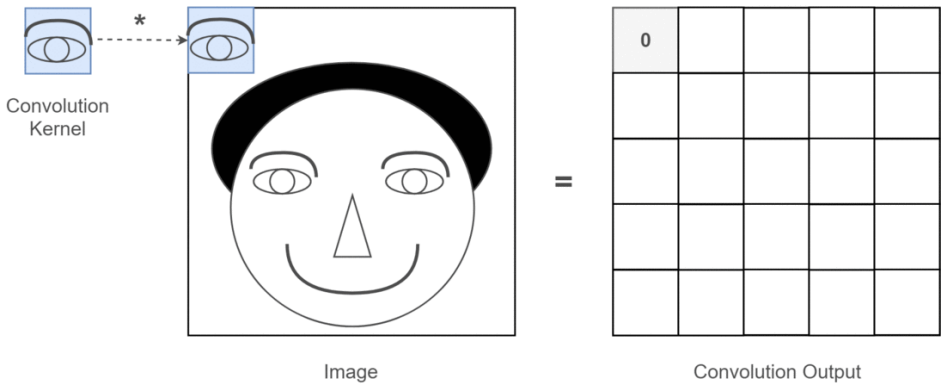
*

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

7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

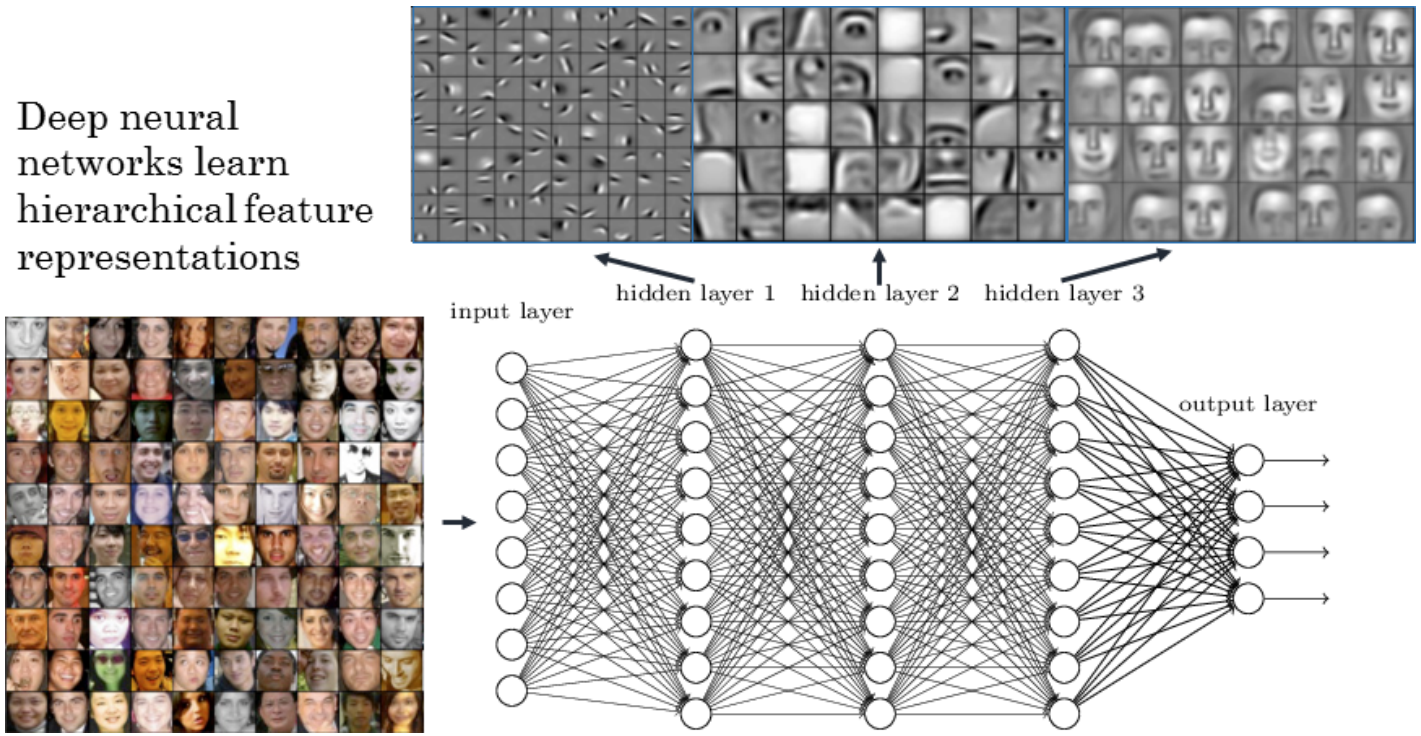
=

6		

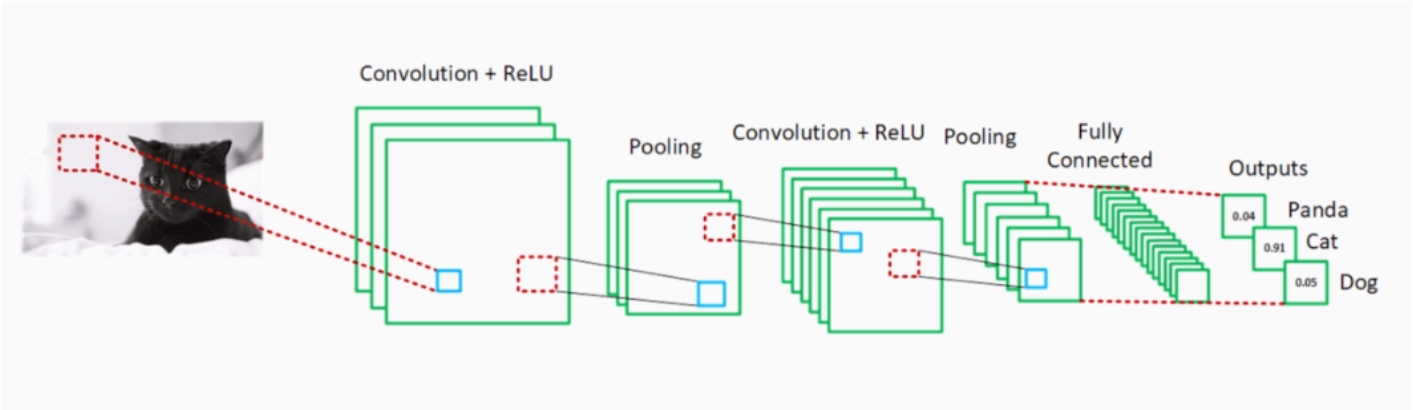


Convolutional Neural Network

Deep neural networks learn hierarchical feature representations

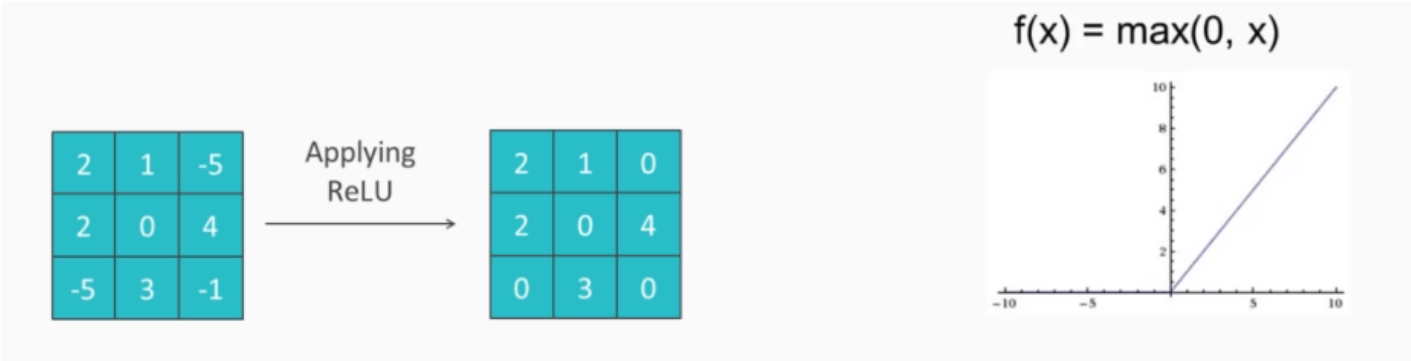


