Image Classification

Giacomo Boracchi,
DEIB, Politecnico di Milano
October, 21th, 2020

giacomo.boracchi@polimi.it
home.deib.polimi.it/boracchi/

Who I am

Giacomo Boracchi (giacomo.boracchi@polimi.it)
Mathematician (Università Statale degli Studi di Milano 2004),
PhD in Information Technology (DEIB, Politecnico di Milano 2008)
Associate Professor since 2019 at DEIB, Polimi (Computer Science)

My research interests are mathematical and statistical methods for:

- Image analysis and processing
- Machine Learning and in particular unsupervised learning, change and anomaly detection

... and the two combined

Materials

https://boracchi.faculty.polimi.it/teaching/AN2DL.htm

What is image classification (and computer vision) about

Computer Vision

An interdisciplinary scientific field that deals with **how computers** can be made to **gain high-level understanding from digital images or videos**

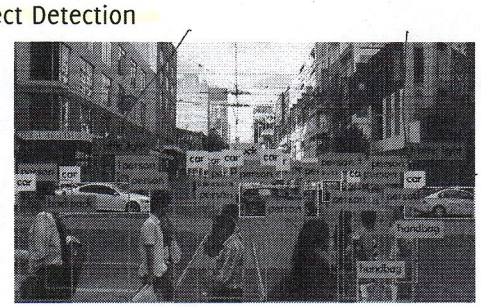
... which has grown incredibly fast in the last years

Computer Vision

An interdisciplinary scientific field that deals with **how computers** can be made to **gain high-level understanding from digital images or videos**

... which has grown incredibly fast in the last years

Object Detection

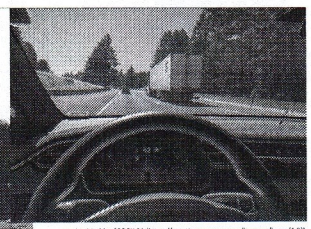


Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

Pose Estimation (skeleton detection)

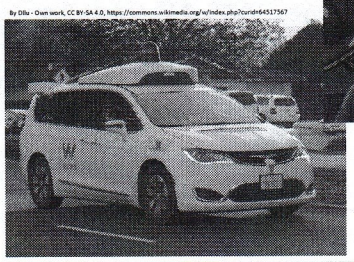
Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7291-7299).

Autonomous Driving

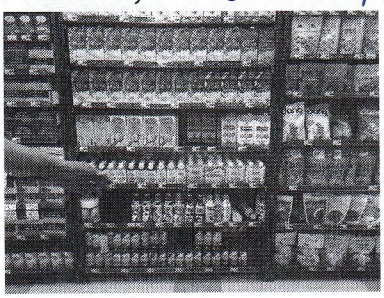


Ian Maddox [CC BY-SA (https://creativecommons.org/licenses/by-sa/4.0)]

By Dilu - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=64517567

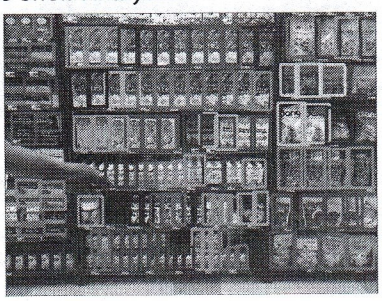


Automatic Shelf Analysis ("how many boxes of – ?")



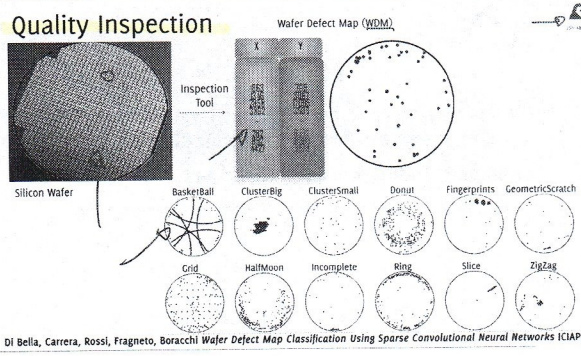
http:focal.systems

Automatic Shelf Analysis



http:focal.systems

Quality Inspection



Di Bella, Carrera, Rossi, Fragneto, Boracchi *Wafer Defect Map Classification Using Sparse Convolutional Neural Networks* ICIAP09

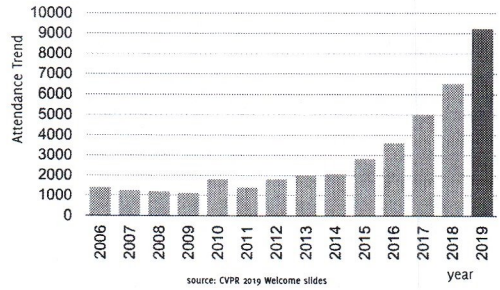
Image Captioning

"little girl is eating piece of cake."

"black cat is sitting on top of suitcase."

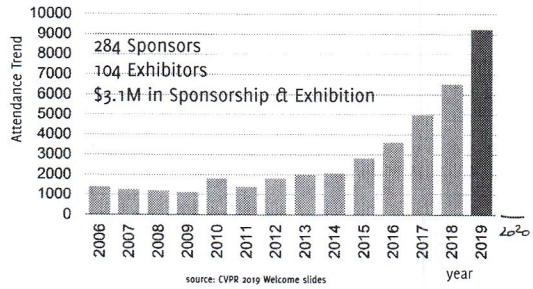
Andrej Karpathy, Li Fei-Fei "Deep Visual-Semantic Alignments for Generating Image Descriptions" CVPR 2015

CVPR



source: CVPR 2019 Welcome slides

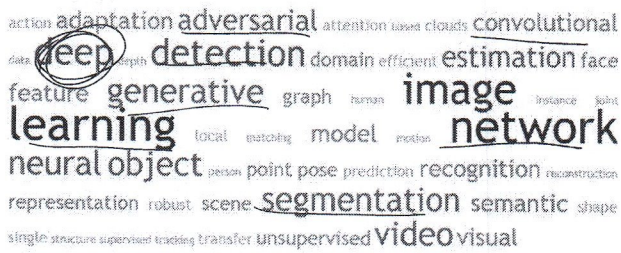
CVPR



source: CVPR 2019 Welcome slides

1,294 papers in CVPR'19 (25.2% acceptance rate)

Acceptance rate is roughly even across topics

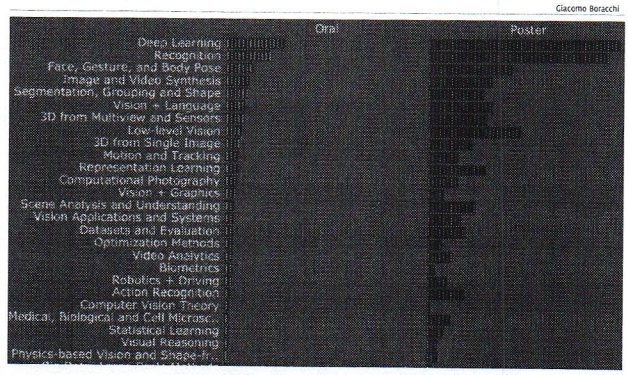


Lately, connection with ML

There has seen a dramatic change in CV:

- Once, most of techniques and algorithms **build upon a mathematical/statistical** description of **images**
- Nowadays, **machine-learning** methods are much more popular

Giacomo Boracchi



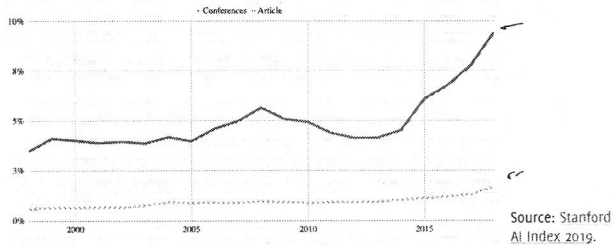
All in all...

If you plan pursuing a research-oriented career:

- If you go towards ML and/or CV, consider these fields are increasingly contaminated
- In any case a strong background in Computer Vision and Pattern Recognition is definitively a plus considering the widespread use of imaging data and the need of automation
- This course is a good way to practice more fundamental theory aspects related to neural networks
- Companies are increasinly interested in Computer Vision and Pattern Recogntion appliction

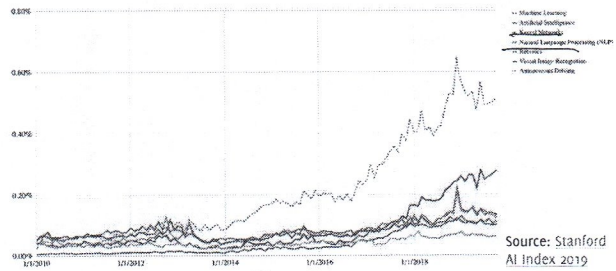
Giacomo Boracchi

AI publications over all the publications



Source: Stanford AI Index 2019.

Percentages of AI posted job positions

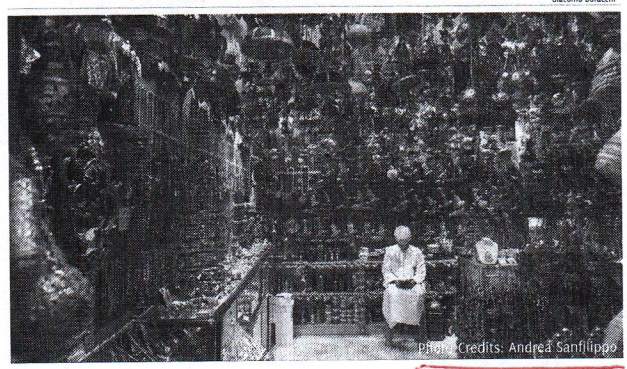


Source: Stanford AI Index 2019

How is an image described?

Images

Giacomo Boracchi

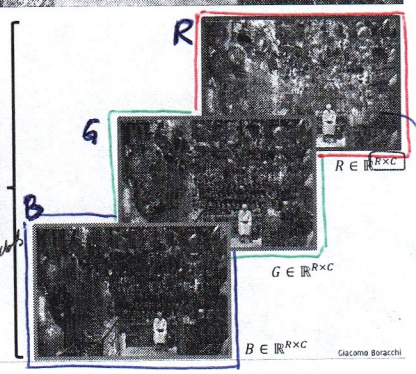


RGB Images

$I \in \mathbb{R}^{R \times C \times 3}$ → # colors channels

rows

cols



Giacomo Boracchi

An image := I is a 3-dim matrix. The 3 dimensions are:

- R = # rows
- C = # columns
- "3" = # color channels

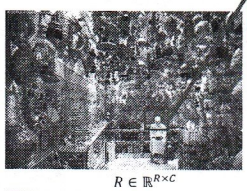
→ Each image visualized on the screen is actually an overlapping of 3 matrices:

1. Red channel
   two-dimensional matrix that contains only the informations about red
2. Green channel
   " about green
3. Blue channel
   " about blue

→ the red image is brighter since the original image is more red

RGB Images

Images are saved by encoding each color information in 8 bits. So images are rescaled and casted in [0,255]



$R \in \mathbb{R}^{R \times C}$

Zoom on the red channel:

each pixel is actually a number $\in [0, 255]$
0 = black, 255 = white

123	122	134	121	132
122	121	125	132	124
119	127	137	119	139

1 Byte

RGB Images

[0, 205, 155] [15, 17, 19] [230,234,233]

almost black

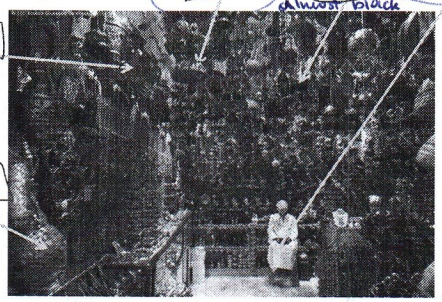
here there is no red at all

almost white

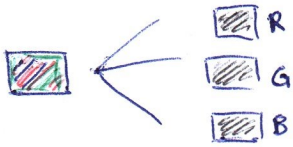
"most" red pixel (the red value is almost max and the other values are almost zero)

[253, 5, 6]

[106,124,138]



In order to build up colors we have to consider the 3 channels:



every pixel of the original image has 3 values: one for the R, one for the G, one for the B. Each of those 3 values are $\in [0, 255]$. Notice that R, G, B are described in gray-scale.

The input of our classifier!

Three dimensional arrays $I \in \mathbb{R}^{R \times C \times 3}$

```
from skimage.io import imread

# Read the image
I = imread('bazar.jpg')

# Extract the color channels of the image
R = I[:, :, 0]
G = I[:, :, 1]
B = I[:, :, 2]
```


Videos

= sequence of frames

Higher dimensional images

Videos are sequences of images (frames)

If a frame is $I \in \mathbb{R}^{R \times C \times 3}$

a video of T frames is $V \in \mathbb{R}^{R \times C \times 3 \times T}$ TEMPORAL DIMENSION

```
print(V.shape)
(144, 180, 3, 30)
```

$144 \times 180 \times 3 \times 30$

space needed for a video of R = 144, C = 180, 3 = number of color channels, 30 = number of frames for 1 second

In this example: $R = 144, C = 180$ thus these 5 color frames contains: 388.800 values in [0,255], thus in principle, 388 KB

Dimension Increases very quickly

Without compression: 1Byte per color per pixel

1 frame in full HD: $R = 1080, C = 1920 \approx 6MB$

1 sec in full HD (24fps) $\approx 150MB$

Fortunately, visual data are very redundant, thus compressible

This has to be taken into account when you design a Machine learning algorithm to be used on images

Higher dimensional images

Multispectral imaging and remote sensing

Giacomo Boracchi

Higher dimensional Images

These images are stacking multiple layers as color-planes

Multi-spectral or Hyper-spectral images:

each band covers a certain wavelength that is meaningful for interpretation soil in areal images

- 0.45-0.52 μm Blue-Green
- 0.52-0.60 μm Green
- 0.63-0.69 μm Red
- 0.76-0.90 μm Near IR
- 1.55-1.75 μm Mid-IR
- 10.40-12.50 μm Thermal IR
- 2.08-2.35 μm Mid-IR

Classification of multispectral images: infer the soil coverage from the vaues in the different spectral bands

In hyperspectral images the number of bands becomes larger and you get a whole energy spectrum in each pixel

Giacomo Boracchi

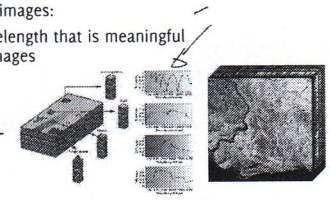
Higher dimensional Images

These images are stacking multiple layers as color-planes

Multi-spectral or Hyper-spectral images:

each band covers a certain wavelength that is meaningful for interpretation soil in areal images

- 0.45-0.52 μm Blue-Green
- 0.52-0.60 μm Green
- 0.63-0.69 μm Red
- 0.76-0.90 μm Near IR
- 1.55-1.75 μm Mid-IR
- 10.40-12.50 μm Thermal IR
- 2.08-2.35 μm Mid-IR



In hyperspectral images the number of bands becomes larger and you get a whole energy spectrum in each pixel

Giacomo Boracchi

Band 1

Giacomo Boracchi

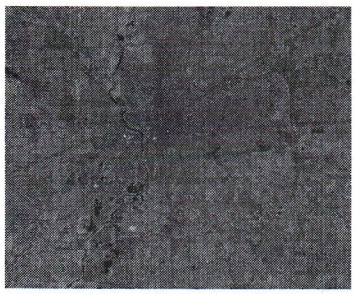
Band 2

Giacomo Boracchi

Band 3

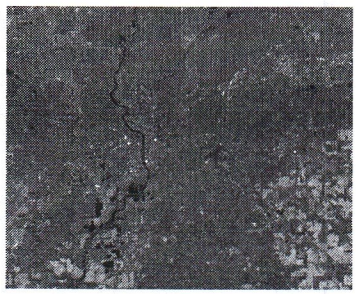
Giacomo Boracchi

Band 4



Giacomo Boracchi

Band 5



Giacomo Boracchi

Band 6

Giacomo Boracchi

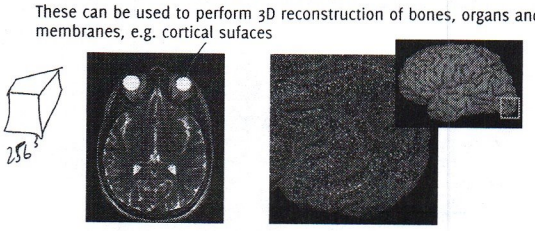
Band 7

Giacomo Boracchi

MRI and TAC

(Structural) MRI provide a 3D stack of grayscale images that are analyzing the body at different depth levels

These can be used to perform 3D reconstruction of bones, organs and membranes, e.g. cortical sufaces



Giacomo Boracchi

Local (Spatial) Transformations: Correlation (and Convolution)

Most important image processing operations for classification problems

Local (Spatial) Transformation

In general, these can be written as

$$G(r,c) = T_U[I(r,c)]$$

Where

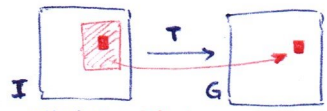
- I is the input image to be transformed
- G is the output
- T_U: $\mathbb{R}^3 \to \mathbb{R}^3$ or T_U: $\mathbb{R}^3 \to \mathbb{R}$ is a function
- U is a neighbourhood, identifies a region of the image that will concur in the output definition

T operates on I "around" U

The output at pixel (r,c) i.e., $T_U[I(r,c)]$ is defined by all pixels $\{I(x,y),\ (x-r, y-c) \in U\}$

Giacomo Boracchi

We have an initial image I and we want to obtain an image G as a transformation of image I. Every pixel in G will be determined not only through the corresponding I-pixel, but also through a close neighborhood of the corresponding I-pixel:



all the neighborhood will determine the new value of the pixel (post transform.)

(This is the main idea of convolutional neural networks).

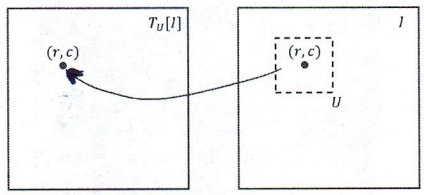
How can we define this neighborhood?

U = neighborhood

$I(r+x, c+y) \quad : \quad (x,y) \in U$

Local (Spatial) Filters

The dashed square represents $\{I(x,y),\ (x-r, y-c) \in U\}$

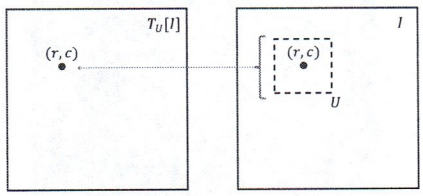


- The location of the output does not change
- The operation is repeated for each pixel
- T can be either linear or nonlinear

Giacomo Boracchi

Local (Spatial) Filters

The dashed square represents $\{I(x,y),\ (x-r, y-c) \in U\}$



- The location of the output does not change
- The operation is repeated for each pixel
- T can be either linear or nonlinear

Giacomo Boracchi

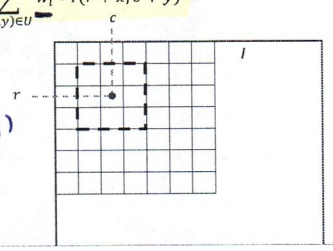
Local Linear Filters

Linear Transformation: Linearity implies that

$$T(I(r,c)) = \sum_{(x,y)\in U} w_i * I(r+x, c+y)$$

Considering some weights $\{w_i\}$

(we consider the weights to be equivalent to pixel we're considering)



If we want the transformation to be linear then we have to linearly combine the pixels of the neighborhood of each pixel.

The set of weights can be arranged in a matrix called **FILTER**:

If we consider a 3x3 U, then

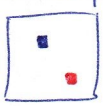
w(-1,-1)	w(-1,0)	w(-1,1)
w(0,-1)	w(0,0)	w(0,1)
w(1,-1)	w(1,0)	w(1,1)

Local Linear Filters

Linear Transformation: Linearity implies that

$$T(I(r,c)) = \sum_{(x,y)\in U} w(x,y) * I(r+x, c+y)$$

Fixed for all the pixels:

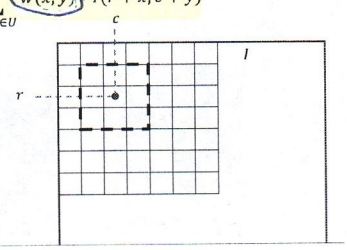


for both the ■ and ■ pixels we'll have that the centered pixel will be weighted w(0,0), the left one to the centered will be weighted w(0,-1) and so on

→ FIXED FILTER

U

w(−1,−1)	w(−1,0)	w(−1,1)
w(0,−1)	w(0,0)	w(0,1)
w(1,−1)	w(1,0)	w(1,1)



We can consider weights as an image, or a filter h

The filter h entirely defines this operation

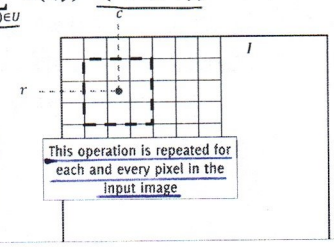
Local Linear Filters

Linear Transformation: Linearity implies that

$$T(I(r,c)) = \sum_{(x,y)\in U} w(x,y) * I(r+x, c+y)$$

U

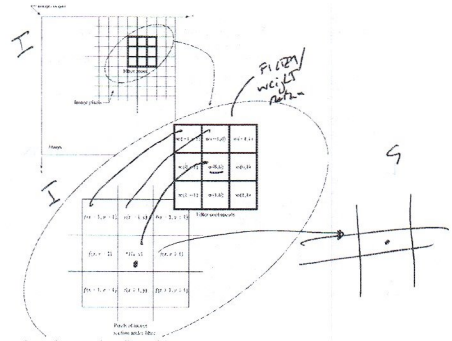
w(−1,−1)	w(−1,0)	w(−1,1)
w(0,−1)	w(0,0)	w(0,1)
w(1,−1)	w(1,0)	w(1,1)



We can consider weights as an image, or a filter h

The filter h entirely defines this operation

This operation is repeated for each and every pixel in the input image



Gonzalez and Woods «Digital Image Processing», Prentice Hall;, 3ª edition

Correlation

= correlation of the image **I** against the weights **w** (set in a filter **h**)
(we'll see neural networks that learn these filters)

The **correlation** among a filter h and an image is defined as

$$(I \otimes h) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} w(u,v) * I(r+u, c+v)$$

where the filter w is of size $(2L+1) \times (2L+1)$

Correlation

The **correlation** among a filter h and an image is defined as

$$(I \otimes h) = \sum_{u=-L}^{L} \sum_{v=-L}^{L} w(u,v) * I(r+u, c+v)$$

where the filter w is of size $(2L+1) \times (2L+1)$

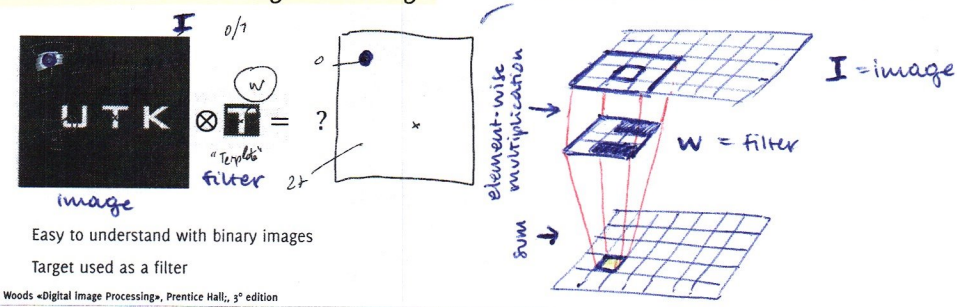
```
from scipy import signal

# correlation
T_corr = signal.correlate(I, h)
```


Correlation for BINARY target matching

how do we know what it will be?

Binary image:
0 - black
1 - white

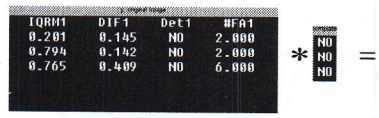


Easy to understand with binary images

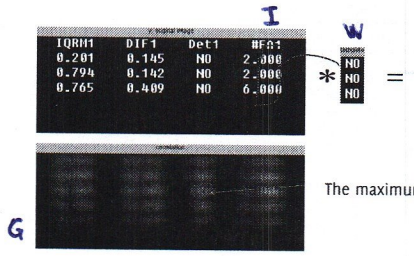
Target used as a filter

Gonzalez and Woods «Digital Image Processing», Prentice Hall;, 3° edition

Another example

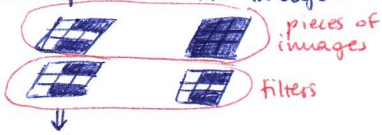


Another example



The maximum is here

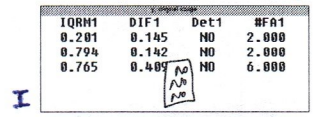
! **Notice**: there is a maximum response when the filter corresponds to the image:



maximum response

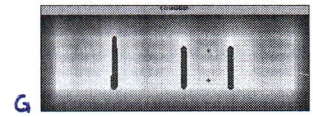
(the closer are, the higher the correlation)

However... if we invert the image (and the filter) everything changes :



IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

I

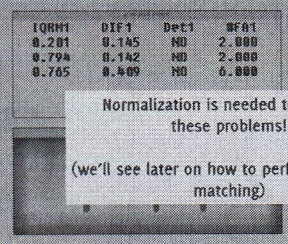


G

$* \begin{bmatrix} NO \\ NO \\ NO \end{bmatrix} =$ W

Each point in a white area is as big as the template achieve the maxium value (togheter with the perfect match)

However...



IQRM1	DIF1	Det1	#FA1
0.201	0.145	NO	2.000
0.794	0.142	NO	2.000
0.765	0.409	NO	6.000

$* \begin{bmatrix} NO \\ NO \\ NO \end{bmatrix} =$

Normalization is needed to prevent these problems!

(we'll see later on how to perform template matching)

a white area is template ... maxium value (togheter with the perfect match)

Setting up the stage..

Image Classification

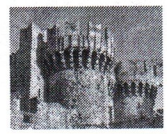
$\Lambda = \{$"wheel", "cars", ..., "castle", "baboon"$\}$

input : image
output : label (categorical)

I

⇨ "wheel"

I



⇨ "castle"

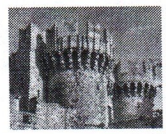
Image Classification

$\Lambda = \{$"wheel", "cars", ..., "castle", "baboon"$\}$

I

⇨ "wheel" 65%, "tyre" 30%..

I



⇨ "castle" 55%, "tower" 43%..

Image Classification, the problem

Assign to an input image $I \in \mathbb{R}^{R \times C \times 3}$

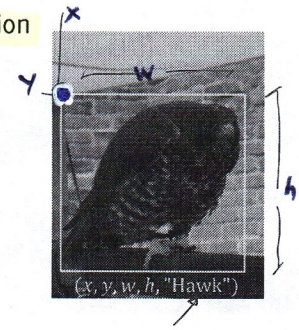
output : • a label l from a fixed set of categories $\Lambda = \{$"wheel", "cars", ..., "castle", "baboon"$\}$

$$I \rightarrow l \in \Lambda$$

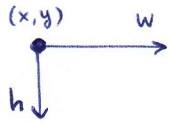
l

Image Classification Example

Localization



together with the label, there are provided also 4 numbers which correspond to the coordinates of the bounding box:



Giacomo Boracchi

Localization, the problem

Assign to an input image $I \in \mathbb{R}^{R \times C \times 3}$:

- a label l from a fixed set of categories
  $\Lambda = \{\text{"wheel"}, \text{"cars"}, \ldots, \text{"castle"}, \text{"baboon"}\}$

output :

- the coordinates (x, y, h, w) of the bounding box enclosing that object

$$I \rightarrow (x, y, h, w, l)$$

Giacomo Boracchi

Object Detection

it provides more than one output: for each object it provides the label and the coordinates of the box

Object Detection, the problem

Assign to an input image $I \in \mathbb{R}^{R \times C \times 3}$:

outputs:

- **multiple** labels $\{l_i\}$ from a fixed set of categories $\Lambda = \{\text{"wheel"}, \text{"cars"}, \ldots, \text{"castle"}, \text{"baboon"}\}$, each corresponding **to an instance of that object**
- the coordinates $\{(x, y, h, w)_i\}$ of the bounding box enclosing **each** object

$$I \rightarrow \{(x, y, h, w, l)_1, \ldots, (x, y, h, w, l)_N\}$$

Giacomo Boracchi

Segmentation

the output is an image which indicates that each pixel has been classified depending on the object it belongs to. The output is a kind of "mask".

Zheng et al. "Conditional Random Fields as Recurrent Neural Networks", ICCV 2015

Image Segmentation, the problem

Assign **to each pixel** of an image $I \in \mathbb{R}^{R \times C \times 3}$:

- a label $\{l_i\}$ from a fixed set of categories $\Lambda = \{$"wheel", "cars", ..., "castle", "baboon"$\}$,

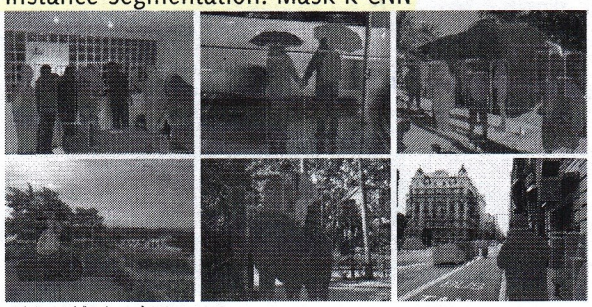
$$I \rightarrow S \in \Lambda^{R \times C}$$

where $S(x, y) \in \Lambda$ denotes the class associated to the pixel (x, y)

S = segmentation mask

Instance Segmentation: Mask R-CNN

– mix: we separate each instance and we provide the bounding box and the label, moreover we provide a mask



He, Kaiming, et al. "Mask r-cnn." *CVPR 2017*

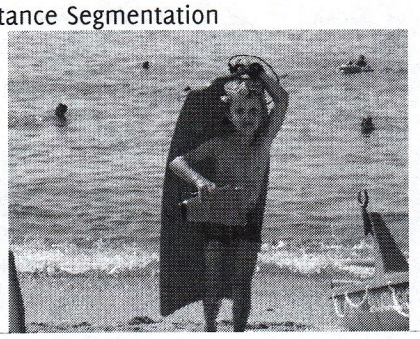
Instance Segmentation, the problem

Assign to an input image I:

- **multiple** labels $\{l_i\}$ from a fixed set of categories $\Lambda = \{$"wheel", "cars", ..., "castle", "baboon"$\}$, each corresponding to **an instance of that object**
- the coordinates $\{(x, y, h, w)_i\}$ of the **bounding box enclosing each object**
- the **set of pixels** S in each bounding box corresponding to that label

$$I \rightarrow \{(x, y, h, w, l, S)_1, \ldots, (x, y, h, w, l, S)_N\}$$

Instance Segmentation



Is Image Classification a Challenging Problem?

Yes.

Yes, it is...

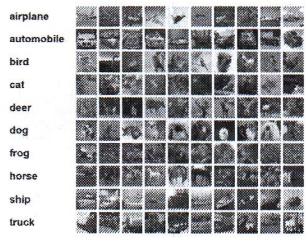
First challenge: dimensionality

Images are very high-dimensional image data

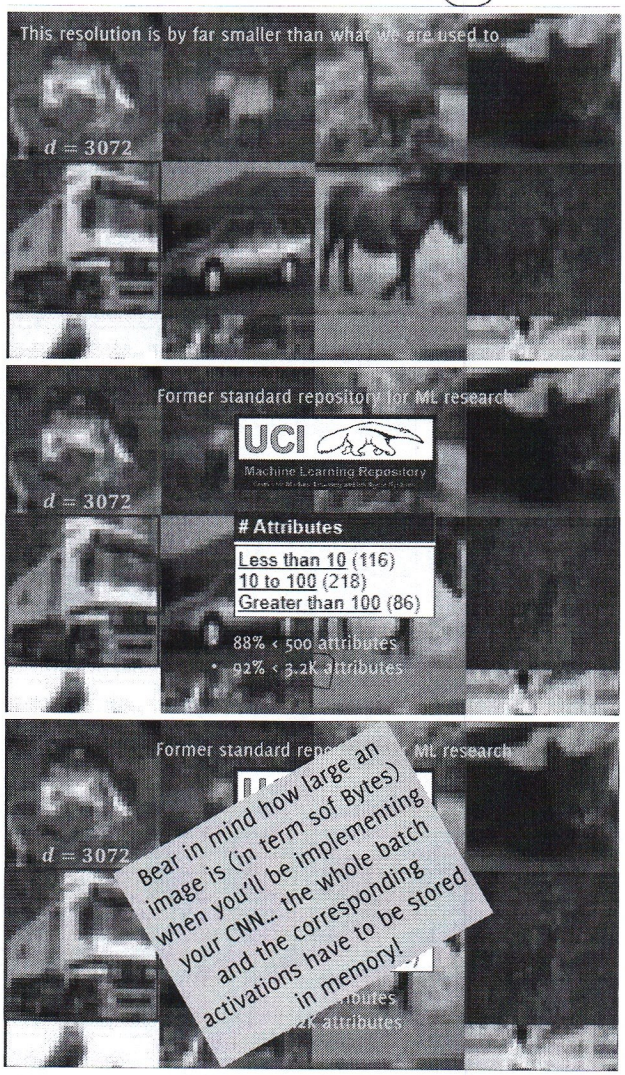
CIFAR-10 dataset

The CIFAR-10 dataset contains 60000 images:
Each image is 32x32 RGB
Images are in 10 classes
6000 images per class

Extremely small images, but high-dimensional:
$d = 32 \times 32 \times 3 = 3072$



Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

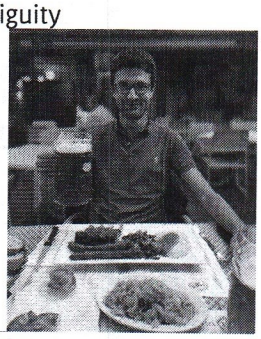


Second challenge: label ambiguity

A label might not uniquely identify the image

Second problem: label ambiguity

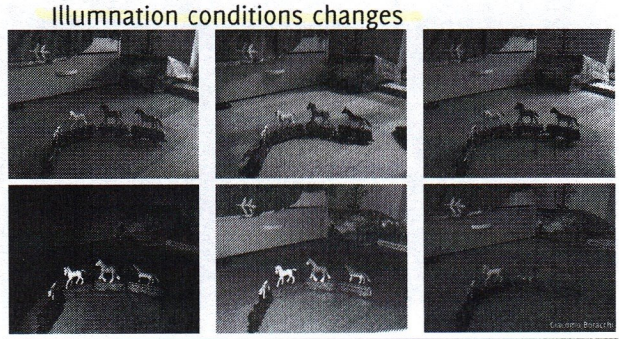
Man?
Beer?
Dinner?
Restaurant?
Sausages?
....



Third challenge: transformations

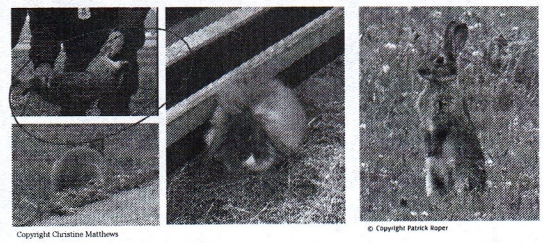
There are many transformations that change the image dramatically, while not its label

Illumnation conditions changes



These 6 images are dramatically different but they should have all the same label (whatever this label is)

Deformations



Copyright Christine Matthews

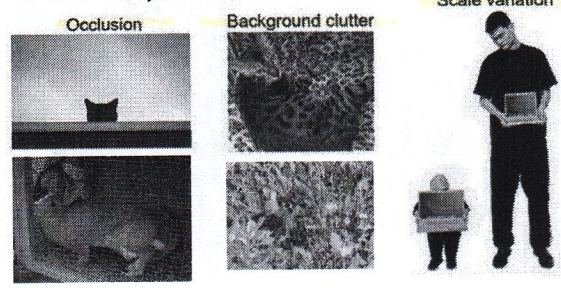
© Copyright Patrick Roper

Giacomo Boracchi

View Point Change

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

... and many othres

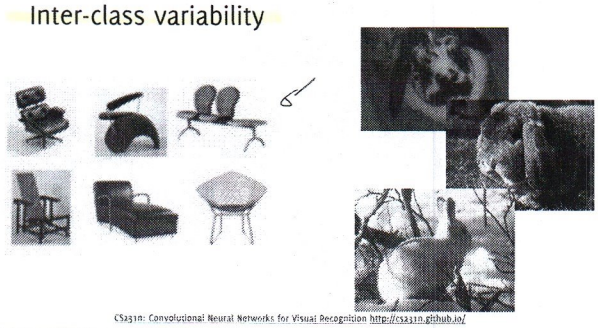


CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Fourth challenge: inter-class variability

Images in the same class might be dramatically different

Inter-class variability



Fifth problem: perceptual similarity

Perceptual similarity in images is not related to pixel-similarity

Nearest Neighborhood Classifiers for Images

Assign to a each test image, **the label of the closest image** in the training set

$$\hat{y}_j = y_{j^*}, \qquad \text{being } j^* = \underset{i=1\ldots N}{\operatorname{argmin}}\, d(\boldsymbol{x}_j, \boldsymbol{x}_i)$$

Distances are typically measured as

$$d(\boldsymbol{x}_j, \boldsymbol{x}_i) = \|\boldsymbol{x}_j - \boldsymbol{x}_i\|_2 = \sqrt{\sum_k \left([\boldsymbol{x}_j]_k - [\boldsymbol{x}_i]_k\right)^2}$$

ℓ^2 norm

if we use this with images we're probably going to fail

Pixel-wise distance among images

test image			
56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image			
10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences			
46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

→ 456 Σ

ℓ^1 norm

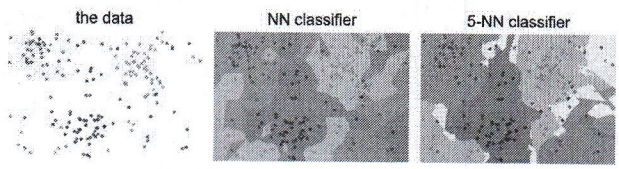
K-Nearest Neighborhood Classifiers for Images

Assign to a each test image, **the most frequent label among the K –closest images in the training set**

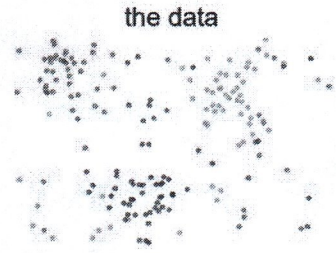
$$\hat{y}_j = y_{j^*}, \qquad \text{being } j^* \text{ the mode of } \mathcal{U}_K(\boldsymbol{x}_j)$$

where $\mathcal{U}_K(\boldsymbol{x}_j)$ contains the K closest training images to $\boldsymbol{x}_j$

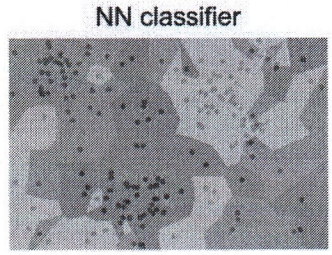
Choosing the right parameter K and the distance measure is often an issue



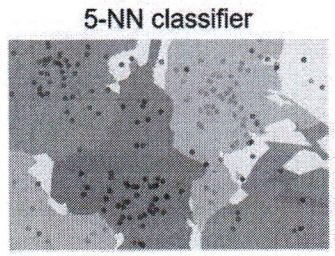
Nearest Neighborhood Classifiers for Images



Nearest Neighborhood Classifiers for Images



Nearest Neighborhood Classifiers for Images



Nearest Neighborhood Classifiers for Images

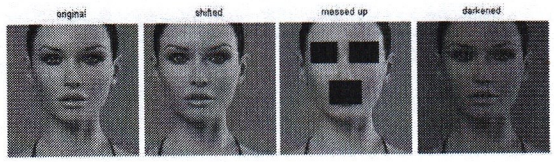
Pros:

- Easy to understand and implement
- It takes no training time

Cons:

- Computationally demanding at test time (in particular when TR is large and d is also large)
- Large training sets have to be stored in memory
- Rarely practical on images: distances on high-dimensional objects are difficult to interpret

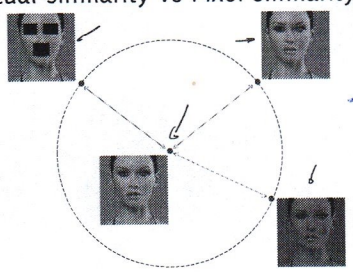
Perceptual Similarity vs Pixel Similarity



The three images have the same pixel-wise distance from the original one...

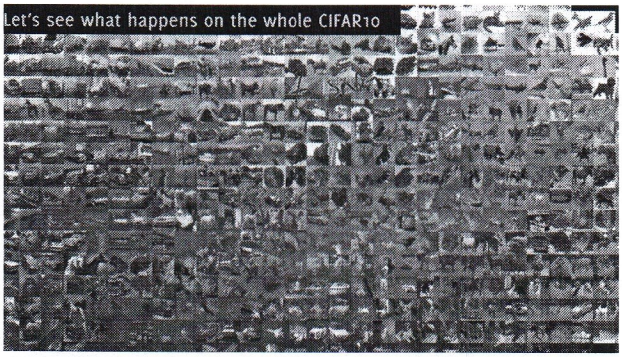
...but perceptually they are very different

Perceptual Similarity vs Pixel Similarity



these 3 pictures are at the same distance from the centered one even though they're pretty much different among each other

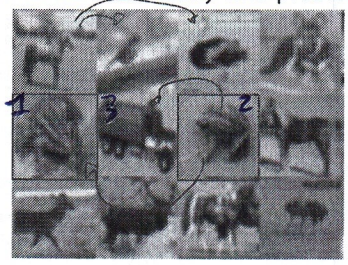
$\Longrightarrow$ we shouldn't use the standard distances ($\ell^1, \ell^2, \ldots$)



t-SNE: places "closer" (w.r.t. some given distance) nearby

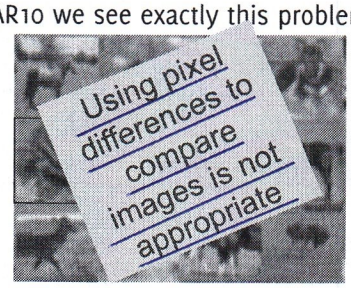
However:

On CIFAR10 we see exactly this problem



the distance used is important: here the image of a horse is close to the image of a bird or of a "strozzo".
Or also: the frog 2 is closer to the truck (3) more than to the other frog (1).

On CIFAR10 we see exactly this problem



Linear Classifier

the basic building block for deep architectures

Linear Classifiers for Images

A classifier can be seen as a function that maps an image $\boldsymbol{x}$ to a confidence scores for each of the L class:

$$\mathcal{K}\colon \mathbb{R}^d \to \mathbb{R}^L$$

where $\mathcal{K}(\boldsymbol{x})$ is a L −dimensional vector and the i −th component

$$s_i = [\mathcal{K}(\boldsymbol{x})]_i$$

contains a score of how likely $\boldsymbol{x}$ belongs to class $\boldsymbol{i}$.

Intuitively, **a good classifier associates to the correct class a score that is larger** than scores associated to incorrect classes.

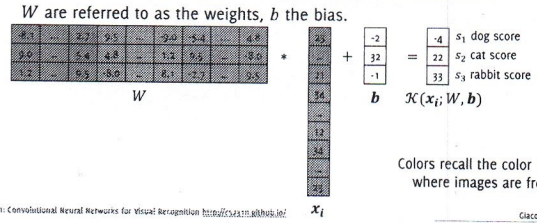
Linear Classifiers for Images

In linear classification $\mathcal{K}$ is a linear function:

$$\mathcal{K}(\boldsymbol{x}) = W\boldsymbol{x} + b$$

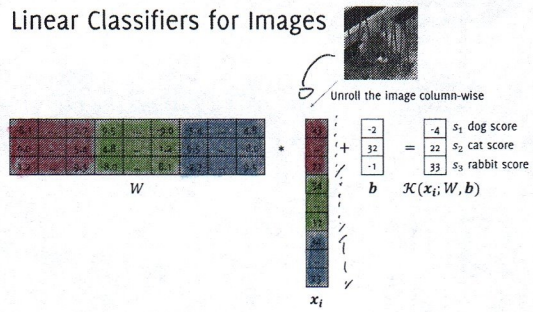
where $W \in \mathbb{R}^{L\times d}$, $b \in \mathbb{R}^L$ are the parameters of the classifier $\mathcal{K}$.

W are referred to as the weights, b the bias.



Colors recall the color plane where images are from

Linear Classifiers for Images



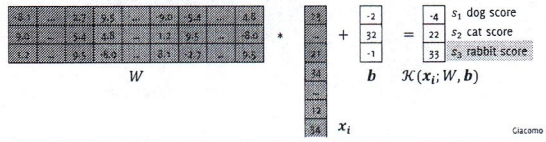
Linear Classifiers for Images

The classifier assign to an input image the class corresponding to the largest score

$$\hat{y}_j = \underset{i=1,..,L}{\operatorname{argmax}} \, [s_j]_i$$

being $[s_j]_i$ the i –th component of the vector

$$\mathcal{K}(x_j) = W x_j + b$$



Linear Classifiers for Images

The score of a class is the weighted sum of all the image pixels.
Weights are actually the classifier parameters.

Weights indicate which are the most important pixels / colors

Training a Classifier

Given a training set TR and a loss function, define the parameters that minimize the loss function over the whole TR

In case of linear classifier

$$[W, b] = \underset{W \in \mathbb{R}^{L \times d},\ b \in \mathbb{R}^L}{\operatorname{argmin}} \sum_{(x_i, y_i) \in TR} \mathcal{L}(x, y_i)$$

Solving this minimization problem provides the weights of our classifier

Loss Function

Loss function: a function $\mathcal{L}$ that measures our unhappiness with the score assigned to training images

The loss $\mathcal{L}$ will be high on a training image that is not correctly classifier, low otherwise.

