

Deep Learning for object detection: fast and accurate results with the YOLO method

David Cornu

IRMIA Deep Learning Summer School 2022



MINERVA



Lessons materials

Slides and Google Colab notebook are available on GitHub :

https://github.com/Deyht/IRMIA_2022

Useful commands for shell:

git clone https://github.com/Deyht/IRMIA_2022

git pull

The Jupyter notebook is to open in Google Colab with a GPU reservation.

(The notebook should work on other environment with some adjustments, untested)

All the data will be downloaded and saved in the temporary Colab environment

=> Download locally anything you want to keep on the long term
(for example network saves and figures)

Neural Networks for images

Fully connected networks has shown one weakness

→ **They are inefficient for handling images !**

- Images are highly dimensional (lots of pixels!)
- They have a very high degree of invariance
(mainly translation but also luminosity, color, rotation, ...).

Classical ANN can deal with images by considering each pixel of an image as an individual input but it is STRONGLY inefficient.

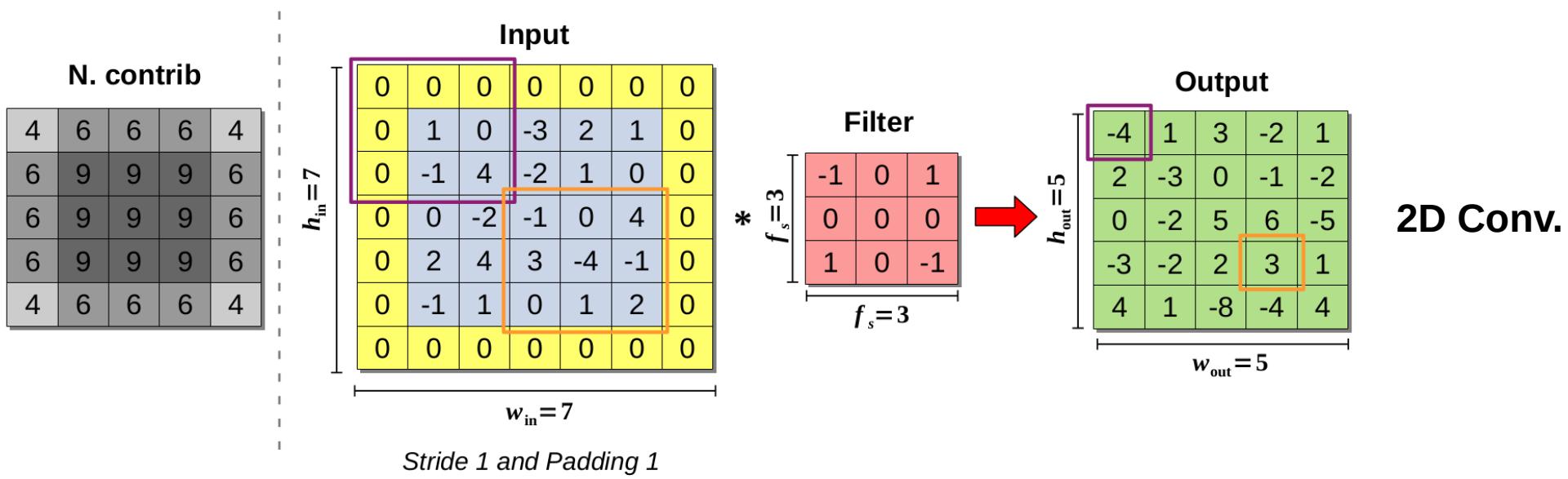
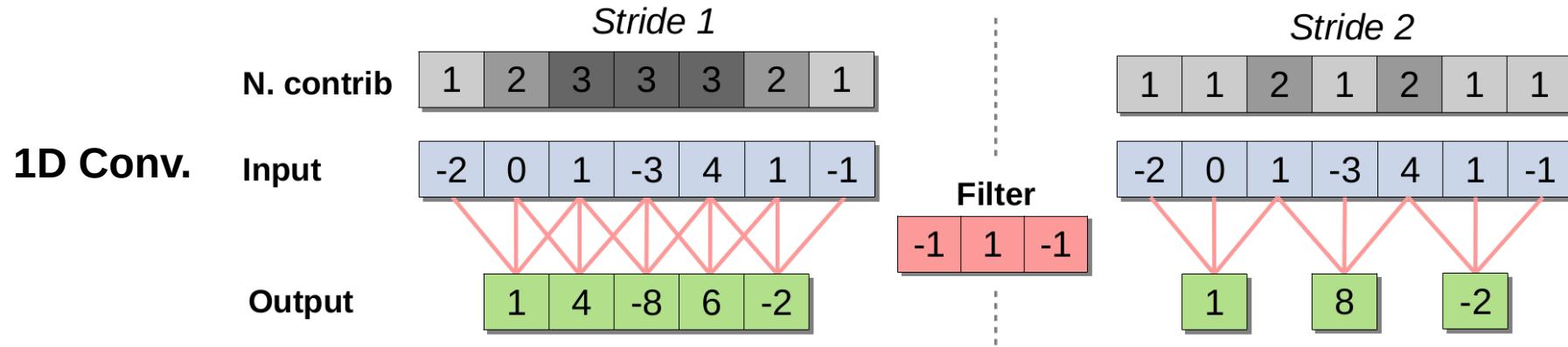
Driven by the computer vision and pattern recognition community these issues have found a solution in the 90s with:

→ **Convolutional Neural Networks !**



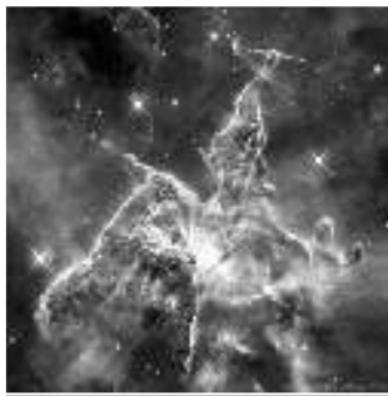
A **highly dimensional** "dog" with ~0.5 Million pixels.
Quite difficult to classify ...

Convolution filter



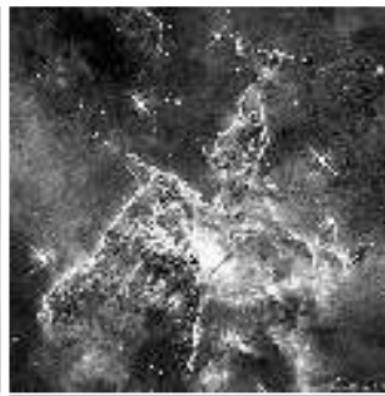
Filter effect examples

No filter



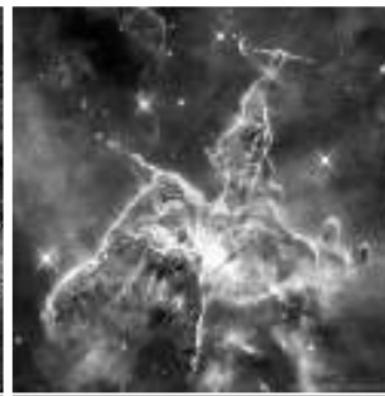
Sharpen

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



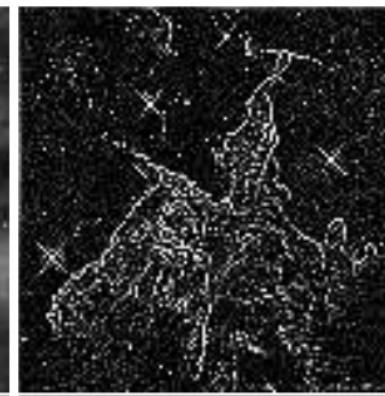
Gaussian blur

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



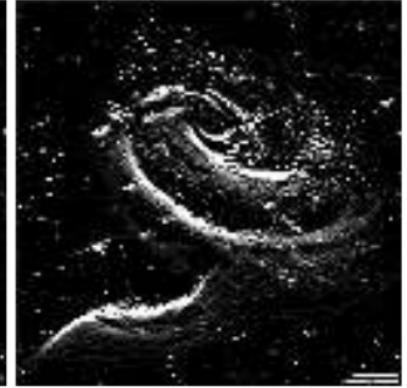
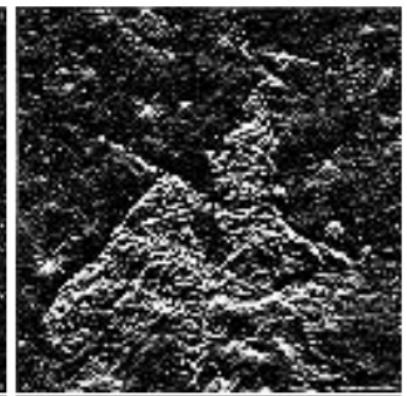
Edge detector