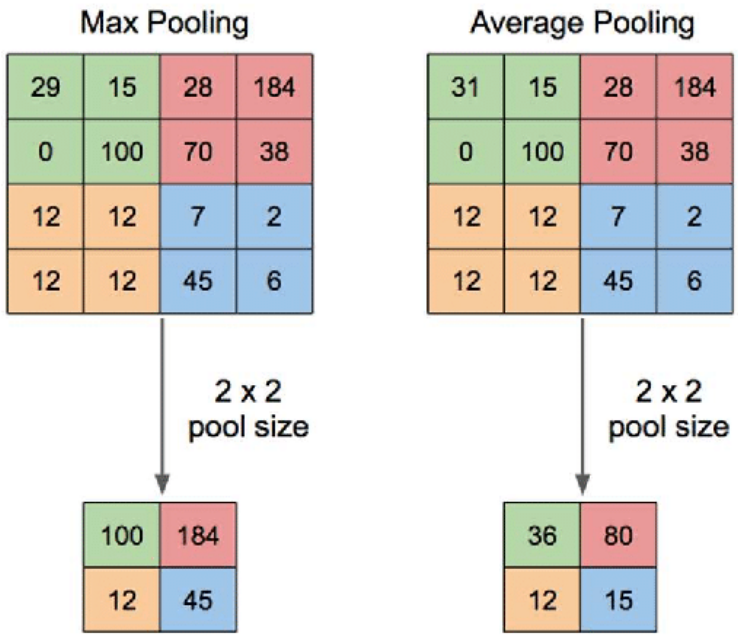
Convolutional Neural Network : Standard architecture



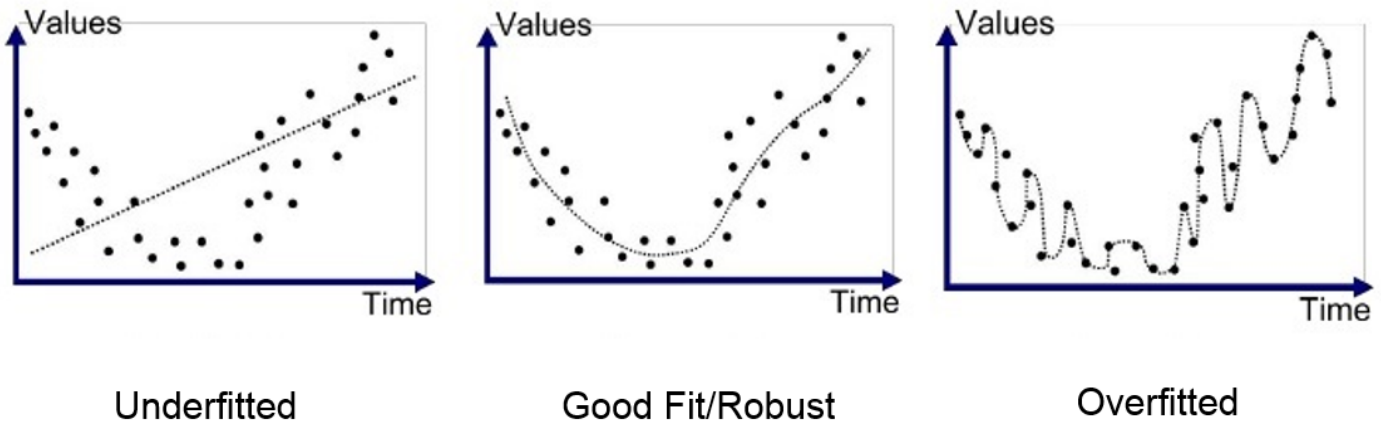
- Input
- Convolution + ReLU (Rectified Linear Unit) layer
- Pool layer (downsampling/subsampling layer)
- Fully connected layer (dense layer)

Convolutional Neural Network : Standard architecture





Convolutional Neural Network : Last problem, Overfitting



Regularization methods

- Dropout (dropping nodes) => ignore some parts of the NN (during training)
- Penalize large weights (L1, L2 techniques)
- Cross validation (K-fold technique) - rotating test/train data
- Early stopping of the training
- Data augmentation (can be done artificially by flipping image e.g)