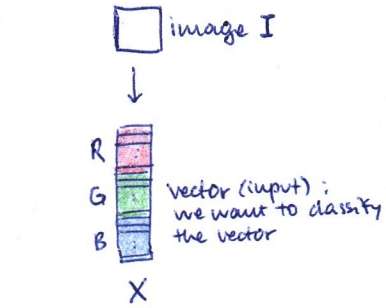
Loss Function Minimization

Loss function can be minimized by specific algorithms (see Matteo' classes)

The loss function has to be typically regularized to achieve a unique solution satisfying some desired property

$$[W, b] = \underset{W \in \mathbb{R}^{L \times d},\ b \in \mathbb{R}^L}{\operatorname{argmin}} \sum_{(x_i, y_i) \in TR} \mathcal{L}(x, y_i) + \lambda \mathcal{R}(W, b)$$

being $\lambda > 0$ a parameter balancing the two terms

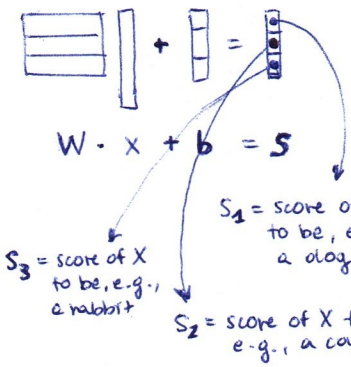


simplest classifier:

$X \mapsto S = WX + b$

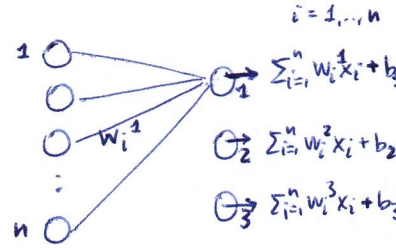
score = $W \cdot X$ + bias

W = matrix (because we want the output to be multiple numbers):



⟹ If we want a linear class. which provides a division of the region in 3 classes we need a matrix W ($3 \times n$, where $n = \#R + \#B + \#G$)

How can we do it with ANN? The input should be a layer of n neurons (one per element of the image). How many outputs? 3, because each one gives the probability of belonging to one class.



(Note: since we're not using the softmax, these are scores, not probabilities)

⇓

We obtain:

$S_1 = W_1 \cdot X + b_1$

$S_2 = W_2 \cdot X + b_2$

$S_3 = W_3 \cdot X + b_3$

part 1

Once Trained

The classifier is expected to provide to the correct class a score that is larger than that assigned to the incorrect classes.

The training data is used to learn the parameters $W, \boldsymbol{b}$

Once the training is completed, it is possible to discard the whole training set and only keep the learned parameters.

Giacomo Boracchi

Geometric Interpretation of a Linear Classifier

In Matlab notation:

$W(i,:)$ is a d −dimensional vector containing the weights of the score function for the i −th class

Computing the score function for the i −th class corresponds to computing the inner product

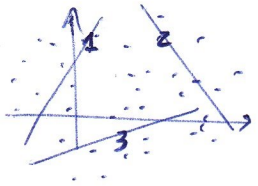
$$W(i,:) * \boldsymbol{x}$$

Thus, linear classifiers identify an hyperplane in a d −dimensional

(in Python $*$ is the element-wise product, here I mean the inner product of vectors):

We get: $\begin{cases} S_1 = W_1 x + b_1 \\ S_2 = W_2 x + b_2 \\ S_3 = W_3 x + b_3 \end{cases}$

but if we take $W_i x + b_i = 0$ we obtain an hyperplane: for each class we're defining an hyperplane which separates one one side something that belongs to that class, on the other side all the other things



Giacomo Boracchi

Geometric Interpretation

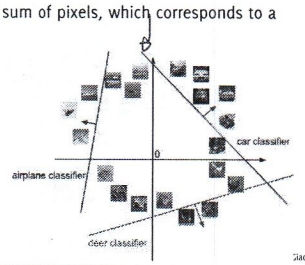
Interpret each image as a point in $\mathbb{R}^d$.

Each classifier is a weighted sum of pixels, which corresponds to a linear function in $\mathbb{R}^d$

In $\mathbb{R}^2$ these would be
$f([x_1, x_2]) = w_1 x_1 + w_2 x_2 + b$

Then, points $[x_1, x_2]$ yielding
$$f([x_1, x_2]) = 0$$
would be lines.

Thus, in $\mathbb{R}^2$ the region that separates positive from negative scores for each class is a line. This region becomes an hyperplane in $\mathbb{R}^d$.



Giacomo Boracchi

Template Matching Interpretation

Using Matlab notation:

- $W(i,:)$ is a d −dimensional vector containing the weights of the score function for the i −th class
- Computing the score function for the i −th class corresponds to computing the inner product
$$W(i,:) * \boldsymbol{x}$$

Then, $\boldsymbol{W(i,:)}$ **can be seen as a template used in matching (the output of correlation in the central pixel)**

The template $W(i,:)$ is learned to match at best images belonging to the i −th class

Let's have a look at these templates

When we're done training we have:

$$\begin{array}{c} 3 \\ \\ \end{array} \begin{array}{c} \boxed{w_1^1 \quad \cdots \quad w_n^1} \\ \\ \\ n \end{array} := W$$

every row is of the dimensions of the input image:

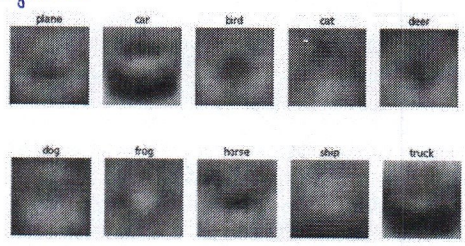
W_1 is $1 \times n$ $\Rightarrow$ we can see this W^1 as an image itself

$\Rightarrow$ we construct 3 images from W_1, W_2, W_3 (:= "TEMPLATES")

Giacomo Boracchi

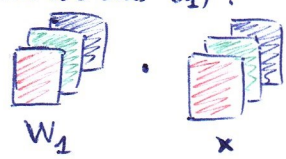
Templates Learned on the CIFAR-10 dataset

weights:



CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

$\Rightarrow$ When we perform:
$S_1 = W_1 \cdot x + b_1$
it's like perform the correlation between the image W_1 and the test image x (and then add the bias b_1):

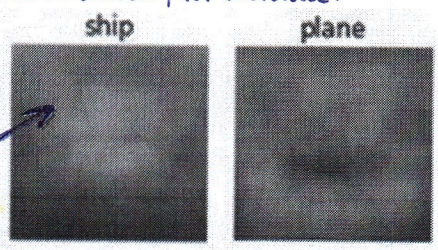


it's like the element-wise product and then we sum them all.

part 2

Templates Learned on the CIFAR-10 dataset

more in details, for instance:

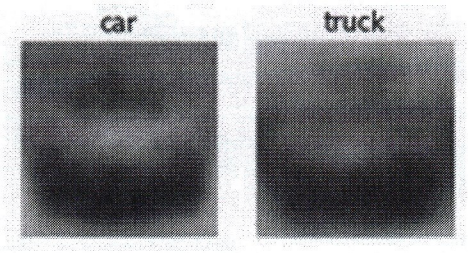


! These are the weights corresponding to the class "ship" and "plane" resembled as images. They're not like this by chance. If we see the color, they're blue! We can interpret it as: the high score in ship is given to those images that looks like this template.

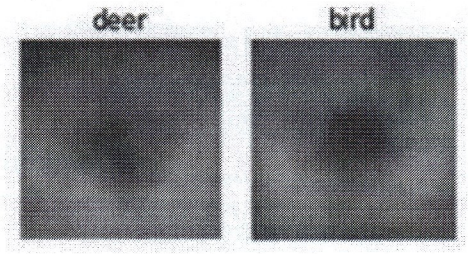
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

$\Rightarrow$ what the network is learning are templates to match the inputs

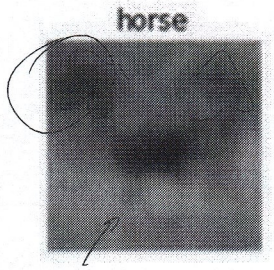
Templates Learned on the CIFAR-10 dataset



Templates Learned on the CIFAR-10 dataset



Templates Learned on the CIFAR-10 dataset



Linear Classifier as a Template Matching

What has the classifier learned?

- That the background of bird and frog is green, (plane and boat is blue)
- Cars are typically red
- Horses have two heads! ☺

The model was definitively too simple / data were not enough for achieving higher performance and better templates

However:

- Linear Classifiers are among the most important layer of NN
- Such a simple model can be interpreted (with more sophisticated models you typically can't)

the templates are optimized to provide the highest score to the right class and low scores to the wrong classes
(it all can be interpreted as VISUAL MATCHING)

can this distinguish, e.g., different races of dogs? No.

Linear Classifier as a Template Matching

What has the classifier learned?

- That the background of bird and frog is ... ne and boat is blue)
- Cars are typically red
- Horses have two heads! ☺

The model was definitivel ... / data were not enough for achieving higher perfor ... better templates

However:

- Linear Classifiers a ... ng the most important layer of NN
- Such a simple model can be interpreted (with more sophisticated models you typically can't)

There should be a better way for handling images

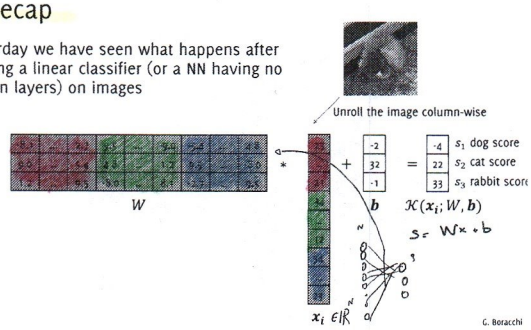
Convolutional Neural Networks

Giacomo Boracchi,
DEIB, Politecnico di Milano
October, 22th, 2020

giacomo.boracchi@polimi.it
https://boracchi.faculty.polimi.it/

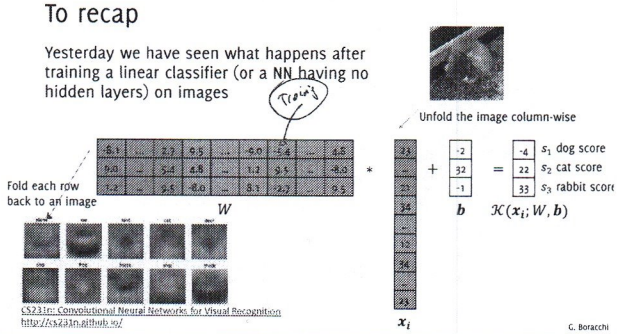
To recap

Yesterday we have seen what happens after training a linear classifier (or a NN having no hidden layers) on images

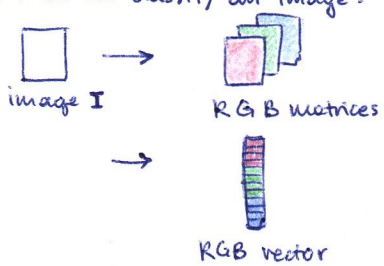


To recap

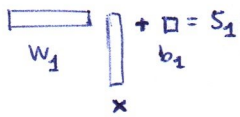
Yesterday we have seen what happens after training a linear classifier (or a NN having no hidden layers) on images



Once we have the weights, how do we classify an image?



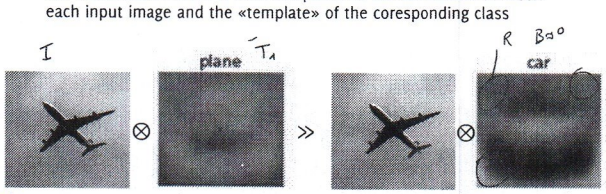
→ we multiply every row of the weights matrix with the RGB vector ($:= x$) and we get the scores:



$[..] \Rightarrow \{S_1, S_2, S_3\}$

The Class Score

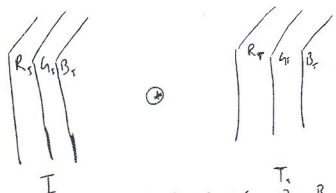
The classification score is then computed as the correlation between each input image and the «template» of the coresponding class



We do the correlation between the image I (x) and the template (in this case T_1) ⟹ $\sum_{(x,y)\in T_1} T_1(x,y) * I(x,y)$ ← "component wise correlation"

Correlation between two RGB images

The image and the filter have the same size



$(R_I * R_T)(0,0) +$
$(G_I * G_T)(0,0) +$
$(R_I * G_T)(0,0)$

$W_1 \cdot x$

$R_I \otimes R_T + G_I \otimes G_T + B_I \otimes B_T =$
$\sum_{x,y} R_I(x,y)\, R_T(x,y) + \dots$

We saw the correlation with a gray scale image. Here we're doing correlation with colored images. Actually, to procede, we do correlation for every level: R with R, G with G, B with B:

$R_I \otimes R_T + G_I \otimes G_T + B_I \otimes B_T$
(CHANNEL-WISE CORRESPONDENCE)

The Feature Extraction Perspective

The Feature Extraction Perspective

Images can not be directly fed to a classifier

We need some intermediate step to:

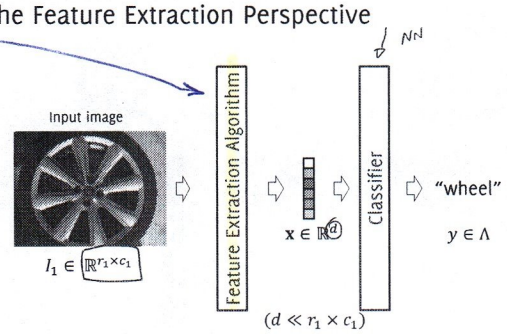
- Extract meaningful information (to our understanding)
- Reduce data-dimension

We need to extract features:

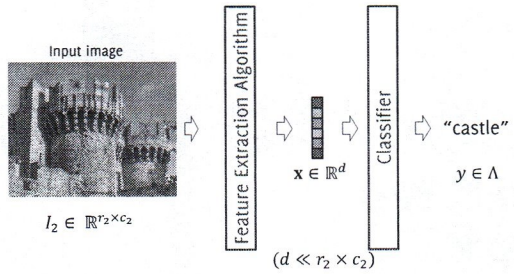
- The better our features, the better the classifier

The Feature Extraction Perspective

"expert driven"
there is someone who
is extracting the features
from the image
(the features that this someone
consider important)



The Feature Extraction Perspective



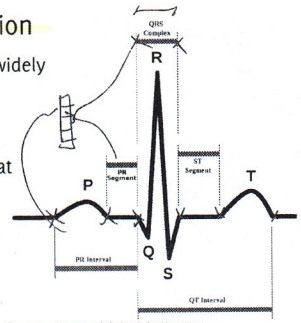
Hand-Crafted Features

Example: ECG classification

Heartbeats morphology has been widely investigated

Doctors know which patterns are meaningful for classifying each beat

Features are extracted from landmarks indicated by doctors:
e.g. QT distance, RR distance...

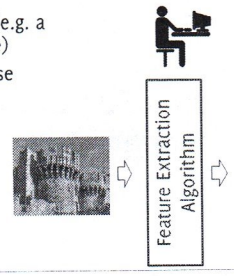


Created by Agateller (Anthony Atkielski), Public Domain, https://commons.wikimedia.org/w/index.php?curid=1560893

Hand Crafted Featues

Engineers:

- know what's meaningful in an image (e.g. a specific color/shape, the area, the size)
- can implement algorithms to map these information in a feature vector



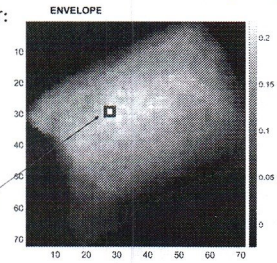
An Illustrative Example: Parcel Classification

analysis of the things going sulla striscia dell'oleodotto

Images acquired from a RGB-D sensor:

- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
  - ENVELOPE
  - PARCEL
  - DOUBLE

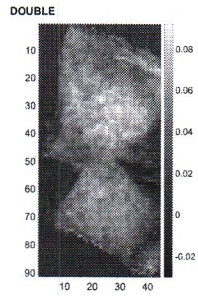
Envelop height at that pixel



An Illustrative Example: Parcel Classification

Images acquired from a RGB-D sensor:

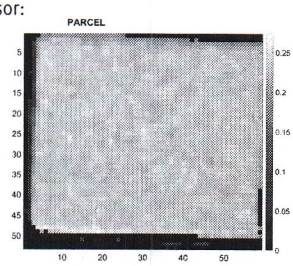
- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
  - ENVELOPE
  - PARCEL
  - DOUBLE



An Illustrative Example: Parcel Classification

Images acquired from a RGB-D sensor:

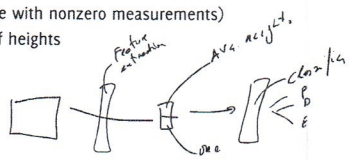
- No color information provided
- A few pixels report depth measurements
- Images of 3 classes
  - ENVELOPE
  - PARCEL
  - DOUBLE



Example of Hand-Crafted Features

Example of features:

- Average height
- Area (coverage with nonzero measurements)
- Distribution of heights
- Perimeter
- Diagonals



we associate to each image a vector of features (feature selection) and then we procede with the fitting of a classifier (classes: E, P, D)

As features we extract:

- average height
- area

→ every image can be represented as a point in a 2-dimensional plane

The Training Set

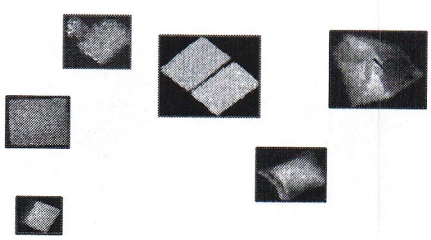
The training set is a set of annotated examples

$$TR = \{(I, l)_i, i = 1, \dots, N\}$$

Each couple $(I, l)_i$ corresponds to:

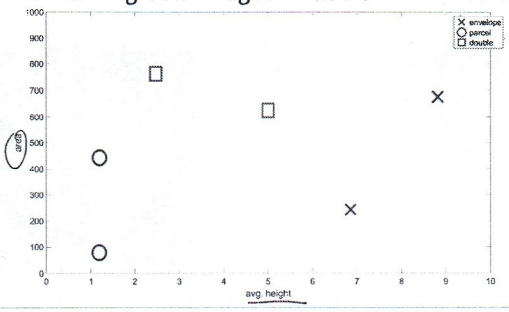
- an image I_i
- the corresponding label l_i

The Training Set: images + labels

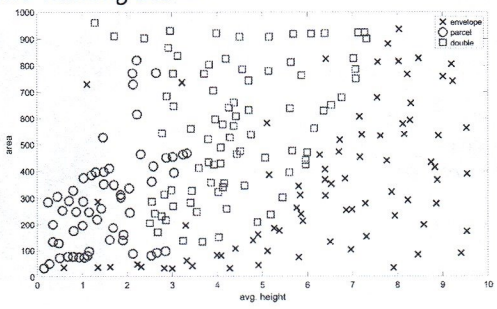


The Training Set: images + labels

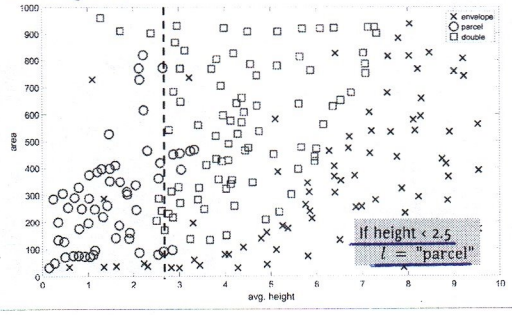
× envelope
○ parcel
□ double



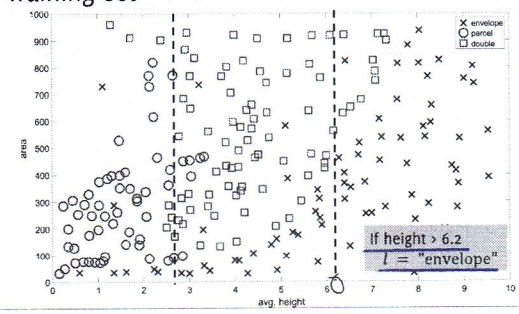
The Training Set



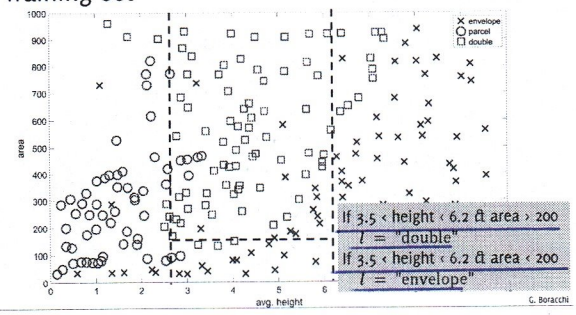
Training Set



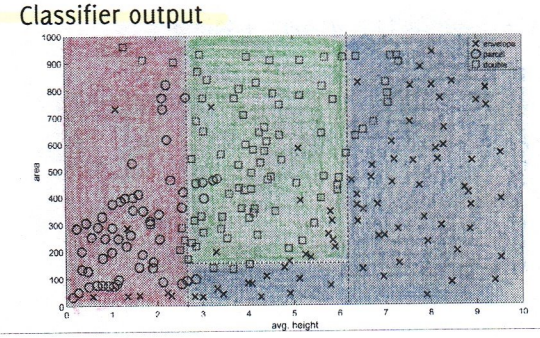
Training Set



Training Set

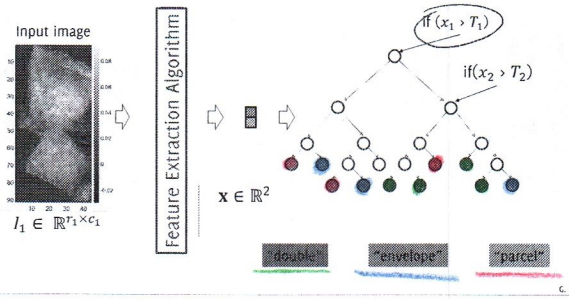


Classifier output



→ this corresponds to a classification tree (next slide)

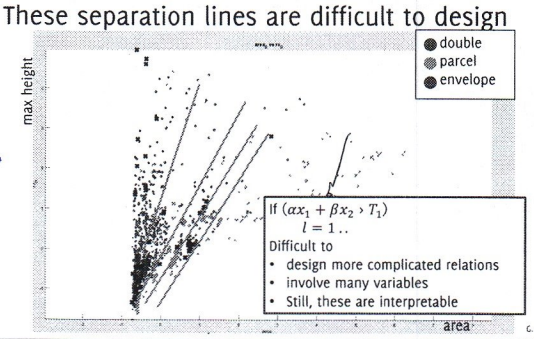
A tree classifying image features



G. Boracchi

These separation lines are difficult to design

another pair of features →



G. Boracchi

Limitations of Rule Based Classifier

It is difficult to grasp what are meaningful dependencies over multiple variables (it is also impossible to visualize these)

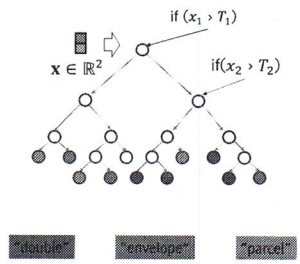
Let's resort to a **data-driven model** for the only task of separating feature vectors in different classes.

How can a classifier achieve better performance?

G. Boracchi

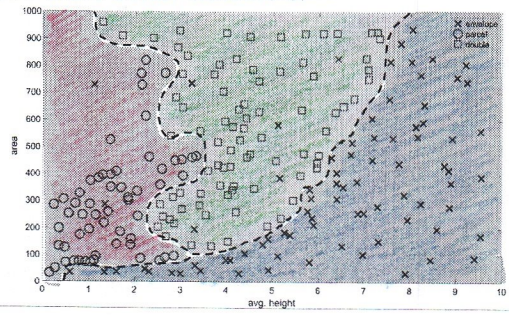
A tree classifying image features

The classifier has a few patameters:
- The **splitting criteria**
- The **splitting thresholds** T_i



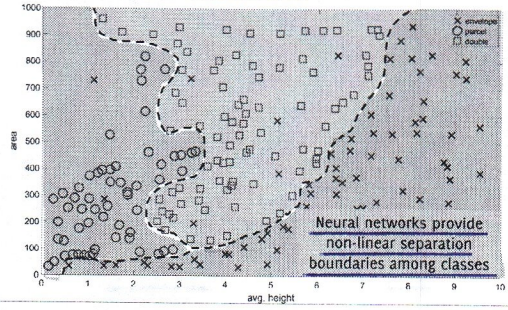
G. Boracchi

Are there better classifiers?



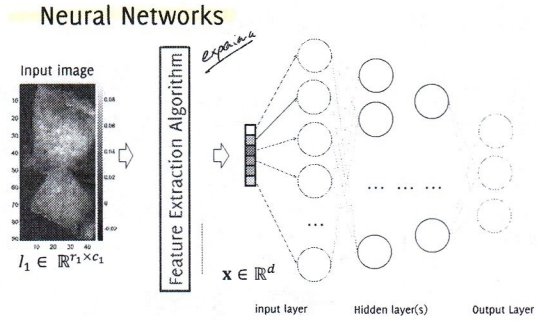
G. Boracchi

Are there better classifiers?



G. Boracchi

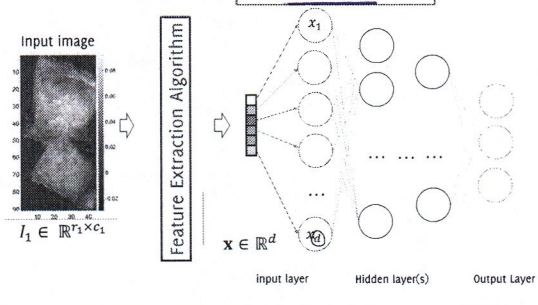
Neural Networks



We can take the features extracted by the experts and use them as inputs for a NN (to train them)

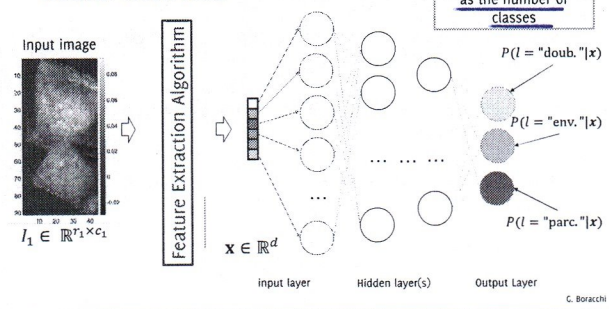
Neural Networks

Input layer: Same size of the feature vector



Neural Networks

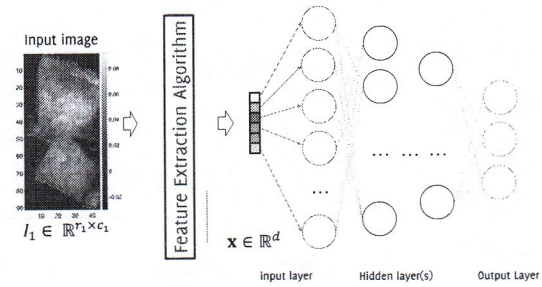
Output layer: Same size as the number of classes



each value corresponds to the probability that the sample belongs to the given class

Neural Networks

Hidden layers: arbitrary size



Neural Networks

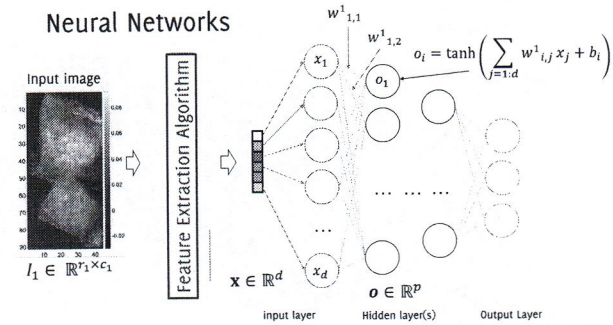
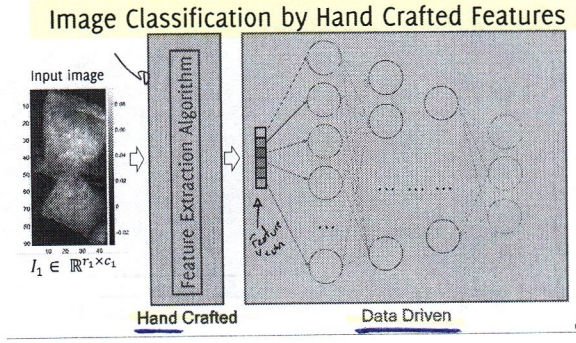


Image Classification by Hand Crafted Features

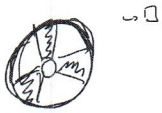


Hand Crafted Featues, pros:

- **Exploit a priori** / expert **information**
- Features are **interpretable** (you might understand why they are not working)
- You can **adjust features** to improve your performance
- **Limited amount of training data** needed
- You can give more relevance to some features

Hand Crafted Featues, cons:

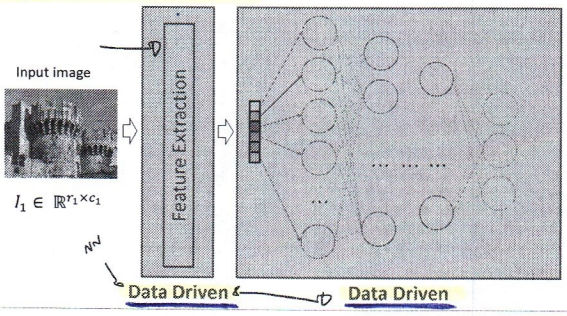
- Requires a lot of **design/programming efforts**
- **Not viable** in many **visual recognition** tasks (e.g. on natural images) which are easily performed by humans
- **Risk of overfitting** the training set used in the design
- **Not very general and "portable"**



Data-Driven Features

... the advent of deep learning

Data-Driven Features

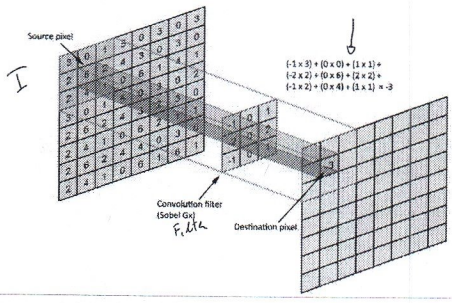


Actually, we want to train the feature extraction and the classifier JOINTLY to optimize both.

Setting up the stage

Convolutional neural networks

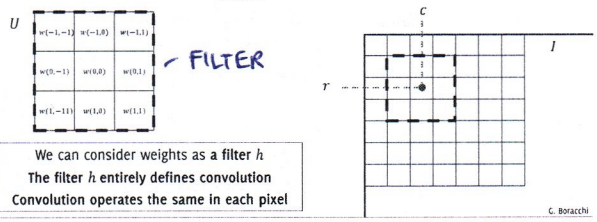
What is a convolution?



2D Correlation

Convolution is a linear transformation. Linearity implies that

$$T[I](r,c) = \sum_{(x,y)\in U} w(x,y) * I(r+x, c+y)$$



We can consider weights as a filter h
The filter h entirely defines convolution
Convolution operates the same in each pixel

G. Boracchi

2D Convolution

Convolution is a linear transformation. Linearity implies that

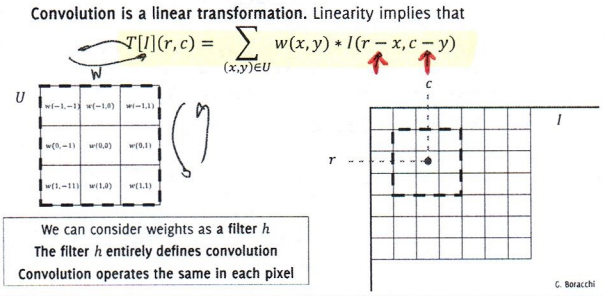
$$T[I](r,c) = \sum_{(x,y)\in U} w(x,y) * I(r-x, c-y)$$

The fact that we use the "−" corresponds to flip the filter (↔, ↕) (and then we can use it as correlation)

⟹ Convolution is the same thing as correlation if we flip the filter.

However it's not important since we learn the weights.

Learning correlation we get the same weights that, flipped, would be the ones learned by convolution.



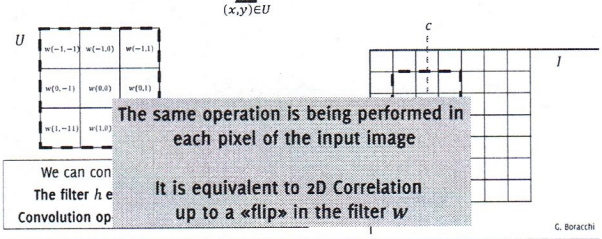
We can consider weights as a filter h
The filter h entirely defines convolution
Convolution operates the same in each pixel

G. Boracchi

2D Convolution

Convolution is a linear transformation. Linearity implies that

$$T[I](r,c) = \sum_{(x,y)\in U} w(x,y) * I(r-x, c-y)$$



The same operation is being performed in each pixel of the input image

It is equivalent to 2D Correlation up to a «flip» in the filter w

We can con
The filter h e
Convolution op

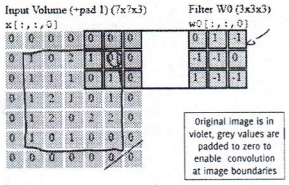
G. Boracchi

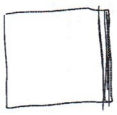
Convolution: Padding

How to define convolution output close to image boundaries?

Padding with zero is the most frequent option, as this does not change the output size. However, no padding or symmetric padding are also viable options

conv2 (I, f, 'same')

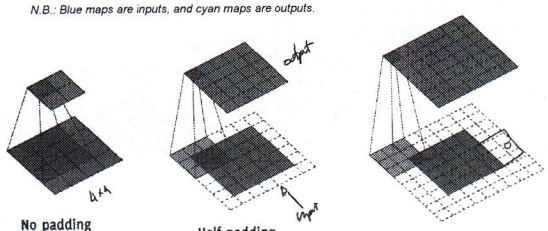




CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Convolution Animation

N.B.: Blue maps are inputs, and cyan maps are outputs.



No padding «valid» | Half padding «same» | full padding «full»

Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

Convolutional Neural Networks

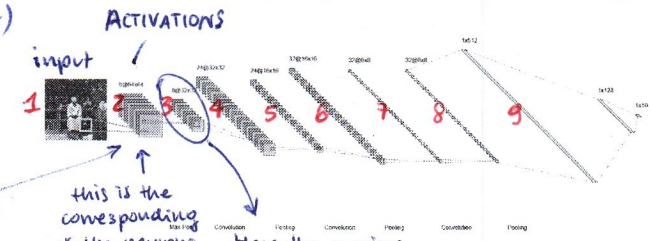
CNNs

G. Boracchi

look also the next slides (più avanti in prende)

The typical architecture of a CNN

what happens to an image:



here the "neurons" are of the same layout of the original image

this is the corresponding of the neurons of a hidden layer

Here the previous neurons get smaller

1: 128 x 128 x 3

2: 128 x 128 x 8
Here the original input becomes an 8 channels image

3: 64 x 64 x 8
Here we reduce the resolution

[..] we reduce again the spatial extent (128 → 64 → 48 → 16 → ..) and we increase the depth (3 → 8 → 24 → ...)

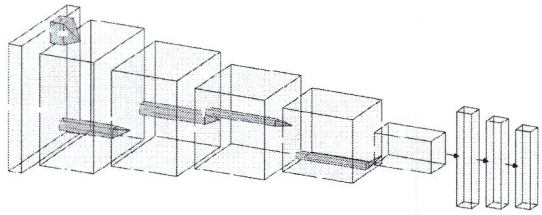
We go on untill we get a 1 x 256 = vector (9)

At the end the image loses in spatial extent and gain in depth.

The procedure is always:

convolution
↓
max-pool
↓
convolution
↓
max-pool
↓
...

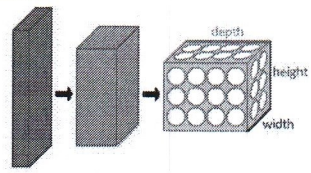
The typical architecture of a CNN



Convolutional Neural Networks (CNN)

CNN are typically made of blocks that include:

- Convolutional layers
- Nonlinearities (activation functions)
- Maxpooling

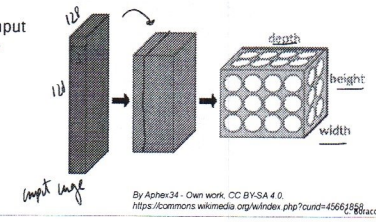


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45662868

G. Boracchi

Convolutional Neural Networks (CNN)

- An image passing through a CNN is transformed in a sequence of volumes.
- As the depth increases, the height and width of the volume decreases
- Each layer takes as input and returns a volume



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45662868

G. Boracchi

What happens to an image (which is a volume) is that the image increases in depth and decreases in height and width

Convolutional Layers

G. Boracchi

Convolutional Layers

identified by a set of weights (a filter)

Convolutional layers "mix" all the input components

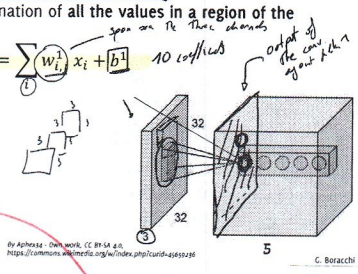
The output is a linear combination of **all the values in a region of the input**

$$y = \sum_i w_i^1 x_i + b^1$$

span on the three channels

10 coefficients

output of the conv. layer with 1 ch

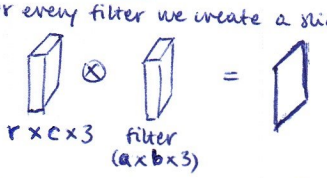


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45662868

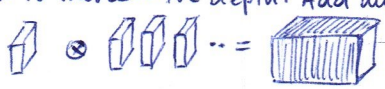
G. Boracchi

How do we increase the depth of an image?

For every filter we create a slice;



How to increase the depth? Add more filter:



to every filter it corresponds a slice

in a convolutional layer we can apply more filters

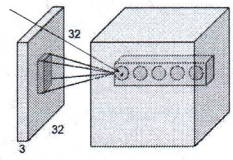
Convolutional Layers

Convolutional layers "mix" all the input components

The output is a linear combination of all the values in a region of the input

$$y = \sum_i \boxed{w^1_{i,j}}\, x_i + \boxed{b^1}$$

The parameters of this layer are called filters.



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

G. Boracchi

Convolutional Layers

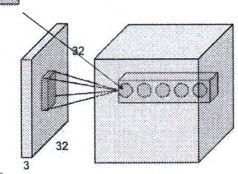
Convolutional layers "mix" all the input components

The output is a linear combination of all the values in a region of the input

$$y = \sum_i \boxed{w^1_{i,j}}\, x_i + \boxed{b^1}$$

The parameters of this layer are called filters.

The same filter is used through the whole spatial extent of the input



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

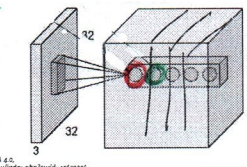
G. Boracchi

Convolutional Layers

Different filters yield different layers in the output

$$\sum_i w^1_{i,j}\, x_i + b^1$$

$$\sum_i w^2_{i,j}\, x_i + b^2$$



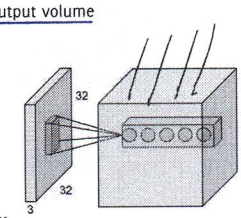
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

G. Boracchi

Convolutional Layers, recap

Convolutional layers "mix" all the input components

- The **output** is also called **volume** or **activation maps**
- Each **filter yields a different slice** of the output volume
- **Each filter has depth equal to the depth of the input volume**



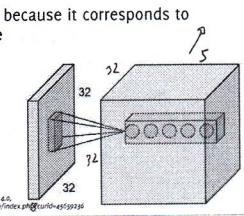
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

G. Boracchi

Convolutional Layers, remarks:

- Convolutional Layers are described by a set of **filters**
- **Filters represent the weights of these linear combination.**
- Filters have **very small spatial extent and large depth extent**,
- The **filter depth** is typically not specified, because it corresponds to the **number of layers of the input volume**

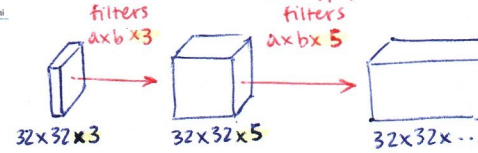
The output of the convolution against a filter becomes a slice in the volume feed to the next layer



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45659236

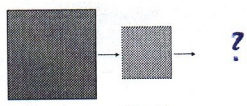
we apply 5 filters
a×b×3

we apply .. filters
a×b×5

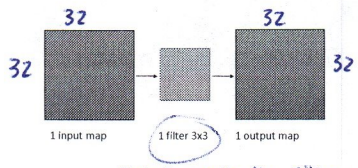


! The filters always have to mix up all the inputs component (component along the depth)

CNN Arithmetic

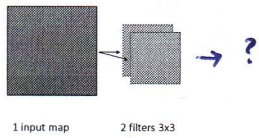


CNN Arithmetic

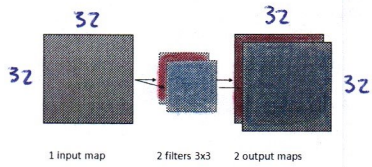


we suppose the filter to have padding (so it mantains the size)

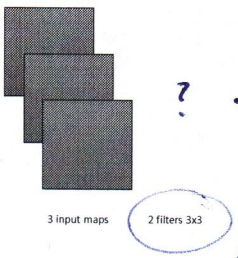
CNN Arithmetic



CNN Arithmetic

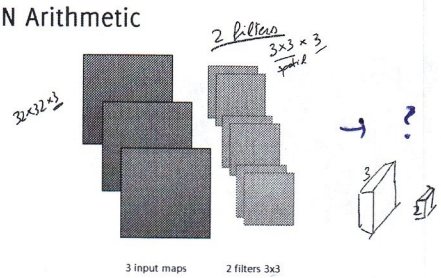


CNN Arithmetic

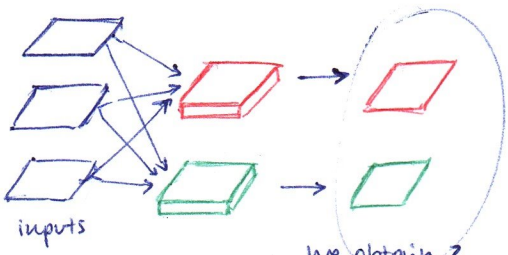


each filter needs to have the depth equal to the depth of the input

CNN Arithmetic



Having 2 filters on 3 inputs:

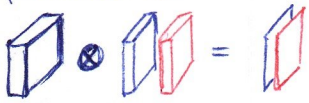


we obtain 2 outputs (2 slices)

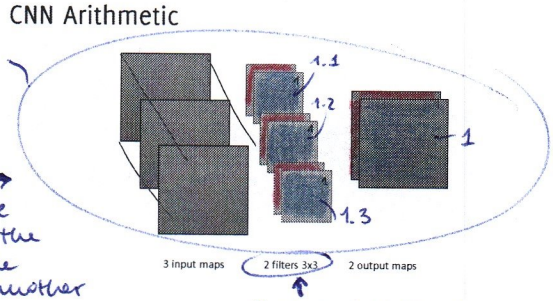
remember: these filters have to have the same depth as the input (the size of the input is equal to the # arrows entering the filter)

CNN Arithmetic

equivalent to:

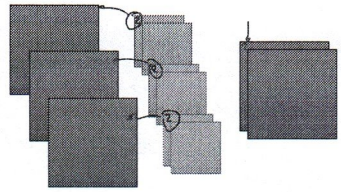


However, we represent it like → because we want to underline that the filter is of the depth of the input <u>and</u> that a slice of the filter may be different from another slice of the filter.
(1.1, 1.2, 1.3 may be different)



these are 2 filters of depth 3

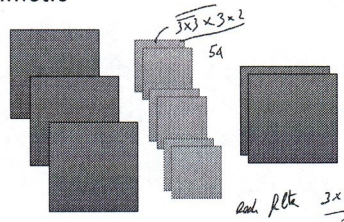
CNN Arithmetic



3 input maps 2 filters 3x3 2 output maps

#

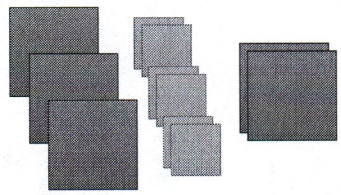
CNN Arithmetic



3 input maps 2 filters 3x3 2 output maps

Quiz: how many parameters does this layer have?

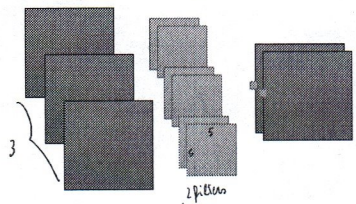
CNN Arithmetic



3 input maps 2 filters 3x3 2 output maps

= 54 parameters in the filters

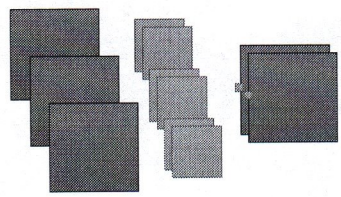
CNN Arithmetic



3 input maps 2 filters 3x3 2 output maps

= 54 ... + 2 biases

CNN Arithmetic



3 input maps 2 filters 3x3 2 output maps

= 54 + 2 biases

= 56 trainable parameters (weights)

How many parameters?