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

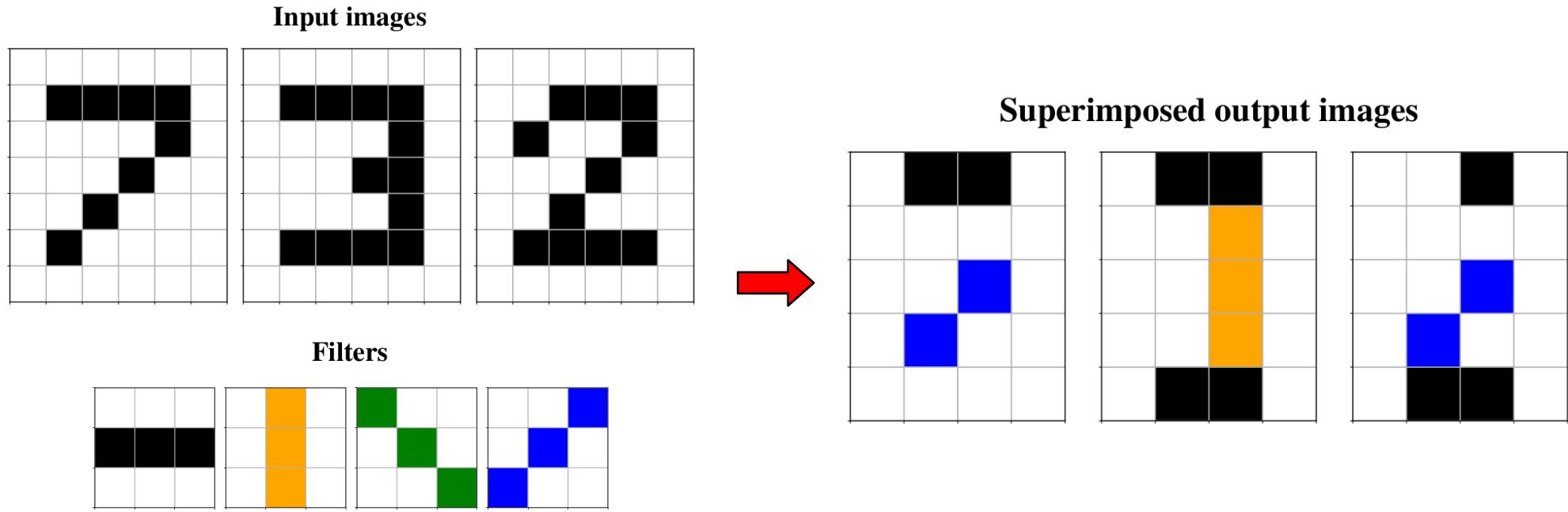


Axis elevation

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$



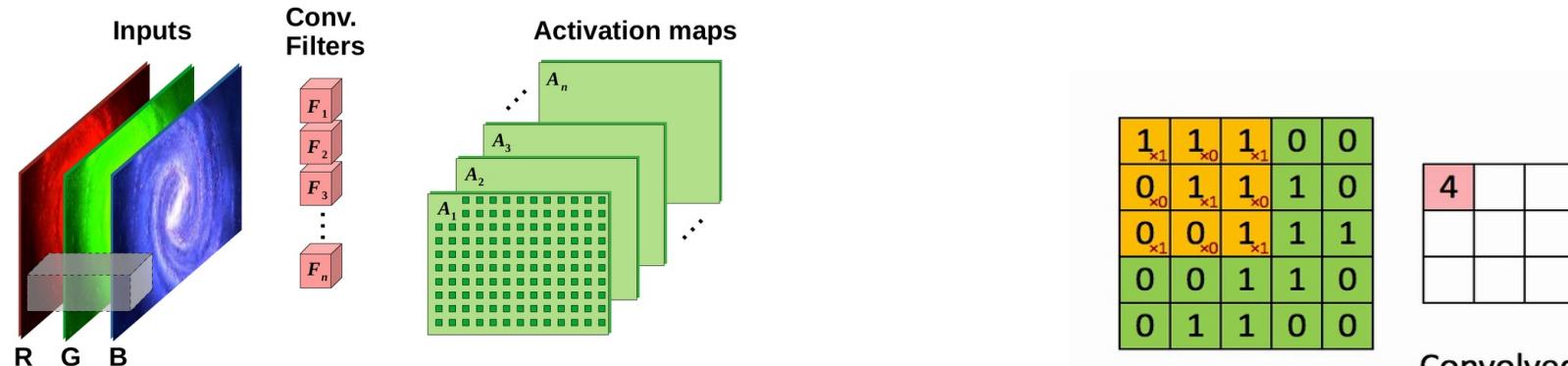
Pattern recognition with several filters



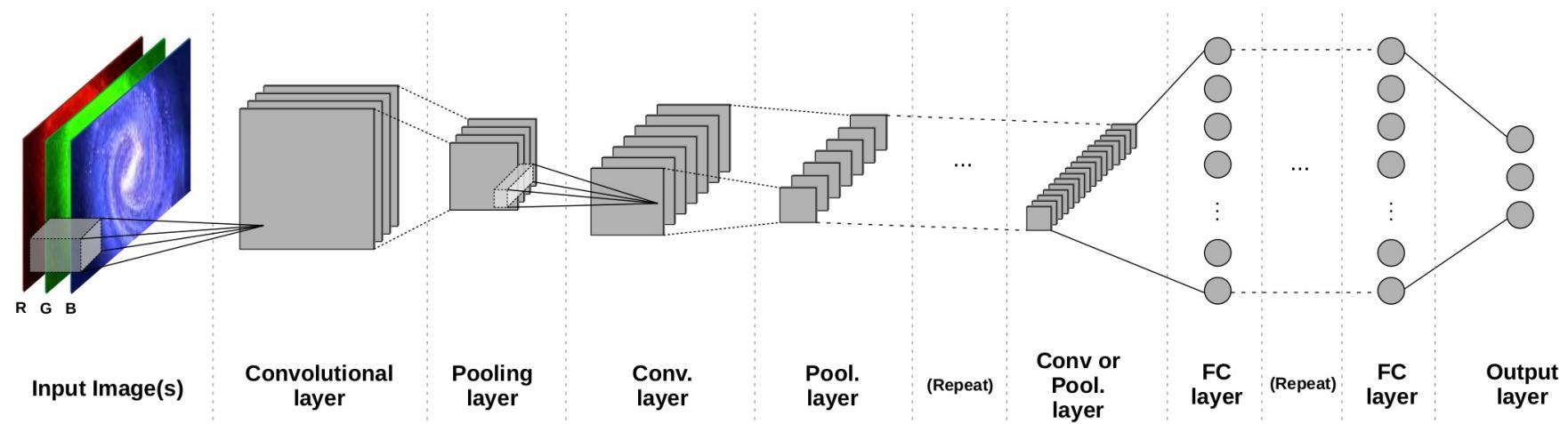
Each filter will produce its own activation map. Combining the information from different activation maps in a new convolutional layer (with a third dimension in the filter corresponding to the number of filters in the previous layer) allow to construct more and more complex patterns.

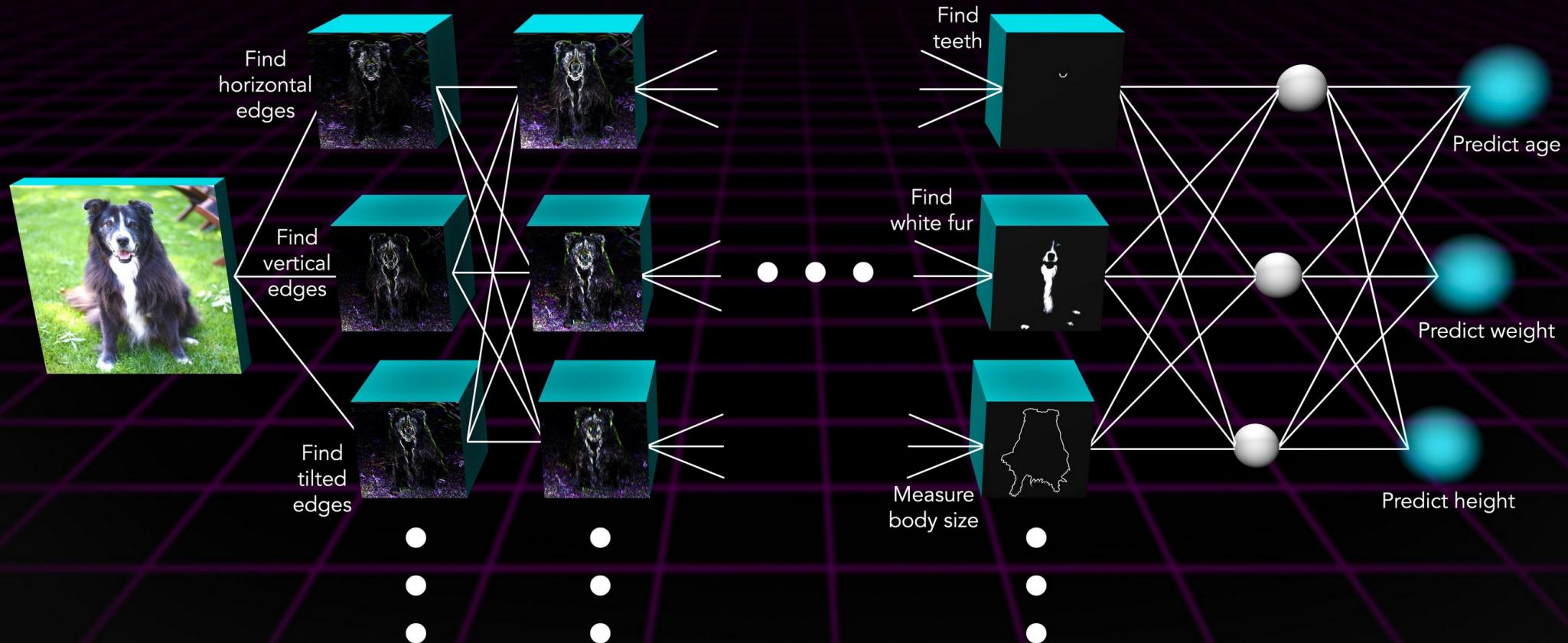
*Here the different activation maps are superimposed using color coding per filter