Convolutional Neural Network : Designing tips

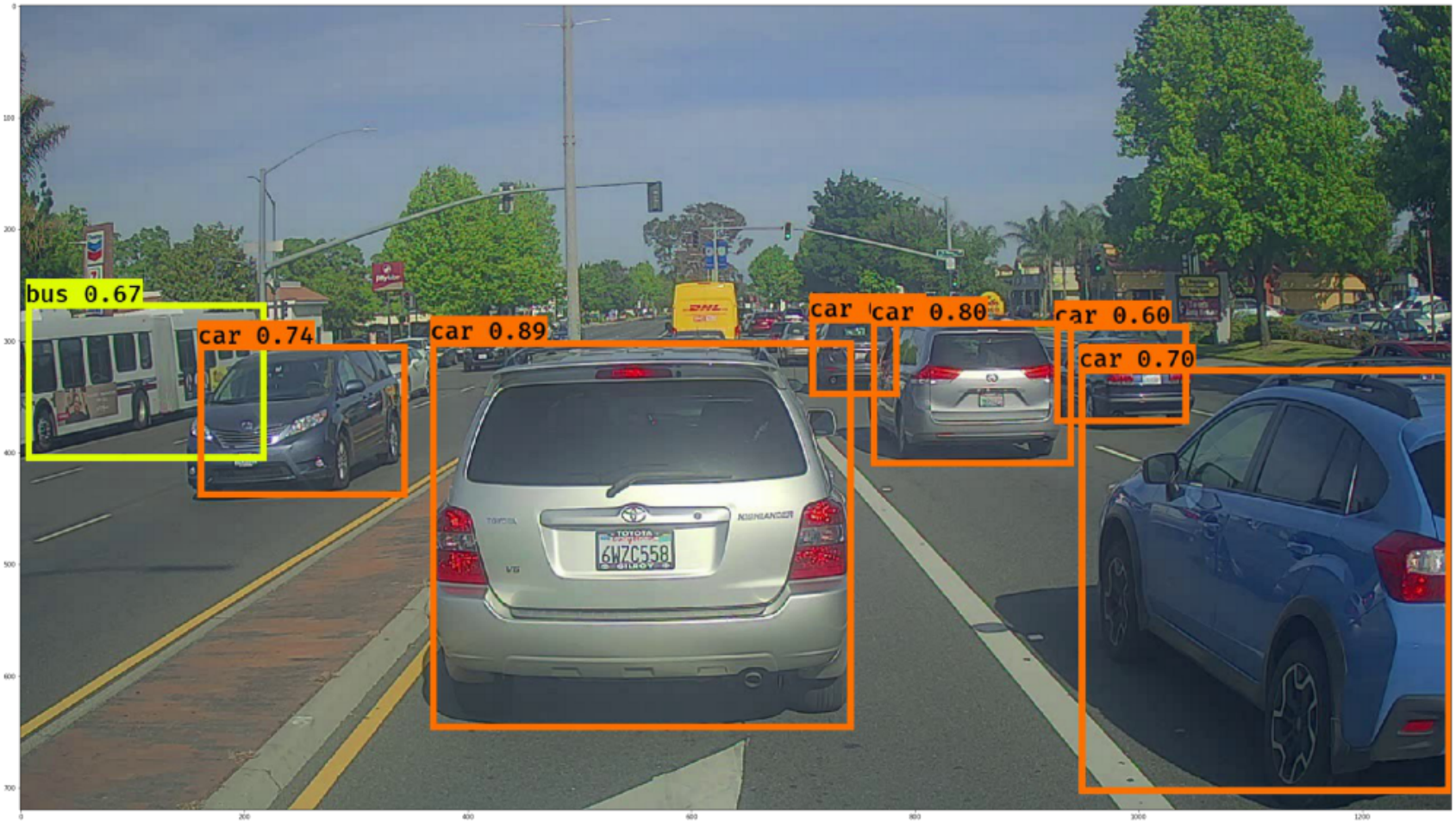
- Input layer: should be a square (no more than 64x64x3 => resize images)
- Input layer: should be divisible by 4 (for downsampling)
- Conv layer : stride (step for kernel movement) is 1 or 2, padding is 0 (so that output size = input size)
- Conv layer : minimum 2 layers
- Pool layer : generally 2x2 kernel
- Dense layer : activation function : 2 classes => sigmoid / 2+ classes => softmax
- Loss function : 2 classes => binary cross entropy / 2 + classes => Mean Squared Error (MSE)
- Gradient descent : use Stochastic Gradient Descent (SGD)
- Use dropout (very efficient against overfitting)
- Limit batch size if you run low on RAM
- Do 50+ epochs (iteration != epoch)
- Flow is : Input -> (Conv -> ReLU -> Pool) * X -> Dense -> Output