We have 2 kernels of dimension 3x3x3 since the inputs are 3

$\Rightarrow 2 \times 3 \times 3 \times 3 = 54$

but we also have a bias for every filter

$\Rightarrow 54 + 2 = 56$

total parameters

Other Layers

Activation and Pooling

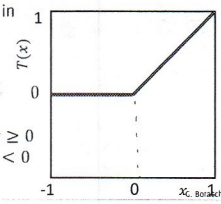
Activation Layers

Relu

Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier...

- **RELU** (Rectifier Linear Units): it's a thresholding on the feature maps, i.e., a max(0,·) operator.
  - By far the most popular activation function in deep NN (since when it has been used in AlexNet)
  - Dying neuron problem: a few neurons become insensitive to the input (vanishing grandient problem)

$$T(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

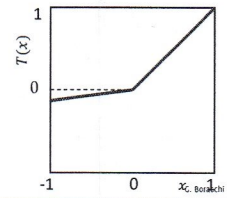


Activation Layers

Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier...

- **LEAKY RELU:** like the relu but include a small slope for negative values

$$T(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0.01 * x & \text{if } x < 0 \end{cases}$$



Activation Layers

Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier...

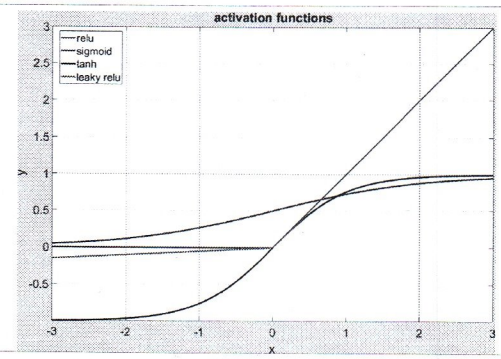
- **TANH** (hyperbolic Tangent): has a range (-1,1), continuous and differentiable

$$T(x) = \frac{2}{1 + e^{-2x}} - 1$$

- **SIGMOID:** has a range (0,1), continuous and differentiable

$$S(x) = \frac{1}{1 + e^{-2x}}$$

These activation functions are mostly popular **in MLP architectures**

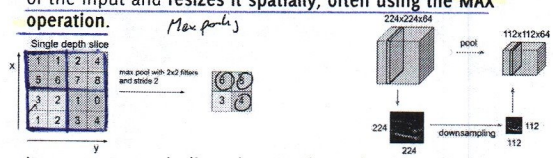


Pooling Layers

Pooling Layers reduce the **spatial** size of the volume.

The Pooling Layer operates **independently on every depth slice** of the input and **resizes it spatially, often using the MAX operation.**

Max pooling



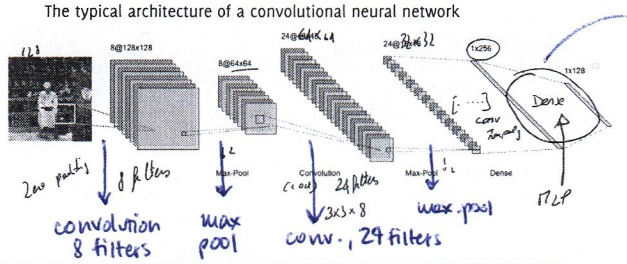
In a 2x2 support it discards 75% of samples in a volume

Dense Layers

Activation and Pooling

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network

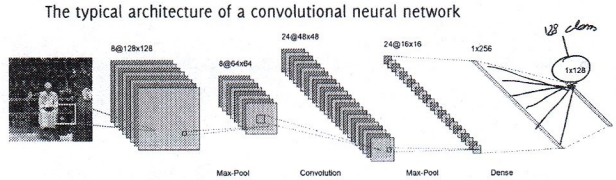


this is a DENSE LAYER, which is a multilayer perceptron network (MLP)

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. *"Gradient-based learning applied to document recognition"* Proceedings of the IEEE, 1998 86(11), 2278-2324.

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network

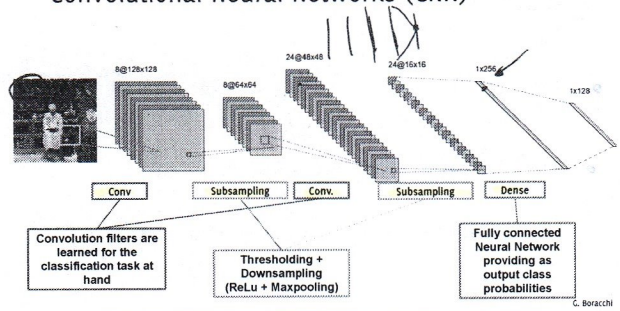


dense = each of the output neuron is connected to every input

The output of the **fully connected (FC) layer** has the same size as the **number of classes**, and **provides a score** for the input image to belong to each class

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. *"Gradient-based learning applied to document recognition"* Proceedings of the IEEE, 1998 86(11), 2278-2324.

Convolutional Neural Networks (CNN)



C. Boracchi

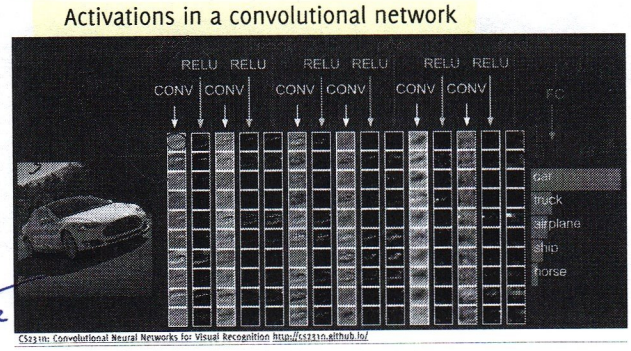
- convolution: increase in number
- subsampling (e.g. max-pool): decrease in size

REMEMBER: convolutions increase in number but doesn't necessarily change the dimension, however the **subsampling** only reduce the dimension, it doesn't modify the number

CNN «in action»

C. Boracchi

Activations in a convolutional network

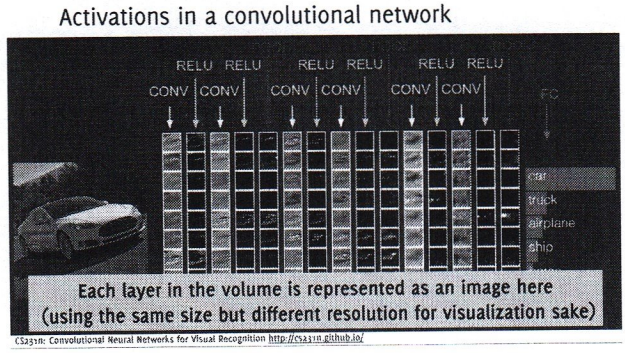


input image (which is a volume)

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

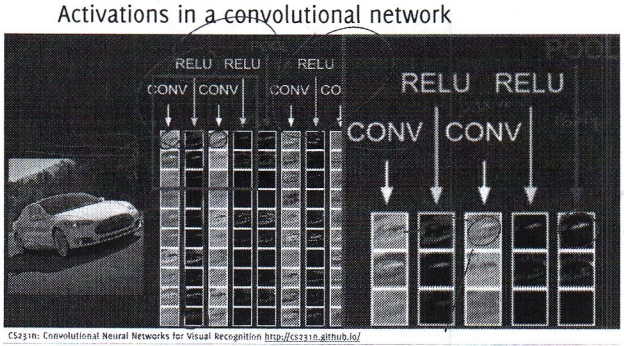
Activations in a convolutional network

we divide slice by slice and we obtain the first column



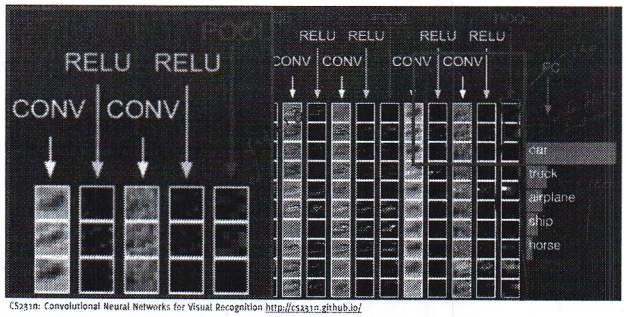
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Activations in a convolutional network



CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

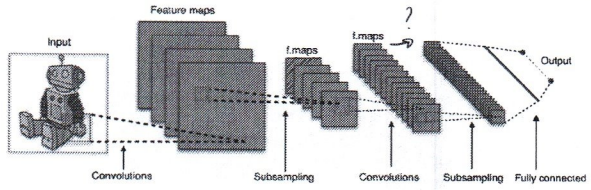
Activations in a convolutional network



CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Convolutional Neural Networks (CNN)

Btw, this figure contains an error.
If you are CNN-Pro, you should spot it!



By Aphex34 - Own work, CC BY-SA 4.0
https://commons.wikimedia.org/w/index.php?curid=45679374

G. Boracchi

The First CNN

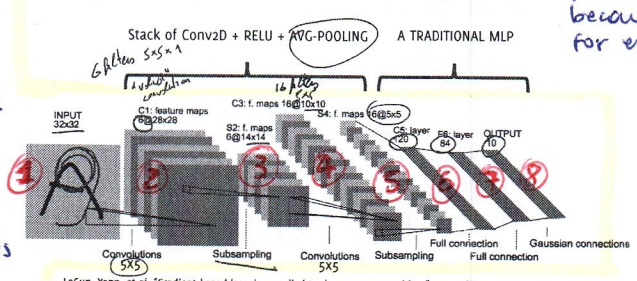
Part 1

1. 32x32 input image
2. From 1. to 2. we have 6 filters (since in 2. we have 6 slices). We have 6 convolutions that are 5x5x1 (we suppose the image to be gray scale).
   How can we know that the filters are 5x5 (x1)?
   Because the input size reduces by 4: 32x32 → 28x28
   (we perform convolution using the "valid option")
3. From 2. to 3. we do a sampling ("AVERAGE-POOL")
   → we reduce the dimension of the images (not the number)
   28x28x6 → 14x14x6
4. From 3. to 4. we have 16 filters (since the outputs are 16 slices). Again filters are 5x5 because we lose 4 in dimension:
   14x14 (x6) → 10x10 (x16)

G. Boracchi

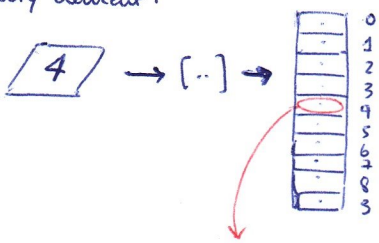
LeNet-5 (1998)

idea: classify images of numbers (the input image should be a number). The output's dimension is 10 because for every image we need a score for every element:



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

chi



We want this score to be the highest

The first CNN

Do not use each pixel as a separate input of a large MLP, because:

- images are highly spatially correlated,
- using individual pixel of the image as separate input features would not take advantage of these correlations.

The first convolutional layer: 6 filters 5x5

The second convolutional layer: 16 filters 5x5

5. From 4. to 5. we have again a sub-sampling:
   10x10 (x16) → 5x5 (x16)

6. The result produced in 5. are directly fed to a fully connected layer of 120 neurons (and then to 7. a hidden layer with 84 layers and then a hidden layer 8. with 10 neurons).

G. Boracchi

Part 2

model.summary()

The first is "None" because it corresponds to the size of the batch

input 32x32x1

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 6)	156
average_pooling2d_1 (Average	(None, 14, 14, 6)	0
conv2d_2 (Conv2D)	(None, 10, 10, 16)	2416
average_pooling2d_2 (Average	(None, 5, 5, 16)	0
flatten_1 (Flatten)	(None, 400)	0
dense_1 (Dense)	(None, 120)	48120
dense_2 (Dense)	(None, 84)	10164
dense_3 (Dense)	(None, 10)	850

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0

6 filters 5x5 + 6 biases

$5 \times 5 \times 1 = 150 + 6$ bias

16 filters 5x5x6 + 16 biases

$400 = 5 \times 5 \times 16$

NN

400

120

$400 \times 120 + 120$

$120 \times 84 + 84$

DENSE LAYER: #params?
The output is 120, the input 400 and they're all connected:
120×400 + 120 biases

G. Boracchi

model.summary()

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 6)	156 (6 x 5 x 5 + 6)
average_pooling2d_1 (Average	(None, 14, 14, 6)	0
conv2d_2 (Conv2D)	(None, 10, 10, 16)	2416 (16 x 5 x 5 x 6 + 16)
average_pooling2d_2 (Average	(None, 5, 5, 16)	0
flatten_1 (Flatten)	(None, 400)	0
dense_1 (Dense)	(None, 120)	48120
dense_2 (Dense)	(None, 84)	10164
dense_3 (Dense)	(None, 10)	850

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0

61k

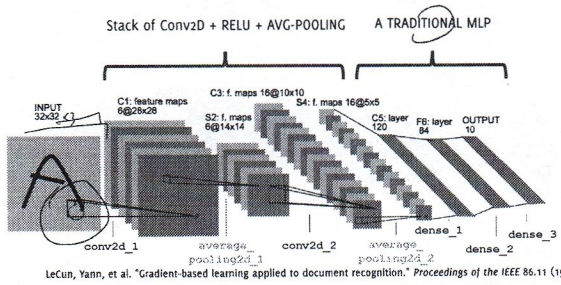
Input is a grayscale image

The input is a volume having depth = 6

Most parameters are still in the MLP

G. Boracchi

LeNet-5 (1998)



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE 86.11 (1998)*

chi

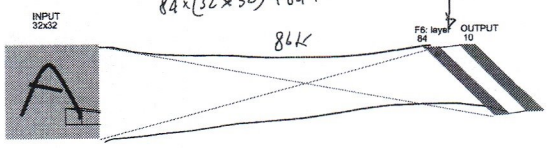
Most of parameters are in MLP

What about a MLP taking as input the whole image?

Input 32 x 32 = 1024 pixels, fed to a 84 neurons (the last FC layers of the network) → 86950 parameters: 1024 * 84 + 84 + 84 * 10 + 10

$84 \times (32 \times 32) + 84 +$

86k



In the previous model we get 61k parameters. Connecting the 32x32 image directly to the NN gives 86k parameters.
What if the input image has colors? So it's no longer 1@32x32 but 3@32x32.

→ The direct connection of the input image (3@32x32) to a dense layer of a NN gives 86k · 3 parameters. Instead, with the previous process, even if we have a colored input (3@28x28) the only things that change are the first convolutional filters.
→ with the CNN we have an increase of a couple of hundreds of parameters → ~ < 62k total #

Most of parameters are in MLP

What about a MLP taking as input the whole image?

Input 32 x 32 = 1024 pixels, fed to a 84 neurons (the last FC layers of the network) → 86950 parameters

$(32 \times 32 \times 3) \times 84$

But.. If you take an RGB input: 32 x 32 x 3,

CNN: only the nr. of parameters in the filters at the first layer increases

$$156 + 61550 \rightarrow 456 + 61550$$

$$(6 \times 5 \times 5) \rightarrow (6 \times 5 \times 5 \times 3)$$

300

gray scale color

MLP: everything increases by a factor 3

$$86950 \rightarrow 86950 \times 3$$

G. Boracchi

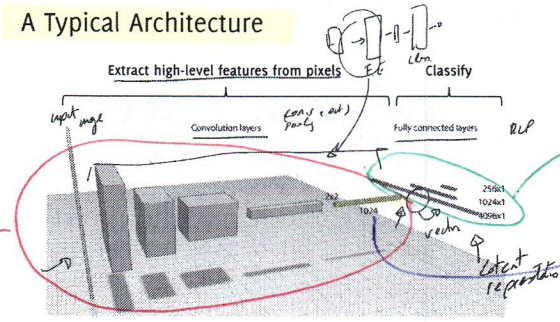
→ CNN really helps with the number of parameters

Latent representation in CNNs

Repeat the «t-SNE experiment» on the CIFAR dataset, using the last layer of the CNN as vectors

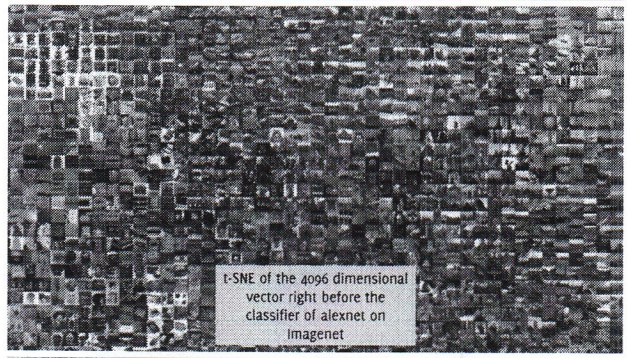
G. Boracchi

A Typical Architecture

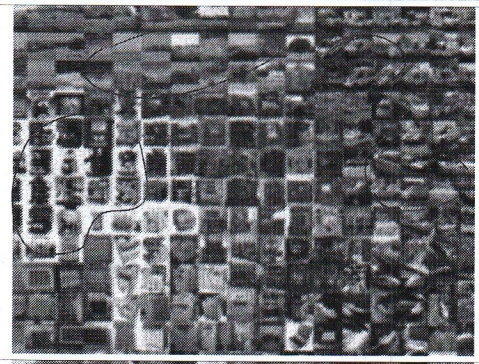


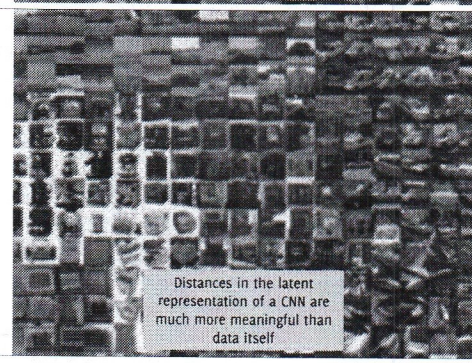
we go on untill we get to a small volume, then we arrange this final volume to a vector and we use the vector to a NN to perform a classification

G. Boracchi



t-SNE of the 4096 dimensional vector right before the classifier of alexnet on Imagenet





Distances in the latent representation of a CNN are much more meaningful than data itself

Keras

G. Boracchi

What is Keras?

An open-source library providing **high-level building blocks** for developing deep-learning models in Python

Designed to enable **fast experimentation with deep neural networks**, it focuses on being **user-friendly, modular, and extensible**

Doesn't handle low-level operations such as tensor manipulation and differentiation.

Relies on **backends** (TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML)

Enables **full access to the backend**

K

G. Boracchi

The software stack

G. Boracchi

Why Keras?

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

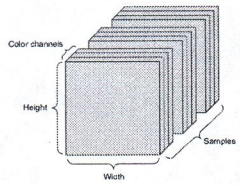
Cons:
Not as flexible
Need more flexibility? Access the backend directly!

G. Boracchi

We will manipulate 4D tensors

— 4D tensors are like 4dimensional arrays

Images are represented in 4D tensors:
Tensorflow convention: (samples, height, width, channels)



G. Boracchi

Building the Network

G. Boracchi

Convolutional Networks in Keras

```
# it is necessary to import some package
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Conv2D, MaxPooling2D

# and initialize an object from Sequential()
model = Sequential()
```

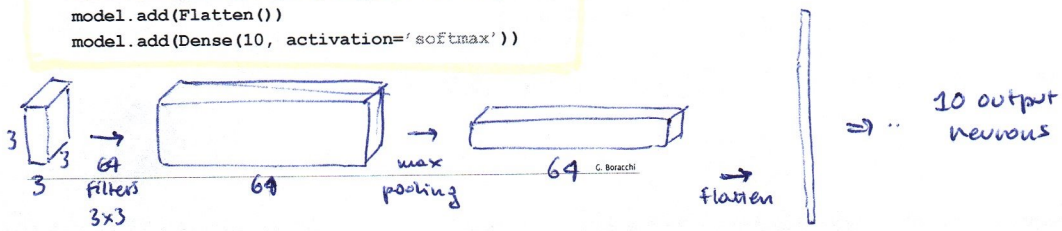
possible layers that we can use

→ place where we stack all the layers
How do we stack them?
model.add(...)

G. Boracchi

A very simple CNN

```
# Network Layers are stacked by means of the
.add() method
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
```



G. Boracchi

Convolutional Layers

```
# Convolutional Layer
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
# the input are meant to define:
# - The number of filters,
# - The spatial size of the filter (assumed
squared), while the depth depends on the network
structure
# - the activation layer (always include a
nonlinearity after the convolution)
# - the input size: (rows, cols, n_channels)
```


Convolutional Layers

```
# Convolutional Layer
model.add(Conv2D(filters = 64, kernel_size=3,
activation='relu', input_shape=(28,28,1)))
# This layer creates a convolution kernel that
is convolved with the layer input to produce a
tensor of outputs.
# When using this layer as the first layer in a
model, provide the keyword argument input_shape
(tuple of integers, does not include the batch
axis), e.g. input_shape=(128, 128, 3) for
128x128 RGB pictures in
data_format="channels_last".
```


Conv2D help

it says how we jump from one pixel to another:
- strides 1 ⇒ we move to the next pixel
- strides 2 ⇒ we move to:

Having a stride > 1 is a way to reduce the output dimension

Arguments

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

padding: one of "valid" or "same" (case-insensitive). Note that "same" is slightly inconsistent across backends with strides != 1, as described here

data_format: A string, one of "channels_last" or "channels_first". The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape (batch, height, width, channels) while "channels_first" corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

Conv2D help

Arguments

dilation_rate: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.

activation: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

use_bias: Boolean, whether the layer uses a bias vector.

kernel_initializer: Initializer for the kernel weights matrix (see initializers).

bias_initializer: Initializer for the bias vector (see initializers).

kernel_regularizer: Regularizer function applied to the kernel weights matrix (see regularizer).

bias_regularizer: Regularizer function applied to the bias vector (see regularizer).

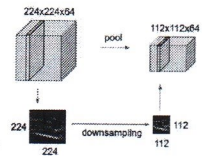
activity_regularizer: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).

kernel_constraint: Constraint function applied to the kernel matrix (see constraints).

bias_constraint: Constraint function applied to the bias vector (see constraints).

MaxPooling Layers

```
# Maxpooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))
# the only parameter here is the (spatial) size
to be reduced by the maximum operator
```



MaxPooling2D help

Arguments:

pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.

strides: Integer, tuple of 2 integers, or None. Strides values. If None, it will default to pool_size.

padding: One of "valid" or "same" (case-insensitive).

data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

MaxPooling2D help

Input shape:

If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

Output shape:

If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)

If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols)

Fully Connected Layers

```
# at the end the activation maps are "flattened" i.e.
they move from an image to a vector (just unrolling)
model.add(Flatten())
# Dense is a Fully Connected layer in a traditional
Neural Network.
model.add(Dense(units=10, activation='softmax'))
# Implements:
# output = activation(dot(input, kernel) + bias)
• activation is the element-wise activation function
  passed as the activation argument,
• kernel is a weights matrix created by the layer,
• bias is a bias vector created by the layer
# "Units" defines the number of neurons
```


Visualizing the model

```
# a nice output describing the model
architecture
model.summary()
```


Overview of the network + number of parameters involved:

batch size

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 26, 26, 64)	640
flatten_3 (Flatten)	(None, 43264)	0
dense_4 (Dense)	(None, 10)	432650

Total params: 433,290
Trainable params: 433,290
Non-trainable params: 0

there there is "max Pooling"

There was no padding option ⇒ the original input was 28x28 but 1 pixel get lost ⇒ 26x26 (due to padding. filters are 3x3)

Training the Model

Compiling the model

Then we need to compile the model using the compile method and specifying:

- **optimizer** which controls the learning rate. Adam is generally a good option as it adjusts the learning rate throughout training.
- **loss function** the most common choice for classification is 'categorical_crossentropy' for our loss function. The lower the better.
- **Metric** to assess model performance, 'accuracy' is more interpretable

```
model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
```


Now we are ready for training (until now we composed the model)

Training the model using

The `fit()` method of the model is used to train the model.

Specify at least the following inputs:

- training data (input images),
- target data (corresponding labels in categorical format),
- validation data (a pair of data, labels to be used only for computing validation performance)
- number of epochs (number of times the whole dataset is scanned during training)

```
model.fit(X_train, y_train,
validation_data=(X_test, y_test), epochs=3)
```


Training output

```
Epoch 22/100
18000/18000 [==============================] - 136s 8ms/step - loss: 0.7567
- acc: 0.6966 - val_loss: 1.9446 - val_acc: 0.4325
Epoch 23/100
18000/18000 [==============================] - 137s 8ms/step - loss: 0.7520
- acc: 0.6959 - val_loss: 1.9646 - val_acc: 0.4275
Epoch 24/100
18000/18000 [==============================] - 137s 8ms/step - loss: 0.7442
- acc: 0.7024 - val_loss: 1.9067 - val_acc: 0.4129
```


Advanced Training Options

used to check how the training progress is (wether the training can be stopped (early stopping), or more generally checkpoints)
Moreover checkpoints "save" the model (if we train it for days or something)

Callbacks in Keras

A callback is a **set of functions** to be **applied at given stages** of the training procedure.

Callbacks give a **view on internal states** and statistics of the model **during training**.

You can pass **a list of callbacks** (as the keyword argument callbacks) to the .fit() method of the Sequential or Model classes.

The relevant methods of the callbacks will then be called at each stage of the training.

```
callback_list = [cb1,…,cbN]
model.fit(X_train, y_train,
validation_data=(X_test, y_test), epochs=3,
callbacks = callback_list)
```


Model Checkpoint

Training a network might take up to several hours

Checkpoints are snapshots of the state of the system to be saved in case of system failure.

When training a deep learning model, the checkpoint is the weights of the model. These weights can be used to make predictions as is, or used as the basis for ongoing training.

```
from keras.callbacks import ModelCheckpoint
[…]
cp = ModelCheckpoint(filepath,
monitor='val_loss', verbose=0,
save_best_only=False, save_weights_only=False,
mode='auto', period=1)
```


Early Stopping

The only stopping criteria when training a Deep Learning model is "reaching the required number of epochs."

However, it might be enough to train a model further, as sometimes the training error decreases but the validation error does not (overfitting)

Checkpoints are used to stop training when a monitored quantity has stopped improving.

```
from keras.callbacks import EarlyStopping
[…]
es = EarlyStopping(monitor='val_loss',
min_delta=0, patience=0, verbose=0, mode='auto',
baseline=None, restore_best_weights=False)
```

Testing the model

Predict() method

```
#returns the class probabilities for the input
image X_test
score = model.predict(X_test)
# select the class with the largest score
prediction_test = np.argmax(score, axis=1)
```


Tensorboard

When training a model it is important to monitor its progresses

Google has developed tensorboard a very useful tool for visualizing reports.

```
from keras.callbacks import TensorBoard
[...]
 tb = TensorBoard(log_dir="dirname")
```

... and add tb to the checkpoint list as well

CNN Parameters and Training

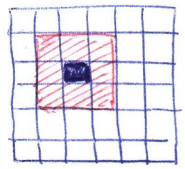
Giacomo Boracchi,
DEIB, Politecnico di Milano
October, 28th, 2

giacomo.boracchi@polimi.it
https://boracchi.faculty.polimi.it/

G. Boracchi

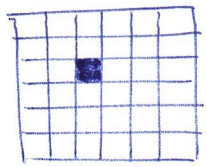
part 1

Can the following CNN be seen as a special case of a Feedforward neural network?
Yes.

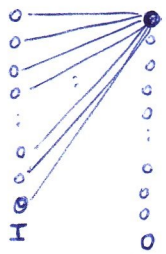


200 x 200

FILTER
3x3x1

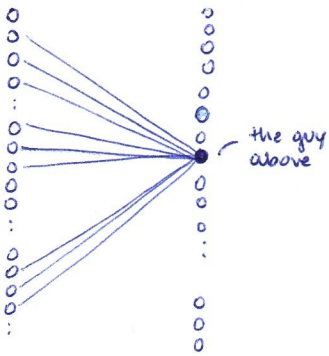


We want to look at it as a Feedforward Neural Network (multilayer perceptron):
1 input, 1 output
input-size = 200·200 = 40k
output-size = 200·200 = 40k
But it's not that every neuron is connected to every neuron:



In this case we would have 40k x 40k + 40k parameters

Having the 3x3 filter changes the structure. We can see in the CNN that every pixel output depends on 9 pixels input (only 9, not 40k):



the guy above

3 + 3 + 3 + 1 connections for every output neuron (+1 for the bias)

⇒ CNN is a feedforward neural network featuring SPARSE CONNECTIVITY

Parameters in a CNN

G. Boracchi

Convolutions as MLP

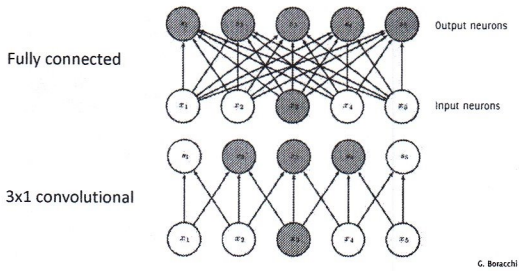
Convolution is a linear operation!

Therefore, if you unroll the input image to a vector, you can consider convolution weights as the weights of a Multilayer Perceptron Network

What are the differences between MLP and CNNs then?

G. Boracchi

CNNs Feature Sparse Connectivity

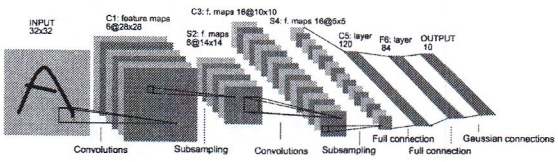


G. Boracchi

Weight Sharing / Spatial Invariance

In a CNN, all the neurons in the same slice of a feature map use the same weights and bias: this reduces the nr. of parameters in the CNN.

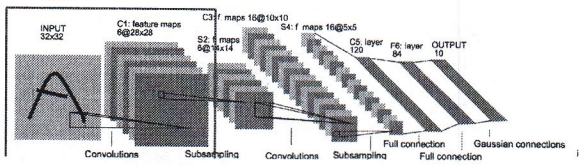
Underlying assumption: **if one feature is useful** to compute **at some spatial position** (x,y), **then it should also** be useful to compute at a **different position** (x2,y2)



Weight Sharing / Spatial Invariance

If the first layer were a MLP:

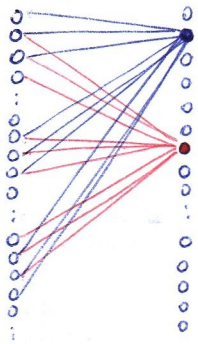
- it should have had 28 x 28 x 6 neurons in the output
- Consider sparse connectivity: 5x5 nonzero weights each neuron
- It would be a 28 x 28 x 6 x 25 weights + 28 x 28 x 6 biases (122 304)
  CNN share the same weights among a few neurons of the same layer



part 2

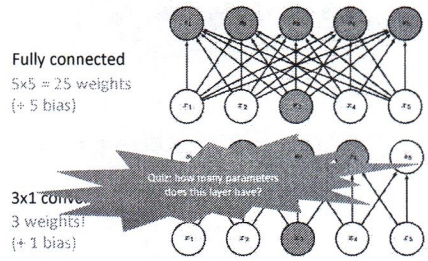
Let's check the #parameters.
If every output is connected to 9 neurons and has 1 bias ⇒ 10 parameters
⇒ the feedforward NN has 400 k parameters
However in the CNN we have only 10 parameters for the whole network (we just have to find out the filter in a CNN).
What are we missing?
In the feedforward NN THE WEIGHTS ARE SHARED.

⇒ • SPARSE CONNECTIVITY
• WEIGHTS SHARING



The weights of the blue connections are the same of the weights of the red connections.

Parameter sharing



The Receptive Field
A very important aspect in CNNs

G. Boracchi

The Receptive Field

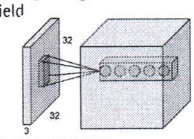
One of the basic concepts in deep CNNs.

Due to sparse connectivity, unlike in FC networks where the value of each output depends on the entire input, **in CNN an output only depends on a region of the input.**

This region in the input is the receptive field for that output

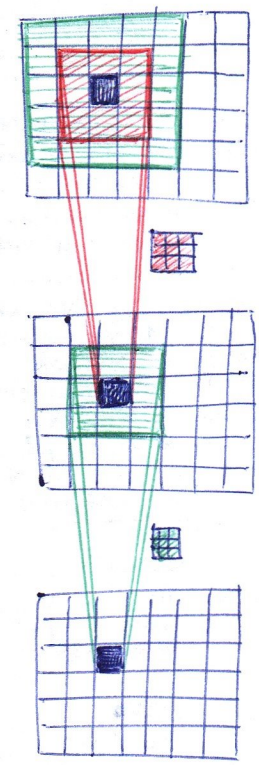
The deeper you go, the wider the receptive field is: maxpooling, convolutions and stride > 1 increase the receptive field

Usually, the receptive field refers to the final **output unit** of the network in relation to the network input, but the same definition holds for intermediate volumes



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45692236

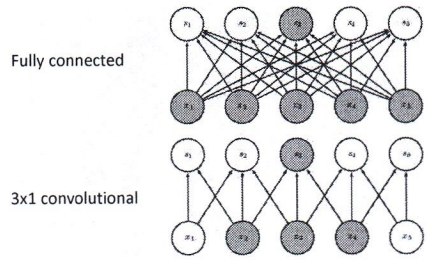
G. Boracchi



The deeper we go, the wider the receptive field gets. The ■ of the second layer has a receptive field of 9 pixels, the ■ of the third layer has a receptive field of 25 neurons.

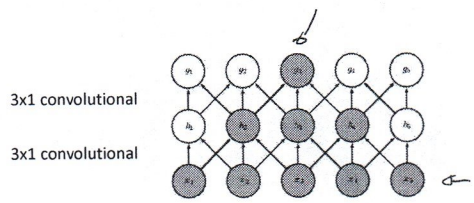
(**Notice**: in the fully connected feed forward neural networks the receptive field is the whole input, no matter how deep we go)

Receptive fields

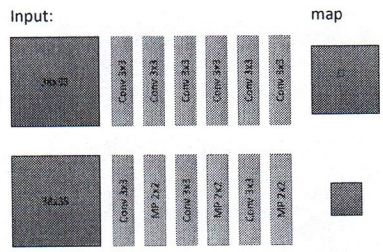


Receptive fields

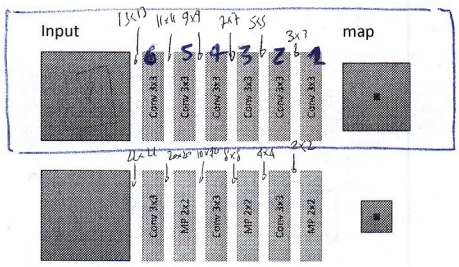
Deeper neurons depend on wider patches of the input (convolution is enough to increase receptive field, no need of maxpooling)



Exercise



Receptive fields

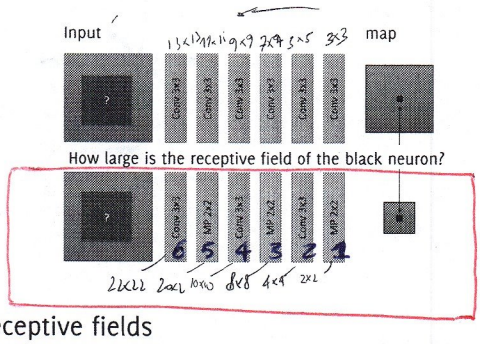


If the filter is 3x3 we're always adding 1 colum left and right and 1 row left and right

1 → 3x3
2 → 5x5
3 → 7x7
4 → 9x9
5 → 11x11
6 → $\boxed{13 \times 13}$

receptive field for every output pixel
(actually there is size-reduction due to padding options, 13x13 is the receptive field of the centered pixels, for the pixels in the border the perceptive field is different)

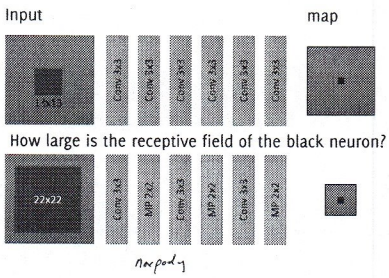
Receptive fields



Here we have the max-pooling:
The maxpooling always half the size:

1 → 2x2
2 → 4x4 (because we add 1 column left-right, 1 row left-right)
3 → 8x8 (because maxpooling always half the size, so before maxpooling we got double the maxpool output)
4 → 10x10
5 → 20x20
6 → $\boxed{22 \times 22}$ ⇒ having maxpooling induces having wider receptive field

Receptive fields



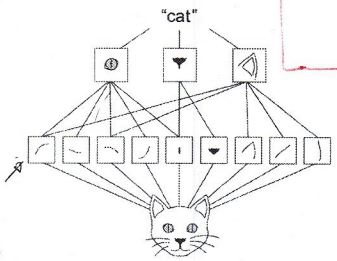
As we move deeper...

As we move to deeper layers:

- spatial resolution is reduced
- the number of maps increases

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

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



that's why we do the weights sharing: if an eye is in the center or in the bottom we want to classify it as an eye (∀ on its position)

CNN Training

G. Boracchi

Training a CNN

- The CNN can be seen as a MLP (with sparse and shared connectivities)
- CNN can be in principle trained by gradient descent to minimize a loss function over a batch (e.g. binary cross-entropy, RMSE, Hinge loss..)
- Gradient can be computed by backpropagation (chain rule) as long as we can derive each layer of the CNN
- Weight sharing needs to be taken into account (fewer parameters to be used in the derivatives) while computing derivatives
- There are just a few details missing

G. Boracchi

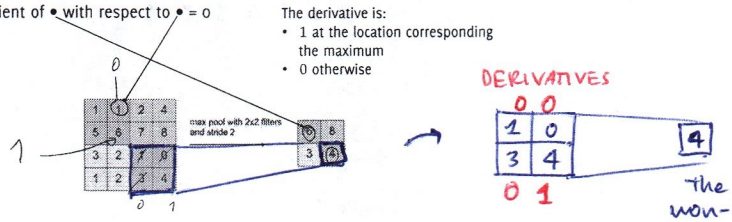
Detail: backprop with max pooling

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

Gradient of • with respect to • = 0

The derivative is:

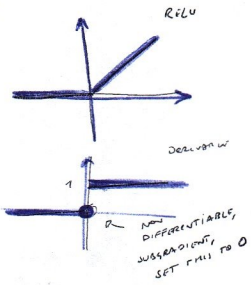
- 1 at the location corresponding the maximum
- 0 otherwise



The derivatives in the non-max locations are going to be 0, the derivatives on the max location is going to be 1

Detail: derivative of ReLU

The ReLU derivative is straightforward



Data Scarcity

Training a CNN with Limited Aumont of Data

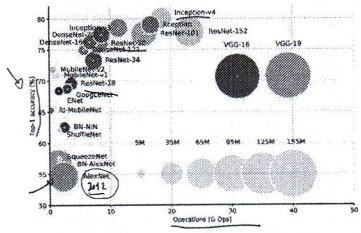
The need of data

Deep learning models are very data hungry.

Networks such as AlexNet have been trained on ImageNet datasets containing tens of thousands of images over hundreds of classes

The need of data

This is necessary to define millions of parameters characterizing these networks



Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications." *arXiv preprint arXiv:1605.07678* (2016).

The need of data

Deep learning models are very data hungry.

... watch out: each image in the training set have to be annotated!

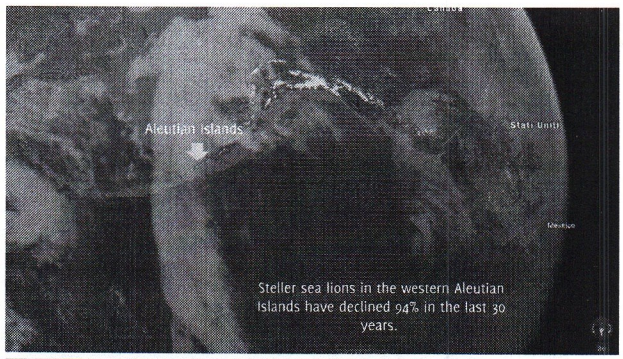
How to train a deep learning model with a few training images?

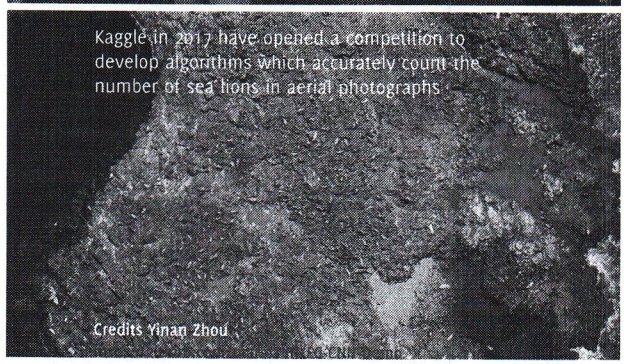
- Data augmentation
- Transfer Learning

Limited Amount of Data:
1. Data Augmentation

Training a CNN with Limited Aumont of Data

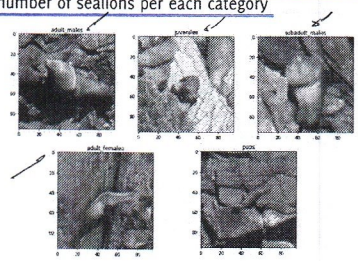
G. Boracchi



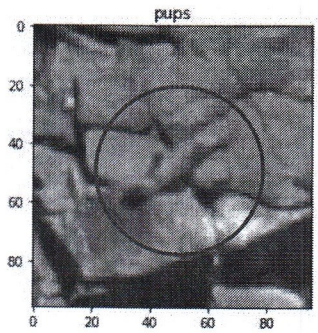


The Challenge

In very large aerial images (≈ 5K x 4K) shot by drones, automatically count the number of sealions per each category

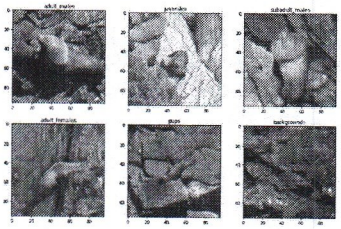


G. Boracchi



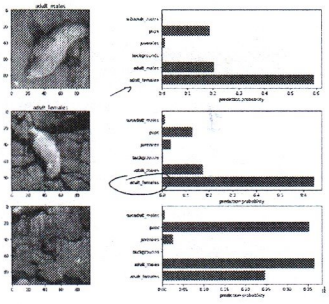
The Challenge

This problem can be naively casted in a patch-by-patch 6-class classification problem, where we include also background



G. Boracchi

An Example of CNN predictions

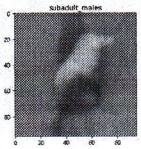


Credits Yinan Zhou
https://github.com/marioZYN/FC-CNN-Demo

Data Augmentation

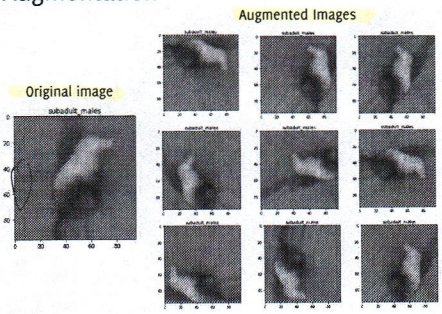
Often, each annotated image represents a class of images that are all likely to belong to the same class

In aereal photograps, for instance, it is normal to have rotated, shifted or scaled images without changing the label



G. Boracchi

Data Augmentation



Data Augmentation

Data augmentation is typically performed by means of

Geometric Transformations:

- Shifts /Rotation/Affine/perspective distortions
- Shear
- Scaling
- Flip

Photometric Transformations:

- Adding noise
- Modifying average intensity
- Superimposing other images
- Modifying image contrast

G. Boracchi

Augmentation in Keras

```
from keras.preprocessing.image import
ImageDataGenerator
ImageDataGenerator(
rotation_range=0,
width_shift_range=0.0, height_shift_range=0.0,
brightness_range=None, shear_range=0.0,
zoom_range=0.0, channel_shift_range=0.0,
fill_mode='nearest',
horizontal_flip=False, vertical_flip=False,
rescale=None,
preprocessing_function=None)
```

G. Boracchi

Augmentation in Keras: flow from images

The Image generator has a method `flow_from_directory` that allows to load images in folder where different classes are arranged in subfolders.

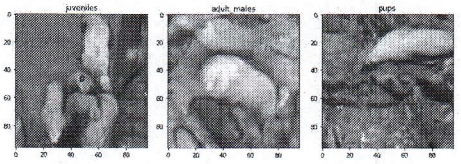
```
ImageDataGenerator.flow_from_directory(
    directory=PATCH_PATH + 'train/',
    target_size=(img_width, img_width),
    batch_size=batch_size,
    shuffle=True)
```

G. Boracchi

However...

This sort of data augmentation might not be enough to capture the inter-class variability of images...

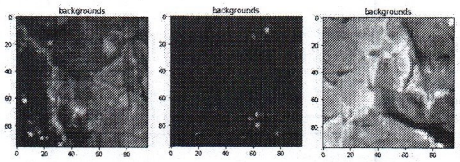
Superimposition of targets (aggregation of sealions)



G. Boracchi

However...

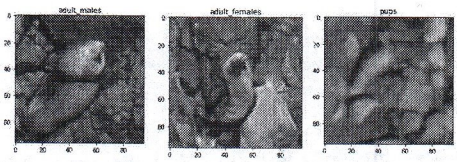
Background variations



G. Boracchi

However...

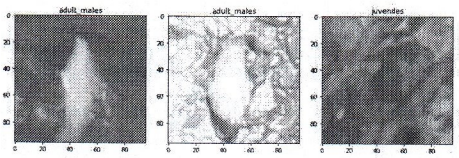
Background variations



G. Boracchi

However...