Convolutional Neural Networks

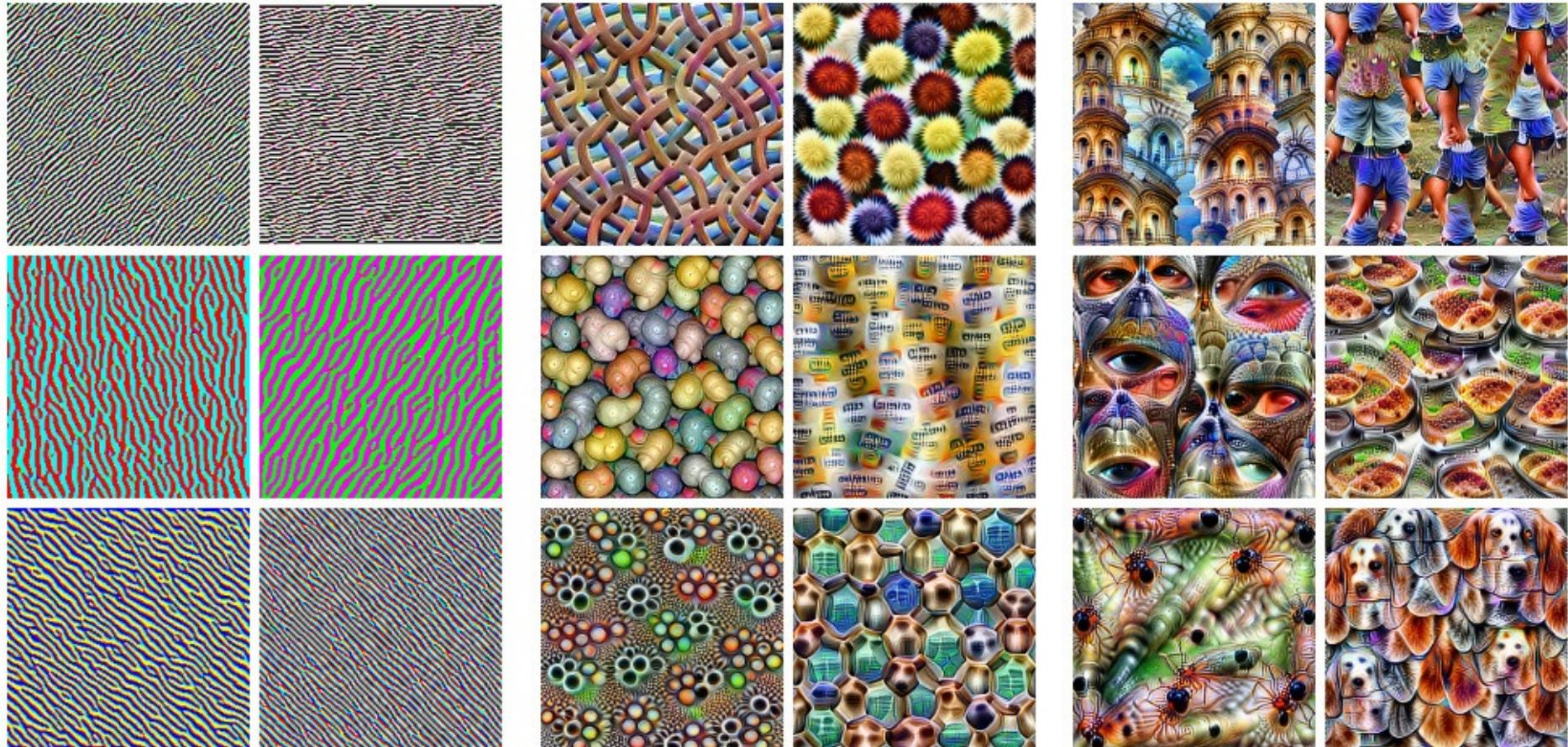


$$g \left(\sum_i X_i \circ W_i \right) = a$$





Examples of filter maximization



Edges (layer conv2d0)

Patterns (layer mixed4a)

Objects (layers mixed4d & mixed4e)

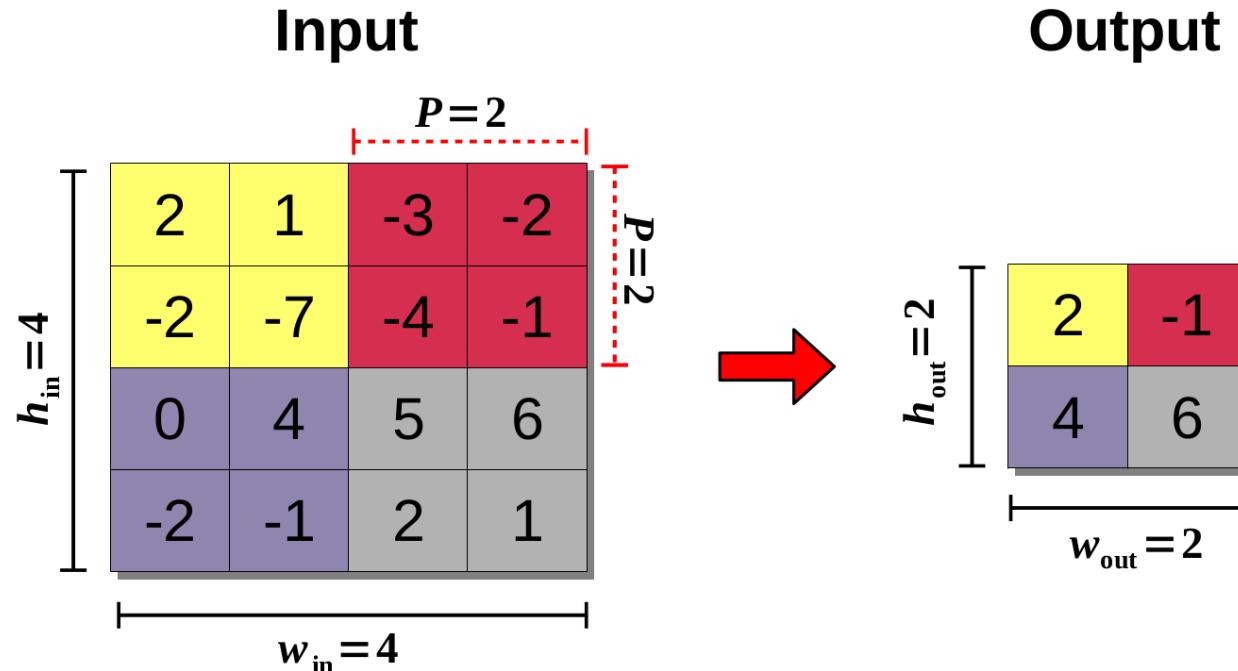
Example of input images that maximize specific filters activation at different depth in a classification network.

Dimensionality reduction: Pooling

A classical convolution operation is parameterized in a way that preserve the spatial dimensionality.

Still, it is most of the time necessary to **progressively reduce the “image” size**.

One way to reduce the dimensionality it to use **Pooling layers** !



Different pooling methods exist, the most common being **Max-Pooling** and **Average-Pooling**.

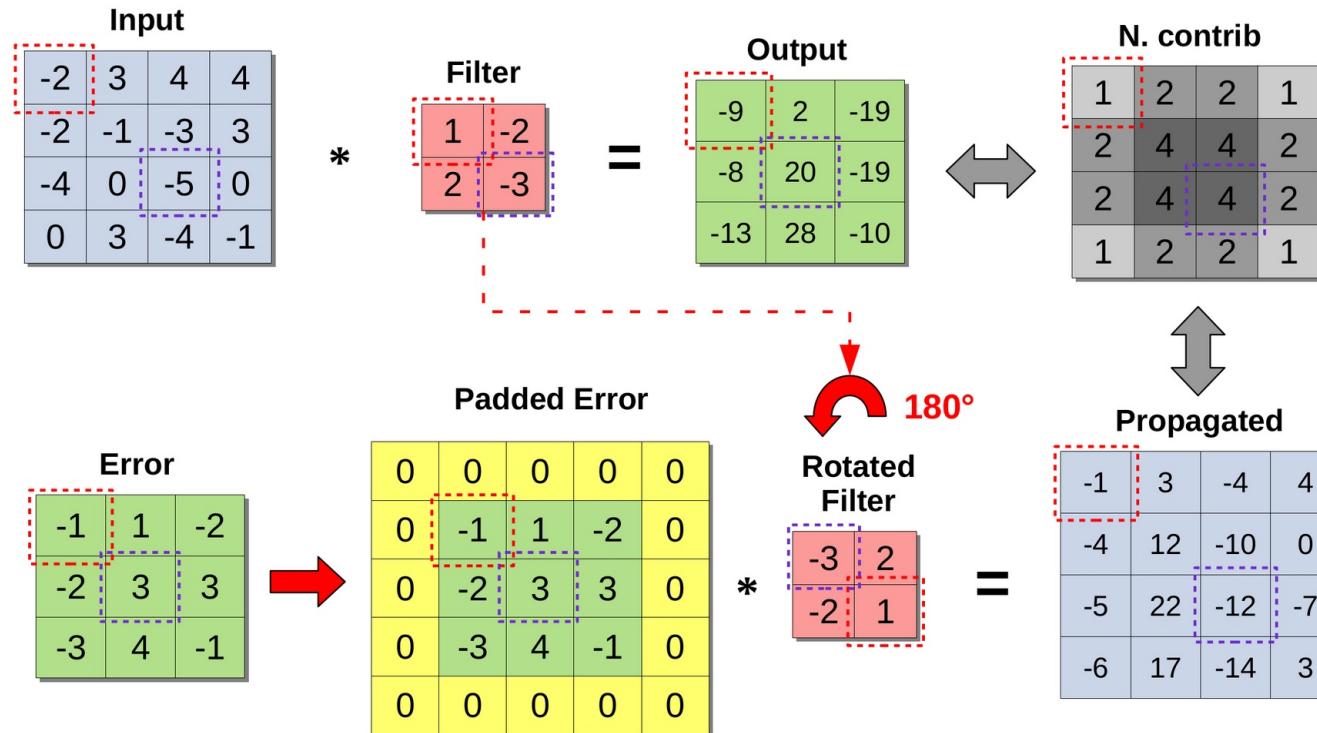
The pooling size can be parametrized, but most of the network architectures reduce each spatial dimension by a factor of two.