Building a face mask detector with custom CNN

Let's create a CNN powered face mask detector (with Keras, high-level framework on top of TensorFlow)

Object detection

Object detection is the Graal of computer vision, it involves both object classification and localization.



Object detection with R-CNN

A first attempt at doing object detection involving deep learning was the R-CNN (2014) algorithm (Regional Convolutional Neural Network).

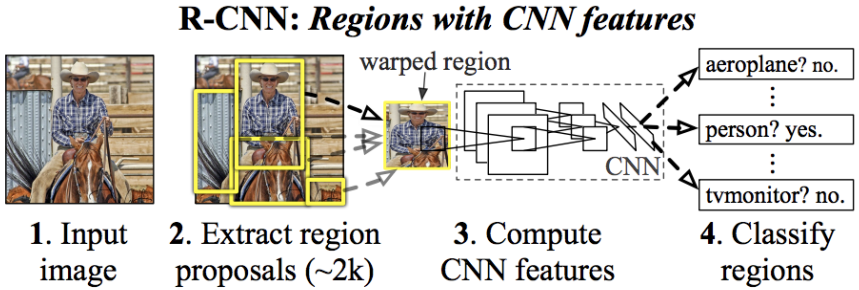
The algorithm is split in two parts :

- Find the objects in the image via Selective Search Algorithm (determine contours in image)
- Inject the content of each box in a CNN to classify what's in the box

It was slow

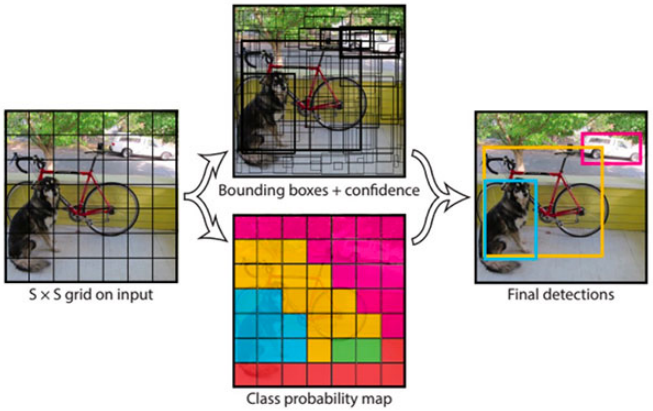
A few slightly better attempts were made with same 2-steps concept

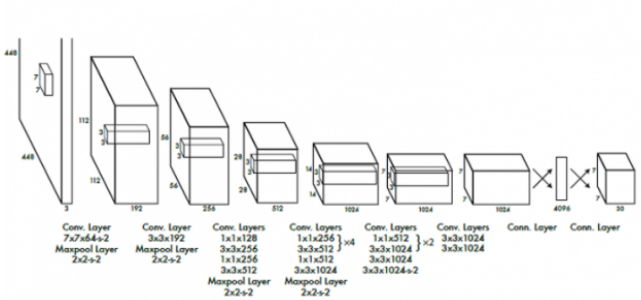
- Fast R-CNN (2015)
- Faster R-CNN (2016)



Object detection with YOLO

Then YOLO cames in (2016) with the idea of doing both the box prediction + classification at once in the same neural network (You Only Look Once).





It has since then been enhanced

- YOLO 2 (2017) : high resolution support (448X448)
- YOLO 3 (2018) : multi-scale training for small objects detection

Object detection with YOLO

Let's use YOLO to do some object detection

Thank you