Out of focus, bad exposure



G. Boracchi

Augmentation in Keras

```
from keras.preprocessing.image import
ImageDataGenerator
ImageDataGenerator(
rotation_range=0,
width_shift_range=0.0, height_shift_range=0.0,
brightness_range=None, shear_range=0.0,
zoom_range=0.0, channel_shift_range=0.0,
fill_mode='nearest',
horizontal_flip=False, vertical_flip=False,
rescale=None,
preprocessing_function=None)
```

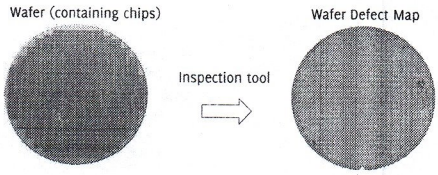
... in case you need some extra flexibility

G. Boracchi

Data Augmentation is often key..

Collaboration with

For instance to monitor wafer manufacturing

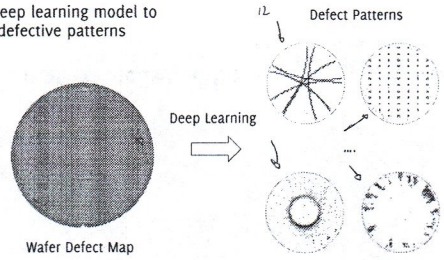


R. di Bella, D. Carrera, B. Rossi, P. Fragneto, G. Boracchi «Wafer Defect Map Classification Using Sparse Convolutional Networks" ICIAP 2019

Data Augmentation is often key..

Collaboration with

Train a deep learning model to identify defective patterns



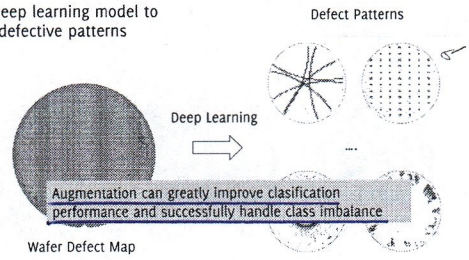
Data Augmentation may have to take into account the class labels!
Some Data Augmentation techniques may not have sense for some labels.

R. di Bella, D. Carrera, B. Rossi, P. Fragneto, G. Boracchi «Wafer Defect Map Classification Using Sparse Convolutional Networks" ICIAP 2019

Data Augmentation is often key..

Collaboration with

Train a deep learning model to identify defective patterns



R. di Bella, D. Carrera, B. Rossi, P. Fragneto, G. Boracchi «Wafer Defect Map Classification Using Sparse Convolutional Networks" ICIAP 2019

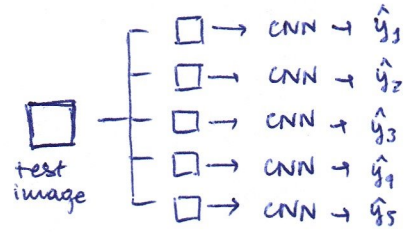
Test Time Augmentation (TTA)

Even if the CNN is trained using augmentation, **it won't achieve perfect invariance** w.r.t. considered transformations

Test time augmentation (TTA): augmentation can be also performed at test time to improve prediction accuracy.

- Perform random augmentation of each test image I
- Average the predictions of each augmented image
- Take the average vector of posterior for defining the final guess

Test Time Augmentation is particularly useful for test images where the model is pretty unsure.



we augment the input

we feed the <u>same CNN</u> with those images

⇒ We use data Augmentation to both the train of the CNN and to classify the new input (:= we augment the input)

The final label will be an aggregation of all the predictions ($\hat{y}_1, \dots, \hat{y}_5$)

G. Boracchi

A bit more of background

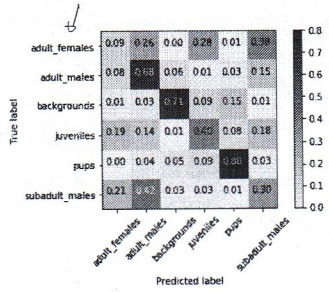
Performance measures

and an overview of successful architectures

G. Boracchi

Confusion Matrix

Case of 6 labels:



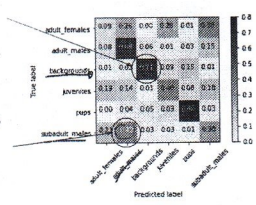
Confusion Matrix

The element $C(i,j)$ i.e. at the i-th row and j-th column corresponds to the percentage of elements belonging to class i classified as elements of class j

⇒ the main diagonal talks about how good it the <u>right</u> (correct) classification

71% of background patches have been correctly classified as background

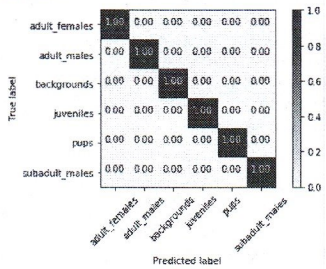
42% of sub-adult males patches have been wrongly classified as adult-males



G. Boracchi

… so, the ideal confusion matrix

Which rarely happens



G. Boracchi

Two-Class Classification (Binary classification)

Classification performance in case of binary classifiers can be also measured in terms of the **ROC** (receiver operating characteristic) **curve**, which does not depend on the threshold you set for each class

This is useful in case you plan to modify this and not use 0.5

The ideal detector would achieve:

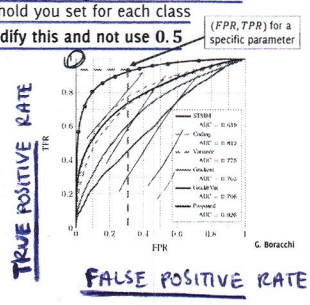
- $FPR = 0\%$,
- $TPR = 100\%$

Thus, the closer to (0,1) the better

The largest the **Area Under the Curve** (AUC), the better

The optimal parameter is the one yielding the point closest to (0,1)

(1,0) corresponds to:
100% true positives
0% false positives



G. Boracchi

0 0 0 0 : 0 0 X ≥ ≠ : 0 IP(class 1)
← : 0 1 − IP(class 1)
1 :

A few popular architectures

AlexNet
VGG

→ Transfer Learning
→ CNN visualization
→ (Fully Convolutional CNN)

G. Boracchi

Outline

Lecture inspired to:

Notes accompanying the Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

… and papers cited in here!

G. Boracchi

PROC. OF THE IEEE, NOVEMBER 1998 1

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—
Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

I. Introduction

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

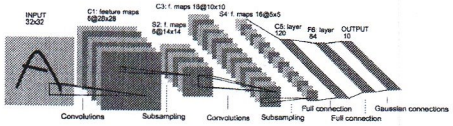
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

LeNet-5 (1998)

Yann Lecun's LeNet-5 model was developed in 1998 to identify handwritten digits for zip code recognition in the postal service.

This pioneering model largely introduced the convolutional neural network as we know it today

Parameters: 60.000



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998)

LeNet-5 (1998)

The idea of using a sequence of 3 layers: **convolution, pooling, non-linearity.**

Pixels in an image are highly correlated: there is no point to separately feed each pixel value through a classification network: add learnable convolutional layers

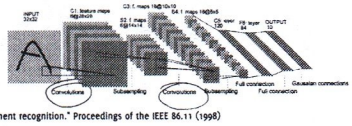
Convolution as a way to extract spatial features and to have sparse connections (thus reducing the number of parameters).

Subsample using spatial average of maps (averaging pooling)

Non-linearities: tanh or sigmoids

MLP as classifier for the final layer

FC layer (fully connected layer)



LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998)

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

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

AlexNet (2012)

Developed by Alex Krizhevsky et al. in 2012 and won Imagenet competition

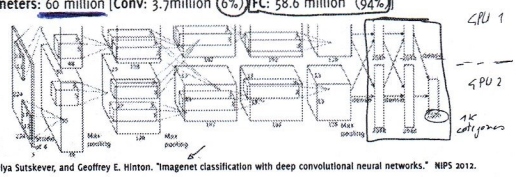
Architecture is quite similar to LeNet-5:

- 5 convolutional layers (rather large filters, 11x11, 5x5),
- 3 MLP

Input size 224 × 224 x 3 (the paper says 227 x 227 x 3)

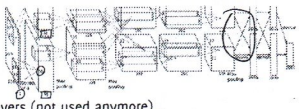
Parameters: 60 million [**Conv:** 3.7million (6%) **FC:** 58.6 million (94%)]

trained over 2 different GPUs



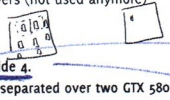
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

AlexNet (2012)



To counteract overfitting, they introduce:

- RELU (also faster than tanh)
- Dropout (0.5), weight decay and norm layers (not used anymore)
- Maxpooling



The first conv layer has 96 11x 11 filters, stride 4.

very large filters

The output are **two volumes of 55 x 55 x 48 separated over two GTX 580 GPUs** (3GB, 90 epochs, 5/6 days to train).

the way a filter jumps from one pixel to another

Most **connections are among** feature maps **of the same GPU**, which will be mixed at the last layer.

Won the ImageNet challenge in 2012

At the end they also trained an **ensemble of 7 models** to drop error: 18.2%->15.4%

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* & Andrew Zisserman†
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

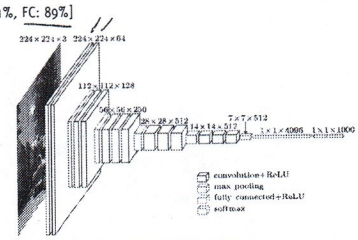
In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16 (2014)

The VGG16, introduced in 2014 is a deeper variant of the AlexNet convolutional structure. Smaller filters are used and the network is deeper

Parameters: 138 million [Conv: 11%, FC: 89%]



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16 (2014)

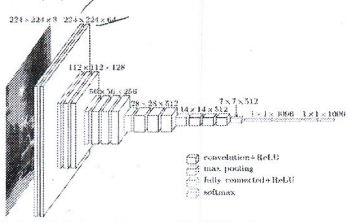
The VGG16, introduced in 2014 is a deeper variant of the AlexNet convolutional structure. Smaller filters are used and the network is deeper

Parameters: 138 million [Conv: 11%, FC: 89%]

conv (? , ? , 64)

These architecture won the first place places (localization) and the second place (classification) tracks in ImageNet Challenge 2014

Input size 224 × 224 x 3



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

VGG16 (2014): Smaller Filter, Deeper Network

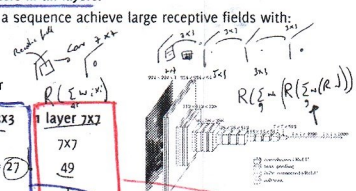
The paper actually present a thorough **study on the role of network depth.**

[...]Fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due to the use of very small (3. × 3) convolution filters in all layers.

Idea: Multiple 3×3 convolution in a sequence achieve large receptive fields with:

- less parameters
- more nonlinearities

than larger filters in a single layer



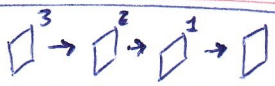
	3 layers 3x3	1 layer 7x7
Receptive field	7x7	7x7
Nr of parameters	3 x 3 x 3 = 27	49
Nr of nonlinearities	3	1

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." ICLR (2015)

SMALLER FILTERS, DEEPER NET.

we prefer small filters (~ 3x3) to the larger ones (11x11 e.g.)
The reason is: with fewer filters we can have less parameters to achieve the same receptive field and with more nonlinearity.

1 filter 7x7
⇒ the receptive field of each pixel of output is 49
also the number of coefficients is 49 (we only need to find this filter).
we just have one non-linearity. (we have just 1 activation)



3 times a 3x3 convolution
The receptive field is:
1 → 3x3
2 → 5x5
3 → 7x7 again 49 !
The number of parameters is 9 x 3 = 27 (< 49) and the number of nonlinearities goes to 3 (so good!)

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[...]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

G. Boracchi

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[...]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

Many convolutional blocks without maxpooling

G. Boracchi

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[...]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

Most parameters in FC layers : 123,642,856

G. Boracchi

VGG16

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)		
block2_conv2 (Conv2D)		
block2_pool (MaxPooling2D)		
block3_conv1 (Conv2D)		
block3_conv2 (Conv2D)		
block3_conv3 (Conv2D)		
block3_pool (MaxPooling2D)		
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
[...]		
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
	14, 14, 512)	2359808
	7, 7, 512)	0
	25088)	0
	4096)	102764544
	4096)	16781312
	1000)	4097000

High memory request, about 100MB per image to be stored in all the activation maps, only for the forward pass (for the backward it's about twice)

Trainable params: 138,357,544
Non-trainable params: 0

G. Boracchi

Popular Architectures to be continued...

G. Boracchi

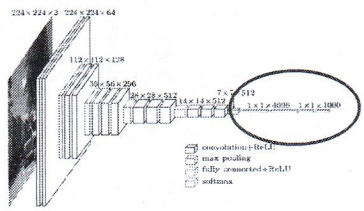
Limited Amount of Data:
2. Transfer Learning

Training a CNN with Limited Aumont of Data

G. Boracchi

Transfer Learning

How to use pre-trained models to solve different problems

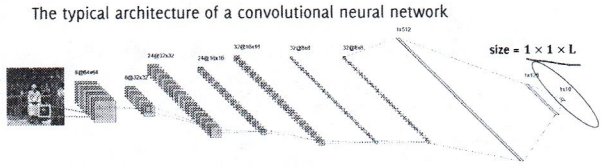


Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

G. Boracchi

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network



The output of the fully connected layer has the same size as the number of classes L, and each component provide a score for the input image to belong to a specific class

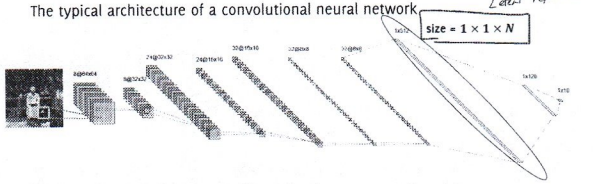
By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

G. Boracchi

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network

Latent representation.



The input of the fully connected layer can be seen as a data-driven descriptor of the input image, i.e. a feature vector. These features are:

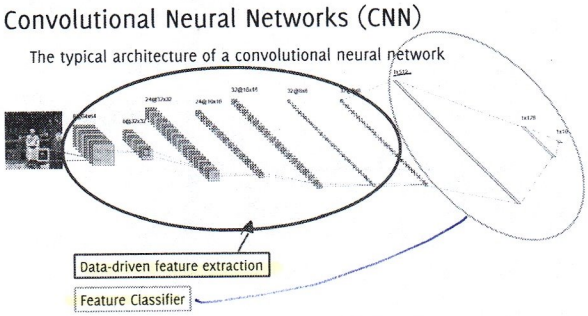
- defined to maximize classification performance
- Trained via backpropagation as the NN they are fed

By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

G. Boracchi

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network

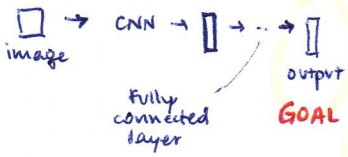


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

G. Boracchi

Once we have our trained model, we can use it in 2 ways:

1. We can simply use it to solve the specific classification problem (we look at the outputs (1x1xL), i.e. the probability scores, and we classify)

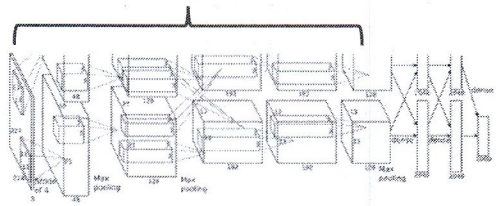


2. We can use it as a kind of FEATURES EXTRACTOR, which means that we're not using the network in order to compute directly the classification output, but we're interested in the latent representation which is fed to the fully connected layer

= the focus is the way we extract features, not the classification itself. If we understand how we're doing the features extraction then we can apply it to other problems

Deep architectures

Deep Networks are great at extracting
(learned) features from images



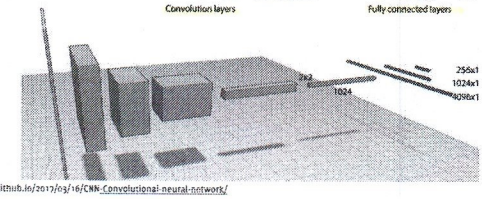
AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

icchi

A typical architecture

Extract high-level features from pixel data — Classify

CONVOLUTION LAYERS — FULLY CONNECTED LAYERS

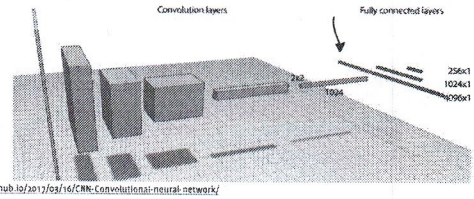


https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

A typical architecture

As we move to deeper layers:

- spatial resolution is reduced
- the number of maps increases

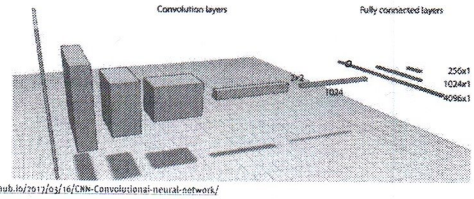


https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

A typical architecture

As we move to deeper layers:

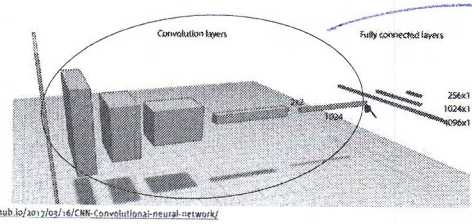
- We search for higher-level patterns, and don't care too much about their exact location.
- There are more high-level patterns than low-level details!



https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

A typical architecture: Convolutional Block

In VGG or Alexnet network that was trained to classify 1000 classes from Imagenet...
the **convolutional part should be a very general feature extractor**

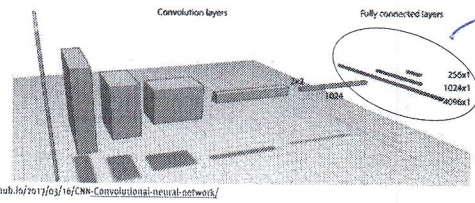


part that we suppose to be general enough (we want to apply this features extraction to other problems)

https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

A typical architecture: FC layers

The **FC layers are instead very custom**, as they are meant to solve the specific classification task at hand

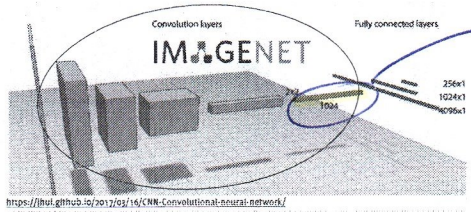


this is extremely problem-related. Every problem has its own, we will not transfer this.

https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning

- Take a successful pre-trained model such as VGG

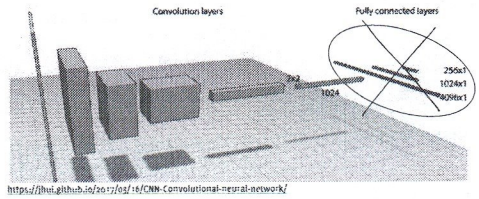


latent representation

https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning

- Take a successful pre-trained model such as VGG
- Remove and modify the fully connected layers for the problem at hand

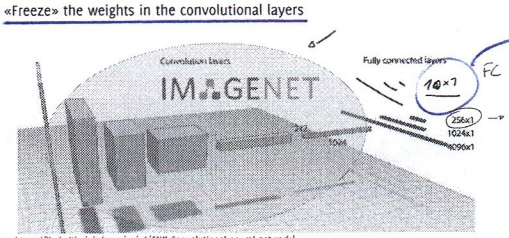


https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

G. Boracchi

Transfer Learning

- Take a successful pre-trained model such as VGG
- Remove and modify the fully connected layers for the problem at hand
- «Freeze» the weights in the convolutional layers

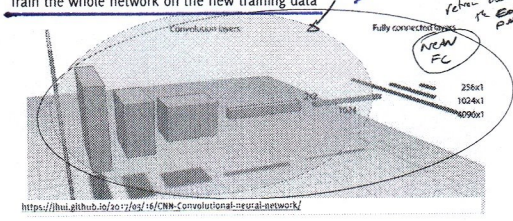


Suppose the fully connected layers part is used for a classification into 10 classes (the output is 10x1)

https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Transfer Learning

- Take a successful pre-trained model such as VGG
- Remove and modify the fully connected layers for the problem at hand
- «Freeze» the weights in the convolutional layers
- Train the whole network on the new training data



Freeze Conv part | T.L
Train FC
retrain the Conv part | Fine Tuning

https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

Two options:

1. train only the fully connected layers (the specific problem classification part). In order to do this we have to "freeze" the weights in the convolutional layers (we cannot update them)

   FREEZE CONVOLUTION + TRAIN FC

   := Transfer learning

2. train both parts (so re-train the convolutional part)

   TRAIN CONVOLUTION + TRAIN FC

   := Fine Tuning

   (where the pre-trained convolutional part is seen as initialization. In this way, in the search for a minima we're already starting (in the gradient descent procedure) close to a very good solution)

In keras...

Pre-trained models are available, typically in two ways:

- **`include_top = True`**: provides the entire network, including the fully convolutional layers. This network can be used to solve the classification problem it was trained for
- **`include_top = False`**: contains only the convolutional layers of the network, and it is specifically meant for transfer learning.

Have a look at the size of these models in the two options!

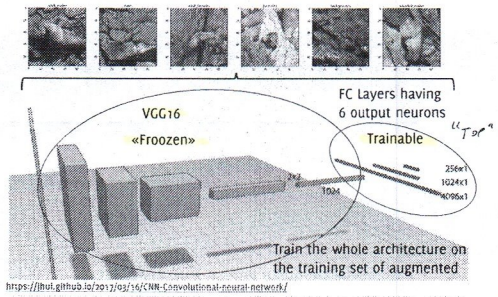
G. Boracchi

VGG16 in keras...

```
from keras import applications
base_model = applications.VGG16(weights =
"imagenet", include_top=False, input_shape =
(img_width, img_width, 3))
```

G. Boracchi

Transfer Learning in the Sealion Case



https://jlhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

G. Boracchi

Transfer Learning in Keras...

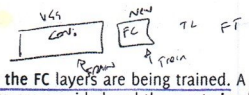
Requires a bit of TensorFlow Backend to add the modified Fully connected layer at the top of a pretrained model

Then, before training it is necessary to loop through the network layers (they are in `model.layers`) and then modify the trainable property

```
for layer in model.layers[: lastFrozen]:
    layer.trainable=False
```

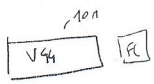
G. Boracchi

Transfer Learning



Different Options:

- **Transfer Learning:** only the FC layers are being trained. A good option when **little training data** are provided and the **pre-trained model** is expected to **match** the problem at hand
- **Fine tuning:** the whole **CNN is retrained**, but the **convolutional layers are initialized to the pre-trained model**. A good option when **enough training data** are provided or when the **pre-trained model is not expected to match** the problem at hand



G. Boracchi

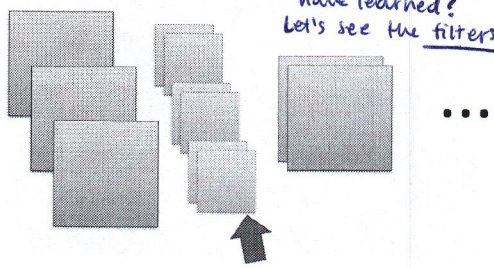
CNN Visualization

= we want to know what a neural network has learned

G. Boracchi

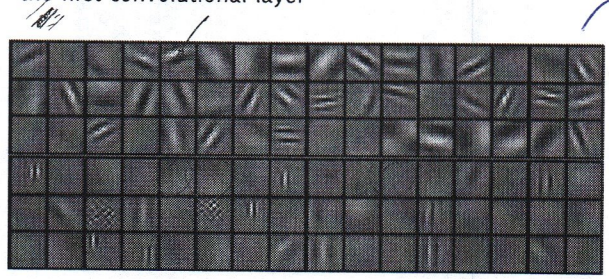
Visualizing CNN Filters

: What does a network have learned? Let's see the filters!



G. Boracchi

11X11X3 filters (visualized in RGB) extracted from the first convolutional layer

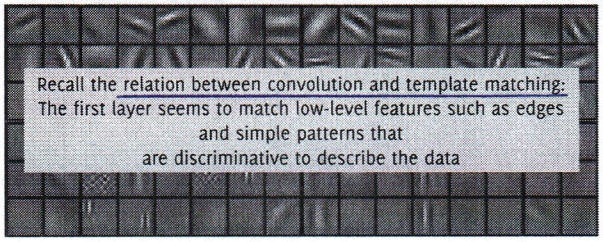


! Very simple patterns. Moreover (in the original PDF it can be better seen) they're gray scale (most of them). This means that we're treating all the colors the same. The patterns are related to the image intensity

Only some of the last are detecting some colors patterns

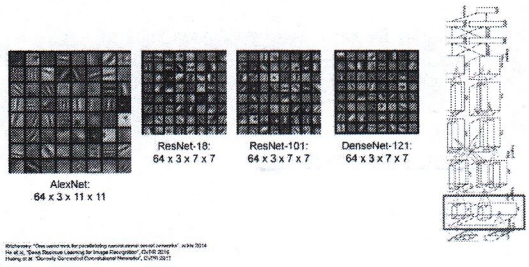
G. Boracchi

11x11x3 filters (visualized in RGB) extracted from the first convolutional layer



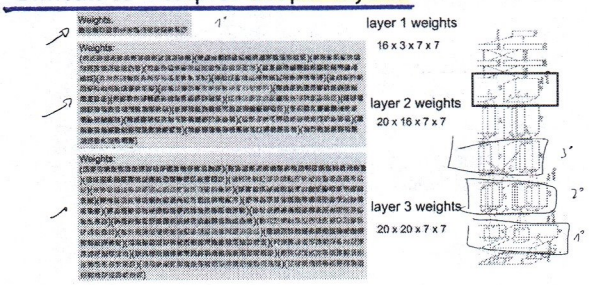
G. Boracchi

First layer's filters are often like these



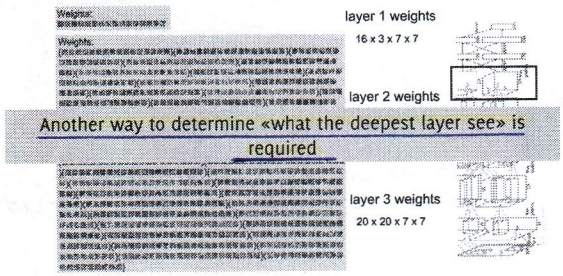
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Difficult to interpret deeper layers



CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

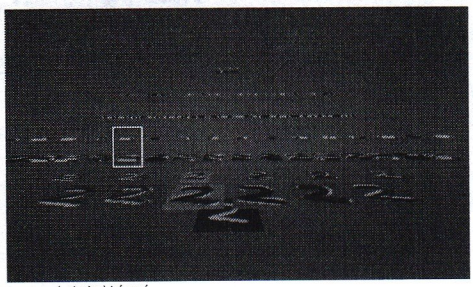
Difficult to interpret deeper layers



! The first filters are "easy" to be interpreted. As we go deeper the filters become more messed up.
(They become difficult to interpret)

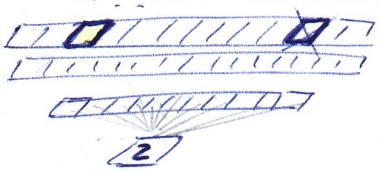
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

What if we look at the activations?

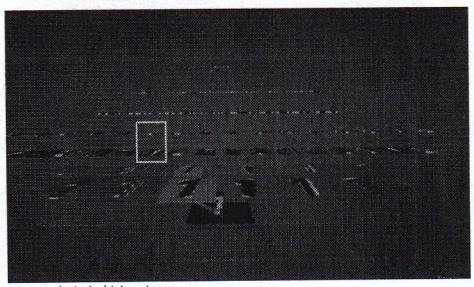


https://www.cs.ryerson.ca/~aharley/vis/conv/

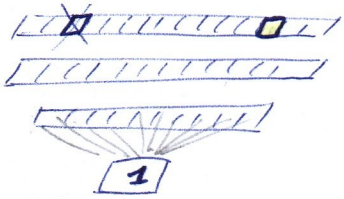
We can try to see the activations:



What if we look at the activations?



https://www.cs.ryerson.ca/~aharley/vis/conv/



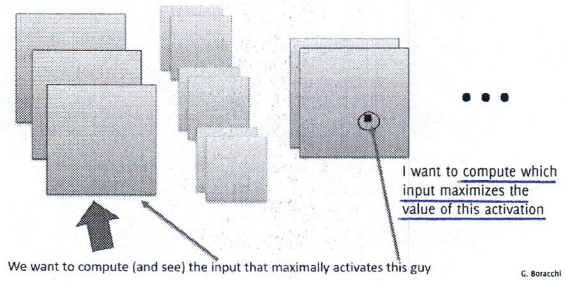
We can see for instance that if we have two different inputs (in the sense of two different labels) then different neurons become active. If we understand what make some specific neurons "excited" then we understand what the network has learned.

However this works mostly for simple networks, let's see an alternative!

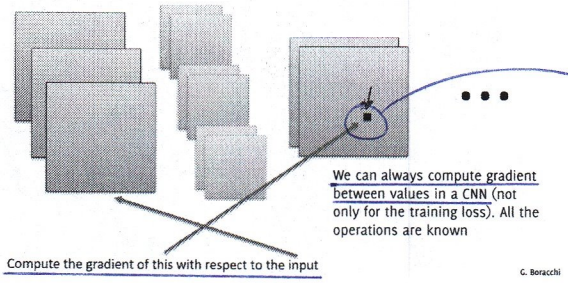
Alternative: (to the "personal interpretation" and to the "what the activation funct. does")

Visualizing the Input that maximally activates some neurons

this is something like the other way around. We look at one output neuron and we look at the input that maximizes it.
(The output neuron in that particular layer, not necessarily on the final output. We're trying to understand what the network is learning at each layer!)

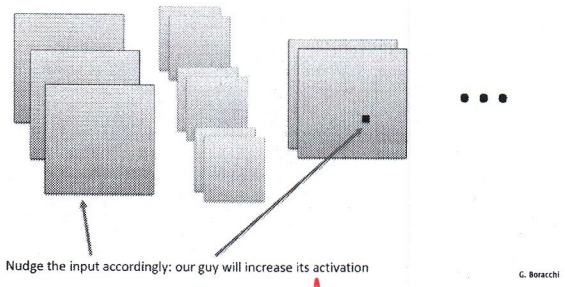


Step 1

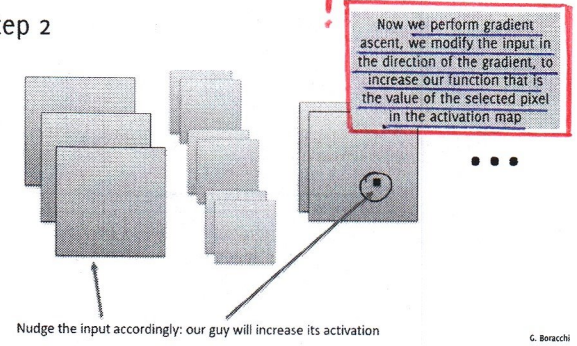


we compute the gradient of this pixel w.r.t. the input. We then vary the input and see how ■ responds.

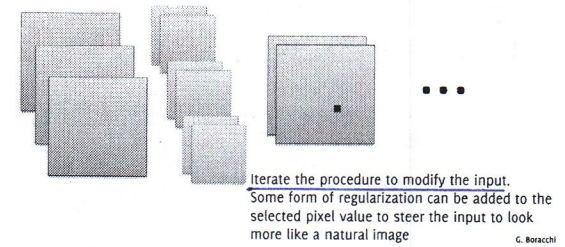
Step 2



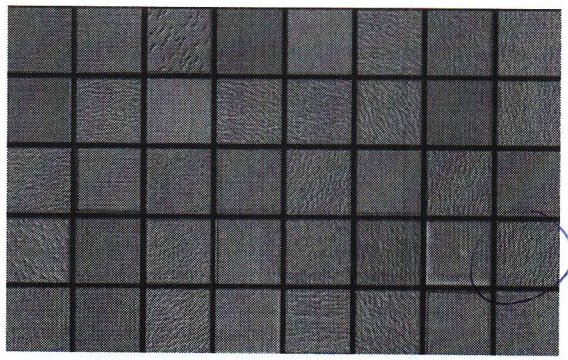
Step 2



Back to step 1 and iterate



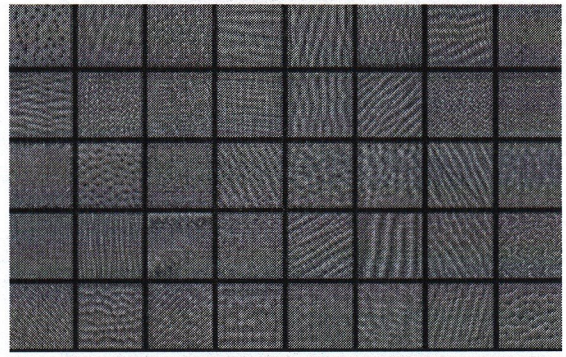
Shallow layers respond to fine, low-level patterns



these are the inputs that provide a response which is maxima in some activations for the first layers of the network.
(It's like detecting patterns)
E.g.

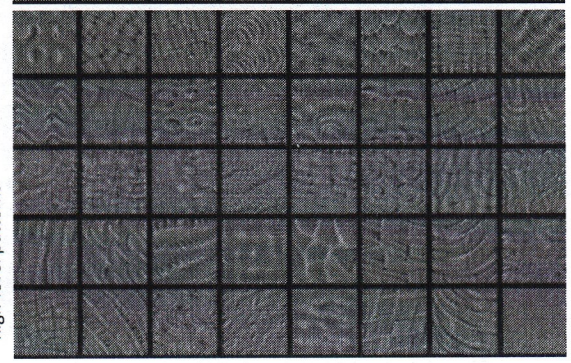
this is the input image which maximaly activates one neuron in the first layer of the network.

As we move deeper, we repeat the procedure (w try to maximize the activation in the second layer) →



Intermediate layers ...

(or the third layer) →



Deep layers respond to complex, high-level patterns

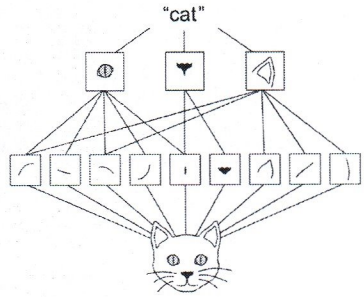
(..)

As we go deeper, we see that what maximaly activates a neuron has a very structured pattern. Details are increasing, high frequencies are popping out, etc.

the first layer(s) doesn't show a precise pattern. We detect patterns only at some stages.

Why CNNs work

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



Fully Convolutional Networks

— "basic ingredient" for image segmentation

What happens when feeding the network with an image having different size?

Fully convolutional Networks

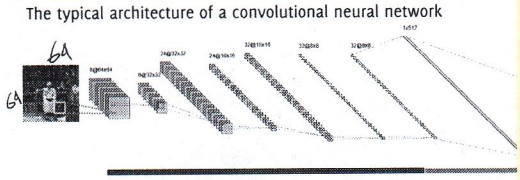
What happens if we feed the network with a bigger image? The convolution layers won't create issues, the fully connected layers will. Why? Because of a dimention problems.
The problem can be solved if we convolutionalize the fully connected layers (:= we see them as a convolution instead of a Feed Forward NN).
In this way, what we obtain is a FULLY CONVOLUTIONAL CNN.

CNN where the fully connected layers were moved to a set of 1x1x? convolutional filters *

* depends on the input size

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network



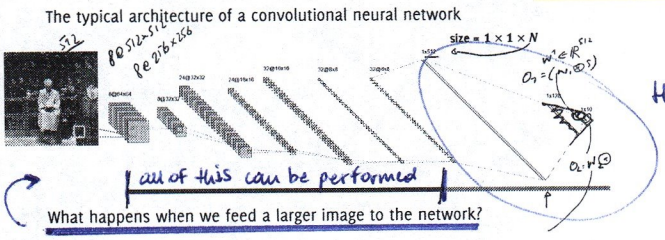
CNNs are meant to process input of a fixed size (e.g. 200 x 200).

The convolutional and subsampling layers operate in a sliding manr over image having arbitrary size

The fully connected layer constrains the input to a fixed size.

Convolutional Neural Networks (CNN)

The typical architecture of a convolutional neural network



Here we got a dimention problem

all of this can be performed

What happens when we feed a larger image to the network?

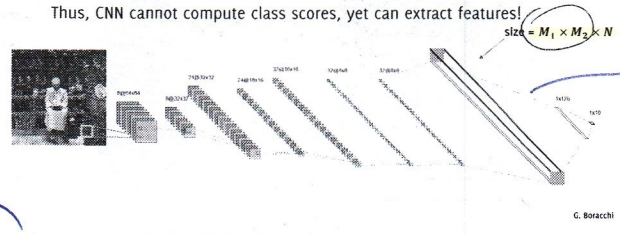
G. Boracchi

Convolutional Neural Networks (CNN)

Convolutional filters can be applied to volumes of any size, yielding larger volumes in the network until the FC layer.

The **FC network however does require a fixed input size**

Thus, CNN cannot compute class scores, yet can extract features!

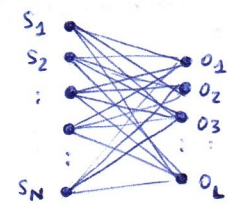


G. Boracchi

since we started with a bigger image here we get something that is no more $1 \times 1 \times N$ but something that still preserves a spatial resolution: $M_1 \times M_2 \times N$

(we're feeding to a fully connected layer an input which is $M_1 \times M_2$ times longer than the expected: this creates problems)

So the problem is not the convolutional part of the network, but the fully connected part



The output of the FC is:

$o_i = \sum_{j=1}^{N} w_j^i s_j + b_i$

and we can look at it as a convolution (we already saw that convolutions can be seen as feedforward NN, so we can proceed also in the other way).

$\sum_{j=1}^{N} w_j^i s_j$ can be seen as:

$$\underbrace{\boxed{w_1^i \cdots w_N^i}\ \boxed{\begin{matrix} s_1 \\ \vdots \\ s_N \end{matrix}}}$$

this is a LINEAR CLASSIFIER and a linear classifier can be interpreted as a kind of template-match, so we can write it down as a filter that performs convolution of the weights to the input.

$o_i = (\vec{w}_i \circledast \vec{s}_i)(0,0)^{*} + b_i$

what size is the filter?

$1 \times 1 \times N$, where N is the input size.

⇒ we can write the output of each fully connected layer as the convolution of the input against a filter whose weights are the weights of the network we learned.

⇒ we can convolutionalize the Fully connected network by moving the weights of the fully connected network into filters, which are size 1×1, and then the (whole) network can be applied also to process images which are of larger sizes.

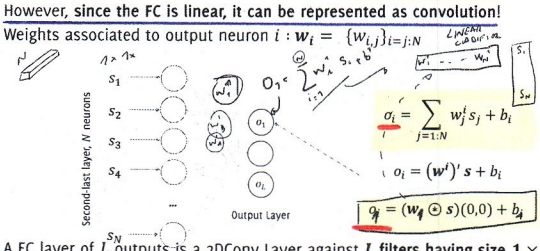
If we convolutionalize the fully connected layer then we can compute the output even if the input size varies.

* the (0,0) means that we're evaluating the convolution w.r.t. the center

Fully Convolutional Neural Networks (f-CNN)

However, **since the FC is linear, it can be represented as convolution!**

Weights associated to output neuron i : $\boldsymbol{w}_i = \{w_{i,j}\}_{i=j:N}$



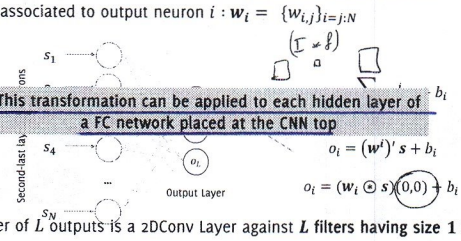
A FC layer of L outputs is a 2DConv Layer against L **filters having size** $1 \times 1 \times N$

G. Boracchi

Fully Convolutional Neural Networks (f-CNN)

However, **since the FC is linear, it can be represented as convolution!**

Weights associated to output neuron i : $\boldsymbol{w}_i = \{w_{i,j}\}_{i=j:N}$



This transformation can be applied to each hidden layer of a FC network placed at the CNN top

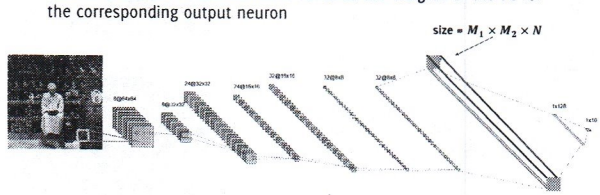
A FC layer of L outputs is a 2DConv Layer against L **filters having size** $1 \times 1 \times N$

G. Boracchi

Fully Convolutional Neural Networks (f-CNN)

However, since the FC is linear, it can be represented as convolution against L filters of size $1 \times 1 \times N$

Each of **these convolutional filters contains the weights of the FC** for the corresponding output neuron



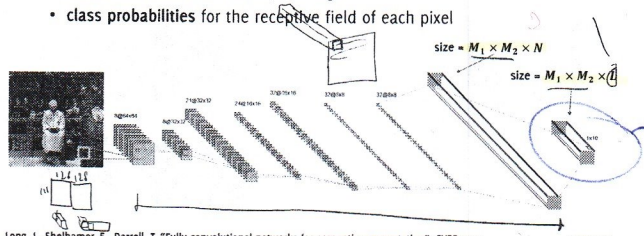
Long, J., Shelhamer, E., Darrell, T. "Fully convolutional networks for semantic segmentation". CVPR 2015

G. Boracchi

Fully Convolutional Neural Networks (f-CNN)

For each output class we obtain an image, having:

- **Lower resolution** than the input image
- **class probabilities** for the receptive field of each pixel



Long, J., Shelhamer, E., Darrell, T. "Fully convolutional networks for semantic segmentation". CVPR 2015

G. Boracchi

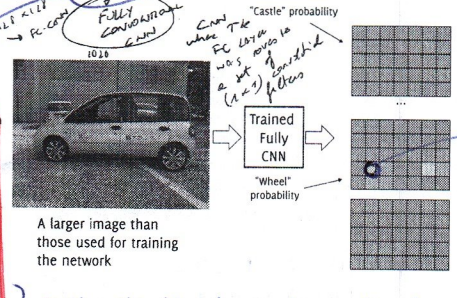
("however")

the output is $M_1 \times M_2 \times L$, not $1 \times 1 \times L$

(so the output are again images; we can these output images: HEATMAPS)

(How many heatmaps? # heatmaps = # classes)

Output of a FCNN as heatmaps



A larger image than those used for training the network

G. Boracchi

The value here is like the probability for the classifier to believe that in a certain region (like in the receptive field related to this value) there was an object of that class (indeed # heatmaps = # classes)

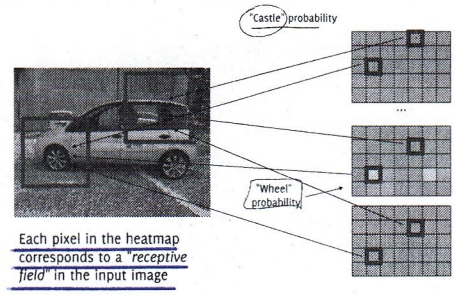
FC-CNN

Fully convolutional CNN

(CNN where the fully connected layer was moved to a set of 1×1 convolutional filters)

↳ the depth depends on the input size

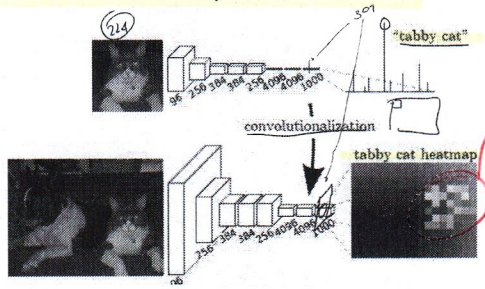
Output of a FCNN as heatmaps



G. Boracchi

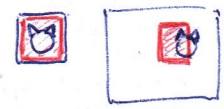
By these heatmaps (which are in very low resolution) we can get a hint in which region of the input image there are elements belonging to a certain class.

Migration to FCNN of a pretrained model



Long, Jonathan, *Evan* Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

we see a region and not a pixel because:

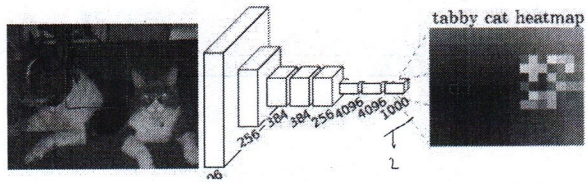


these two overlapped will not produce 0 correlation

Migration to FCNN of a pretrained model

This stack of convolutions operates on the whole image as a filter.