Learning the filters

Just like fully connected layers, the convolutional filters can be learned by using **backpropagation** of the error measured at the output layer.

In practice the error is propagated using a transposed convolution operation, which can be expressed with a classical convolution operation using some simple transformations on specific layer elements.

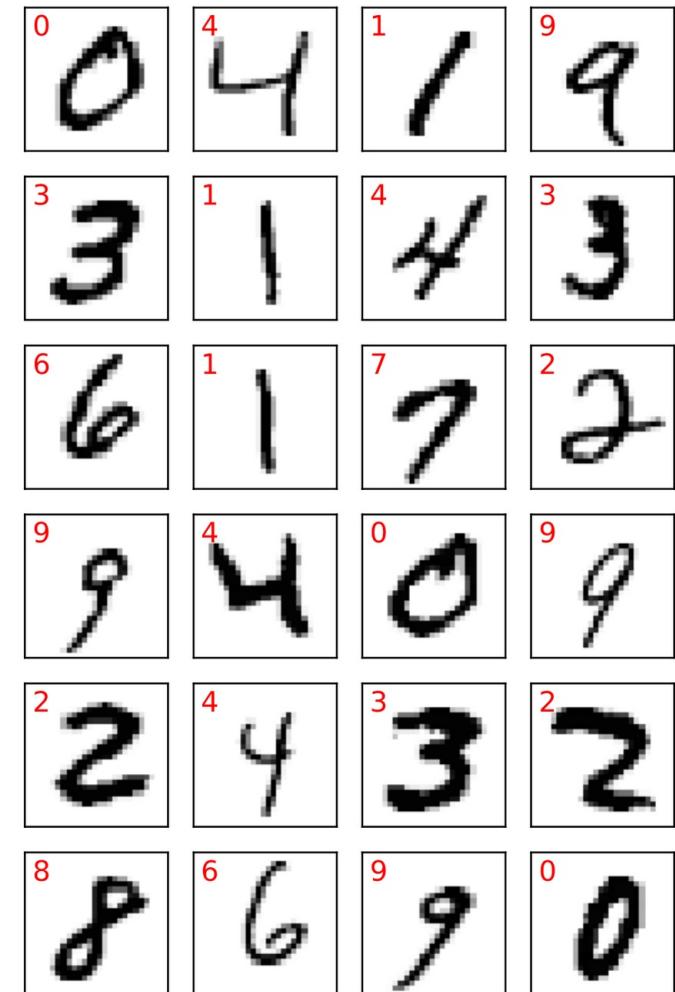
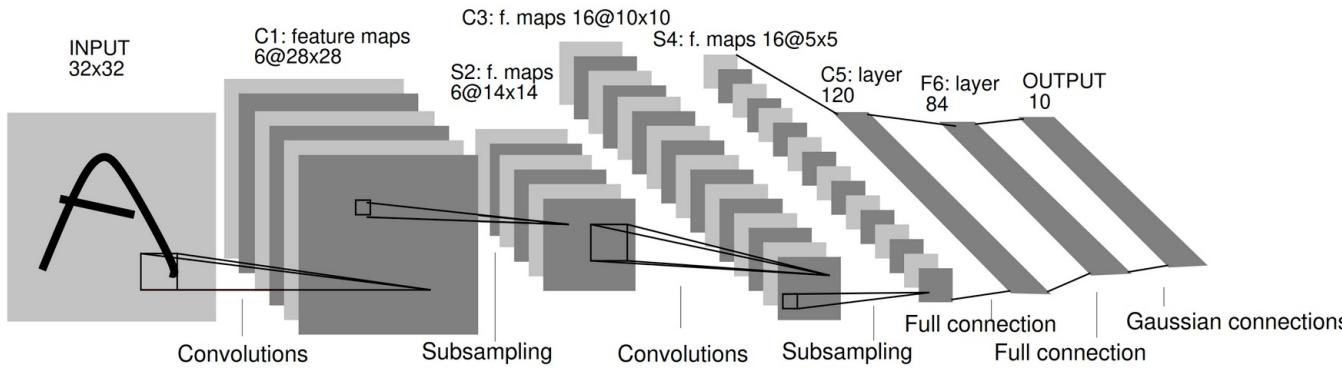
Learning the convolutional filters is the boundary at which most researchers define “**Deep Learning**”.



Simple examples: MNIST

The well known **MNIST** (Modified NIST's special dataset) dataset consist of handwritten digits from 500 different writers expressed as 28x28 grayscale images. It is freely accessible in the form of a 60000 image training dataset and a 10000 image test dataset.

Illustration of the [LeNet 5](#) network architecture (from LeCun et al. 2018) for this specific dataset.



Simple examples: MNIST

Actual

Class	Predicted										Recall
	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	
C0	976	0	1	0	0	0	1	1	1	0	99.6%
C1	0	1132	1	0	1	0	0	0	1	0	99.7%
C2	1	1	1027	0	1	0	0	1	1	0	99.5%
C3	0	0	1	1004	0	3	0	1	1	0	99.4%
C4	0	0	1	0	972	0	1	0	1	7	99.0%
C5	0	0	0	4	0	886	1	0	0	1	99.3%
C6	3	2	0	0	1	2	949	0	1	0	99.1%
C7	0	2	3	0	0	0	0	1020	1	2	99.2%
C8	0	0	1	1	0	1	1	1	968	1	99.4%
C9	0	0	0	0	3	1	0	4	0	1001	99.2%
Precision	99.6%	99.6%	99.2%	99.5%	99.4%	99.2%	99.6%	99.2%	99.3%	98.9%	99.35%

Obtained using CIANNA with the following architecture

=> I-28.28, C-6.5, P-2, C-16.5, P-2, C-48.3, D-1024_d0.5, D-256_d0.2, D10

All inner layers uses leaky ReLU activation (factor 0.1) and the output uses a Softmax activation.

The batch size is $b=64$, the learning rate is $\eta=2 \times 10^{-4}$ with a small decay up to $\eta=1 \times 10^{-4}$, and a momentum of $\alpha=0.9$.

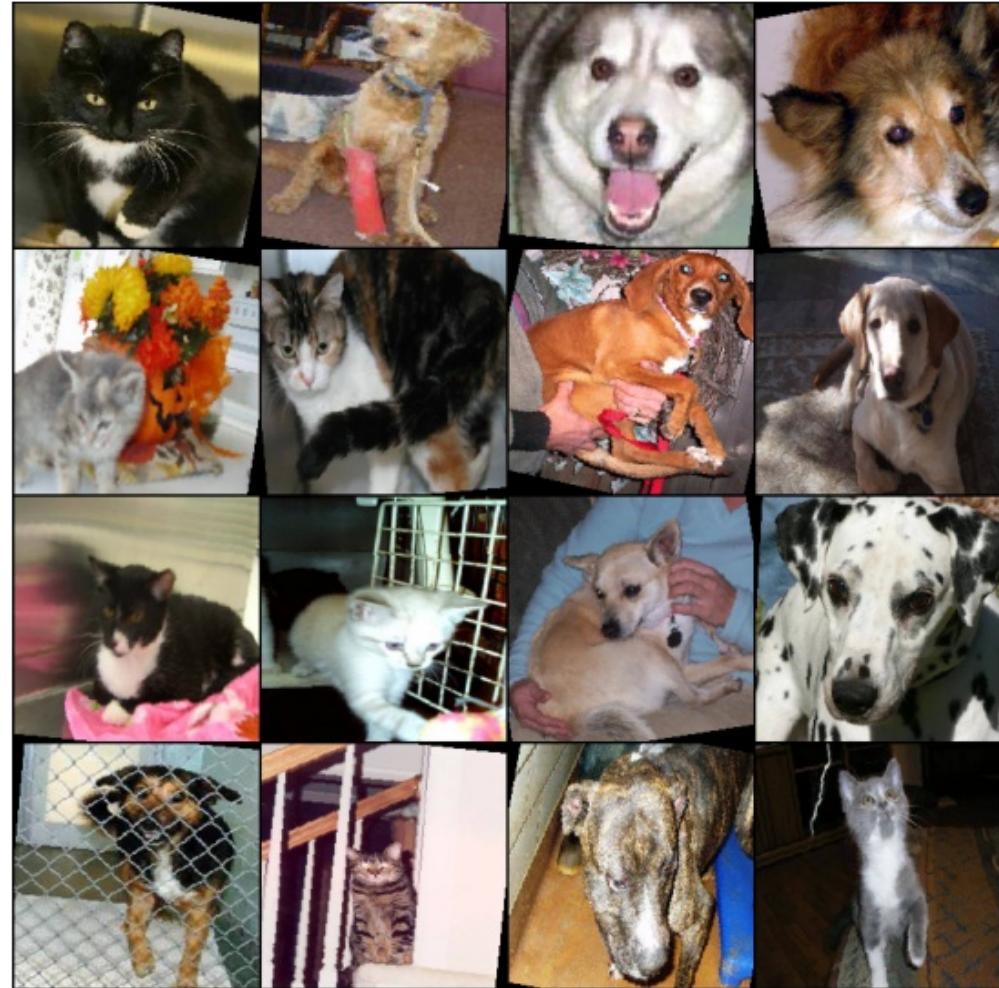
The network is trained for 40 epochs.

Simple examples: ASIRRA

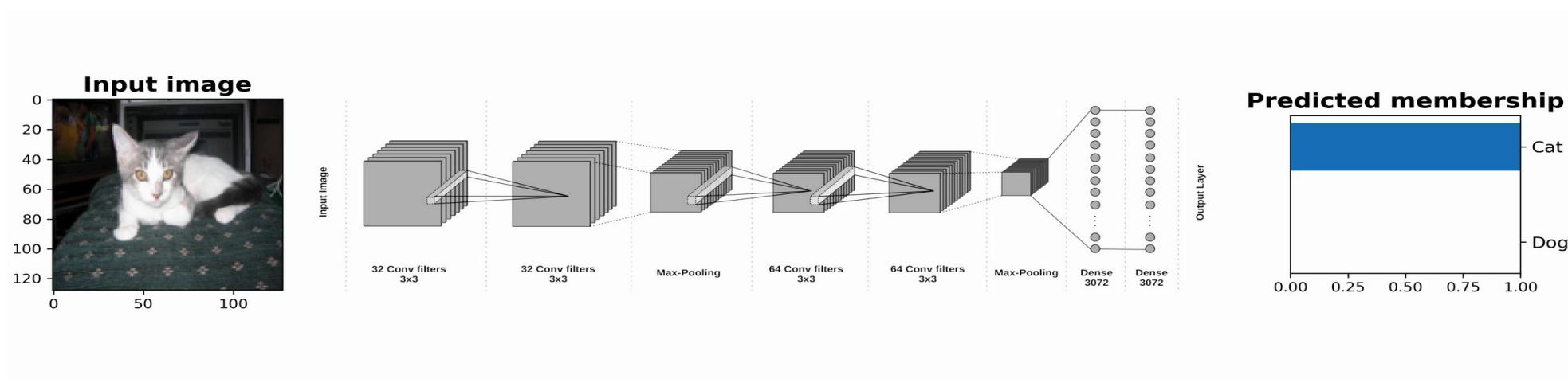
ASIRRA (Animal Species Image Recognition for Restricting Access): 25000 images, 50 % cats, 50 % dogs.

Typical of a **CAPTCHA** task or **HIP** (Human Interactive Proof) because the task is easy and quick for humans.

→ Nowadays modern CNN tools have more than **90% accuracy** on ASIRRA !



Simple examples: ASIRRA



Net. forward perf.: 13597.95 items/s
Cumulated error: 0.333725

ConfMat (valid set)

		Recall		
		925	75	92.50%
		68	932	93.20%
Precision :	93.15%	92.55%	Correct :	92.849998%

It is possible to obtain good results with relatively small networks (~90% accuracy), while more complex networks can get up to 97-98% accuracy.

Obtained using CIANNA with the following architecture =>
I-128.128.3, C-32.3, P-2, C-64.3, P-2, C-128.3, P2, C128.3, P2, D-512_d0.2, D2

All inner layers use leaky ReLU activation (factor 0.1) and the output uses a Softmax activation.

The batch size is $b=32$, the learning rate is $\eta=2 \times 10^{-4}$ with a small decay up to $\eta=5 \times 10^{-5}$, and a momentum of $\alpha=0.8$.

The network is trained for 20 epochs.

“Object detection” types

Classification



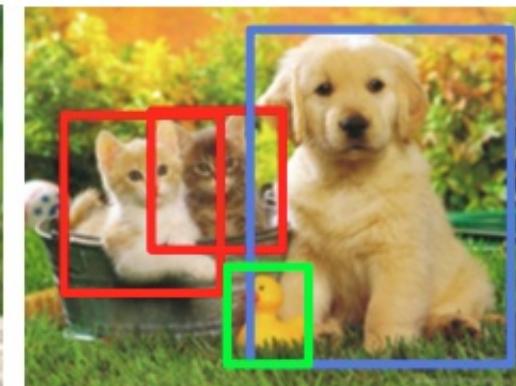
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation



CAT, DOG, DUCK

Our objective : multiple object detection



The PASCAL Visual Object Classes dataset(s)

Standardized image datasets for object class recognition organized in the form of annual challenges from 2005 to 2012



Targets for various “tasks” :

<http://host.robots.ox.ac.uk/pascal/VOC/index.html>

Classification



[Person, Horse]

Detection



List of bounding boxes
and their associated class

Segmentation



List of segmentation masks
and their associated class

The PASCAL Visual Object Classes dataset(s)

Standardized image datasets for object class recognition organized in the form of annual challenges from 2005 to 2012



Targets for various “tasks” :

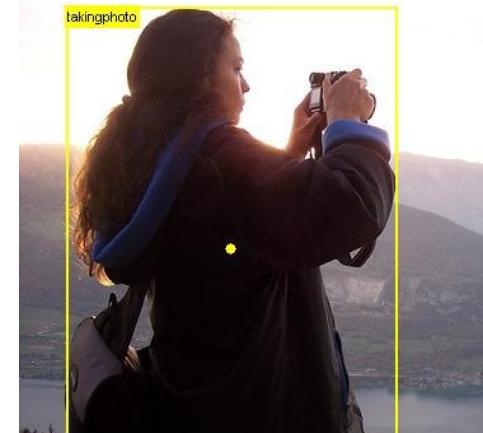
<http://host.robots.ox.ac.uk/pascal/VOC/index.html>

Boxless Action



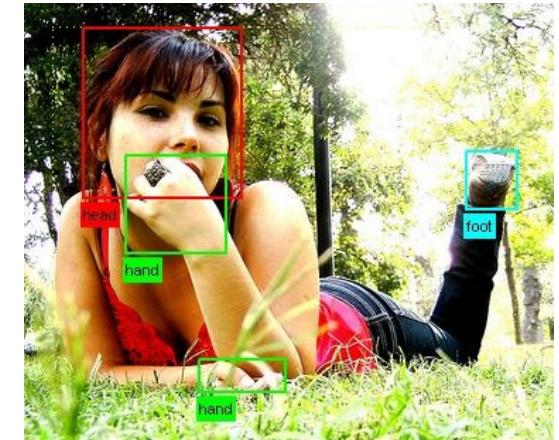
A single reference point and the associated action

Action



List of person boxes and their associated action

Person Layout



List of bounding boxes corresponding to body parts

Pascal detection task

Dataset construction and properties

VOC - 2005 : Standalone data

1578 Train (and 1293 Test) annotated images, detection task with 4 classes

VOC - 2006 : Standalone data

5618 Train (and 2686 Test) annotated images, detection task with 10 classes

VOC - 2007 : Standalone data

5011 Train (and 4952 Test) annotated images, detection task with 20 classes

VOC - 2012 : incremental from 2008 to 2012

11540 Train annotated images (test set hidden), detection task with 20 classes

One can **combine the Train and Test data from 2007 with the Train data from 2012** to obtain a larger training dataset of **21503 images, containing 52090 objects**

Total	plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	m-bike	person	p-plant	sheep	sofa	train	tv
52090	1456	1401	2064	1403	2233	1035	4468	1951	3908	1091	1030	2514	1420	1377	17784	1967	1312	1053	1207	1416

Practical objectives for the course

Switching to practice with the Google Colab notebook !

Introduction

- Get the Google Colab environment working
- Get the Pascal VOC data and visualize it
- Install and understand the CIANNA Deep Learning framework

Using a combination of the data from Pascal VOC 2007 and 2012:

A - Training a simple classifier using object cutouts

B - Training a sliding window detector

C - Training a dedicated YOLO object detector



Convolutional Interactive Artificial Neural Networks by/for Astrophysicists

General purpose framework (like Keras, PyTorch, ...)