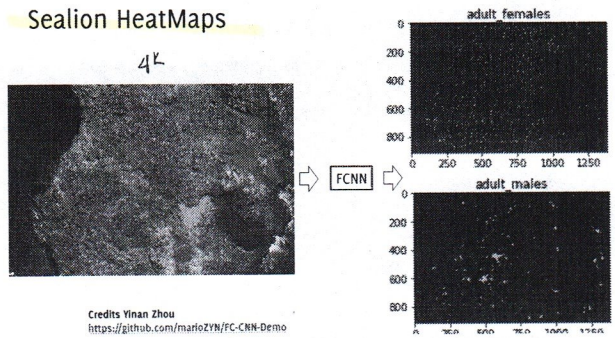
Significantly more efficient than patch extraction and classification (avoids multiple repeated computations within overlapping patches)



Long, Jonathan, *Evan* Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

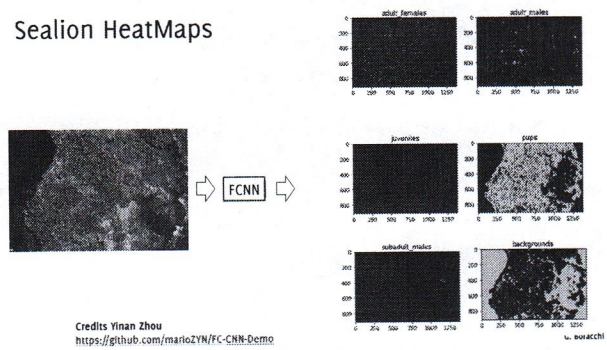
Sealion HeatMaps

4K



Credits Yinan Zhou
https://github.com/marioZYN/FC-CNN-Demo

Sealion HeatMaps



Credits Yinan Zhou
https://github.com/marioZYN/FC-CNN-Demo

G. Boracchi

Fully convolutional networks in keras

It is necessary to get and set the weights of networks by means of the methods `get_weights` and `set_weights`

- get the weights of the trained CNN

```
w7, b7 = model.layers[7].get_weights()
```

- reshape these weights to become ten 1x1 convolutional filters having depth equal to 256 (these sizes are dictated by the sizes of the dense layer in the "traditional" CNN

```
w7.reshape(1, 1, 256, 10)
```

- assign these weights to the FCNN architecture

```
model2.layers[7].set_weights(w7, b7)
```

↳ this has 10 filters that are 1x1

model 2 made of convolutions only (in particular we have 1x1xN convolutions instead of dense layers)

filters used = # neurons in the output layer of the original fully connected layers

G. Boracchi

Convolutional Neural Networks for Semantic Segmentation

Giacomo Boracchi
giacomo.boracchi@polimi.it

Semantic Segmentation Task

http://www.robots.ox.ac.uk/~szheng/crfasrnndemo

Zheng et al. *"Conditional Random Fields as Recurrent Neural Networks"*, ICCV 2015

Semantic Segmentation Task → tipically performed in a supervised setting

The goal of semantic segmentation is:

Given an image I, **associate to each pixel (r, c) a label** from Λ.

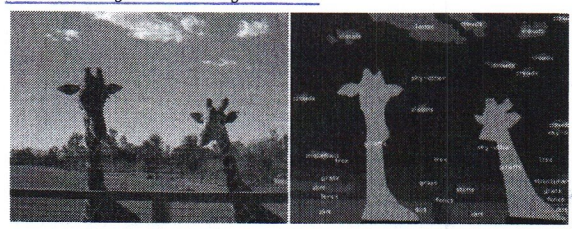
The result of segmentation is a map of labels containing in each pixel the estimated class.

Remark: In this image there is no distinction among persons.
Segmentation **does not separate different instances belonging to the same class.**
That would be instance segmentation.

http://www.robots.ox.ac.uk/~szheng/crfasrnndemo

Training Set

The training set is made of pairs (I, GT), where the GT is a pixel-wise annotated image over the categories in Λ



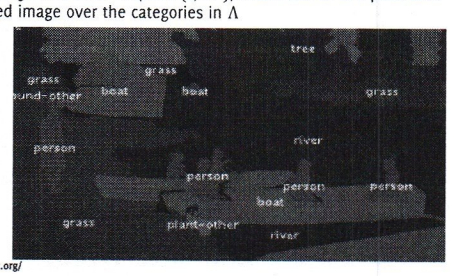
I GT

← each image has its corresponded annotated image

http://cocodataset.org/

Training Set

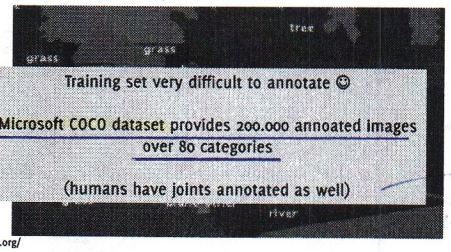
The training set is made of pairs (I, GT), where the GT is a pixel-wise annotated image over the categories in Λ



http://cocodataset.org/

Training Set

The training set is made of pairs (I, GT), where the GT is a pixel-wise annotated image over the categories in Λ



Training set very difficult to annotate ☺

Microsoft COCO dataset provides 200.000 annoated images over 80 categories

(humans have joints annotated as well)

There are provided the location (coordinates) of the joints (= key points)

http://cocodataset.org/

Semantic Segmentation by Fully Convolutional Neural Networks

Predicting dense outputs for arbitrary-sized inputs

G. Boracchi

This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

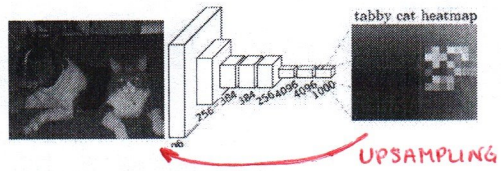
Fully Convolutional Networks for Semantic Segmentation

Jonathan Long* Evan Shelhamer* Trevor Darrell
UC Berkeley
{jonlong,shelhamer,trevor}@cs.berkeley.edu

G. Boracchi

Simple Solution (1): Direct Heatmap Predictions

We can assign the predicted label in the heatmap to the whole receptive field, however that would be a **very coarse estimate** (not good enough for a image segmentation)



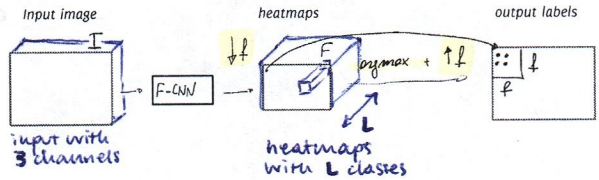
Long, Jonathan, *Evan* Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

We would like to keep a fully convolutional neural network for segmentation and identify an effective scheme in order to go back from the heatmap to the full-resolution image. (:= UPSAMPLING)

What is the most straightforward solution?

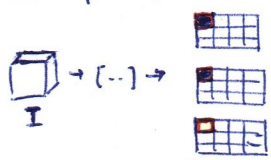
Simple Solution (1): Direct Heatmap Predictions

Very coarse estimates

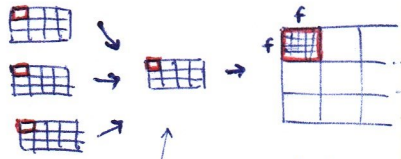


Solution 1

What we can do is:
when we get to the L final heatmaps, for every pixel we associate the label of the most probable class:



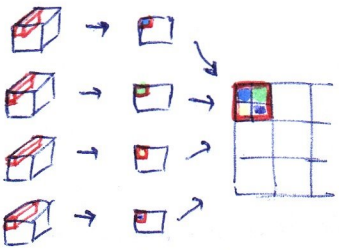
Here we have 3 classes (L=3). The red pixel is more probable to be of the 3rd class. To go back to the original dimension (the final output of this procedure will not be an image which has a size of the heatmap), if the heatmap has dimensions of the input image (I) reduced by f, the output will be the final heat-map-dimension image (with labels) augmented by f:



If the ▫ in is of label "3", then the final output will have a f×f square of pixels of label "3" in this way the final output will have the same size of the original input image.

Solution 2

In the solution 1 we produced only one set of heatmaps for the input image I (set of L heatmaps). Instead of just one set of heatmaps, we can compute f^2 heatmaps by shifting the input. The resulting f^2 sets of heatmaps will be different since the input image is different (shifted). In the example we have a down-sampling of 2, so we produce 4 heatmaps sets. Then, for every heatmap set we get one image (heatmap-size) with argmax and then we ensemble.



However, this is still not optimal. (There is no learning in the way we aggregate the outputs)

Simple Solution (2): The Shift and Stich

Shift and Stich: Assume there is a ratio f between the size of input and of the output heatmap

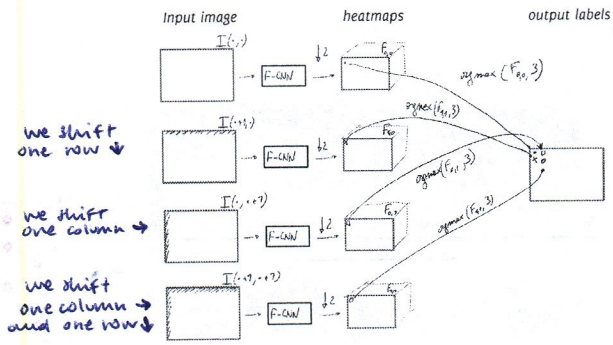
- Compute **heatmaps** for all f^2 possible shifts of the input ($0 \le r, c < f$)
- **Map predictions from the** f^2 heatmaps to the image: each pixel in the heatmap **provides prediction of the central pixel of the receptive field**
- **Interleave the heatmaps** to form an image as large as the input

This **exploits the whole depth of the network**

Efficient implementation through the à trous algorithm in wavelet

However, **the upsampling method is very rigid**

Long, Jonathan, *Evan* Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015



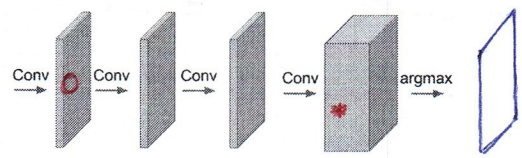
G. Boracchi

What about having a network which is fully convolutional and never reduces the spatial extent of the volume?
(NO maxpooling, no stride and padding that mantains the right sizes)
In this way, after the argmax we get an image already of the size of the image input.

Simple Solution (3): Only Convolutions

What if we avoid any pooling (just conv2d and activation layers)?

- Very small receptive field
- Very inefficient



In this way however, when we get in * the receptive field is not very large (O). This is not good for semantic segmentation purpose

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Drawbacks of convolutions only

On the one hand **we need to "go deep" to extract high level information** on the image

On the other hand **we want to stay local not to loose spatial resolution** in the predictions

Semantic segmentation faces an inherent tension between semantics and location:

- ***global information resolves what**, while*
- ***local information resolves where***

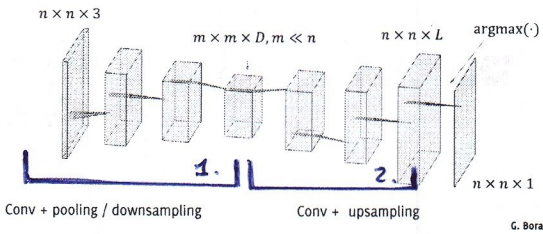
***Combining fine layers and coarse layers** lets the model make local predictions that respect global structure.*

How can we obtain all the informations? (↓)

Long, Jonathan, *Evan* Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

Reduce latent representation dimension

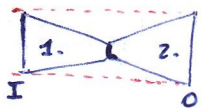
An architecture like the following would probably be more suitable for semantic segmentation



G. Boracchi

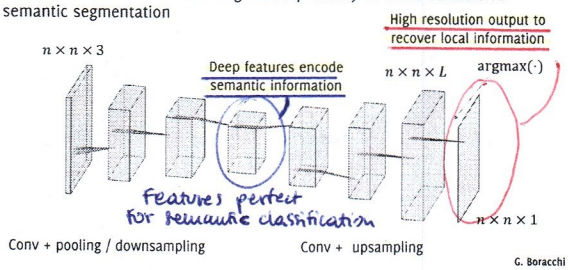
1. contractive part of the network (maxpooling and downsampling operations). From here we get the lobal informations
2. we recover the original spatial resolution (we increase the spatial extent, possibly reducing the depth)

This is a kind of bottleneck architecture:



Reduce latent representation dimension

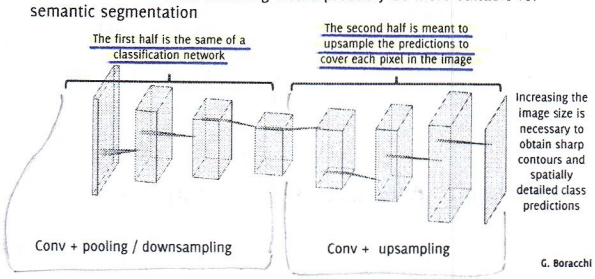
An architecture like the following would probably be more suitable for semantic segmentation



G. Boracchi

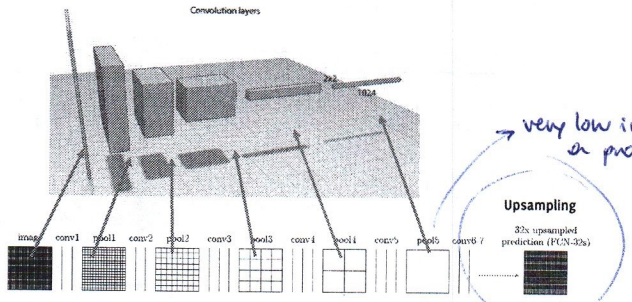
Reduce latent representation dimension

An architecture like the following would probably be more suitable for semantic segmentation



G. Boracchi

downsample = reduce the spatial extent
upsample = augment the spatial extent



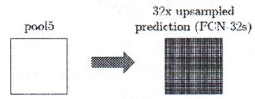
very low in resolution. How do we perform a proper upsampling to get back to a 32x image?

We take always the same size of the image but we change the resolution of the grid to indicate the depth where the slices are from

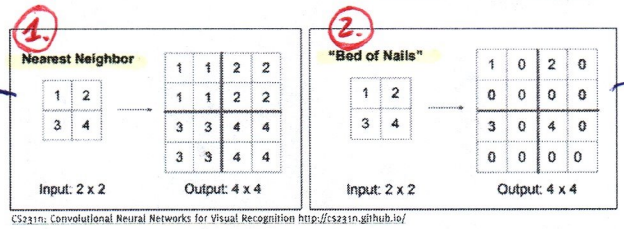
Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

(4 approaches)

How to perform upsampling?



based on replications, not so good

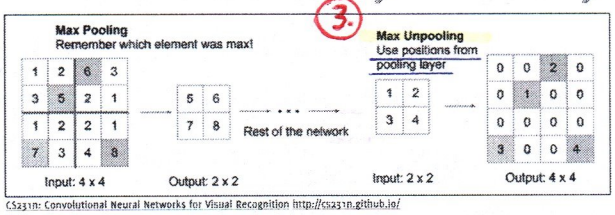


still not effective

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Max Unpooling

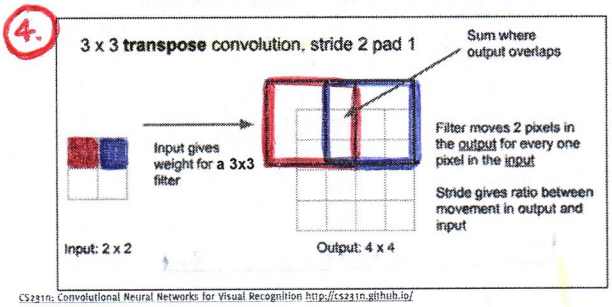
You have to keep track of the locations of the max during maxpooling



CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

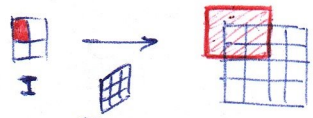
We can see this as a case where the filter is longer than the input.

Transpose Convolution



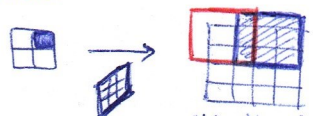
CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

We take a filter that is 3x3, which has its own coefficients, and we overlap it to the input image (which in this case is 2x2) and we compute the output even if the filter cannot be included (entirely). What's happening is that the input image is giving weights for the filter values. The red square in the image (I, 2x2) multiply the whole filter and we obtain



this red is the filter scaled by the red pixel of the input

Then:



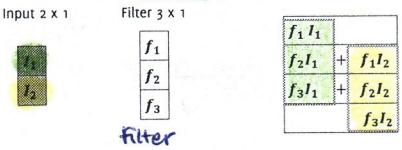
this blue is the filter scaled by

In the overlapped zone we sum up the values.

One dimensional example:

Transpose Convolution

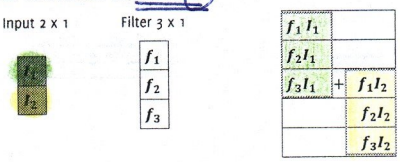
Transpose convolution with stride 1



G. Boracchi

Transpose Convolution

Transpose convolution with stride 2



The stride determines the output size.

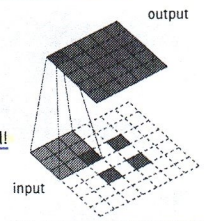
G. Boracchi

How to perform upsampling?

Transpose Convolution can be seen as a traditional convolution after having upsampled the input image

Many names for transpose convolution: fractional strided convolution, backward strided convolution, deconvolution (very misleading!!!)

Upsamping based on convolution gives more degrees of freedom, since the filters can be learned!



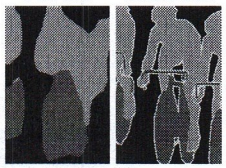
https://github.com/vdumoulin/conv_arithmetic

Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." *arXiv preprint arXiv:1603.07285* (2016).

Prediction Upsampling

this is what we get with this approach

FCN-32s | Ground truth



this is what we would like to get

How can we include more details?

(↓)

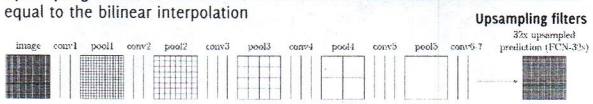
Linear upsampling of a factor f can be implemented as a convolution against a filter with a fractional stride $1/f$.

Upsampling filters can thus be learned during network training.

These **predictions** however are **very coarse**

Upsampling filters are learned with initialization equal to the bilinear interpolation

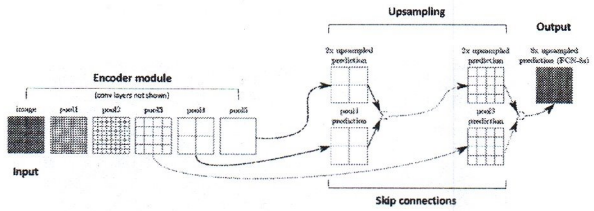
Upsampling filters



Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

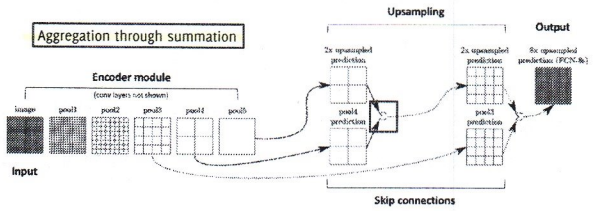
Solution: Skip Connections!

Why we do this? Adding pools we're adding deeper informations! (informations which have global meaning)



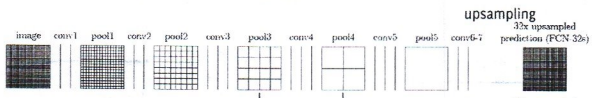
https://www.jeremyjordan.me/semantic-segmentation/

Solution: Skip Connections!

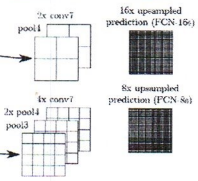


https://www.jeremyjordan.me/semantic-segmentation/

Solution: Skip Connections!

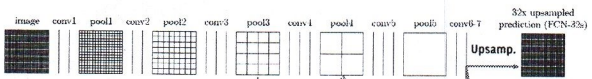


- Supplement a traditional «contracting» network by successive layers **where convolution is replaced by transpose convolution**
- Upsampling is ubiquitous
- **Upsampling filters are learned** during training
- Upsampling filters are **initialized using bilinear interpolation**

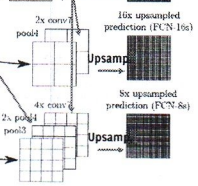


Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

Solution: Skip Connections!

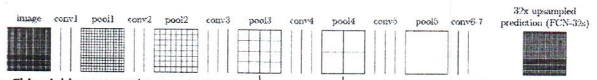


- Supplement a traditional «contracting» network by successive layers **where convolution is replaced by transpose convolution**
- Upsampling is ubiquitous
- **Upsampling filters are learned** during training
- Upsampling filters are **initialized using bilinear interpolation**

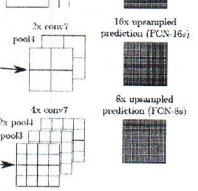


Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

Solution: Skip Connections!



- **This yields 3 networks.** *Combining fine layers and coarse layers lets the model make local predictions that respect global structure.*
- Train first the lowest resolution network (FCN-32s)
- Then, the weights of the next network (FCN-16s) are initialized with (FCN-32s)
- The same for FCN-8s
- All the FC layers are converted to convolutional layers 1x1



Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

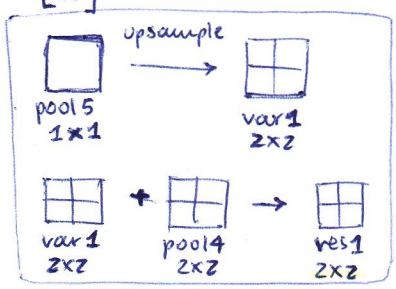
Instead of doing the entire upsample from pool5 (1x) to the 32x, we can:

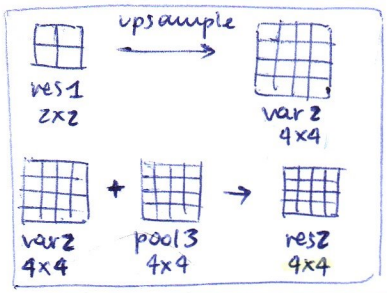
we have:

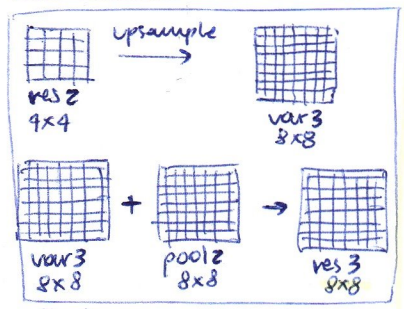
pool 5 → 1x1
pool 4 → 2x2
pool 3 → 4x4
pool 2 → 8x8
pool 1 → 16x16
image → 32x32

- we do the upsample for pool5, we go from 1x1 to 2x2 (not like before from 1x1 to 32x32).
- When we have the upsampled pooled 5, we have something of the dimensions of pool4, and so we sum them up.
- now we do the upsample of (pooled4 + upsample(pooled5)) and we go from 2x2 to 4x4
- the result is of the size of pool3 and so we sum the result and pool3.
- we upsample the previous sum from 4x4 to 8x8, obtaining *
- now * has the size of pool2, so we sum * and pool2
- we perform upsampling of * + pool2 and we go from 8x8 to 16x16

[..]





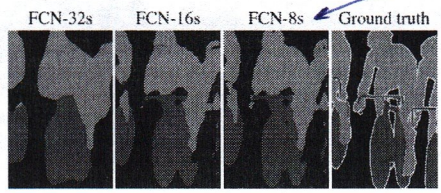


[..]

Semantic Segmentation

Retaining intermediate information is beneficial, the deeper layers contribute to provide a better refined estimate of segments

This new procedure gets closer to what we want to obtain

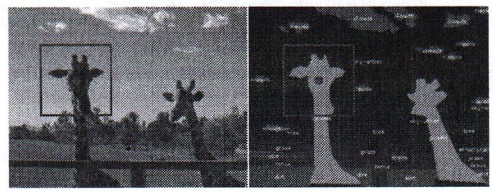


Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

How do we train the network?

FCNN Training Options: the «patch-based» way

- Prepare a training set for a classification network
- Crop as many patches x_i from annotated images and assign to each patch, the label corresponding to the patch center



FCNN Training Options

The «patch-based» way:

- Prepare a training set for a classification network
- Crop as many patches x_i from annotated images and assign to each patch, **the label corresponding to the patch center**
- **Train a CNN for classification** from scratches, or fine tune a pre-trained model over the segmentation classes
- Convolutionalization: once trained the network, **move the FC layers to 1x1 convolutions**
- **Design the upsampling side of the network** and train these filters

G. Boracchi

FCNN Training Options

The «patch-based» way:

- The classification network is trained to minimize the classification loss ℓ over a mini-batch

$$\hat{\theta} = \min_{\theta} \sum_{x_j} \ell(x_j, \theta)$$

where x_j belongs to a mini-batch

- Batches of patches are **randomly assembled during training**
- **It is possible to resample patches for solving** class imbalance
- It is **very inefficient**, since convolutions on overlapping patches are repeated multiple times

G. Boracchi

FCNN Training Options: The «full-image» way

Since the network provide dense predictions, it is possible to directly train a FCNN that includes upsampling layers as well

Learning becomes:

$$\min_{x_j \in I} \sum_{x_j} \ell(x_j, \theta)$$

Where x_j **are all the pixels in a region of the input image** and the loss is evaluated over the corresponding labels in the annotation. Therefore, each patch provides already a mini-batch estimate for computing gradient.

G. Boracchi

FCNN Training Options

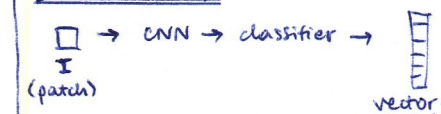
The «full-image» way:

- FCNN are trained in an end-to-end manner to predict the segmented output $S(\cdot,\cdot)$
- This loss is the sum of losses over different pixels. Derivatives can be easily computed through the whole network, and this can be trained through backpropagation
- No need to pass through a classification network first
- Takes **advantage of FCNN efficiency**, does not have to re-compute convolutional features in overlapping regions

G. Boracchi

Patch-based way

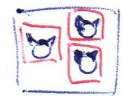
Standard way:



and then we convolutionize the classifier (from FC layers to 1x1 convolutions, so that we can take any image as input)

However this is not the most efficient way for proceeding.

Why? We train the network over **PATCHES** (:= subsection of the input image), more precisely:



From this input image we extract some patches (the cat patches) and we use these patches as inputs for the CNN.

This may be not efficient if, for example, some of the patches overlap. We would like to train the CNN over the **whole image** input and not over some extracted patches.

If we have an image of a dog and a cat, for example, we would like to put directly it to the CNN as input, instead of creating two patches: dog, cat.

(If we procede with the whole image as the CNN input we're doing the "full-image" way)

FCNN Training Options

?

(but we're not entering into details of the training over the full image)

Limitations of full-image training and solutions:

- Minibatches in patch-wise training are assambled randomly. Image regions in full-image training are not. To make the estimated loss a bit stochastic, adopt random mask

$$\text{minimize} \sum_{x_j} M(x_j)\ell(x_j, \theta)$$

being $M(x_j)$ a binary random variable

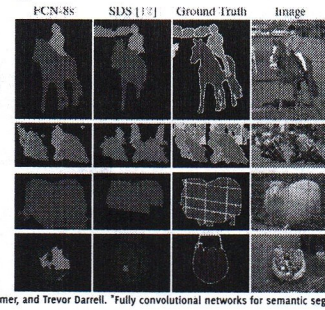
- It is not possible to perform patch resampling to compensate for class imbalance. One should go for weighting the loss over different labels

$$\text{minimize} \sum_{x_j} w(x_j)\ell(x_j, \theta)$$

being $w(x_j)$ a weight that takes into account the true label of x_j

we can give different weights to different classes in such a way that we give more relevance to classification errors of the under-represented classes to compensate the class imbalance

Semantic Segmentation Results



Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." CVPR 2015

Comments

- Both learning and inference can be performed on the whole-image-at-a-time
- Both in full-image or batch-training it is possible to **perform transfer learning/fine tuning** of **pre-trained classification models** (segmentation typically requires fewer labels than classification)
- **Accurate pixel-wise prediction** is achieved by upsampling layers
- **End-to-end training is more efficient** than patch-wise training
- Outperforms state-of the art in 2015
- Being fully convolutional, this network **handles arbitrarily sized input**

U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOSS Centre for Biological Signalling Studies,
University of Freiburg, Germany
ronneber@informatik.uni-freiburg.de,
WWW home page: http://lmb.informatik.uni-freiburg.de/

U-Net

Network formed by:

- A contracting path
- An expansive path

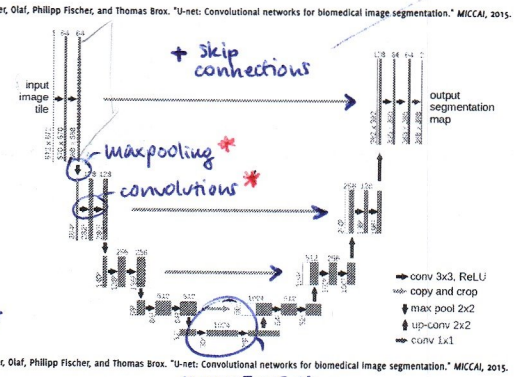
No fully connected layers

Major differences w.r.t. (long et al. 2015):

- use **a large number of feature channels** in the upsampling part, while in (long et al. 2015) there were a few upsampling. The network become symmetric
- **Use excessive data-augmentation** by applying elastic deformations to the training images

skip connections are bringing again informations from the low layers of the network (= low semantic high resolution content of the network) to the corresponding level of the expanding path

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, 2015.



contractive path
(goal: extract meaningful features)

expansive path

* convolutions increase the depth, maxpoolings reduce the spatial extent

(subsampling + convolutional layers)

BOTTLENECK:
from here: upsampling

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, 2015.

U-Net: Contracting path

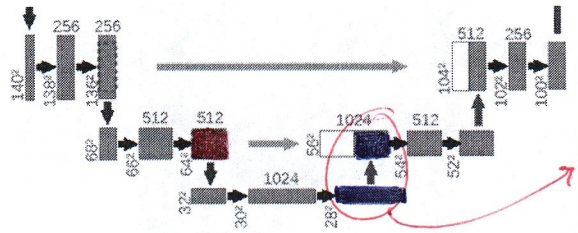
Repeats blocks of:

- 3 × 3 convolution + ReLU ('valid' option, no padding)
- 3 × 3 convolution + ReLU ('valid' option, no padding)
- Maxpooling 2 × 2

At each downsampling the number of feature maps is doubled

G. Boracchi

U-Net: Skip connections



Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, 2015.

when we do the up-conv we use less filters (decrease the depth) if we had 28 x 28 x 1024, after the up-conv we have 56 x 56 x 512

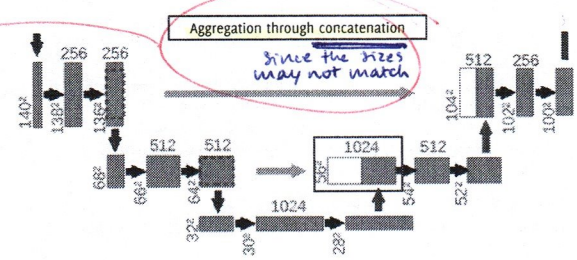
(↓ depth, ↑ spatial extent)

When we obtain the we mix it with the corresponding level of the contractive path by using the convolutional filters. (transpose convolution)

So we always decrease the depth and increase the spatial extent (going ↑) but still, while decreasing the depth, we are recovering part of the informations from the previous layers (through the skip connections). Notice that, since the regions don't always match, we recover only part of the information (e.g. is 64 x 64 x 512, while is 56 x 56 x 512)

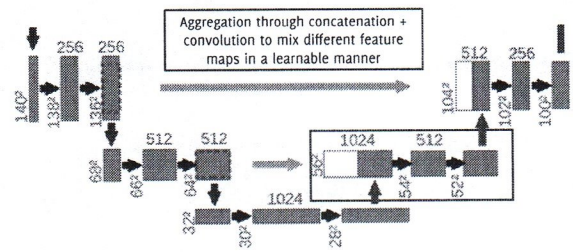
U-Net: Skip connections

In the other case (some slides ago) when we were doing the skip connections we summed the informations connected (only summed). Here we're concatenating them. (transpose convolution)



Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, 2015.

U-Net: Skip connections



Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, 2015.

U-Net: Expanding path

Repeats blocks of:

- 2 × 2 transpose convolution, halving the number of feature maps (but doubling the spatial resolution)
- Concatenation of corresponding cropped features
- 3 × 3 convolution + ReLU
- 3 × 3 convolution + ReLU

} Aggregation during upsampling

U-Net: Network Top

No fully connected layers: there are L convolutions against filters $1 \times 1 \times N$, to yield predictions out of the convolutional feature maps

Output image is smaller than the input image by a constant border

(however we can get the original size output if we want)

U-Net: Training

Full-image training by a weighted loss function

$$\hat{\theta} = \min_{\theta} \sum_{x_j} w(x_j)\ell(x_j, \theta)$$

where the weight

$$w(x) = w_c(x) + w_0\, e^{-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2}}$$

- w_c is used to balance class proportions (remember no patch resampling in full-image training)
- d_1 is the distance to the border of the closest cell
- d_2 is the distance to the border of the second closest cell

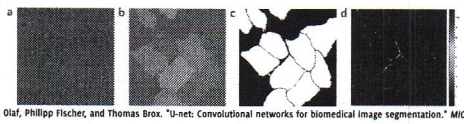
U-net: Training

Full-image training by a weighted loss function

$$w(x) = w_c(x) + w_0\, e^{-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2}}$$

- w_c is used to balance class proportions (remember no patch resampling in full-image training)
- d_1 is the distance to the border of the closest cell
- d_2 is the distance to the border of the second closest cell

Weights are large when the distance to the first two closest cells is small



Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, 2015.

U-net: Training

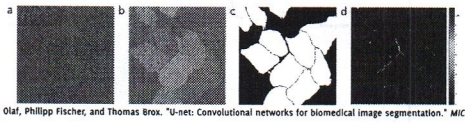
Full-image training by a weighted loss function

$$w(x) = \boxed{w_c(x)} + \boxed{w_0\, e^{-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2}}}$$

Takes into account class unbalance in the training set — (w_c is higher when the class is under-represented)

Enhances classification performance at borders of different objects — (takes into account how critical the situation is: how the pixel is in between two different objects (if the pixel is on the boundary of two different objects))

true division



Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, 2015.

U-net: Training

This term is large at pixels close to borders delimiting objects of different cells

Full-image training by a weighted loss function

$$w(x) = w_c(x) + \boxed{w_0\, e^{-\frac{(d_1(x)+d_2(x))^2}{2\sigma^2}}}$$

true division

we have larger weights in regions that are close to two different cells / categories

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *MICCAI*, 2015.

Global Averaging Pooling

G. Boracchi

Network In Network

Min Lin[1,2], Qiang Chen[2], Shuicheng Yan[2]
[1]Graduate School for Integrative Sciences and Engineering
[2]Department of Electronic & Computer Engineering
National University of Singapore, Singapore
{linmin,chenqiang,eleyans}@nus.edu.sg

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

Network in Network : two innovations :

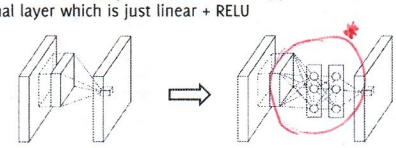
- Mlpconv layers (not very successful)
- Global averaging pooling layers

MULTILAYER PERCEPTRON CONVOLUTIONAL layers (Mlpconv layers)

Mlpconv layers: instead of traditional convolutions, **a stack of 1x1 convolutions + RELU**

- 1x1 convolutions used in a stack followed by RELU corresponds to a MLP networks **used in a sliding manner on the whole image**

Each layer features a **more powerful functional approximation** than a convolutional layer which is just linear + RELU



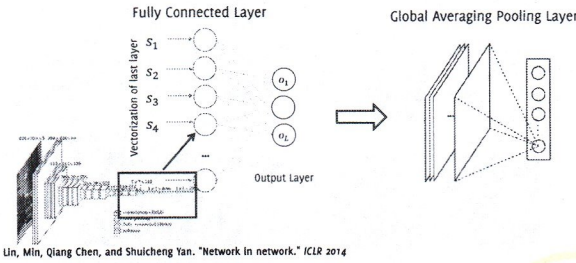
(a) Linear convolution layer (b) Mlpconv layer

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

The idea is: convolution (which is just a linear combination with learned weights in an image region plus a non-linearity) is not general enough. It'll be better to use something else: a multilayer perceptron*. This will provide more non-linear features as output of each layer of the network.

Network in Network

They introduce Global Averaging Pooling Layers

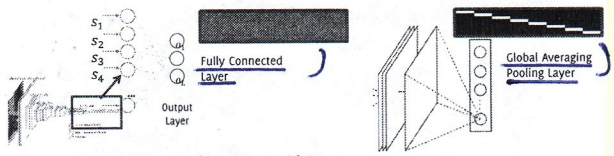


Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

Network in Network: GAP

Global Averaging Pooling Layers: instead of a FC layer at the end of the network, **compute the average of each feature map.**

- The transformation corresponding **to GAP is a block diagonal, constant matrix** (consider the input unrolled layer-wise in a vector)
- The transformation of each layer in **MLP corresponds to a dense matrix.**



Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

Rationale behind GAP

Fully connected layers are prone to overfitting

- They have many parameters
- Dropout was proposed as a regularized that randomly sets to zero a percentage of activations in the FC layers during training

The GAP strategy is:

- **Remove the fully connected layer** at the end of the network!
- **Predict by a simple soft-max after the GAP.**
- **Watch out: the number of feature maps has to be equal to the number of output classes!**

G. Boracchi

The Advantages of GAP Layers:

- No parameters to optimize, lighter networks less prone to overfitting
- More interpretability, creates a direct connection between layers and classes output (we'll see soon)
- This makes GAP a structural regularizer (to prevent overfitting)
- More robustness to spatial transformation of the input images
- **The network can be used to classify images of different sizes**
- Classification is performed by a softMax layer at the end of the GAP

Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." ICLR 2014

Network in Network

The whole NiN stacks

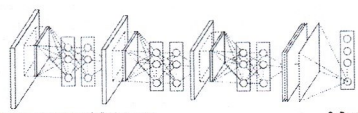
- mlpconv layers (RELU) + dropout
- Maxpooling

} A few layers of these

- Global Averaging Pooling (GAP) layer
- Softmax

} At the end of the network

simple NiNs achieve state-of-the-art performance on «small» datasets (CIFAR10, CIFAR100, SVHN, MNIST) and that **GAP effectively reduces overfitting** w.r.t. FC



Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *ICLR 2014*

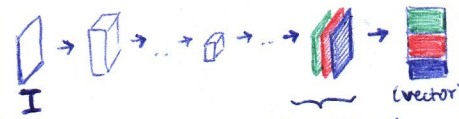
G. Boracchi

GAP

One of the major drawbacks of the most powerful architectures is the large number of coefficients spent on the fully connected layer at the end. This very large fully connected layer make the network inclined to overfitting.

⟹ The idea is to replace all the fully connected layers (at the end of a CNN) simply by a global averaging pooling layer (GAP layer).

The GAP layer computes the average of each slice of the volume and place it in a vector:



when we get at the end we average every slice (and then we perform the SOFTMAX)

Of course we need the number of the slices to be exactly the number of classes.

In this way we're avoiding all the fully connected layer part. We go from the last convolutional part of the network to the softmax (directly through this GAP).

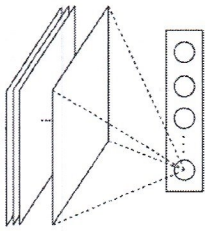
We can see GAP as spatial average-tor.

We're using just the CNN to learn the classificator (getting rid of a huge amount of parameters). In this way we prevent the network to do overfitting.

The Global Averaging Pooling (GAP) Layer

We indeed see that GAP is acting as a (structural) regularizer

Method	Testing Error
mlpconv + Fully Connected	11.59%
mlpconv + Fully Connected + Dropout	10.88%
mlpconv + Global Average Pooling	10.41%



Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400v3* (2014). G. Boracchi

Weakly-Supervised Localization

... Global Averaging Pooling Revisited

G. Boracchi

Weakly supervised localization

Perform localization over an image without images with annotated bounding box

- Training set provided as for classification with image-label pairs (I, ℓ) where no localization information is provided

G. Boracchi

This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Learning Deep Features for Discriminative Localization

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba
Computer Science and Artificial Intelligence Laboratory, MIT
{bzhou,khosla,agata,oliva,torralba}@csail.mit.edu

G. Boracchi

The GAP revisited

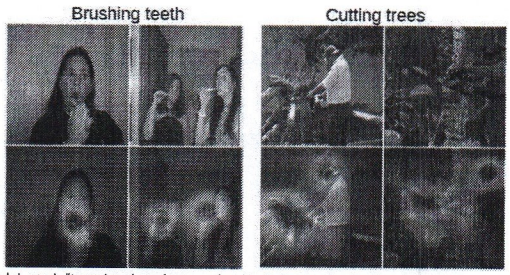
The advantages of GAP layer **extend beyond** simply acting as a structural **regularizer** that prevents overfitting

In fact, the **network can retain a remarkable localization ability** until the final layer. By a simple tweak it is possible to easily identify the discriminative image regions leading to a prediction.

*A **CNN trained on object categorization** is successfully able to **localize the discriminative regions for action classification** as the objects that the humans are interacting with rather than the humans themselves*

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping



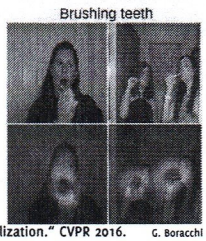
Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016. G. Boracchi

Class Activation Mapping (CAM)

Identifying exactly **which regions** of an image **are being used for discrimination.**

CAM are very easy to compute. It just requires:

- FC layer after the GAP
- a minor tweak

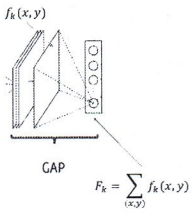


Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016. G. Boracchi

The Global Averaging Pooling (GAP) Layer

A very simple architecture made only of convolutions and activation functions leads to a final layer having:

- n feature maps $f_k(\cdot,\cdot)$ having resolution "similar" to the input image
- a vector after GAP made of n averages F_k



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016. G. Boracchi

The Global Averaging Pooling (GAP) Layer

Add (and train) a **single FC layer** after the GAP.

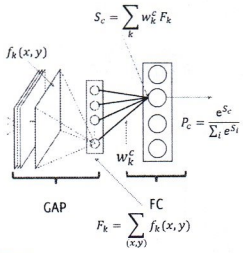
The FC computes S_c for each class c as the weighted sum of $\{F_k\}$, where weights are defined during training

Then, the class probability P_c via soft-max (class c)

Remark: when computing

$$S_c = \sum_k w_k^c F_k$$

w_k^c encodes the importance of F_k for the class c.



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

The Global Averaging Pooling (GAP) Layer

However

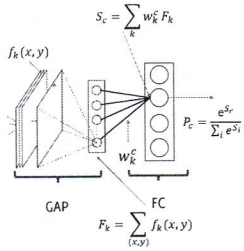
$$S_c = \sum_k w_k^c \sum_{x,y} f_k(x,y) = \sum_{x,y} \boxed{\sum_k w_k^c f_k(x,y)}$$

And CAM is defined as

$$M_c(x,y) = \sum_k w_k^c f_k(x,y)$$

where $M_c(x,y)$ directly indicates the importance of the activations at (x,y) for predicting the class c

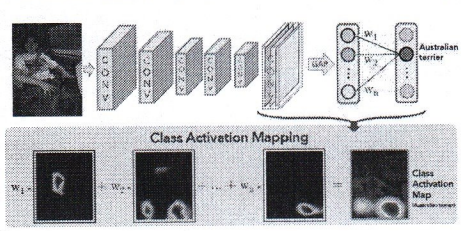
Thanks to the softmax, the depth of the last convolutional volume can be different from the number of classes



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

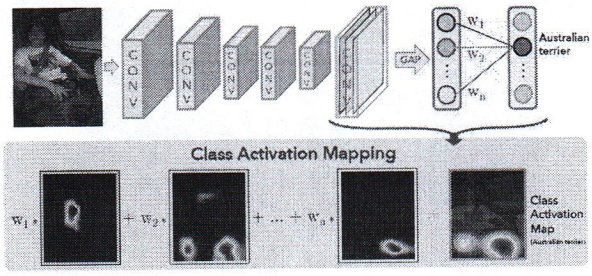
Class Activation Mapping

Now, the weights represents the importance of each feature map to yield the final prediction. Upsampling might be necessary to match the input image



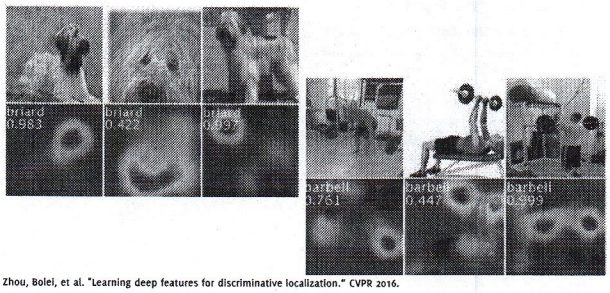
Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

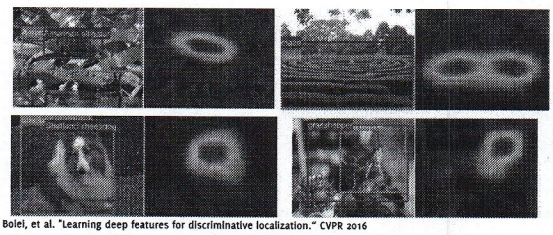
Remarks

- CAM can be **included in any pre-trained network**, as long as all the FC layers at the end are removed
- The **FC used for CAM is simple**, few neurons and no hidden layers
- **Classification performance might drop** (in VGG removing FC means loosing 90% of parameters)
- **CAM resolution** (localization accuracy) can improve by **«anticipating»** GAP to larger convolutional feature maps (but this reduces the semantic information within these layers)
- **GAP**: encourages the **identification of the whole object**, as all the parts of the values in the activation map concurs to the classification
- **GMP (Global Max Pooling)**: it is enough to have a high maximum, thus **promotes specific discriminative features**

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

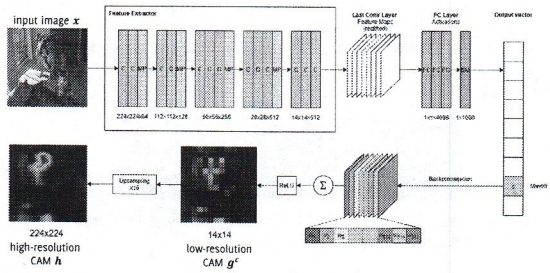
Weakly Supervised Localization

Use thresholding CAM values: > 20% max(CAM), then take the largest componet of the thresholded map (green GT, red estimated location)



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016

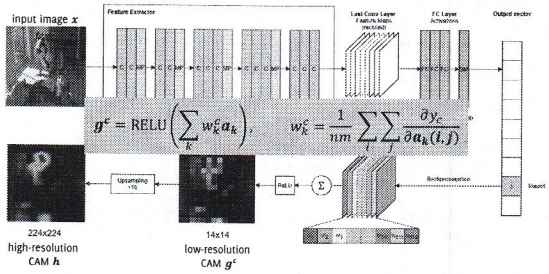
Grad-CAM and CAM-based techniques



Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV 2017*

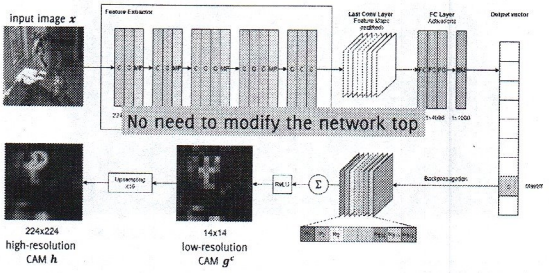
Type equation here.