BUT developed for **astronomical applications**

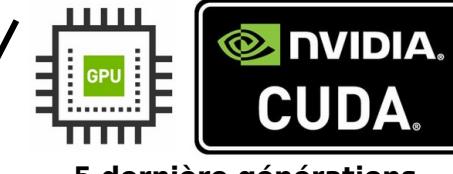


github.com/Deyht/CIANNA

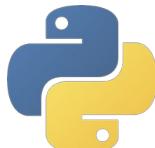
Open source – Apache 2 license



x86 Intel/AMD & ARM



Full user interface



Successfully deployed on

- Laptops / Workstation
- Local compute serveurs
- Mesocenteurs
- Large computing facilities

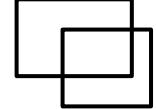
Custom YOLO implementation



Activation



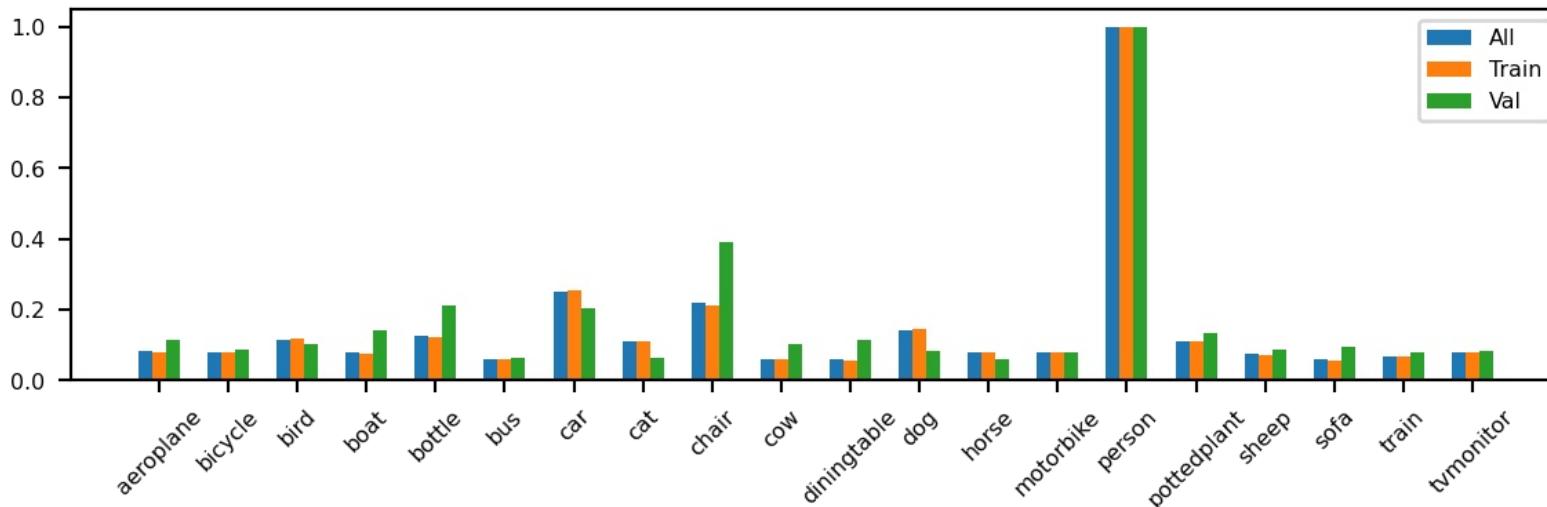
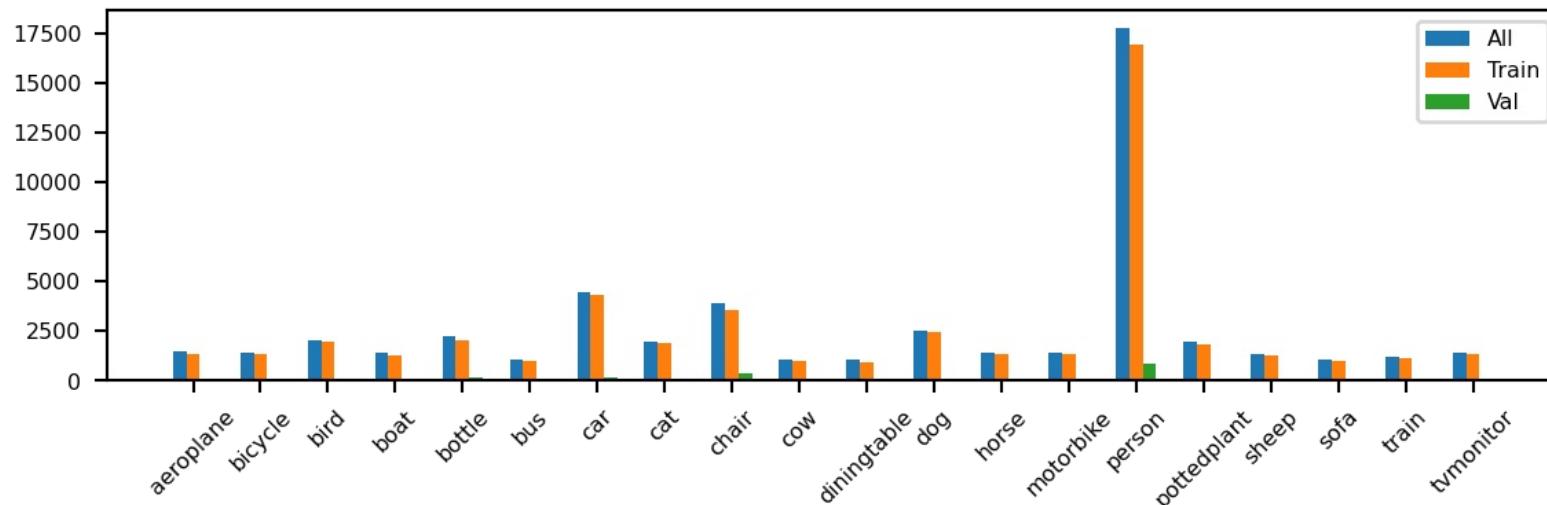
Cost



Association

And specificities : supplementary parameters per box, cascading loss, custom NMS process, ...

Training dataset summary statistics



Training dataset summary statistics

All the images are made square and resized to 224x224 pixels, using uint8 encoding.

The resulting dataset is saved in a binary format for use over all the applications in the notebook.

Target bounding boxes are encoded in a specific format for CIANNA, and also saved as binary.

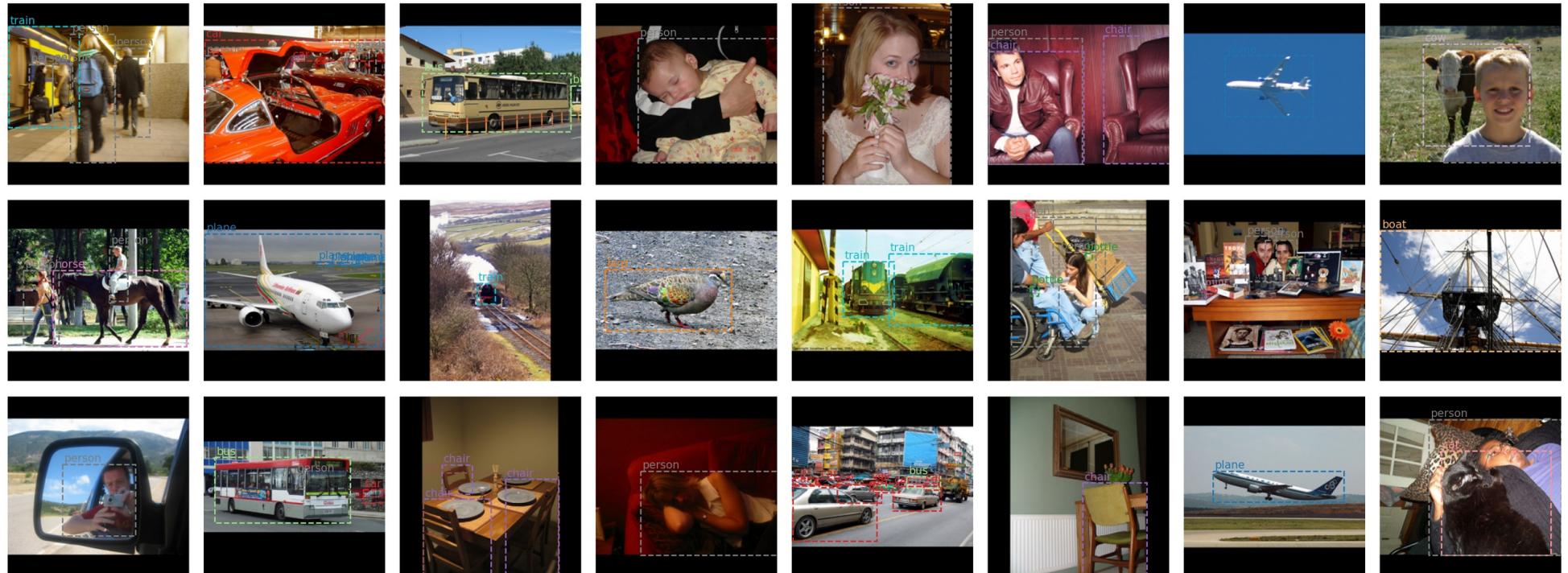


table	dog	horse	m-bike	person	p-plant	sheep	sofa	train	tv
plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow

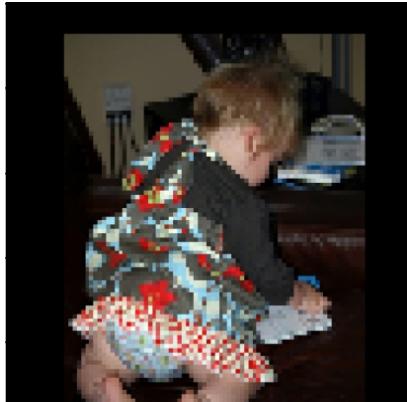
A - Simple classifier using object cutouts

We define a classifier with an **input dim. of 96x96**, and an **output dim. of 20** (number of classes) using **Softmax** activation. 1000 original images are kept apart for validation.

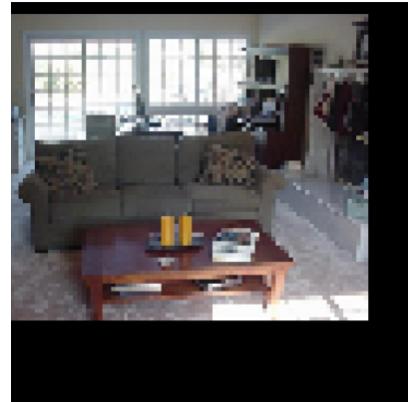
The training images dynamically generated as followed

- Pick a random image in training set
- Pick a random target box in the image
- Apply simple augmentation (zoom, and translate)
- **Cutout** the augmented region and **rescale** it to the input size

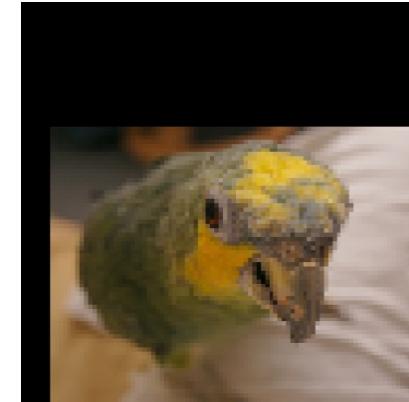
The validation images correspond to all the objects in the validation set with no augmentation



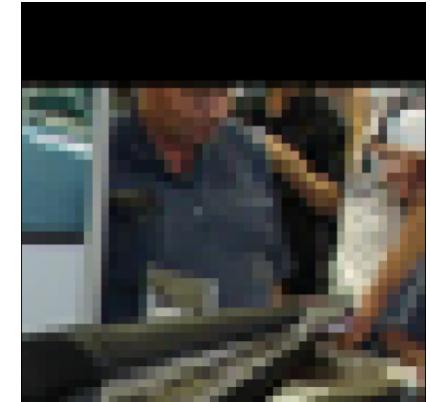
[person]



[sofa]



[bird]



[person]

A - Simple classifier using object cutouts

Suggested network architecture :

```
cnn.conv(f_size=i_ar([3,3]), nb_filters=48, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=96, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=128, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=256, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=512, padding=i_ar([1,1]), activation="RELU")
cnn.conv(f_size=i_ar([3,3]), nb_filters=512, padding=i_ar([1,1]), activation="RELU")
cnn.dense(nb_neurons=1024, activation="RELU", drop_rate=0.4)
cnn.dense(nb_neurons=512, activation="RELU")
cnn.dense(nb_neurons=nb_class, activation="SMAX")
```

Confusion matrix on the validation set at epoch 1100 :

ConfMat (valid set)																				Recall		
68	1	2	7	0	3	10	1	1	0	0	1	0	1	3	0	0	0	0	1	68.69%		
2	24	3	0	3	0	4	1	7	1	0	6	0	6	15	2	0	0	1	0	31.58%		
3	0	61	1	0	0	0	3	0	1	0	5	0	0	6	1	1	0	0	9	67.03%		
9	2	4	74	0	3	8	0	0	0	0	3	0	1	6	1	0	0	2	0	60.66%		
1	0	1	2	106	0	2	2	9	0	0	2	0	1	40	5	0	0	0	0	57.61%		
1	0	0	2	1	39	8	0	1	1	0	0	0	0	1	0	0	0	2	0	69.64%		
0	1	0	2	1	5	136	1	3	0	0	0	0	0	12	2	0	0	1	0	76.84%		
0	0	0	0	0	0	1	29	0	0	0	18	0	0	2	2	0	1	0	0	52.73%		
0	6	6	1	14	2	8	4	186	0	3	2	0	0	45	4	1	10	2	10	54.87%		
0	0	0	0	0	2	0	0	1	2	50	0	8	5	0	12	1	8	0	0	56.18%		
0	0	3	0	0	0	0	1	4	15	0	42	3	0	21	3	0	5	0	0	41.58%		
0	0	0	0	0	0	1	13	0	0	0	42	2	1	5	1	5	0	0	2	58.33%		
0	1	1	0	0	0	0	1	2	14	0	7	17	0	5	0	3	0	1	0	32.08%		
0	2	2	0	0	0	7	0	1	0	0	4	0	37	12	0	1	0	1	0	53.62%		
1	3	7	0	15	0	6	7	17	1	0	21	1	12	768	1	0	1	1	2	88.17%		
0	2	2	0	5	0	1	0	2	0	0	0	0	0	23	76	0	0	1	4	65.52%		
0	0	0	0	5	0	0	0	0	8	0	4	2	0	3	0	52	0	0	1	69.33%		
0	0	0	4	2	0	2	2	14	0	2	2	1	0	17	1	0	29	0	3	34.52%		
3	0	0	3	0	3	5	0	2	0	0	0	0	0	3	1	0	2	44	0	4.62.86%		
0	0	0	0	0	0	0	0	1	0	0	1	0	0	6	0	0	0	60	3	84.51%		
0	0	0	1	0	0	0	1	2	6	1	0	0	0	4	1	1	3	0	479	95.80%		
Precision	77.27%	57.14%	66.30%	76.29%	68.83%	70.91%	67.66%	40.85%	69.14%	64.94%	89.36%	32.56%	60.71%	61.67%	76.11%	74.51%	72.22%	56.86%	75.86%	78.95%	80.91%	Acc 71.78%

B - Training a sliding window detector

We just slightly modify the previous classifier

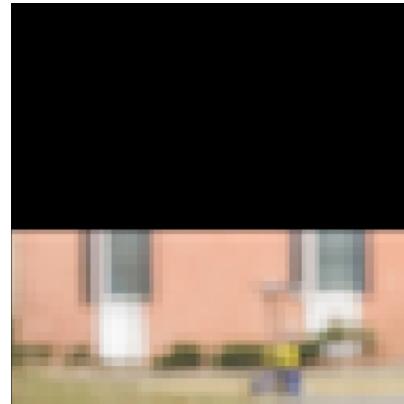
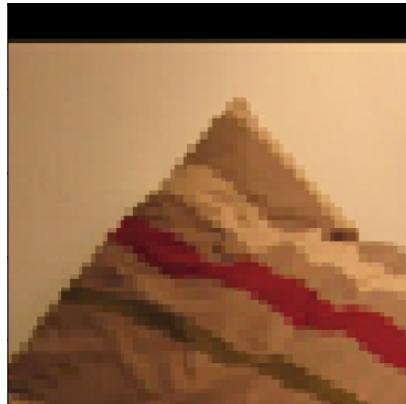
- We add a new “background / empty” class, raising the output dimension to 21
- When generating data

→ **80% of the time, generate an image and class as before**

→ **Generate a “background / empty” example**

- a) Pick a random image, and define a starting *size*
- b) Randomly select a position in the image and define a box around it using *size*
- c) Check if box overlap a target : If no accept the box and rescale it, if yes retry random position
- d) If the box is rejected X times, reduce *size* and retry X times
- e) If *size* become two small, stop the process and use an empty (zero) input instead

The validation is enriched with “selected” background examples using the validation images



The performance of this new classifier should be similar with ~95% recall on background examples

B - Training a sliding window detector

The sliding window principle

Our classifier is trained to identify individual objects relatively centered in the input image.

To use it as a **sliding window detector**, one can split a larger image on “chunks” that each goes through the network.

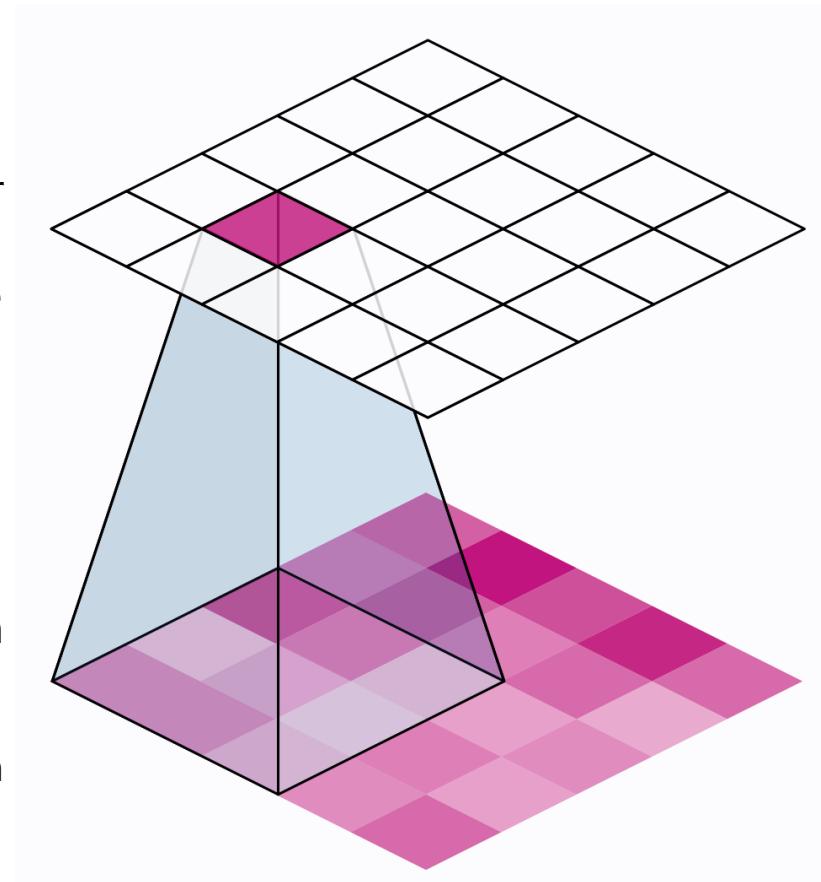
To increase the chance of the object to be centered on the “input window”, each chunks **partially overlap each others**.