Grad-CAM and CAM-based techniques



Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV2017*

Grad-CAM and CAM-based techniques



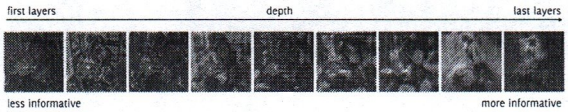
Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV2017*

Heatmaps Desiderata

Should be **class discriminative**

Should **capture fine-grained details** (high-resolution)

- This is critical in many applications (*e.g.* medical imaging, industrial processes)



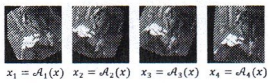
G. Boracchi

Augmented Grad-CAM

We consider the augmentation operator $\mathcal{A}_l: \mathbb{R}^{N\times M} \rightarrow \mathbb{R}^{N\times M}$, including random rotations and translations of the input image $\boldsymbol{x}$

Augmented Grad-CAM: increase heat-maps resolution through image augmentation

All the responses that the CNN generates to the **multiple augmented versions of the same input image** are very informative for reconstructing the high-resolution heat-map $\boldsymbol{h}$

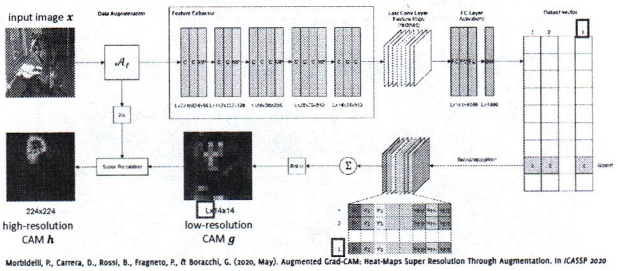


$x_1 = \mathcal{A}_1(x)$ $x_2 = \mathcal{A}_2(x)$ $x_3 = \mathcal{A}_3(x)$ $x_4 = \mathcal{A}_4(x)$

g_1 g_2 g_3 g_4

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

Augmented Grad-CAM

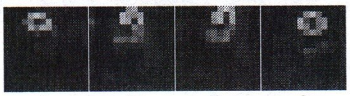


Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

The Super-Resolution Approach

We perform heat-map Super-Resolution (**SR**) by taking advantage of the information shared in multiple low-resolution heat-maps computed from the **same input under different - but known - transformations**

CNNs are in general invariant to roto-translations, in terms of predictions, but each $\boldsymbol{g}_\ell$ actually contains different information



g_1 g_2 g_3 g_4

General approach, our SR framework can be combined with any visualization tool (not only Grad-CAM)

G. Boracchi

The Super-Resolution Formulation

We model heat-maps computed by Grad-CAM as the result of an unknown downsampling operator $\boldsymbol{\mathcal{D}}: \mathbb{R}^{N\times M} \rightarrow \mathbb{R}^{n\times m}$

The high-resolution heat-map $\boldsymbol{h}$ is recovered by solving an inverse problem

$$\underset{h}{\operatorname{argmin}} \frac{1}{2} \sum_{l=1}^{L} \|\mathcal{D}\mathcal{A}_\ell h - g_\ell\|_2^2 + \lambda TV_{\ell_1}(h) + \frac{\mu}{2}\|h\|_2^2 \quad (1)$$

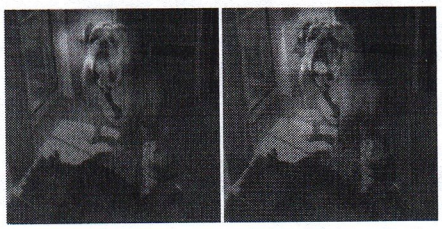
TV_{ℓ_1}: Anistropic Total Variation regularization is used to preserve the edges in the target heat-map (high-resolution)

$$TV_{\ell_1}(\boldsymbol{h}) = \Sigma_{i,j} \|\partial_x \boldsymbol{h}(i,j)\| + \|\partial_y \boldsymbol{h}(i,j)\| \quad (2)$$

This is solved through Subgradient Descent since the function is convex and non-smooth

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G "Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation". ICASSP 2020

Augmented Grad-CAM



(a) Grad-CAM. (b) Augmented Grad-CAM.

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

Other popular architectures

Outline

Lecture inspired to:

Notes accompanying the Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

... and papers cited in here!

This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Going Deeper with Convolutions

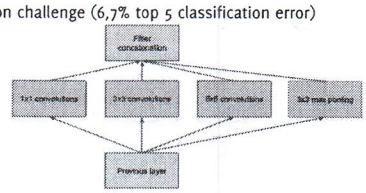
Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,draqomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

GoogLeNet and Inception v1 (2014)

Deep network, with **high computational efficiency**

Only **5 million parameters**, 22 layers of Inception modules

Won 2014 ILSVR-classification challenge (6,7% top 5 classification error)

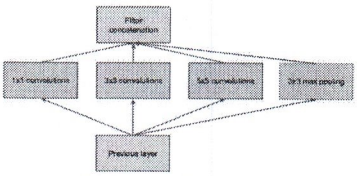


(a) Inception module, naïve version

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

GoogLeNet and Inception v1 (2014)

It is based on inception modules, which are sort of «networks inside the network» or «local modules»

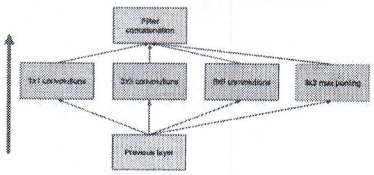


(a) Inception module, naïve version

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

Inception Module (2014)

- Choosing the **right kernel size** for the convolution operations is difficult, as image might show relevant features at different scale
- Deep networks **tend to over fit** and to become very computationally expensive

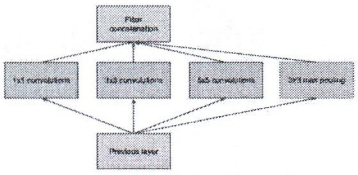


(a) Inception module, naïve version

Szegedy et al. "Going deeper with convolutions." *CVPR 2015.*

Inception Module (2014)

The solution is to **exploit multiple filter size** at the same level (1x1, 3x3, 5x5) and then **merge by concatenation** the output activation maps together



(a) Inception module, naïve version

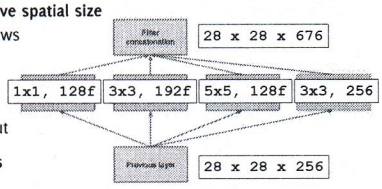
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Inception Module (2014)

The solution is to exploit multiple filter size at the same level (1x1, 3x3, 5x5) and then **merge by concatenation** the output activation maps together

- **Zero padding to preserve spatial size**
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



(a) Inception module, naïve version

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

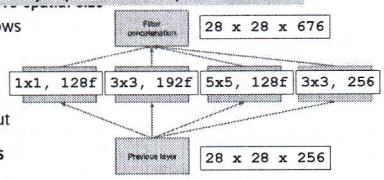
G. Boracchi

Inception Module (2014)

The sol 3x3, 5x5) an togethe

The spatial extent is preserved, but the depth of the activation map is much expanded. This is very expensive to compute

- Zero
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



(a) Inception module, naïve version

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

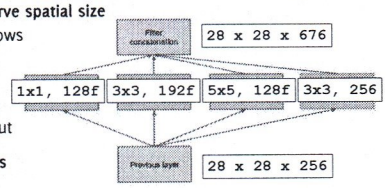
G. Boracchi

Inception Module (2014)

The sol 3x3, 5x5) an togethe

Computational problems will get significantly worst when stacking multiple layers...

- **Zero padding to preserve spatial size**
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



(a) Inception module, naïve version

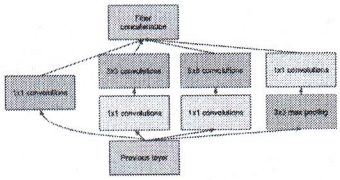
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Inception Module (2014)

To reduce the computational load of the network, the number of **input channels** is reduced by adding an **1x1 convolution** layers before the 3x3 and 5x5 convolutions

Using these 1x1 conv is referred to as **"bottleneck" layer**

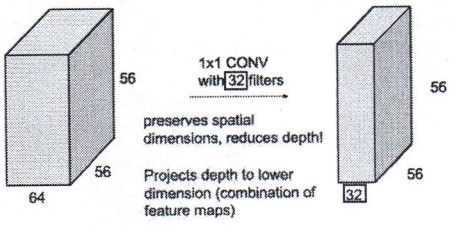


(b) Inception module with dimension reductions

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

1x1 convolution layers as bottleneck

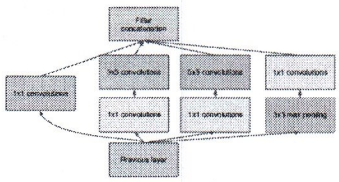


Inception Module (2014)

To reduce the computational load of the network, the number of **input channels** is reduced by adding an **1x1 convolution** layers before the 3x3 and 5x5 convolutions

The output volume has similar size, but the number of operation required is significantly reduced due to the 1x1 conv:
358M operations now

Adding 1x1 convolution layers increases the number of nonlinearities



(b) Inception module with dimension reductions

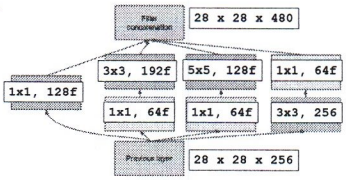
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Inception Module (2014)

To reduce the computational load of the network, the number of **input channels** is reduced by adding an **1x1 convolution** layers before the 3x3 and 5x5 convolutions

The output volume has similar size, but the number of operation required is significantly reduced due to the 1x1 conv:
358M operations now



(b) Inception module with dimension reductions

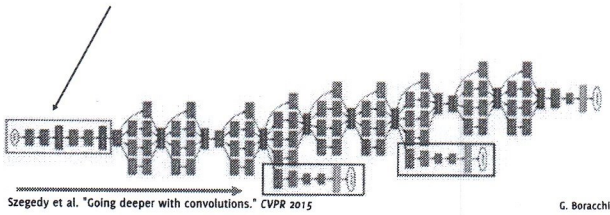
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

GoogLeNet (2014)

GoogleNet stacks many inception modules: 27 layers considering pooling ones.

At the beginning there are two blocks of conv + pool layers



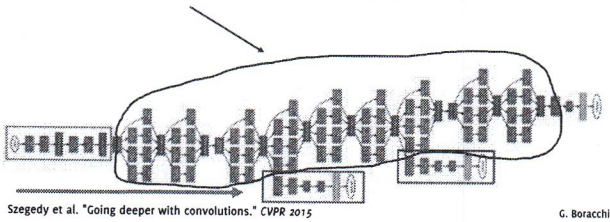
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

GoogLeNet (2014)

GoogleNet stacks many inception modules: 27 layers considering pooling ones.

Then, there are a stack of 9 of inception modules



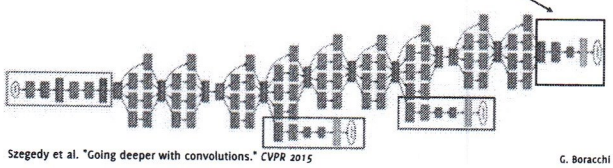
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

GoogLeNet (2014)

GoogleNet stacks many inception modules: 22 layers considering pooling ones.

No Fully connected layer at the end, simple averaging pooling, linear classifier + softmax



Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

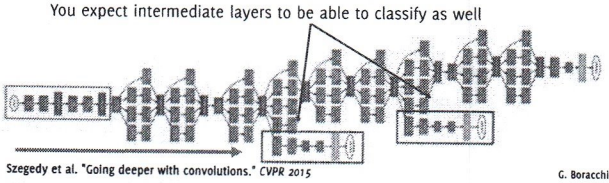
G. Boracchi

GoogLeNet (2014)

GoogleNet stacks many inception modules: 27 layers considering pooling ones.

It also suffers of the **dying neuron problem**, therefore the authors **add two extra classifiers** on the intermediate representation **to compute an intermediate loss that is used during training.**

You expect intermediate layers to be able to classify as well



Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Convolutional Neural Networks for Detection

Giacomo Boracchi
Advanced Neural Networks and Deep Learning
https://boracchi.faculty.polimi.it/

Localization

we want to predict the location of a certain object inside an image
⇒ **output**: label + box surrounding the object

The Localization Task

The input image contains a single relevant object to be classified in a fixed set of categories

The task is to:

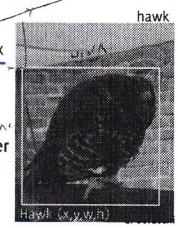
1) assign the object class to the image
2) locate the object in the image by its bounding box

A training set of annotated images with **label and a bounding box** around each object is required

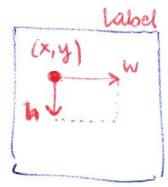
Extended localization problems involve regression over more complicated geometries (e.g. human skeleton)

(regression because we predict the location of a certain object)

(x, y, h, w, label)

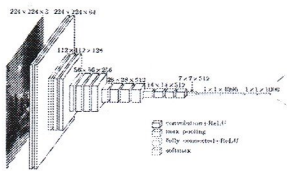


For each image, as output we have:
(x, y, h, w, label):



The Simplest Solution

Train a network to predict both the class label and the bounding box



Target class
Soft-max loss $\mathcal{S}$

$\| [\hat{x}, \hat{y}, \hat{w}, \hat{h}] - [x, y, w, h] \|_2$

Bounding Box Coordinates
(x, y, w, h)
Regression loss $\mathcal{R}$, e.g. the $\ell^2, \ell^1, ..$ bounding box.

We may have 2 errors:
- target class
- bounding box coordinates

(In the simple image classification we have only one possible error: the target class)

error evaluated as:
$\underbrace{\| [\hat{x}, \hat{y}, \hat{w}, \hat{h}]}_{\text{predicted}} - \underbrace{[x, y, w, h]}_{\text{correct}} \|_{2/1}$

The Simplest Solution

The training loss has to be a single scalar since **we compute gradient of a scalar function** with respect to network parameters.

So one tend to **minimize a multitask loss to merge two losses:**

$$\mathcal{L}(x) = \alpha\, \mathcal{S}(x) + (1 - \alpha)\mathcal{R}(x)$$

and α is an hyper parameter of the network.

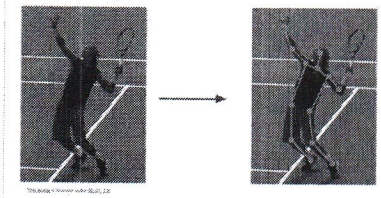
Watch out that α **directly influences the loss definition**, so tuning might be difficult, better to do cross-validation looking at some other loss.

It is also possible to **adopt a pre-trained model** and then train the two FC separately... however it is always better to perform at least some fine tuning to train the two jointly.

simple combination of the two loss functions (target class and bounding box coordinates)

Extension to Human Pose Estimation

Pose estimation is formulated as a **CNN-regression problem towards body joints.**



Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

28 - dimensional regression problem

Alexander Toshev and Christian Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks", CVPR 2014

what is "cool" is that the network doesn't change w.r.t. a classification network. What we have to change w.r.t. a classification network is just the final layer to perform regression instead of classification and the annotations we fit in the network. So, in principle the network could be the same: the same network used for classification can be used to a completely different purpose (like human pose estimation) just by defining different target and different loss.

Extension to Human Pose Estimation

Pose estimation is formulated as a **CNN-regression problem towards body joints.**

- **The network receives as input the whole image, capturing the full-context of each body joints.**
- The approach is **very simple to design and train.** Training problems can be **alleviated by transfer learning** of existing classification networks

Pose is defined as a vector of k joints location for the human body, possibly normalized w.r.t. the bounding box enclosing the human

Train a CNN to predict **a $2k$ vector as output** by using an Alexnet-like architecture

Alexander Toshev and Christian Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks", CVPR 2014

Training Human Pose Estimation Networks

Adopt a ℓ^2 regression loss of the estimated pose parameters over the annotations.

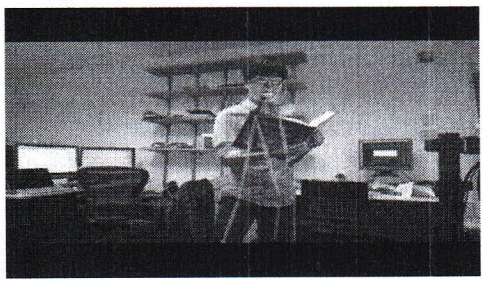
- This can be also defined when a few joints are not visible

Reduce overfitting by augmentation (translation and flips)

Multiple networks have been trained to improve localization by refining joint localtions in a crop around the previous detection

Alexander Toshev and Christian Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks", CVPR 2014

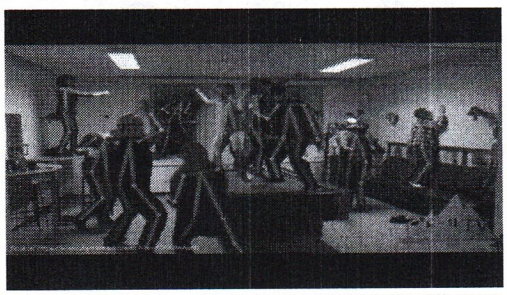
Open Pose



Cao, Z et al.. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. CVPR 2017

regression problem performed frame by frame

Pose Estimation



Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. Realtime multi-person 2d pose estimation using part affinity fields. CVPR2017

Weakly-Supervised Localization

... Global Averaging Pooling Revisited

... visualizing what matters most for CNN predicitons

G. Boracchi

Weakly supervised localization

Perform localization over an image without images with annotated bounding box

- Training set provided as for classification with image-label pairs $\{(I, \ell)\}$ where no localization information is provided

For training the network is getting only images and labels, without bounding boxes

G. Boracchi

This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore

Learning Deep Features for Discriminative Localization

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba
Computer Science and Artificial Intelligence Laboratory, MIT
{bzhou,khosla,agata,oliva,torralba}@csail.mit.edu

— global average pooling

G. Boracchi

The GAP revisited

The advantages of GAP layer **extend beyond** simply acting as a structural **regularizer** that prevents overfitting

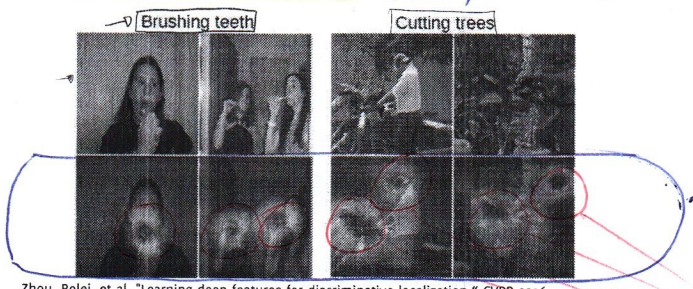
In fact, the **network can retain a remarkable localization ability** until the final layer. By a simple tweak it is possible to easily identify the discriminative image regions leading to a prediction.

*A **CNN trained on object categorization** is successfully able to **localize the discriminative regions for action classification** as the objects that the humans are interacting with rather than the humans themselves*

GAP gives informations on what made the network to take a certain decision, so which portion of the image leads the network to make a certain decision. So GAP are used also to get an explaination of the network decisions.

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping (CAM)



here are underlined the parts of the images that influenced most the network decision (these are the parts that led the net to a decision)

the highlighted are the peculiar features that the network has learned and use to determine that label

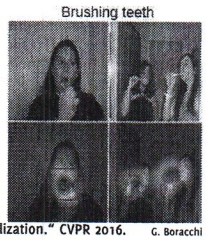
Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016. G. Boracchi

Class Activation Mapping (CAM)

Identifying exactly **which regions** of an image **are being used for discrimination.**

CAM are very easy to compute. It just requires:

- FC layer after the GAP
- a minor tweak



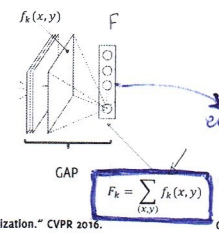
Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016. G. Boracchi

The Global Averaging Pooling (GAP) Layer

A very simple architecture made only of convolutions and activation functions leads to a final layer having:

- n feature maps $f_k(\cdot,\cdot)$ having resolution "similar" to the input image
- a vector after GAP made of n averages F_k

we obtain localization as a consequence of searching the region of the input that contributes at most for the classification



each element is an average

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016. G. Boracchi

The Global Averaging Pooling (GAP) Layer

FC (fully connected)

Add (and train) a **single FC layer** after the GAP.

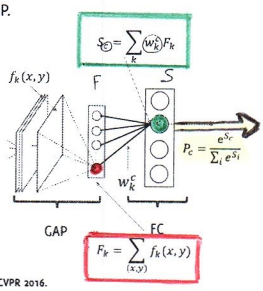
The FC computes S_c for each class c as the weighted sum of $\{F_k\}$, where weights are defined during training

Then, the class probability P_c via soft-max (class c)

Remark: when computing

$$S_c = \sum_k w_k^c F_k$$

w_k^c encodes the importance of F_k for the class c.



S_c = probability score for each class

P_c = probability via soft-max

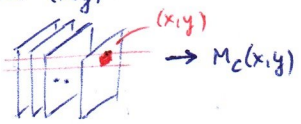
the highest the score, the highest the probability of belonging to a class. This score is a weighted sum of F_k.

($k = k^{th}$ slice of the GAP)

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

$$S_c = \sum_k w_k^c F_k = \sum_k w_k^c \left(\sum_{(x,y)} f_k(x,y) \right) = \sum_{(x,y)} \left(\sum_k w_k^c f_k(x,y) \right) = \sum_{(x,y)} M_c(x,y)$$

$M_c(x,y)$ is like the sum of all the features' maps for a given pixel (x,y):



slices of GAP

The larger $M_c(x,y)$ is, the larger is the probability of (x,y) to be in the c class.

The Global Averaging Pooling (GAP) Layer

However

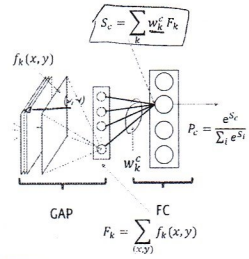
$$S_c = \sum_k w_k^c \sum_{(x,y)} f_k(x,y) = \sum_{x,y} \sum_k w_k^c f_k(x,y)$$

And CAM is defined as

$$M_c(x,y) = \sum_k w_k^c f_k(x,y)$$

where $M_c(x,y)$ directly indicates the importance of the activations at (x,y) for predicting the class c

Thanks to the softmax, the depth of the last convolutional volume can be different from the number of classes



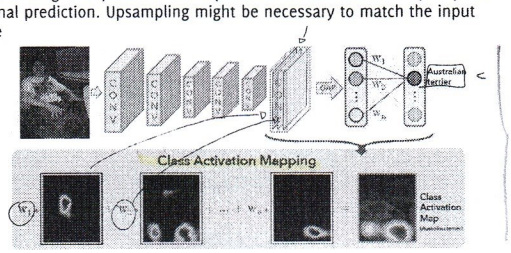
Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

F_k = spatial sum over the k-th slice of the GAP

⇒ S_c is the linear combination of all these F_k

Class Activation Mapping

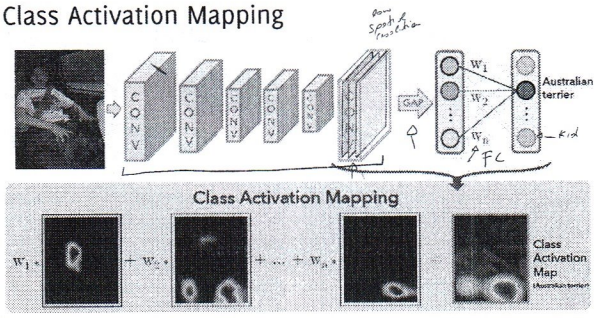
Now, the weights represents the importance of each feature map to yield the final prediction. Upsampling might be necessary to match the input image



better with colors

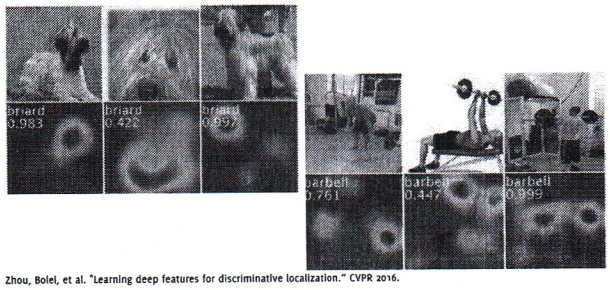
Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping



the number is the confidence of the classifier

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Remarks

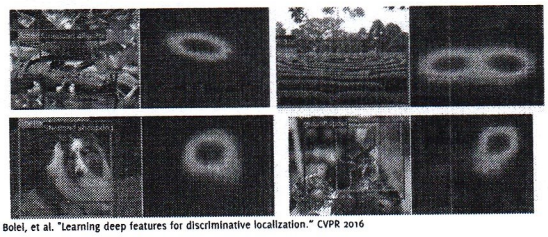
- CAM can be **included in any pre-trained network**, as long as all the FC layers at the end are removed
- The **FC used for CAM is simple**, few neurons and no hidden layers
- **Classification performance might drop** (in VGG removing FC means loosing 90% of parameters)
- **CAM resolution** (localization accuracy) can improve by **«anticipating»** GAP to larger convolutional feature maps (but this reduces the semantic information within these layers)
- **GAP**: encourages the **identification of the whole object**, as all the parts of the values in the activation map concurs to the classification
- **GMP (Global Max Pooling)**: it is enough to have a high maximum, thus **promotes specific discriminative features**

less good than GAP

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Weakly Supervised Localization

Use thresholding CAM values: > 20% max(CAM), then take the largest componet of the thresholded map (green GT, red estimated location)



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016

Weakly supervised localization:

Localization & CAMs

Localization:

CNN → GAP → FC

How to pass from the simple classification to a localization?

For classification we have CNN + FC (where FC classify). For the localization we can simply take the CNN + FC of the classification problem, we remove the FC at the end, we place a GAP and a new FC, then we re-train the network to perform CLASSIFICATION and we get FOR FREE a way to create these CAMs (class activation maps (the blue-yellow-red maps)), which tell us which regions are mostly contributing to get the classification output.

(

The GAP layer allow the localization throug the creation of CAMs.

! **Notice**: we're not training networks to have heatmaps, we train networks for CLASSIFICATION.
(CAMS ≠ HEATMAPS)

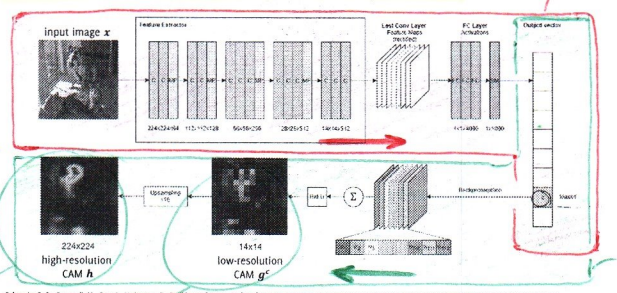
CAMs have as major drawback that they have to change the network architecture (we have to get rid of the FC layer). Modifying the network, we're making the network less powerful to perform classification.

⟹ There is an other way of performing CAMs which does not involve the architecture modification!

It's called: **GRAD-CAM**

original classifier (CNN + FC)

Grad-CAM and CAM-based techniques



Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV 2017*

Type equation here.

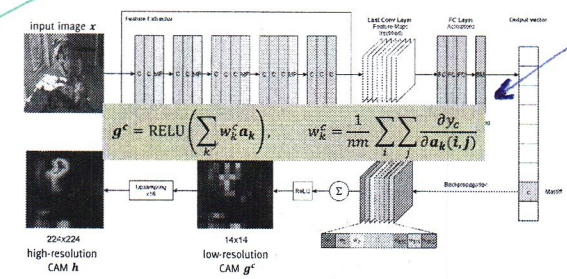
We define the CAM starting from the output of the original classification network: we take the class we're interested in ("MASTIFF") from the last FC and we compute a few gradients

CAMs constructor

Grad-CAM and CAM-based techniques

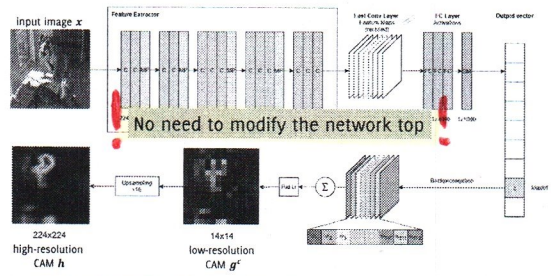
low-resolution CAM

smoothed CAM



Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV2017*

Grad-CAM and CAM-based techniques



Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV2017*

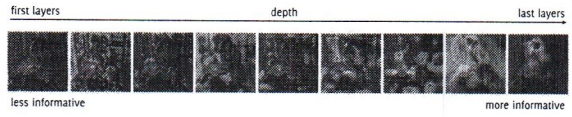
Heatmaps Desiderata

intended as CAMs (not heatmaps intended as alternative to the classification)

Should be **class discriminative**

Should **capture fine-grained details** (high-resolution)

- This is critical in many applications (*e.g.* medical imaging, industrial processes)



Notice: we can get CAMs from each CNN layer. If we get a CAM from the very last CNN layer (like we did before) we get a very informative CAM but with very low resolution. If we obtain the CAM from the first layers we obtain CAMs of very high resolution but not very informative.

G. Boracchi

Augmented Grad-CAM

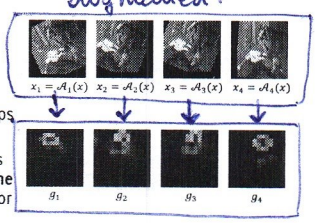
include augmentation as a form to improve the CAM resolution (↓)

We consider the augmentation operator $\mathcal{A}_i: \mathbb{R}^{N\times M} \to \mathbb{R}^{N\times M}$, including random rotations and translations of the input image $\boldsymbol{x}$

Augmented Grad-CAM: increase heat-maps resolution through image augmentation

All the responses that the CNN generates to the **multiple augmented versions of the same input image** are very informative for reconstructing the high-resolution heat-map $\boldsymbol{h}$

same image but augmented:



each image provides an estimate (CAM)

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

part 1

image

↓

augmentation:

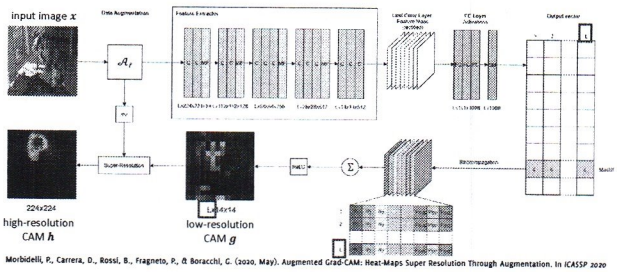
↓

CNN + FC (for each one): we obtain a "last" FC for each augmented image:

↓

we select a class and we create a CAM for each augmented image *

Augmented Grad-CAM



Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

part 2

* Notice: we know the transformation we did for augmentation (we're able to go back)

↓

we obtain a CAM for every augmented image. We do the "inversion" of the augmentation and we end-up with multiple CAMs (for the same class) for a given image

↓

we reconstruct a high-quality CAM from many low-quality CAMs

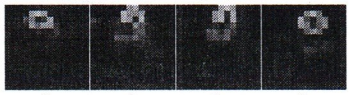
In the previous method, how are we going to merge the informations from multiple CAMs to obtain a high-resolution CAM?

again, heatmap in terms of CAM

The Super-Resolution Approach

We perform heat-map Super-Resolution (**SR**) by taking advantage of the information shared in multiple low-resolution heat-maps computed from the **same input under different – but known – transformations**

CNNs are in general invariant to roto-translations, in terms of predictions, but each $\boldsymbol{g}_\ell$ actually contains different information



g_1 g_2 g_3 g_4

General approach, our SR framework can be combined with any visualization tool (not only Grad-CAM)

G. Boracchi

The Super-Resolution Formulation

We model heat-maps computed by Grad-CAM as the result of an unknown downsampling operator $\boldsymbol{D} : \mathbb{R}^{N \times M} \rightarrow \mathbb{R}^{n \times m}$

The high-resolution heat-map $\boldsymbol{h}$ is recovered by solving an inverse problem

$$\underset{h}{\operatorname{argmin}} \frac{1}{2} \sum_{l=1}^{L} \|\mathcal{D}\mathcal{A}_\ell h - g_\ell\|_2^2 + \lambda TV_{\ell_1}(h) + \frac{\mu}{2}\|h\|_2^2 \quad (1)$$

TV_{ℓ_1}: Anistropic Total Variation regularization is used to preserve the edges in the target heat-map (high-resolution)

$$TV_{\ell_1}(\boldsymbol{h}) = \Sigma_{i,j} \|\partial_x \boldsymbol{h}(i,j)\| + \|\partial_y \boldsymbol{h}(i,j)\| \quad (2)$$

This is solved through Subgradient Descent since the function is convex and non-smooth

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G "Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation". ICASSP 2020

ℓ = # augmentations

h = high resolution CAM (unknown)

g_ℓ = non-resolution Grad-CAM

D = unknown downsampling operation

A_ℓ = known augmentation that has been performed

(just know that) two terms used to make the final (high-resolution) CAM look smoother

Augmented Grad-CAM

(a) Grad-CAM. (b) Augmented Grad-CAM.

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

The Augmented Grad-CAM is better covering the shape of the dog! (✓)

⇒ The Augmented Grad-CAM performs better than a simple up-sampling of the low resolution CAM

17.05

Object Detection

Thanks to Davide Rutigliano, former AN2DL student!

G. Boracchi

Object Detection Task

Given a fixed set of categories and an input image which contains an unknown and varying number of instances

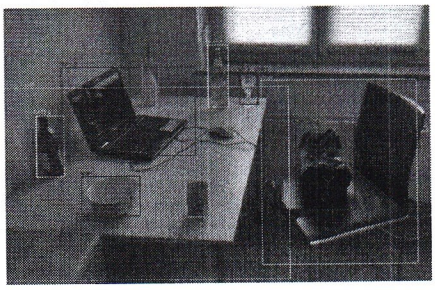
Draw a bounding box on each object instance

A training set of annotated images with labels and bounding boxes for each object is required

Each image requires **a varying number of outputs**

MAN: (x,y,h,w)
KID: (x,y,h,w)
GLOVE: (x,y,h,w)

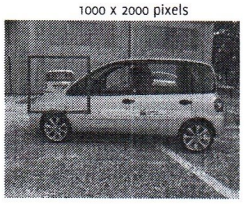
Annotated Dataset for Object Detection



G. Boracchi

The Straightforward Solution: Sliding Window

1000 x 2000 pixels

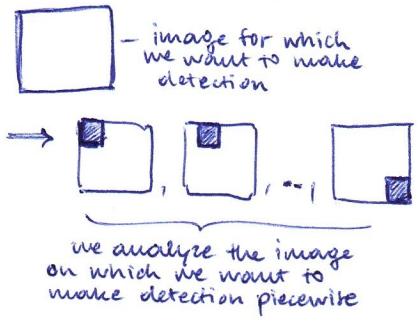


- Similar to the sliding window for semantic segmentation
- A pretrained model is meant to process a fixed input size (e.g. 224 x 224 x 3)
- Slide on the image a window of that size and classify each region.
- Assign the predicted label to the central pixel

Adopt the whole machinery seen so far to each crop of the image

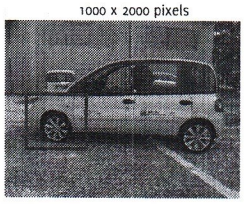
G. Boracchi

We train a model on fixed images, then we feed a larger image and we analyze the larger image from piece to piece with smaller dimensions:



The Straightforward Solution: Sliding Window

1000 x 2000 pixels



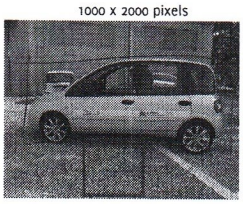
- Similar to the sliding window for semantic segmentation
- A pretrained model is meant to process a fixed input size (e.g. 224 x 224 x 3)
- Slide on the image a window of that size and classify each region.
- Assign the predicted label to the central pixel

Adopt the whole machinery seen so far to each crop of the image

G. Boracchi

The Straightforward Solution: Sliding Window

1000 x 2000 pixels



- Similar to the sliding window for semantic segmentation
- A pretrained model is meant to process a fixed input size (e.g. 224 x 224 x 3)
- Slide on the image a window of that size and classify each region.
- Assign the predicted label to the central pixel

Adopt the whole machinery seen so far to each crop of the image

The background class has to be included! ← common in all the detection methods

G. Boracchi

Many drawbacks...

Cons:

- **Very inefficient!** Does not re-use features that are «shared» among overlapping crops
- How to choose the crop size?
- Difficult to detect objects at different scales!
- A huge number of crops of different sizes should be considered....

Plus:

- No need of retraining the CNN

Region Proposal

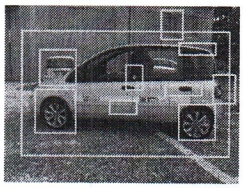
Region proposal algorithms (and networks) are meant **to identify bounding boxes** that correspond to a **candidate object** in the image.

Algorithms with **very high recall** (but low precision) were there before the deep learning advent

if there is an object, most probably the bounding box associated to this object will be returned by this algorithm (but at very low precision)

The idea is to:

- Apply a region proposal algorithm
- Classify by a CNN the image inside each proposal regions



This CVPR2014 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

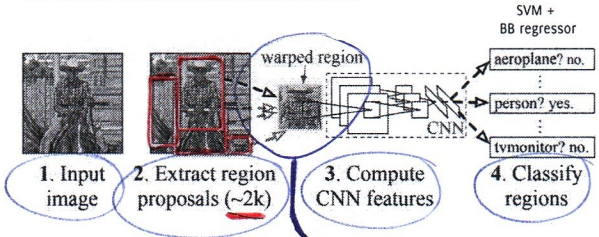
Rich feature hierarchies for accurate object detection and semantic segmentation

Ross Girshick[1] Jeff Donahue[1,2] Trevor Darrell[1,2] Jitendra Malik[1]
[1]UC Berkeley and [2]ICSI
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

G. Boracchi

R-CNN — Region Convolutional Neural Network

Object detection by means of region proposal (R stands for **regions**)

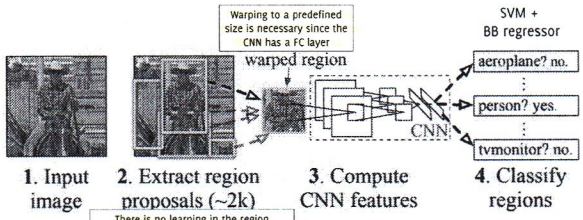


Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." *CVPR 2014*

to classify each proposed image (region) we need to feed a CNN with the image and so we have to re-size the image.
⇒ WARPED REGION (regione deformata)
This is definitely not good. We're changing the proportions of the region. We're not preserving all the informations.

R-CNN

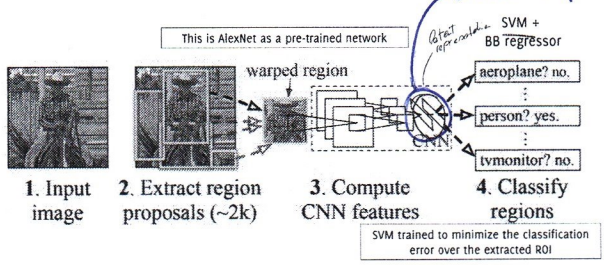
Object detection by means of region proposal



Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." *CVPR 2014*

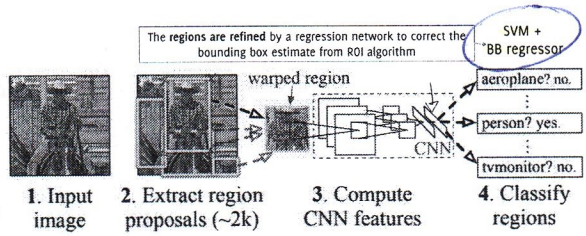
R-CNN

latent representation



SVM is in charge of the classification

R-CNN



Support Vector Machine for the classification of each region

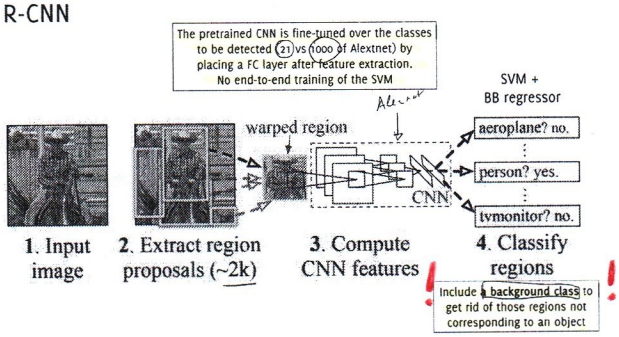
Bounding Box regressor to refine and correct the bounding box estimate from ROI algorithm

the 2 are separately trained

Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." *CVPR 2014*

ROI = region of interest

R-CNN



R-CNN limitations

- **Ad-hoc training objectives** and not an end-to-end training
  - Fine-tuning network with softmax classifier (log loss) before training SVM
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- **Region proposals are from a different algorithm** and that part has not been optimized for the detection by CNN
- **Training is slow** (84h), takes **a lot of disk space to store features**
- **Inference** (detection) **is slow** since the CNN has to be executed on each region proposal (**no feature re-use**)
  - 47s / image with VGG16

This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

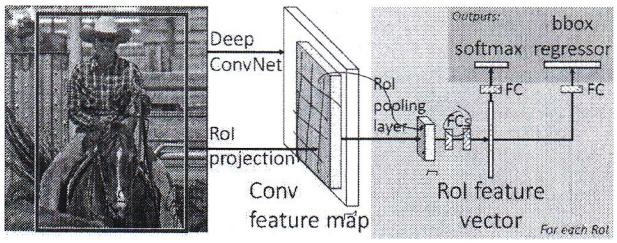
Fast R-CNN

Ross Girshick
Microsoft Research
rbg@microsoft.com

G. Boracchi

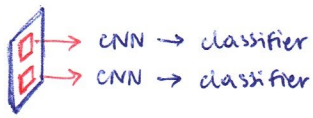
Fast R-CNN

avoid to run the entire CNN for every proposed region

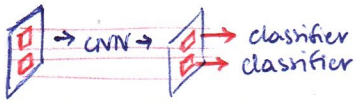


Girshick, Ross. "Fast r-cnn." ICCV 2015

R-CNN

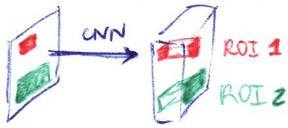


Fast R-CNN



Here the CNN is used only one time, the regions are identified in the original image and projected into the features map. Then they're classified from the features map.

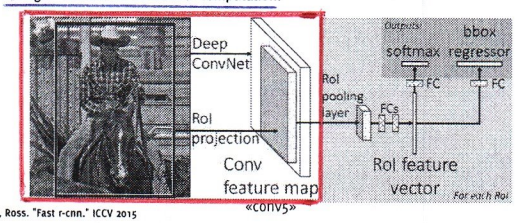
Problem: when we're already in the features map we are connected to a classifier which wants always the same dimentions as input but the ROI (region of interest) may have different size.



and so we introduce the **ROI POOLING LAYER**: extract a fixed size from the last convolutional layer of the network for whatever region selected. (and this is done by maxpooling)

Fast R-CNN

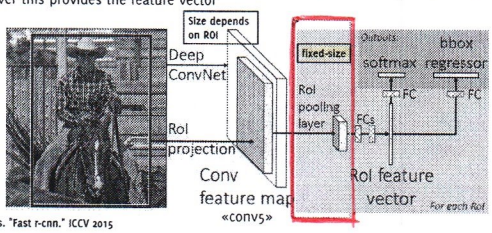
1. The whole image is fed to a CNN that extracts feature maps.
2. **Region proposals** are identified from the image and **projected into the feature maps.** Regions are directly cropped form the feature maps, instead from the image: →**re-use convolutional computation.**



Girshick, Ross. "Fast r-cnn." ICCV 2015

Fast R-CNN

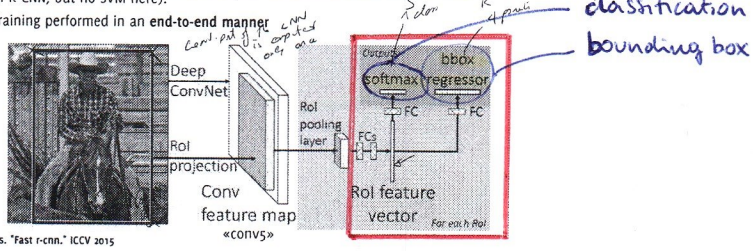
3. Fixed size is still required to feed data to a fully connected layer. **ROI pooling** layers extract a **feature vector of fixed size $H \times W$** from each region proposal. Each **ROI in the feature maps is divided in a $H \times W$ grid** and then **maxpooling** over this provides the feature vector



Girshick, Ross. "Fast r-cnn." ICCV 2015

Fast R-CNN

4. **The FC layers estimate both classes and BB location** (bb regressor)
   A convex combination of the two is used as a **multitask loss** to be optimized (as in R-CNN, but no SVM here).
5. Training performed in an **end-to-end manner**



Girshick, Ross. "Fast r-cnn." ICCV 2015

Fast R-CNN

In this new architecture it is possible to **back-propagate through the whole network**, thus train the whole network in an end-to-end manner

It becomes **incredibly faster than R-CNN during testing**.

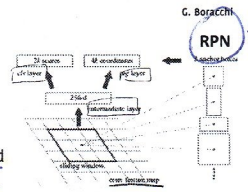
Now that convolutions are not repeated on overlapping areas, **the vast majority of test time is spent on ROI extraction** (e.g. selective search)

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren* Kaiming He Ross Girshick Jian Sun
Microsoft Research
{v-shren, kahe, rbg, jiansun}@microsoft.com

Faster R-CNN

- Instead of the ROI extraction algorithm, **train a region proposal network** (RPN), which is a F-CNN (3x3 filter size)
- **RPN operates on the same feature maps** used for classification, thus at the last conv layers
- RPN can be seen as an additional module that **improves efficiency** and **focus Fast R-CNN** over the most promising regions for object detection



RPN is based on 3 different blocks:
- Intermediate layer
- Regression layer
- Classification layers

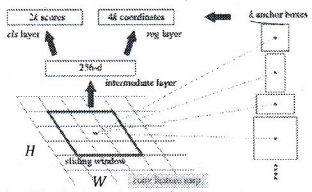
This allows the detection network also to learn how to extract the region proposed.

Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

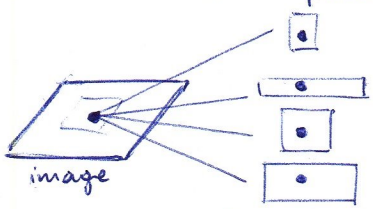
RPN

Goal: Associate to each spatial location k anchor boxes, i.e. ROI having different scales and rations (e.g. $k = 3 \times 3$, 3 sizes of the anchor side, 3 height/width ratios)

The network outputs $H \times W \times k$ candidates anchor and estimate *scores* for each anchor



The goal is to provide as output **k** different estimates of possible bounding boxes. These estimates of bounding boxes are called **ANCHORS**. Anchors are associated to each pixel in the convolutional feature map and have different shapes of bounding box to be centered in that pixel.



For each pixel are estimated **k** possible bounding boxes (**k** anchor boxes)

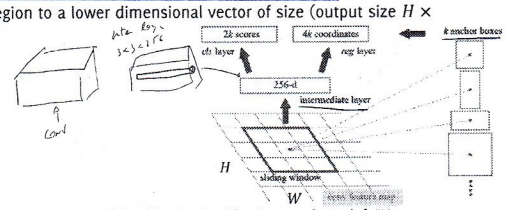
Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

RPN: Intermediate layer

Intermediate layer: is a standard CNN layer that takes as input the last layer of the feature-extraction network and uses 256 filters of size 3×3.

It is used to **reduce the dimensionality of the feature map.**

Maps the region to a lower dimensional vector of size (output size $H \times W \times 256$)



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

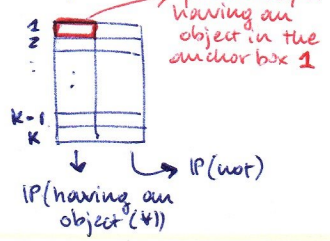
part 1

before the whole anchors thing we reduce the dimensionality of the feature map: **INTERMEDIATE LAYER**

The output of the intermediate layer is used to train 2 different networks:
- classification network
- regression network

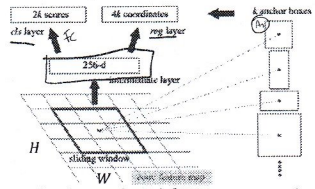
The classification network is meant to predict wether in that pixel there is or not an object associated to one of those ANCHORS. The output of the classification layer is **2k**, where **k** is the number of anchors and the **2** because for each is a "contain" or "**not** contain" the object (not an object of a specific class, any object).

⇒ output:



RPN estimating k anchors

- The **cls** (classification) **network** is trained to predict the ***object probability***, i.e. the probability for an anchor to contain and object *[contains / does not contain]* → $2k$ probability estimates
- Made of a stack of convolutional layers having size 1×1
- Each of these k proabity pairs corresponds to a specific anchor (having a specific dimension) **and expresses the probability for that anchor in that spatial location to contain *any object***



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

part 2

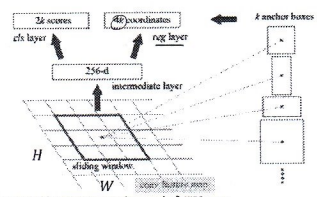
The regression network instead, adjust each anchor to better match the location. This network provides 4k output. The 4 because we have the displacements that we want to adjust for every anchor.

4-tuples for every anchor:

[x, y, w, h]

RPN estimating k anchors

- The **reg** (regression) **network** is trained to *adjust* each of the k predicted anchor to better match object ground truth → $4k$ estimates for the 4 bounding box cooridnates
- Each of these k 4 —tuples **expresses the** refinements for a specific anchor

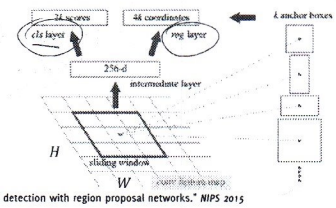


Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

RPN estimating k anchors

different anchors sizes (**note**: if we want to change the number of anchors for each pixel then we have to change the RPN)

- If you want to train the network to predict different anchors, there is no need to design different RPN, bust just to **define different labels when training the RPN, associated to different anchors**
- Each of these k 4 –tuples **expresses the** refinements for a specific anchor



There is no encoding of the anchors dimensions inside the network

Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

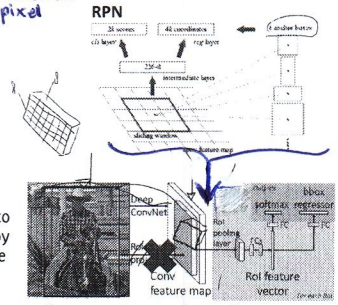
Faster R-CNN

k proposals for each pixel

RPN returns $H \times W \times k$ region proposals,
thus replaces the region proposal algorithms
(selective search)

After RPN there is a **non-maximum suppression**
based on the *objectiveness score*

Remaining proposals are then fed to the ROI pooling and then clssifier by the standard Fast-RCNN architecture

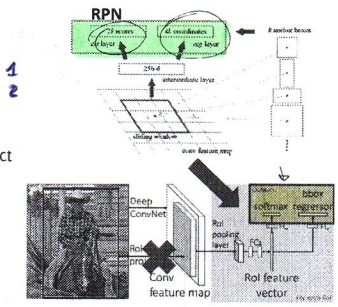


We can feed the classificator with $\forall$anchor.

Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

Faster R-CNN Training

- **Training** now involves 4 losses:
  - RPN classify object/non object 1
  - RPN regression coordinates 2
  - Final classification score 3
  - Final BB coordinates 4
- During training, object/non object ground truth is defined by measuring the overlap with annotated BB



1. tells us wether an anchor contains or not an object
2. regression loss for the bounding box of the anchor

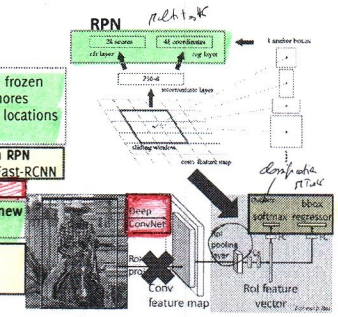
+ 3., 4. standard outputs of the fast R-CNN

Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

Faster R-CNN Training

Training procedure

1. **Train RPN** keeping backbone network frozen and training only RPN layers. This ignores object classes but just bounding box locations (Multi-task loss cls + reg)
2. **Train Fast-RCNN** using proposals from **RPN trained before**. Fine tune the whole Fast-RCNN **including the backbone**
3. **Fine tune the RPN in cascade of the new backbone**
4. **Freeze backbone and RPN and fine tune only the last layers of the Faster R-CNN**



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

Faster R-CNN

At test time,

- Take the top ~ 300 anchors according to their object scores
- Consider the refined bounding box location of these 300 anchors
- These are the ROI to be fed to a Fast R-CNN
- Classify each ROI and provide the non-background ones as output

Faster R-CNN provides as output to each image a **set of BB** with their **classifier posterior**

The network becomes much faster (0.2s test time per image)

Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

Faster R-CNN

It's still a **two stage detector**

first we extract a region and then we classify it

First stage:

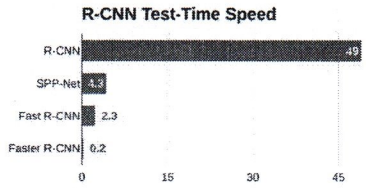
- run a **backbone network** (e.g. VGG16) to **extract featues**
- run the **Region Proposal Network** to **estimate** ~ 300 **ROI**

Second stage (the same as in Fast R-CNN):

- **Crop Features** through ROI pooling (with alignment)
- **Predict object class** using FC + softmax
- **Predict bounding box offset** to improve localization using FC + softmax

Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

Faster R-CNN



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NIPS 2015*

This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

You Only Look Once:
Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

– You Only Look Once

G. Boracchi

YOLO/SSD

R-CNN methods are based on region proposals
There are also region-free methods, like:
Yolo: You Only Look Once
SSD: Single Shot Detectors

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." CVPR 2016.

The rationale

Detection networks are indeed a pipeline of multiple steps.

In particular, **region-based methods make it necessary to have two steps** during inference

This can be slow to run and hard to optimize, because each individual component must be trained separately.

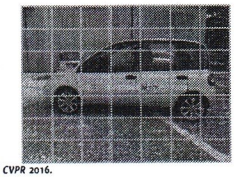
In Yolo *"**we reframe the object detection as a single regression problem,** straight from image pixels to bounding box coordinates and class probabilities"*

And solve these regression problems **all at once**, with a large CNN

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *CVPR* 2016.
Liu, Wei, et al. "SSD: Single shot multibox detector." *European conference on computer vision*. Springer, Cham, 2016.

YOLO

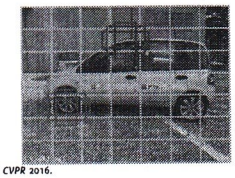
1. divide the image in a coarse grid (e.g. 7x7)



Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *CVPR* 2016.

YOLO

1. divide the image in a coarse grid (e.g. 7x7)
2. each grid cell contains ***B anchors*** *(base bounding box)* associated



Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *CVPR* 2016.

YOLO

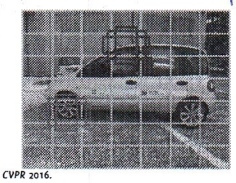
the adjustment of the bounding box ([x,y,w,h]) and the objectness score (exactly as before in the RPN)

3. For each cell and anchor **we predict**:
   - The **offset of the base bounding box**, to better match the object: $(dx, dy, dh, dw, objectness_score)$
   - The **classification score of the base-bounding box** over the C considered categories (including background)

classification score of the anchor given the categories (we have a score associated to each category)

So, the output of the network has dimension
$7 \times 7 \times B \times (5 + C)$

! **Fixed size**: for each cell (7x7) and for each anchor (B) it provides 5+C values



Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *CVPR* 2016.

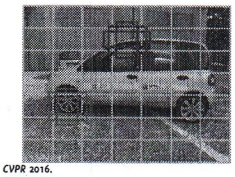
YOLO

The whole **prediction is performed in a single forward pass** over the image, **by a single convolutional network**

Training this network is sort of **tricky to assess the** loss (matched / not matched)

YOLO/SSD shares a similar ground of the RPN used in Faster R-CCN

Typically networks based on region-proposal are more accurate, single shot detectors are faster but less accurate

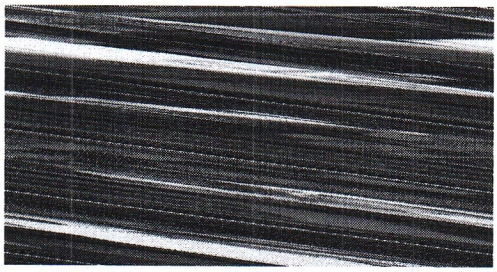


Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *CVPR* 2016.

Object Detection

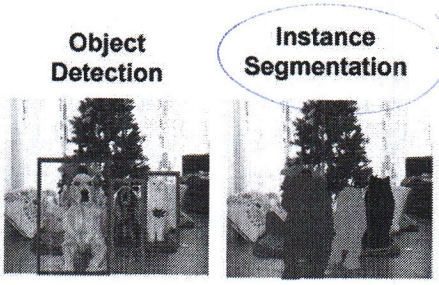
Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

Object Detection (**video ↓** : the process is done frame by frame)



Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

Object Detection vs Instance Segmentation



segment each instance separately

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

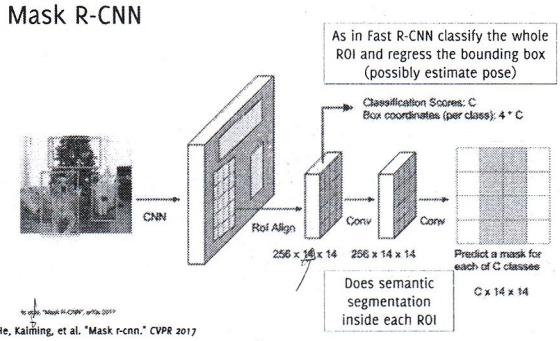
Instance Segmentation

It combines the challenges of:

- **Object detection** (multiple instances present in the image)
- **Semantic segmentation** (associate a label to each pixel) separating each object instance

(just to introduce)

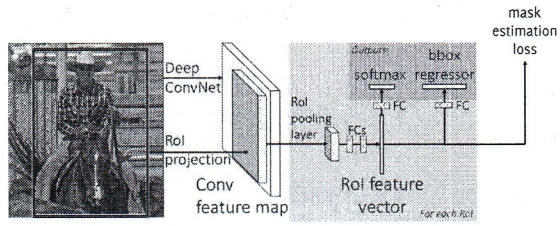
Mask R-CNN



He, Kaiming, et al. "Mask r-cnn." *CVPR 2017*

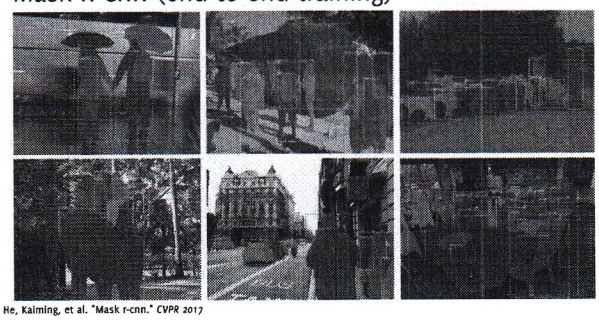
Mask R-CNN

Mask is estimated for each ROI and each class



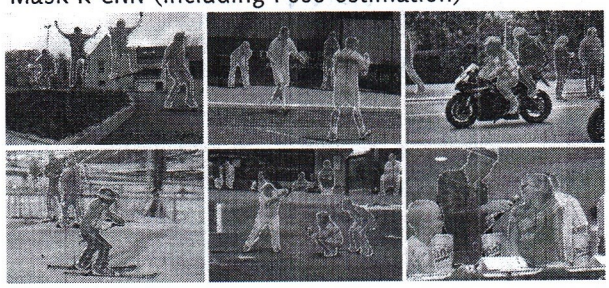
He, Kaiming, et al. "Mask r-cnn." *CVPR 2017*

Mask R-CNN (end-to-end training)



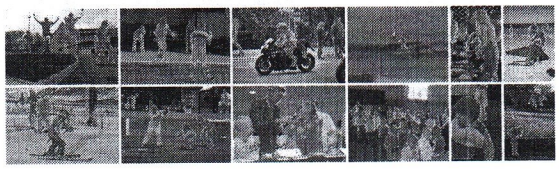
He, Kaiming, et al. "Mask r-cnn." *CVPR 2017*

Mask R-CNN (including Pose estimation)



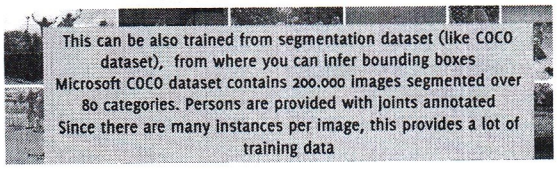
He, Kaiming, et al. "Mask r-cnn." *CVPR 2017*

Mask R-CNN (including Pose estimation)



He, Kaiming, et al. "Mask r-cnn." *CVPR 2017*

Mask R-CNN (including Pose estimation)



He, Kaiming, et al. "Mask r-cnn." *CVPR 2017*

Popular Architectures and Generative Adversarial Networks

Giacomo Boracchi
Advanced Neural Networks and Deep Learning
https://boracchi.faculty.polimi.it/

→ Other popular architectures

→ Generative Models (GAN)
→ Metric Learning (Siamese Networks)

Outline

Lecture inspired to:

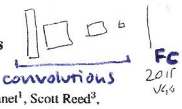
Notes accompanying the Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

... and papers cited in here!

This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

All the networks have this form: (all that we have seen till now, there's a second generation of neural networks)

Going Deeper with Convolutions



2011
VGG

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3], Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Inception Module

The most straightforward way of improving the performance of deep neural networks is by **increasing their size (either in depth or width)**

Backdraws:

Bigger size typically means

- a larger number of parameters, which makes the enlarged network more prone to overfitting
- dramatically increase in the use of computational resources

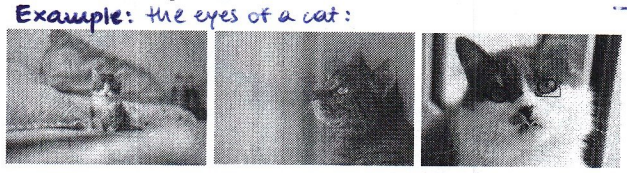
Moreover image features might appear at different scale, and it is difficult to define the right filter size

Features might appear at different scales

kernel intended as filter

Difficult to set the right kernel size!

Example: the eyes of a cat:

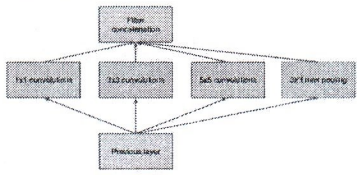


→ It's not guaranteed (instead most of the times it does not happen) that the change of scale does not influence the performance.

Photos from Unsplash

GoogLeNet and Inception v1 (2014)

Deep network, with **high computational efficiency**
Only **5 million parameters**, 22 layers of Inception modules
Won 2014 ILSVR-classification challenge (6,7% top 5 classification error)



(a) Inception module, naïve version

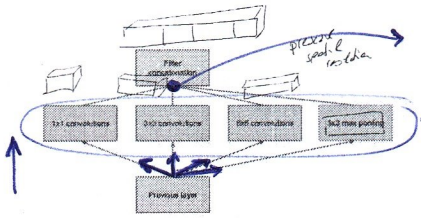
Inception block: it performs multiple convolutions, of different size, simultaneously over the same input. Then, at the end, the outputs are concatenated into one.

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

GoogLeNet and Inception v1 (2014)

It is based on inception modules, which are sort of «networks inside the network» or «local modules»



(a) Inception module, naïve version

here it's preserved the original spatial resolution.

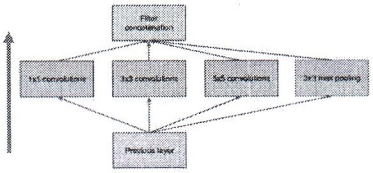
The depths can be different, the important sizes to be the same are width and height (they have to be the same because of the final concatenation)

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Inception Module (2014)

- Choosing the **right kernel size** for the convolution operations is difficult, as image might show relevant features at different scale
- Deep networks **tend to over fit** and to become very computationally expensive



(a) Inception module, naïve version

Szegedy et al. "Going deeper with convolutions." *CVPR 2015.*

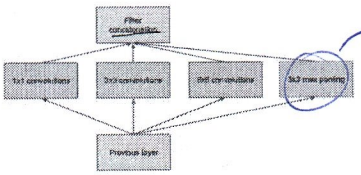
G. Boracchi

Inception Module (2014)

The solution is to **exploit multiple filter size** at the same level (1x1, 3x3, 5x5) and then **merge by concatenation** the output activation maps together

All the blocks preserve the spatial dimension by zero padding (convolutional filters) or by fractional stride (for Maxpooling)

Thus, outputs can be concatenated dept-wise



(a) Inception module, naïve version

this is a fake max pooling: every cell is replaced with the max over a certain zone but the sizes are preserved

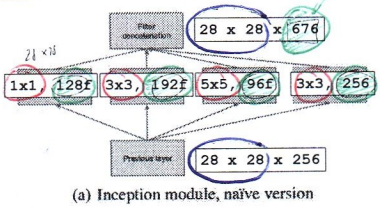
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Inception Module (2014)

The solution is to exploit multiple filter size at the same level (1x1, 3x3, 5x5) and then **merge by concatenation** the output activation maps together

- **Zero padding to preserve spatial size**
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



(a) Inception module, naïve version

Concatenation means that all the images obtained are put in a pile: 676 = 128 + 192 + 96 + 256

filter sizes (width × height)
filters (depth)

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

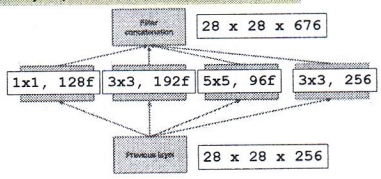
G. Boracchi

Inception Module (2014)

The solu ... id
then me ...

The spatial extent is preserved, but the depth of the activation map is much expanded.
This is very expensive to compute

- Zero ...
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



(a) Inception module, naïve version

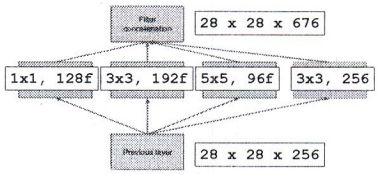
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Inception Module (2014)

The solu **Computational problems will get significantly worst when stacking multiple layers...** nd then me

- **Zero padding to preserve spatial size**
- The activation map grows much in depth
- A large number of operations to be performed due to the large depth of the input of each convolutional block: **854M operations** in this example



(a) Inception model, naive version

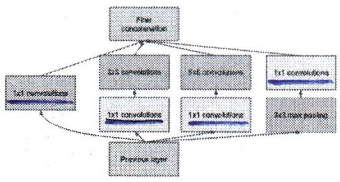
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Inception Module (2014)

To reduce the computational load of the network, the number of **input channels** is reduced by adding an **1x1 convolution** layers before the 3x3 and 5x5 convolutions

Using these 1x1 conv is referred to as **"bottleneck" layer**



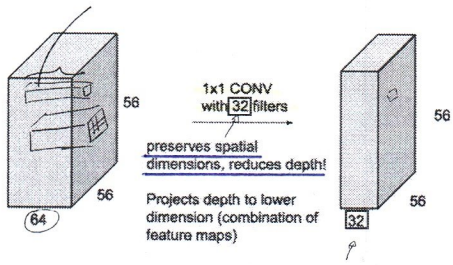
1x1 convolutions are used to reduce the depth

(b) Inception module with dimension reductions

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

1x1 convolution layers as bottleneck



CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

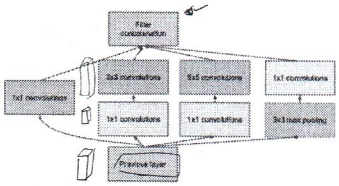
G. Boracchi

Inception Module (2014)

To reduce the computational load of the network, the number of **input channels** is reduced by adding an **1x1 convolution** layers before the 3x3 and 5x5 convolutions

The output volume has similar size, but the number of operation required is significantly reduced due to the 1x1 conv: **358M operations** now

Adding 1x1 convolution layers increases the number of nonlinearities



(b) Inception module with dimension reductions

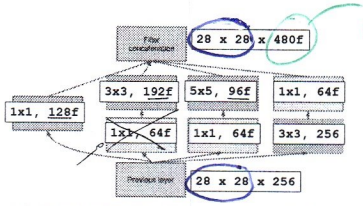
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

Inception Module (2014)

To reduce the computational load of the network, the number of **input channels** is reduced by adding an **1x1 convolution** layers before the 3x3 and 5x5 convolutions

The output volume has similar size, but the number of operation required is significantly reduced due to the 1x1 conv: **358M operations** now



definitely better (before we had 676)

(b) Inception module with dimension reductions

Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

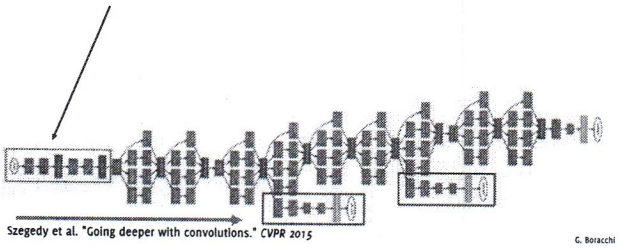
G. Boracchi

Here's the resulting network:

GoogLeNet (2014)

GoogleNet stacks 27 layers considering pooling ones.

At the beginning there are two blocks of conv + pool layers



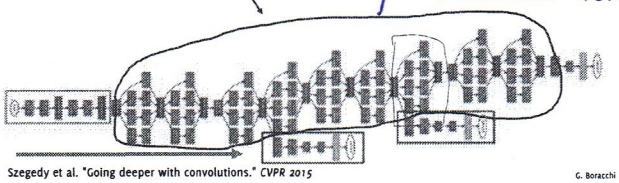
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

GoogLeNet (2014)

GoogLeNet stacks 27 layers considering pooling ones.
Then, there are a stack of 9 of inception modules

concatenation of the (upgraded) Inception blocks mentioned before



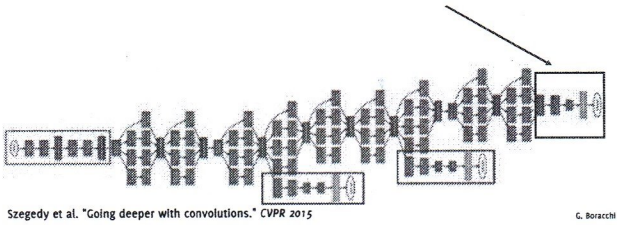
Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

GoogLeNet (2014)

57 pooling

GoogLeNet stacks 27 layers considering pooling ones.
No Fully connected layer at the end, simple averaging pooling, linear classifier + softmax

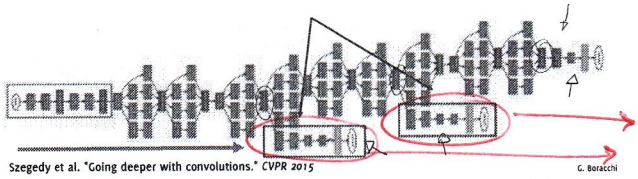


Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

GoogLeNet (2014)

GoogLeNet stacks 27 layers considering pooling ones.
It also suffers of the **dying neuron problem**, therefore the authors **add two extra auxiliary classifiers** on the intermediate representation **to compute an intermediate loss that is used during training.**
You expect intermediate layers to be able to classify as well



Szegedy et al. "Going deeper with convolutions." *CVPR 2015*

G. Boracchi

these two are intermediate classifiers: they just take the information that they can take and try to do the classification.
They're present just for training purposes (they're not directly considered for the final prediction)

This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

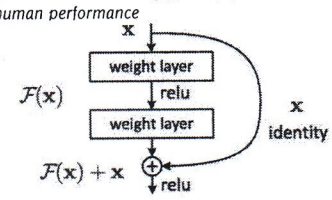
Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

ResNet (2015) (from Microsoft)

Very Deep network: 152 layers for a deep network trained on Imagenet!
1202 layers on CIFAR!
2015 ILSVR winner both localization and classification (3.57% top 5 classification error). Better than *human performance*

The main investigation was: is it possible to continuosly improve accuracy by stacking more and more layers

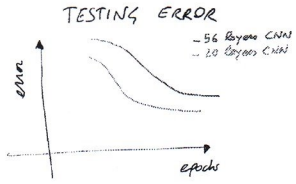


Kaiming He, et al. "Deep Residual Learning for Image Recognition" CVPR 2016

ResNet (2015): The rationale

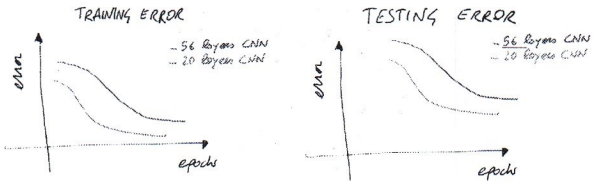
Increasing the network depth, by stacking an increasingly number of layers, **does not always improve performance**

TESTING ERROR



ResNet (2015): The rationale

! But **this is not due to overfitting**, since the same trend is shown in the training error !

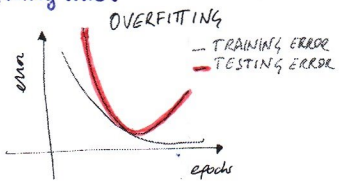


hi

ResNet (2015): The rationale

But **this is not due to overfitting**, since the same trend is shown in the training error, while for overfitting we have that training and test error diverge

If it would have been overfitting, it would have been something like:



but it wasn't the case!

G. Boracchi

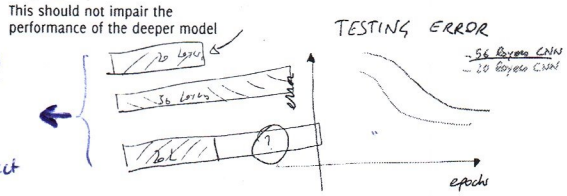
ResNet (2015): The rationale

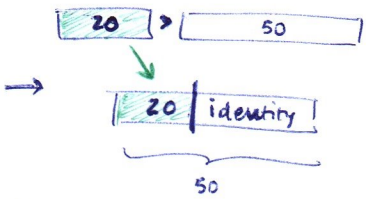
Deeper model are harder to optimize than shallower models.

However, we might in principle **copy the parameters of the shallow** network **in the deeper one** and then **in the remaining part**, set the weights to **yield an identity mapping.**

This should not impair the performance of the deeper model

If we have two networks: one of 20 layers and one of 50 layers and the one with 20 layers is better than the other, with a structure (yet to train) of 50 layers we expect the first 20 to be the ones of the 20-layers network (since it's the best between the networks that we have) and then some sort of identity in the remaining layers:





However, it comes out that the identity function is not easy to train!

So the fact is: if the optimal number of layers is 20, it's not that if we use 100 layers the first 20 we'll be trained as if they were only 20 layers (and so in the optimal way) and then the other 80 we'll be trained as identity function!

hi

ResNet (2015): The rationale

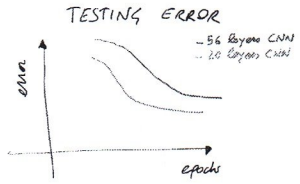
Deeper model are harder to optimize than shallower models.

However, we might in principle **copy the parameters of the shallow** network **in the deeper one** and then in the remaining part, set the weights to **yield an identity mapping.**

This should not impair the performance of the deeper model

Deeper networks should be in principle as good as the shallow ones

Then, the identity function is not easy to learn!

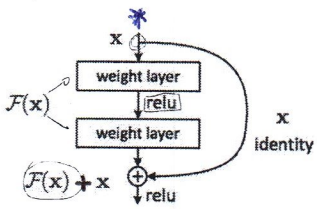


hi

ResNet: Very deep by residual connections

Adding an "Identity shortcut connection" :

- helps in **mitigating the vanishing gradient problem** and enables deeper architectures
- Does not add parameters
- In case the **previous network was optimal**, the **weights to be learned goes to zero** and information is propagated by the identity
- The network can still be trained through back-propagation



Kaiming He, et al. "Deep Residual Learning for Image Recognition" CVPR 2016

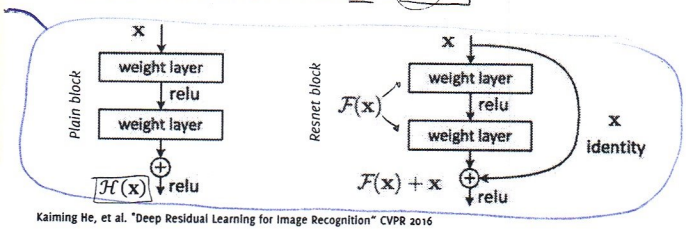
ResNet block: we have some standard layers (convolutionals) but we also have the possibility of a skip connection (that represent the identity function): if the output of the previous block (*) is already the best representation we can get then in this block we follow the identity path

ResNet: Very deep by residual connections

Intuition: force the network to learn a different task in each block.

If $\mathcal{H}(\mathbf{x})$ is the **ideal mapping** to be learned from a plain network, by skip connections **we force the network to learn** $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$,

! If $H(x)$ is the best we can do in the $H(x)$'s block, then $F(x)$ only learns the residual, namely $H(x) - x$. $F(x)$ does not learn anything that $H(x)$ already knows.

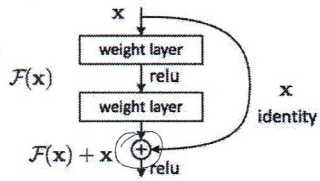


Kaiming He, et al. "Deep Residual Learning for Image Recognition" CVPR 2016

ResNet: Very deep by residual connections

- $\mathcal{F}(\mathbf{x})$ is called the **residual** (something to add on top of identity), which turns to be easier to train in deep networks.
- **Weights in between the skip connection can be used to learn a «delta», a residual i.e., $F(\mathbf{x})$ to improve over the solution that can be achieved by a shallow network**
- The weights (convolutional layers) are such that to **preserve dimension depth-wise or are re-arranged** by 1x1 convolutions

The convolutions preserve the size because post-convolutions we have to make a summation
⇒ preserve the SPATIAL EXTENT

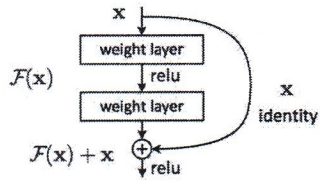


Kaiming He, et al. "Deep Residual Learning for Image Recognition" CVPR 2016

ResNet (2015)

The rationale behind adding this identity mapping is that:

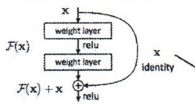
- It is easier for the following layers to learn features on top of the input value
- In practice the weights between the identity mapping fails at learning the identity function to transfer the input to the output
- The performance achieved by resNet suggests that **probably most of thee deep layers have to be close to the identity!**

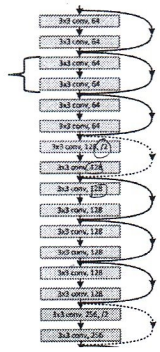


Kaiming He, et al. "Deep Residual Learning for Image Recognition" CVPR 2016

ResNet (2015)

The ResNet is a stack of 152 layers of this module





The network alternates

- some spatial **pooling by convolution with stride 2**
- **Doubling the number of filters**

At the beginning there is a **convolutional layer**
At the end, no FC but just a **GAP to be fed in the final softmax**

Deeper networks are able to achieve lower errors as expected

Kaiming He, et al. "Deep Residual Learning for Image Recognition" CVPR 2016

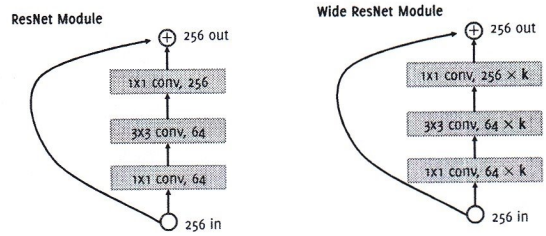
CvF This CVPR paper is the Open Access version, provided by the Computer Vision Foundation. Except for this watermark, it is identical to the version available on IEEE Xplore.

Aggregated Residual Transformations for Deep Neural Networks

Saining Xie[1] Ross Girshick[2] Piotr Dollár[2] Zhuowen Tu[1] Kaiming He[2]
[1]UC San Diego [2]Facebook AI Research
{s9xie,ztu}@ucsd.edu {rbg,pdollar,kaiminghe}@fb.com

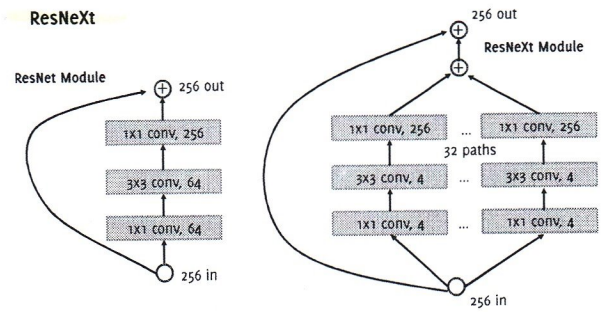
Some alternatives: (we're not entering into the details)

Wide Resnet



Xie et al "Aggregated Residual Transformations for Deep Neural Networks", CVPR 2017

ResNeXt



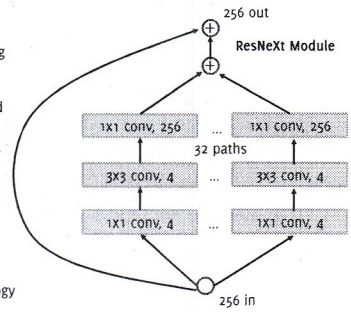
Xie et al "Aggregated Residual Transformations for Deep Neural Networks", CVPR 2017

ResNeXt

Widen the ResNet module by adding multiple pathways in parallel (previous wide Resnet was just increasing the number of filters and showing it achieves similar perf. with fewer blocks

Similar to inception module where the activation maps are being processed in parallel

Different from inception module, all the paths share the same topology



Xie et al "Aggregated Residual Transformations for Deep Neural Networks", CVPR 2017

This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

DenseNet

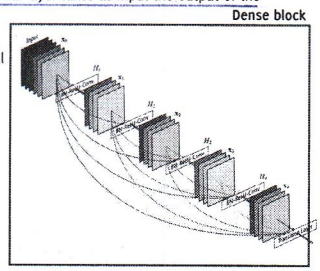
(not in details)

In the block a DenseNet, each convolutional layer takes as input the output of the previous layers

Short connections between convolutional layers of the network

Each layer is connected to every other layer in a feed-forward fashion

Dense block



Huang G. et al, "Densely Connected Convolutional Network" CVPR 2017

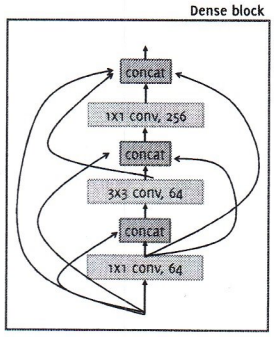
DenseNet

In the block a DenseNet, each **convolutional layer takes as input the output of the previous layers**

Each layer is connected to every other layer in a feed-forward fashion

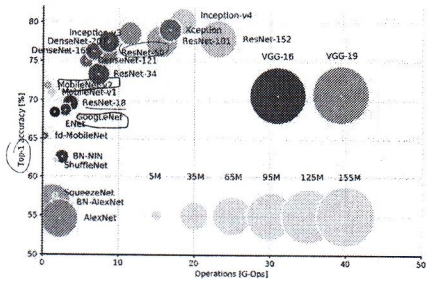
This alleviates vanishing gradient problem, promotes feature re-use since each feature is spread through the network

Dense block



Huang G. et al, "Densely Connected Convolutional Network" CVPR 2017

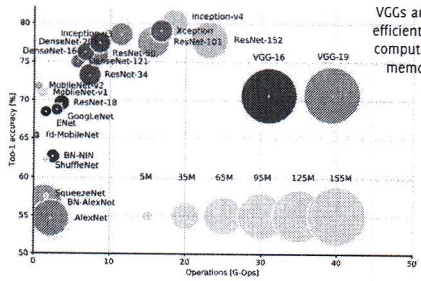
Comparison



Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

G. Boracchi

Comparison



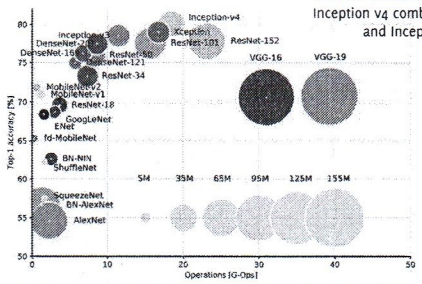
VGGs are the least efficient: very large computational and memory usage

Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

G. Boracchi

Comparison

Inception models are the most efficient and best performing
Inception v4 combines ResNet and Inception

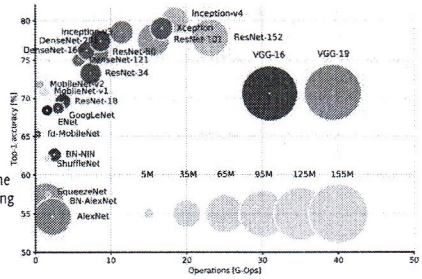


Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

G. Boracchi

Comparison

AlexNet are the least performing and not particularly efficient

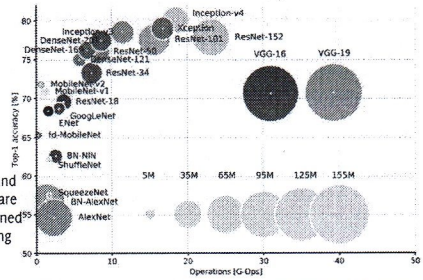


Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

G. Boracchi

Comparison

Mobile net and squeezenet are instead designed for improving efficiency



Canziani, A, Paszke A., Culurciello E.. "An analysis of deep neural network models for practical applications." *arXiv preprint* (2016).

G. Boracchi

Generative Adversarial Networks

: new problem: generating images

Giacomo Boracchi
giacomo.boracchi@polimi.it

November 18, 2020

giacomo.boracchi@polimi.it
home.deib.polimi.it/boracchi/

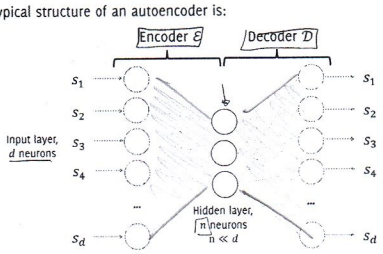
Background: Autoencoders

Autoencoders using MLP

multi layer perception

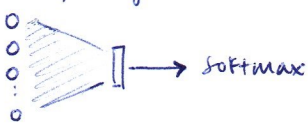
Autoencoders are neural networks used for data reconstruction (unsupervised learning)

The typical structure of an autoencoder is:

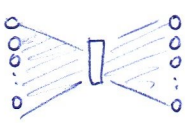


With the structure we want the network to learn inputs. The point is; if the bottleneck has smaller dimensions than the beginning/end, then we found a way to reduce the infos of the inputs (encode them) s.t. we're able to reconstruct (fully) the inputs from these informations.

Usually we go:



autoencoders are:



Autoencoders using MLP

Autoencoders are non-parametric models that can be trained to reconstruct all the data in a training set.

The reconstruction loss is

$$\boxed{\sum_{s \in S} \|s - \mathcal{D}(\mathcal{E}(s))\|_2} + \lambda R(\ )$$

$\mathcal{E}(\bullet)$ encoding
$D(\mathcal{E}(\bullet))$ decoding

and training of $\mathcal{D}(\mathcal{E}(\cdot))$ is performed through standard backpropagation algorithms (e.g. SGD)

Notice that there are no labels (external labels) involved in training the autoencoder

Bengio, Y., Courville, A., Vincent, P. *"Representation learning: A review and new perspectives"*. IEEE TPAMI 2013

G. Boracchi

Autoencoders

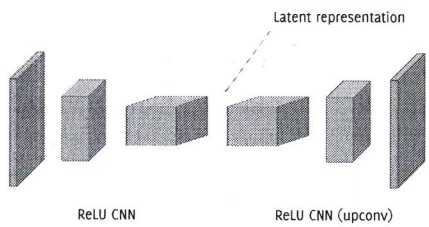
Trained in an unsupervised manner (we don't have labels, we're just training the network to reconstruct the input)

Remark:

- Features z are typically referred to as **latent representation**
- AE typically do not provide exact reconstruction since $n \ll d$, by doing so we **expect the latent representation to be a meaningful and compact representation** of the input
- Additional **regularization terms** might be included in the loss function
- More powerful and nonlinear representations can be learned by **stacking multiple hidden layers** (deep architectures)

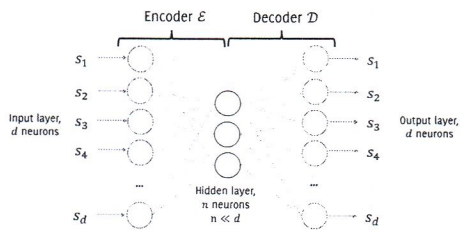
G. Boracchi

Convolutional AutoEncoders



Using Autoencoders for classifier initialization

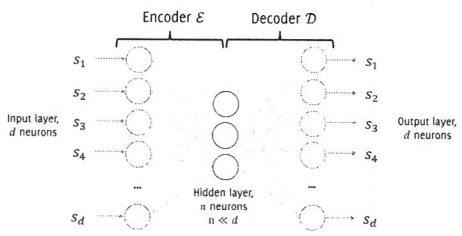
Autoencoders can be used to **inizialize the classifier when few annotated data** are provided for training together with many unlabeled ones



Using Autoencoders for classifier initialization

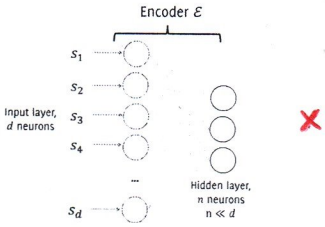
1) **Train the autoencoder** in a fully unsupervised way, using all the unlabeled data

even if they are unlabeled, because we just want to reconstruct them



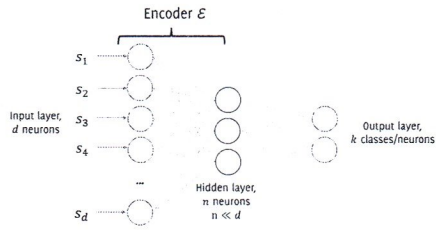
Using Autoencoders for classifier initialization

2) **Get rid of the decoder** and keep the encoder weights



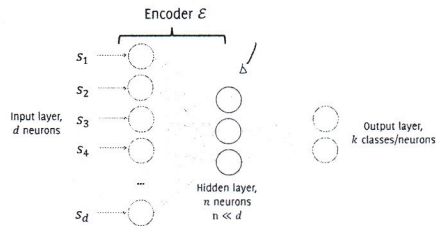
Using Autoencoders for classifier initialization

3) **Plug in a FC layer for classifying** samples from the latent representation



Using Autoencoders for classifier initialization

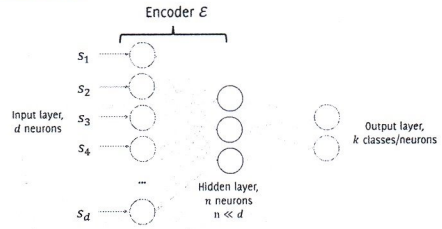
4) **Fine tune** the autoencoder using the few supervised samples provided. This is perfectly in line with «Transfer Learning» and holds for whatever model



This is an alternative to the pre-trained models (transfer learning)

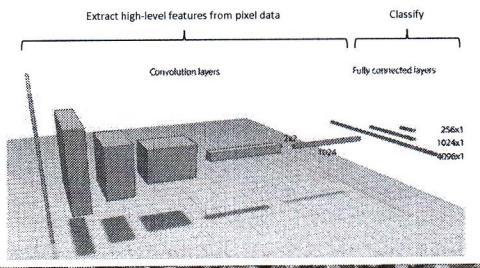
Using Autoencoders for classifier initialization

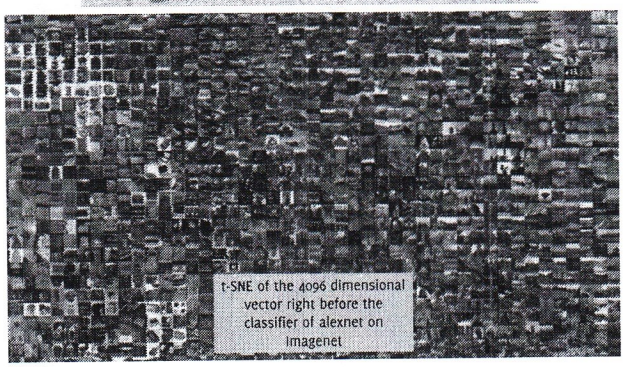
Autoencoders provide a **good initialization** (and reduce the risk of overfitting) because their latent vector is actually **a good (latent) representation** of the inputs used for training.



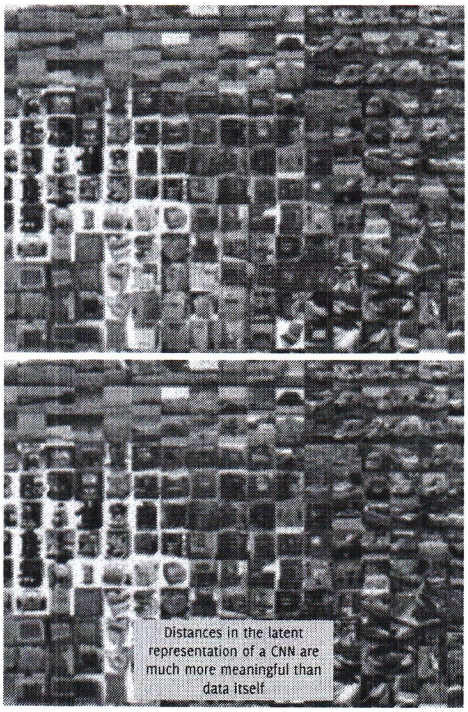
The Latent Representation of CNNs

A typical architecture





t-SNE of the 4096 dimensional vector right before the classifier of alexnet on imagenet

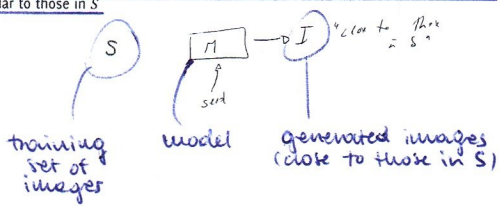


Generative Models

Networks able to generate realistic images

Generative Models

Goal: generate, given a training set of images (data) S, other images (data) that are similar to those in S



G. Boracchi

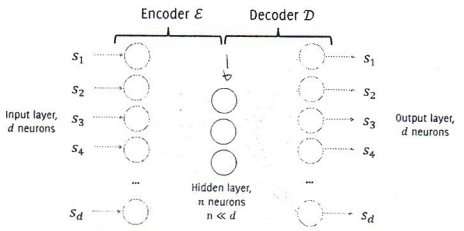
What for generative models?

- Generative models can be used for **data augmentation**, simulation and planning
- Training generative models can also enable inference of latent representations that can be useful as general features
- Realistic samples for artwork, super-resolution, colorization, etc.
- You are getting close to the "holy grail" of modeling the distribution of natural images

Radford, et al "Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

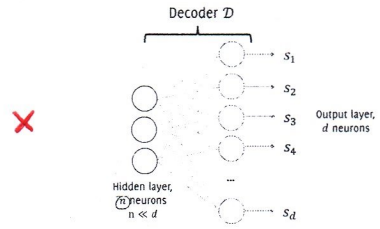
What about using Autoencoders as Generative Models?

One option would be to
1) **train an autoencoder** on S



What about using Autoencoders as Generative Models?

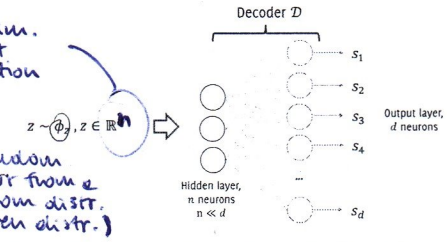
2) Discard the encoder



What about using Autoencoders as Generative Models?

3) Draw random vectors to replace the latent representation and feed this to the decoder input

same dim. of the latent representation

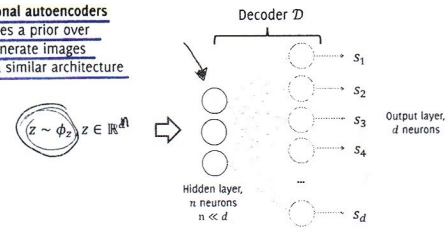


random vector from a random distr. (or a given distr.)

What about using Autoencoders as Generative Models?

This approach does not work since **we do not know the distribution of proper latent representation** (or it is very difficult to estimate).

Variational autoencoders leverages a prior over z to generate images using a similar architecture



Generative Adversarial Networks

do not use the autoencoder approach

A very effective way to generate images

Generative Adversarial Networks (GAN)

everything is trained in an unsupervised manner (we don't need labeled images, we only need images similar to the ones we want to construct)

The GAN approach:

- Do not look for an explicit density model ϕ_S describing the manifold of natural images.
- Just find out a model able to generate samples that «looks like» training samples $S \subset \mathbb{R}^n$

Instead of sampling from ϕ_S, just:

- **Sample a seed** from a known distribution ϕ_z . This is defined a priori and also referred to as noise.
- Feed this seed to a learned transformation that generates realistic samples, as if they were drawn from ϕ_S

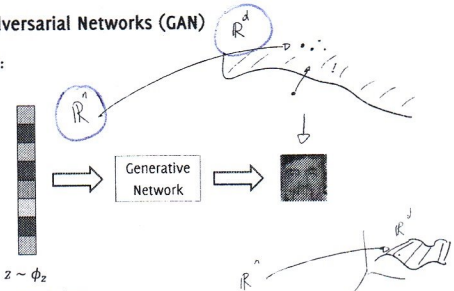
Use a neural network to learn this transformation

The neural network is going to be **trained in an unsupervised manner, no label needed**

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

Generative Adversarial Networks (GAN)

The GAN approach:



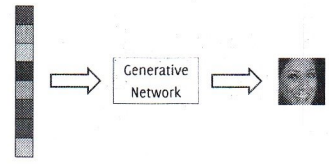
which is known

Draw a sample from the noise distribution

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

Generative Adversarial Networks (GAN)

The GAN approach:



$z \sim \phi_z$

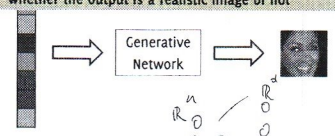
Draw a sample from the noise distribution

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

Generative Adversarial Networks (GAN)

The GAN approach:

The biggest challenge is to define a suitable loss for assessing whether the output is a realistic image or not



$z \sim \phi_z$

Draw a sample from the noise distribution

$\mathbb{R}^n$ $\mathbb{R}^d$

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

How can we associate to an image a measure which tells how close the image is to a human face? ?

Instead of defining a loss function (impossible to define) we train a second neural network. This second ANN will be trained for an artificial problem and will tell if an image is real or fake.

Generative Adversarial Networks (GAN)

The GAN solution: Train a pair of neural networks addressing two different tasks that compete in a sort of **two player** (adversarial) **game.**

These models are:

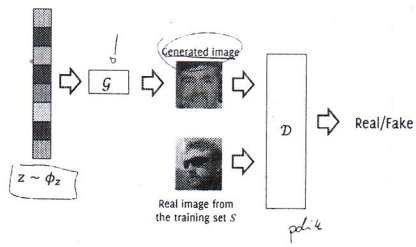
- **Generator $\mathcal{G}$ that produces realistic samples** e.g. taking as input some random noise. $\mathcal{G}$ tries to fool the discriminator
- **Discriminator $\mathcal{D}$** that takes as input an image **and assess whether it is real or generated by $\mathcal{G}$**

Train the two and **at the end, keep only $\mathcal{G}$**

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

GAN Architecture

$D: \mathbb{R}^d \to \{0,1\}$



GAN Architecture:

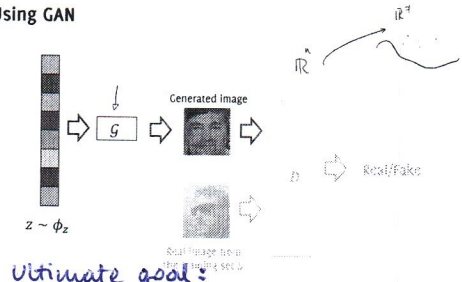
- random sample
- generating an image from the random sample (through $\mathcal{G}$)
- testing if the image generated is real or fake (through $\mathcal{D}$)

$\mathcal{G}: \mathbb{R}^n \to \mathbb{R}^d$

$\mathcal{D}: \mathbb{R}^d \to \{0,1\}$

The loss of training $\mathcal{G}$ is related to the loss of $\mathcal{D}$. The idea is to obtain a good $\mathcal{G}$ such that the generated images are not distinguishable from the fake through $\mathcal{D}$.

Using GAN



Ultimate goal:

Discriminator $\mathcal{D}$ is completely useless and as such dropped. After a successful GAN training, $\mathcal{D}$ is not able to distinguish the real/fake

It's like training a generator of money and taking care of the police. At the beginning the police will be easily able to tell if the money is fake, but thanks to the feedback from police, the generator can improve and can make so good fake money that the police won't be able to distinguish them.

GAN

Both $\mathcal{D}$ and $\mathcal{G}$ are conveniently chosen as MLP / CNN

Setting up the stage

Our networks take as input: generated real

- $\mathcal{D} = \mathcal{D}(s, \theta_d)$
- $\mathcal{G} = \mathcal{G}(z, \theta_g)$ $z \sim \phi_z$

θ_g and θ_d are network parameters, $s \in \mathbb{R}^d$ is an input image (either real or generated by $\mathcal{G}$) and $z \in \mathbb{R}^n$ is some random noise to be fed to the generator.

Now:

- $\mathcal{D}(\cdot, \theta_d): \mathbb{R}^d \to [0,1]$ gives as output the posterior for the input be a true image
- $\mathcal{G}(\cdot, \theta_d): \mathbb{R}^n \to \mathbb{R}^n$ gives as output the generated image

θ_d, θ_g
(d = discriminator, g = generator)

$\vec{z}$ = random noise

$\vec{s}$ = image (generated / real)

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

GAN Training

A good discriminator is such:

- $\mathcal{D}(\boldsymbol{s}, \theta_d)$ is maximum when $\boldsymbol{s} \in S$ (true image from the training set)
- $1 - \mathcal{D}(\boldsymbol{s}, \theta_d)$ is maximum when $\boldsymbol{s}$ was generated from $\mathcal{G}$
- $1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)$ is maximum when $\boldsymbol{z} \sim \phi_Z$

Training $\mathcal{D}$ consists in maximizing the binary cross-entroy

$$\max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))]\right)$$

for training $\mathcal{D}$

G. Boracchi

GAN Training

A good discriminator is such:

- $\mathcal{D}(\boldsymbol{s}, \theta_d)$ is maximum when $\boldsymbol{s} \in S$ (true image from the training set)
- $1 - \mathcal{D}(\boldsymbol{s}, \theta_d)$ is maximum when $\boldsymbol{s}$ was generated from $\mathcal{G}$
- $1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)$ is maximum when $\boldsymbol{z} \sim \phi_Z$

Training $\mathcal{D}$ consists in maximizing the binary cross-entroy

$$\max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))]\right)$$

⇩ This has to be 1 since $s \sim \phi_S$, thus images are real

⇩ This has to be 0 since $\mathcal{G}(\boldsymbol{z}, \theta_g)$ is a generated (fake) image

G. Boracchi

GAN Training

A good discriminator is such:

- $\mathcal{D}(\boldsymbol{s}, \theta_d)$ is maximum when $\boldsymbol{s} \in S$ (true image from the training set)
- $1 - \mathcal{D}(\boldsymbol{s}, \theta_d)$ is maximum when $\boldsymbol{s}$ was generated from $\mathcal{G}$
- $1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)$ is maximum when $\boldsymbol{z} \sim \phi_Z$

Training $\mathcal{D}$ consists in maximizing the binary cross-entroy

$$\max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))]\right)$$

A good generator $\mathcal{G}$ makes $\mathcal{D}$ to fail, thus minimizes the above

$$\min_{\theta_g} \boxed{\max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))]\right)}$$

for training $\mathcal{G}$ (which want $\mathcal{D}$ "to fail", because $\mathcal{G}$ wants to fool $\mathcal{D}$ ⇒ the purpose of $\mathcal{G}$ is to minimize what $\mathcal{D}$ is trying to optimize: min [max ...])

G. Boracchi

GAN Training

Solve by an iterative numerical approach

$$\min_{\theta_g} \max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))]\right)$$

(take one parameter fixed and minimize w.r.t. the other, etc)

G. Boracchi

GAN Training

Solve by an iterative numerical approach

$$\min_{\theta_g} \max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))]\right)$$

Alternate:

- k-steps of Stochastic Gradient Ascent w.r.t. θ_d to solve

$$\max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right]\right) \quad (\mathcal{D})$$

- 1-step of Stochastic Grandient ~~Ascent~~ Descent w.r.t. θ_g being θ_d fixed

$$\min_{\theta_g}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right]\right) \quad (\mathcal{G})$$

G. Boracchi

GAN Training

Solve by an iterative numerical approach

$$\min_{\theta_g} \max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))]\right)$$

Alternate:

- k-steps of Stochastic Gradient Ascent w.r.t. θ_d to solve

$$\max_{\theta_d}\left(\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z \sim \phi_Z}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right]\right)$$

- 1-step of Stochastic Grandient ~~Ascent~~ w.r.t. θ_g being θ_d fixed

$$\min_{\theta_g}\left(\cancel{\mathrm{E}_{s \sim \phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)]} + \mathrm{E}_{z \sim \phi_Z}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right]\right)$$

and since the first term does not depend on θ_g, this consists in maximizing

$$\max_{\theta_g}\left(\mathrm{E}_{z \sim \phi_Z}\left[\log\left(\mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right]\right) \quad (\mathcal{G})$$

G. Boracchi

GAN Training

Solve by an iterative numerical approach

$$\min_{\theta_g} \max_{\theta_d} \left(\mathrm{E}_{s\sim\phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z\sim\phi_Z}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d))] \right)$$

Alternate:

- k-
- 1-step of Stochastic Grandient Ascent w.r.t. θ_g being θ_d fixed

$$\min_{\theta_g} \left(\mathrm{E}_{s\sim\phi_S}[\log \mathcal{D}(\boldsymbol{s}, \theta_d)] + \mathrm{E}_{z\sim\phi_Z}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right] \right)$$

and since the first term does not depend on θ_g, this consists in maximizing

$$\max_{\theta_g} \left(\mathrm{E}_{z\sim\phi_Z}\left[\log\left(\mathcal{D}(\mathcal{G}(\boldsymbol{z}, \theta_g), \theta_d)\right)\right] \right)$$

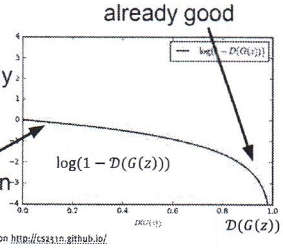
Maximizing w.r.t θ_g (instead of minimizing as before) does not modify the global solution of the min-max problem, but provides a stronger gradient during the early learning stages

G. Boracchi

Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data [ndr thus $D(G(z)) \approx 0$. In this case, $\log(1 - D(G(z)))$ saturates.

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!

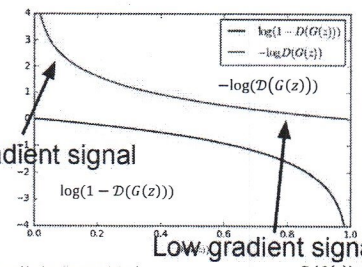


from the paper

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

Rather than training G to minimize $\log(1 - D(G(z)))$ we can train G to maximize $\log(\mathcal{D}(G(z)))$ [ndr or as in this figure, minimize $-\log(\mathcal{D}(G(z)))$] This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

High gradient signal



Low gradient signal

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

GAN Training

for $i = 1 \ldots$ #number of epochs

for k –times # gradient ascent steps for θ_d

- Draw a minibatch $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ of noise realization
- Sample a minibatch of images $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m\}$
- Update $\mathcal{D}$ by stochastic gradient ascend:

$$\nabla_{\theta_d}\left[\sum_i \log \mathcal{D}(\boldsymbol{s}_i, \theta_d) + \log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}_i, \theta_g), \theta_d)\right)\right]$$

Draw a minibatch $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ of noise realizations # gradient descent steps for θ_g

Update $\mathcal{G}$ by stochastic gradient descent:

$$\nabla_{\theta_g}\left[\sum_i \log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}_i, \theta_g), \theta_d)\right)\right]$$

!

G. Boracchi

GAN Training

This was presented as a best practice, latest GANs such as Wasserstein GANs do not use this

for $i = 1 \ldots$ #number of epochs

for k –times # gradient ascent steps for θ_d

- Draw a minibatch $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ of noise realization
- Sample a minibatch of images $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m\}$
- Update $\mathcal{D}$ by stochastic gradient ascend:

$$\nabla_{\theta_d}\left[\sum_i \log \mathcal{D}(\boldsymbol{s}_i, \theta_d) + \log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}_i, \theta_g), \theta_d)\right)\right]$$

Draw a minibatch $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ of noise realizations # gradient descent steps for θ_g

Update $\mathcal{G}$ by stochastic gradient descent:

$$\nabla_{\theta_g}\left[\sum_i \log\left(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}_i, \theta_g), \theta_d)\right)\right]$$

G. Boracchi

Algorithm outline

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Goodfellow, I. et al "Generative adversarial nets" NIPS 2014

GAN Training

for $i = 1 \ldots$

for k −times

- Draw a minibatch $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ of noise realizations
- Sample a minibatch of images $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m\}$
- Update $\mathcal{D}$ by stochastic gradient ascend:

$$\nabla_{\theta_d}\left[\sum_i \log \mathcal{D}(\boldsymbol{s}_i, \theta_d) + \log\left(1 - \mathcal{D}\big(\mathcal{G}(\boldsymbol{z}_i, \theta_g), \theta_d\big)\right)\right]$$

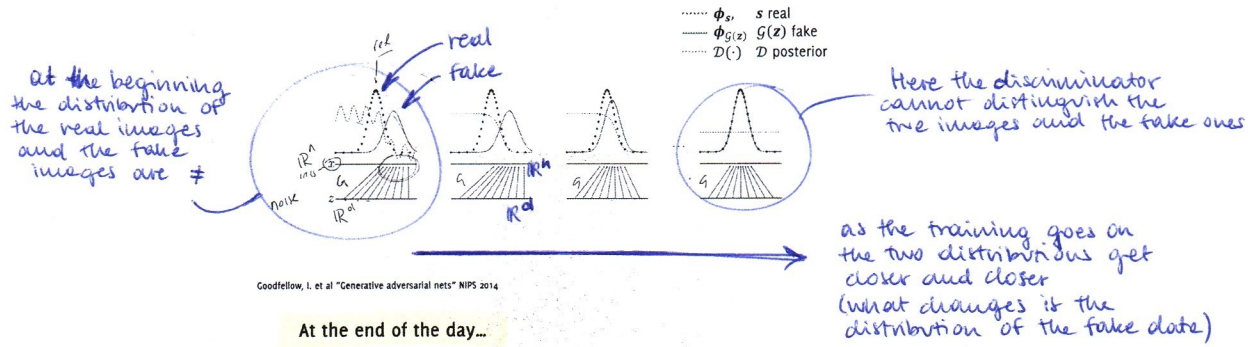
Draw a minibatch $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m\}$ of noise realization

Update $\mathcal{G}$ by stochastic gradient **ascent:**

$$\nabla_{\theta_g}\left[\sum_i \log\left(\mathcal{D}\big(\mathcal{G}(\boldsymbol{z}_i, \theta_g), \theta_d\big)\right)\right]$$

G. Boracchi

Illustration



Goodfellow, I. et al "Generative adversarial nets" NIPS 2014

At the end of the day...

The discriminator $\mathcal{D}$ is discarded

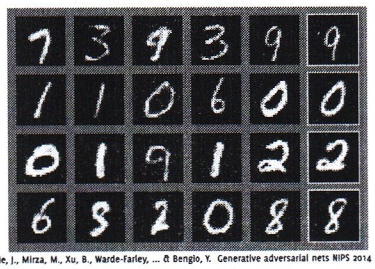
The generator $\mathcal{G}$ and ϕ_Z are preserved as generative model

Remarks:

- **The training is rather unstable**, need to carefully synchronize the two steps (many later works in this direction, e.g. Wasserstein GAN)
- **Training by standard tools**: backpropagation and dropout
- Theoretical results provided
- **Generator does not use S directly during training**
- **Generator performance is difficult to assess quantitatively**
- There is **no explicit expression for the generator**, it is provided in an implicit form -> you cannot compute the likelihood of a sample w.r.t. the learned GAN

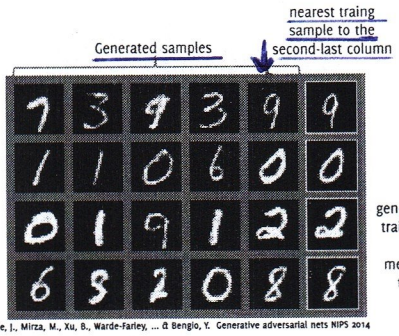
G. Boracchi

MNIST



Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

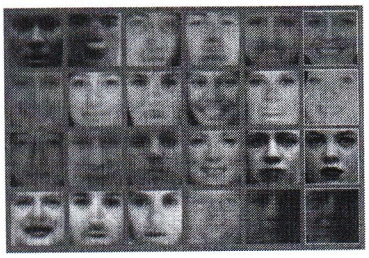
MNIST



This GAN generates realistic training samples without memorizing the training set

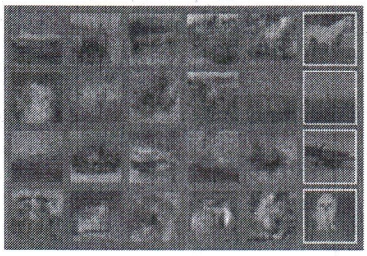
Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

Toronto Face Database (TFD)



Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

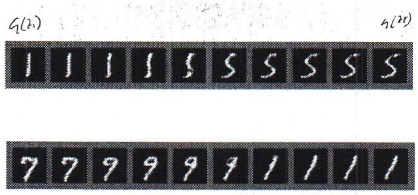
CIFAR-10



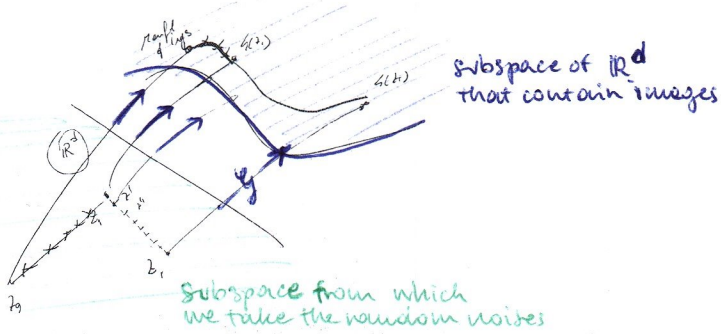
Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014

Outputs of interpolated trajectories

Select two noise realization z_1 and z_2 yielding reasonable outputs, and interpolate among the two. Generate the images of intermediate values

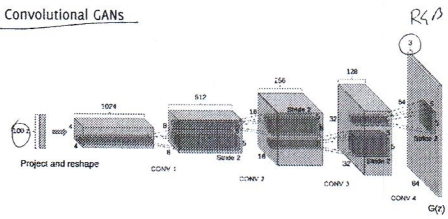


Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, ... & Bengio, Y. Generative adversarial nets NIPS 2014



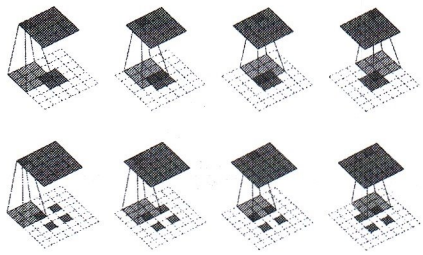
GANs have much improved in the last few years

Deep Convolutional GANs



Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

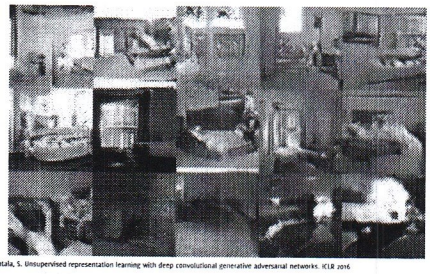
How is it possible to "increase the network" size?



Dumoulin V., Visin F. - *A guide to convolution arithmetic for deep learning*

GANs have much improved in the last few years

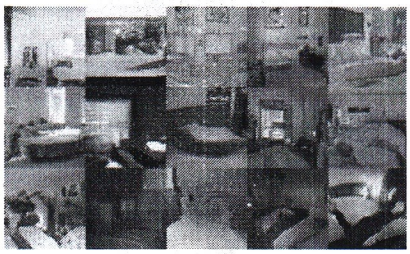
Images generated after 1 training epochs



Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

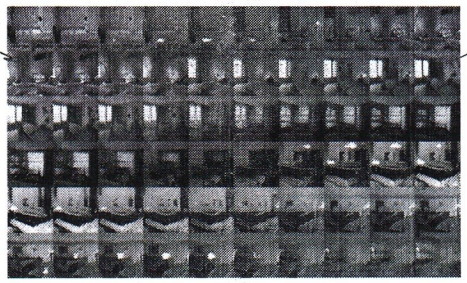
GANs have much improved in the last few years

Images generated after 5 training epochs



Radford, A., Metz, L., & Chintala, S. Unsupervisnd representation learning with deep convolutional generative adversarial networks. ICLR 2016

Interpolation between a series of 9 random points



Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

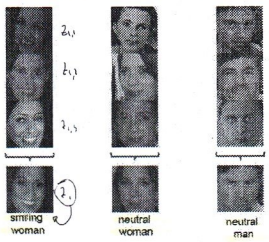
Vector Arithmetic

Take randomly generated samples of smiling women, neutral women, and neutral men

Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

Vector Arithmetic

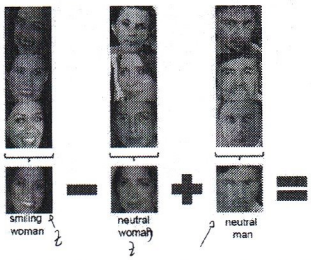
Average the corresponding noise seeds



Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

Vector Arithmetic

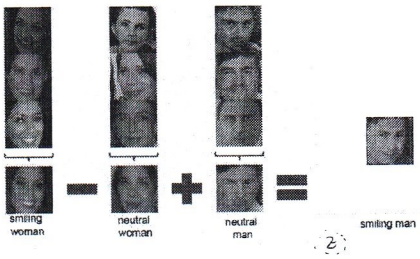
Perform some arithmetic



Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

Vector Arithmetic

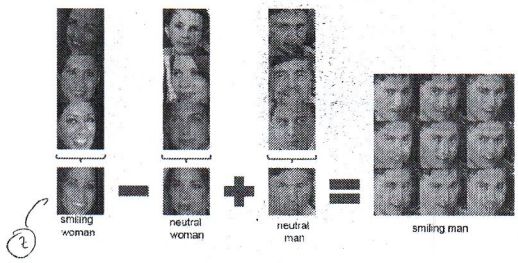
Perform some arithmetic



Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

Vector Arithmetic

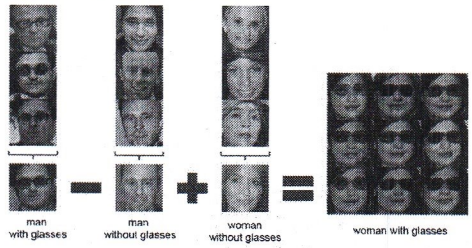
Add some noise to the input vector and that's pretty robust



Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

Vector Arithmetic

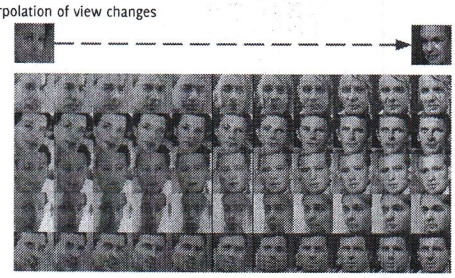
Similar example as word embedding



Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

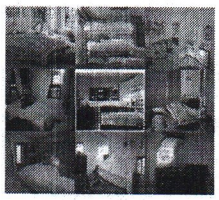
Vector Arithmetic

Interpolation of view changes

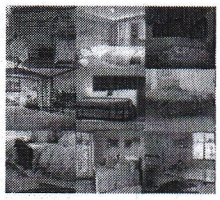


Radford, A., Metz, L., & Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. ICLR 2016

GAN has been a very active research field



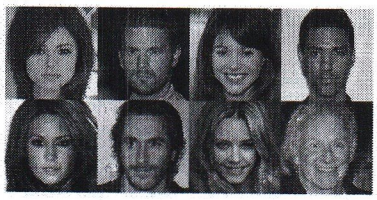
LSGAN, Zhu 2017.

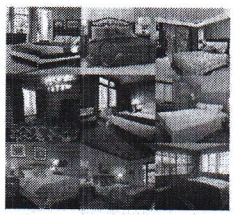


Wasserstein GAN, Arjovsky 2017.
Improved Wasserstein GAN, Gulrajani 2017.

CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/

GAN has been a very active research field

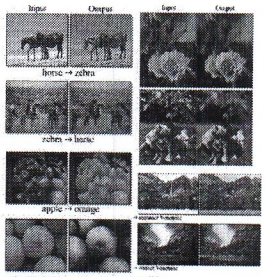




Karras, Tero, et al. "Progressive growing of gans for improved quality, stability, and variation." *ICLR 2018*

GANs for source-target domain transfer

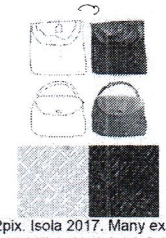
This can also be used for photo enhancement and data augmentation



CycleGAN. Zhu et al. 2017.

... «The GAN ZOO» and https://github.com/soumith/ganhacks

Reed et al. 2017.



Pix2pix. Isola 2017. Many examples at https://phillipi.github.io/pix2pix/

G. Boracchi

A few project and thesis opportunities

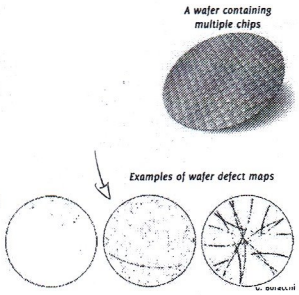
Updated thesis opportunities and furthre information on https://boracchi.faculty.polimi.it/teaching.html

Wafer Defect Maps Monitoring

Semiconductor manufacturing is a **long, complex and expensive** process involving several specialized steps

Inspection tools provide a **huge amount of data**, that can be entirely analysed only by **Deep Learning** techniques

Deep Learning enables the **automatic** monitoring of the production lines, **identifying defective patterns**. The results of this procedure allow to discard defective products and might promptly reveal failures of the machines



Stage ST1: Wafer Defect Maps Monitoring

Currently, our automatic solution based on a CNN is classifying WDMs on the major production lines of STMicroelectronics site in Agrate. Future research will address

- The study of **domain adaptation** techniques to handle data coming from different production processes
- Design a solution to **identify the type of malfunctioning** and localize the faulty machinery in the production pipeline (metric learning)
- The development of an **anomaly-detection network** to identify never seen defective patterns

Materials and methods:

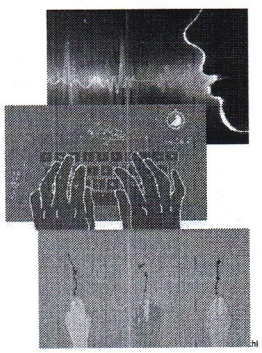
- Annotated WDM dataset
- Support from ST's engineers
- Reference algorithm based on sparse convolutions

G. Boracchi

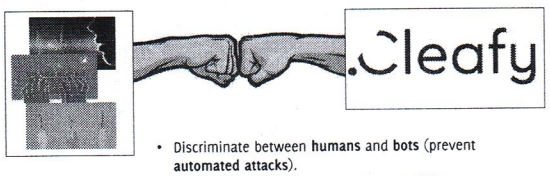
Behavioral biometrics

Behavioral biometrics is the study related to **uniquely identifiable** and **measurable patterns** in human **activities**.

Behavioral biometric verification methods include **keystroke dynamics, gait analysis, voice ID, mouse use characteristics, signature analysis** and **cognitive biometrics**.



Behavioral biometrics for fraud detection



- Discriminate between **humans** and **bots** (prevent **automated attacks**).
- Identify **fraudsters** (previously seen **malicious behaviors**).
- Predict the **credentials** through which the user will **authenticate** himself (likelihood of being a **specific known user**).
- Corroborate the identity of an **authenticated user**.

G. Boracchi

Thesis: User Identification

Goal: Design and prototype an algorithm able to **verify** the user **identity** through his **behavioral biometrics** during a navigation session.

- Getting familiar with **state-of-the-art** behavioral biometrics verification **methods.**
- Robustly **profile users** through the various aforementioned methods.
- **Continuously** (and **frictionless**) **verify** the users **identities** during sessions.

Materials and Methods:

- Access to a server mounting GPUs will be provided.
- Anonymized data acquired from a large database of users.

Collaboration with

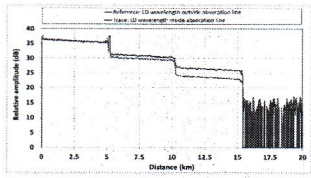
Cleafy

racchi

Stage CISCO: OTDR events recognition

OTDR (optical time domain reflectometer) is a technique to analyse a portion of an optical fiber.

The output of OTDR is a signal representing the optical power (in dB) as a function of the distance (in Km) along the fiber.



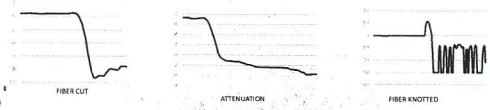
CISCO

G. Boracchi

Stage CISCO: OTDR events recognition

Every span of optical fiber in the field features its own characteristic. ODTR signal has the potential to identify whether some event has occurred in the fiber span.

In practice, by analyzing OTDR signal it is posible to identify at which location in the fiber there has been a cut, a strong attenuation, the presence of reflection sources, or concentrated losses, to name a few examples:



CISCO

G. Boracchi

Stage CISCO: OTDR events recognition

Goals:

- Implement a deep learning model to detect (multiple) patterns inside an ODTR signal. The model has to be 1-dimensional and able to cope with input signals of different dimensions.
- Define realistic data augmentation procedures in collaboration with CISCO engineers and scientists.
- Data set manipulation and preparation.
- Comparison among different solutions (expert-driven vs data-driven)
- Porting the final model on a real embedded systems.

Materials and Methods:

- OTDR measurements from real fiber deployed in multiple sites.
- CISCO engineers will provide all the guidance and equipments needed.

CISCO

G. Boracchi

A Laboratory Course (but not on deep learning)

Next year, second semester (hopefully)

If you are planning to be here in a year...

Laboratory of Mathematical Models and Techniques for Image and Signal Processing (5 CFU)

Spring AA 2021 – 2022, Laboratory under preparation for Mathematical Engineers (and hopefully Computer Science Engineers)

*The course presents **mathematical models underpinning** several **successful image/signal processing and pattern recognition algorithms**, including the solution of inverse problems and fitting in noisy data. Particular emphasis will be given to models describing images as elements of a vector space, and the implications of linear algebra theory in imaging. These include transformed domain methods and other techniques to solve inverse problems.*

G. Boracchi

Restoration & Inverse Problems

Dabov, K., Foi, A., Katkovnik, V., & Egiazarian, K. Image denoising by sparse 3-D transform-domain collaborative filtering. *TIP 2007*

G. Boracchi

Restoration & Inverse Problems

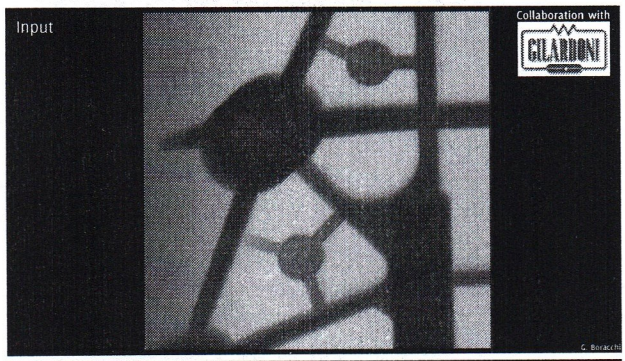
Dabov, K., Foi, A., Katkovnik, V., & Egiazarian, K. Image denoising by sparse 3-D transform-domain collaborative filtering. *TIP 2007*

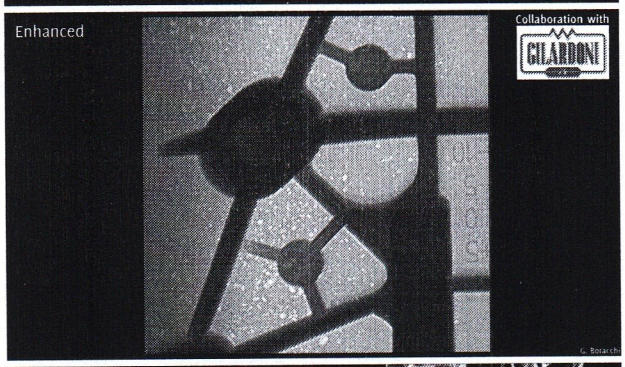
G. Boracchi

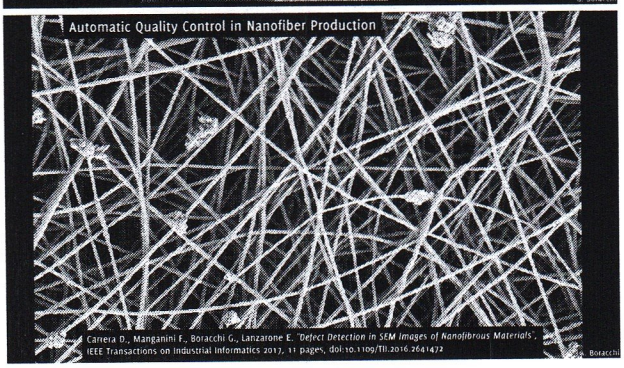
Filtering

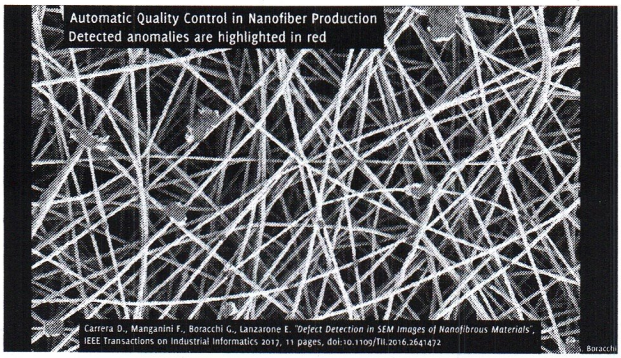
LASIP c/o Tampere University of Technology http://www.cs.tut.fi/~lasip/

G. Boracchi

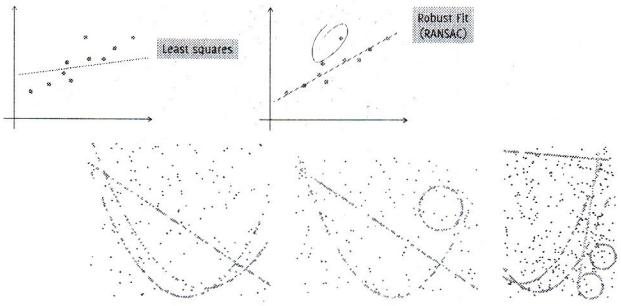






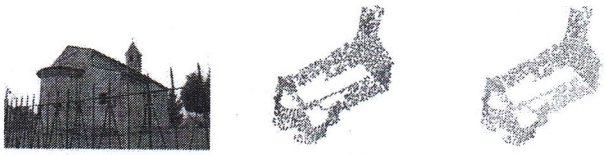


Model Fitting



Model Fitting

Model Fitting



Magri, Luca, and Andrea Fusiello. "Fitting Multiple Heterogeneous Models by Multi-Class Cascaded T-Linkage." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

G. Boracchi

Learning Sparse Represenations for Image and Singal Modeling

a PhD Course open to MSc Students
on Febrauary 2021

A PhD Course open to MSc Students on Febrauary 2021

The main goal of this course is to provide the student with an understanding of the most important aspects of the theory underlying sparse representation and, more in general, of sparsity as a form of regularization in learning problems. Students will have the opportunity to develop and understand the main algorithms for learning sparse models and computing sparse representations. These methods have wide applicability in computer science, and these will be a useful background for their research.

More information on the Course program page

https://boracchi.faculty.polimi.it/teaching/LearningSparse.htm
(dates to be updated)

G. Boracchi

Dates

The course consists of traditional classes and computer-laboratory hours. During computer laboratory, students will be guided in the implementation of simplified versions of the presented algorithms. Imaging problems, including denoising, inpainting and anomaly detection, will be considered as running examples through the computer labs. Matlab is the preferred programming language (but you are of course allowed to use Python) and students are invited to work on their own laptop

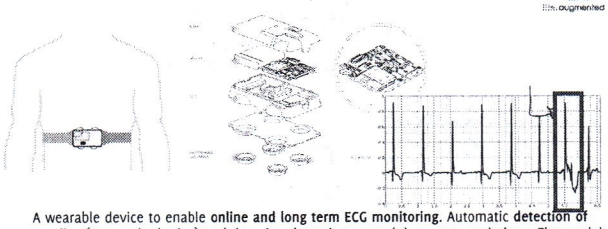
- 02/02/2021 from 14:00 til 18:00
- 05/02/2021 from 14:00 til 19:00
- 09/02/2021 from 14:00 til 18:00
- 12/02/2021 from 14:00 til 18:00
- 16/02/2021 from 14:00 til 18:00
- 19/02/2021 from 14:00 til 18:00

G. Boracchi

An Example of algorithm we will see as an application: AD for Long Term ECG monitoring in wearables

Collaboration with

life.augmented



A wearable device to enable **online and long term ECG monitoring**. Automatic **detection of anomalies** (e.g., arrhythmias) and **domain adaptation** to track heart rate variations. The model is based on **a learned and user-specific model** of heartbeats.

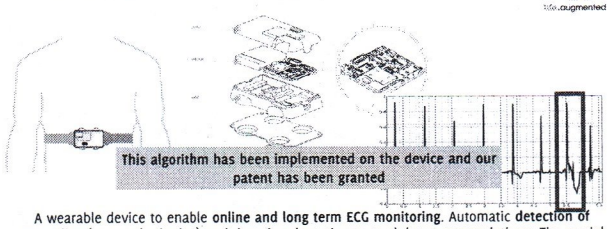
IJCAI Demo webpage: http://home.deib.polimi.it/carrerad/IJCAI_2018_Demo.html

G. Boracchi

An Example of algorithm we will see as an application: AD for Long Term ECG monitoring in wearables

Collaboration with

life.augmented



This algorithm has been implemented on the device and our patent has been granted

A wearable device to enable **online and long term ECG monitoring**. Automatic **detection of anomalies** (e.g., arrhythmias) and **domain adaptation** to track heart rate variations. The model is based on **a learned and user-specific model** of heartbeats.

IJCAI Demo webpage: http://home.deib.polimi.it/carrerad/IJCAI_2018_Demo.html

G. Boracchi