This all process is similar to a convolution, with the filter being the classifier

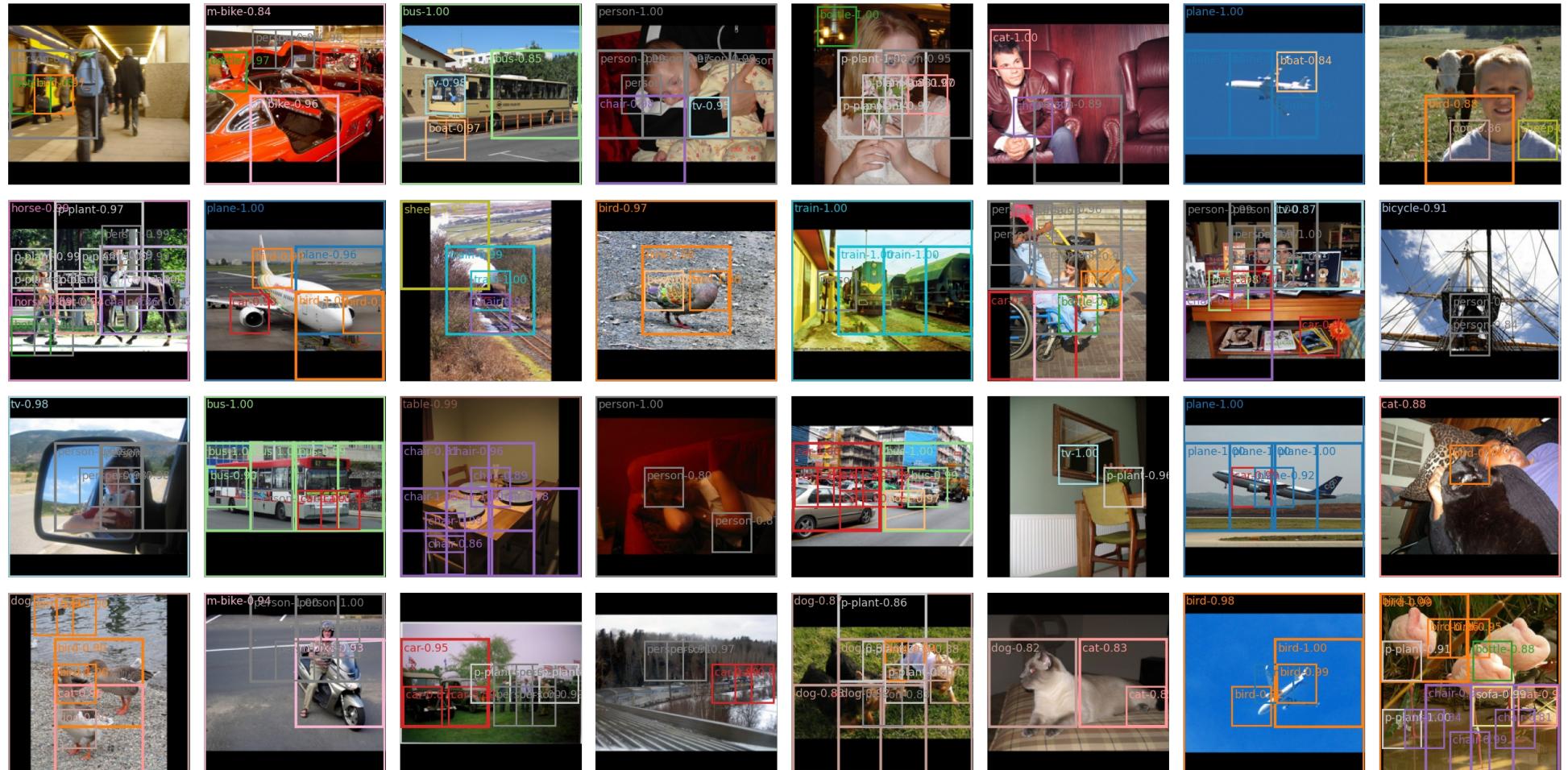
To improve prediction across scales, one can use several window sizes (each time rescale to the classifier input size) in order to detect larger and small objects in the image.

In the present case we use **3 window sizes** [224, 112, 56] with a respective stride of [0, 56, 28].

This leads to a number of « chunk » par size of [1, 9, 49], corresponding to grids of [1x1, 3x3, 7x7].



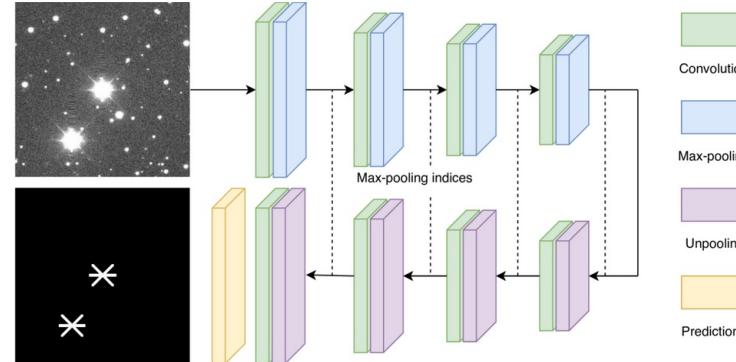
B - Training a sliding window detector



Questions : How to filter overlapping detections ? How to evaluate prediction performance ?

U-Nets

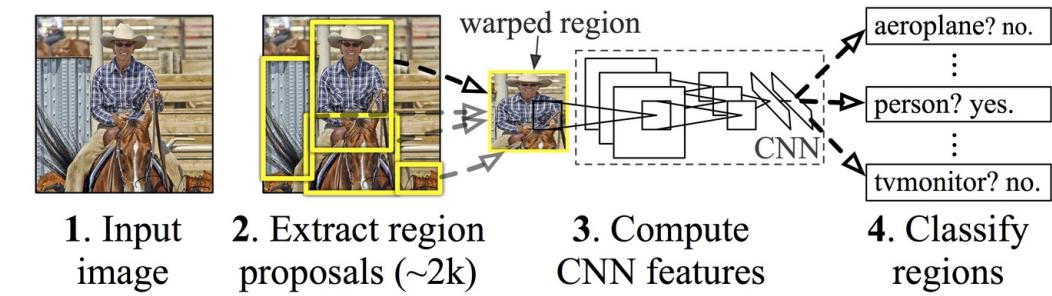
Pros: segmentation maps, shallow latent space, ...



Region-Based

Methods : R-CNN (Fast and Faster), SPP-net, Mask R-CNN, ...

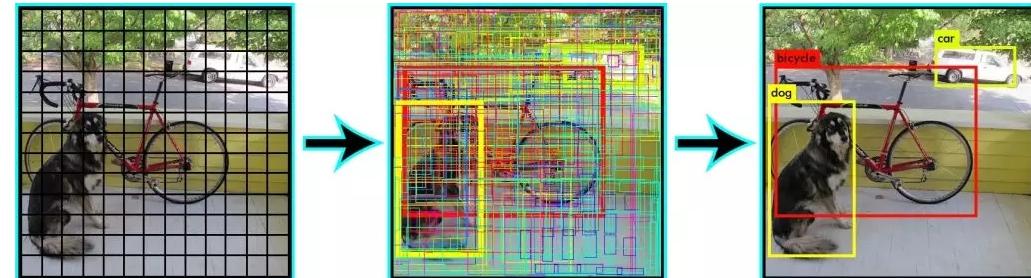
Pros: Best accuracy, ...



Regression-based

Methods : SSD (Single Shot Detector), YOLO (You Only Look Once), ...

Pros: Very Fast, straightforward architecture, ...



Types of object detection in images

Classification



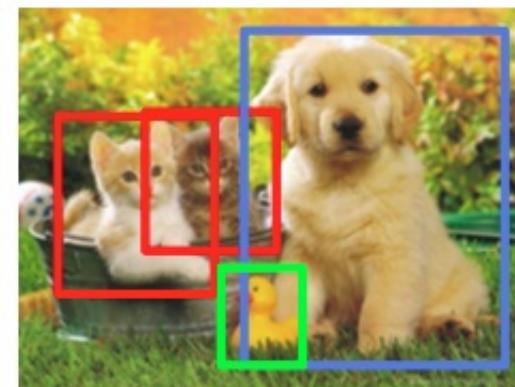
CAT

Classification + Localization



CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation



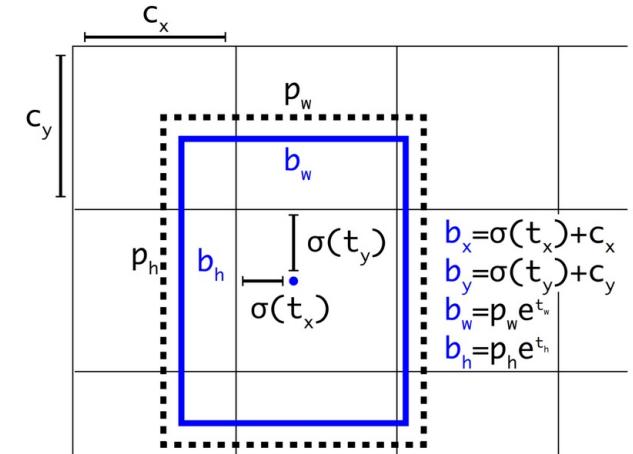
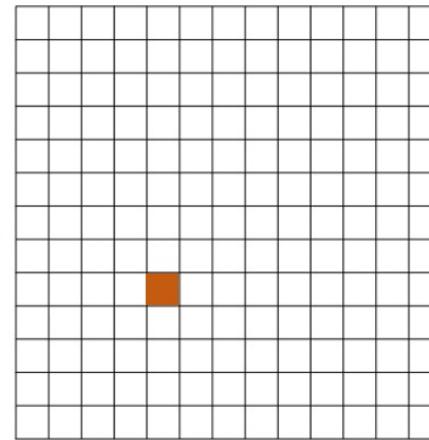
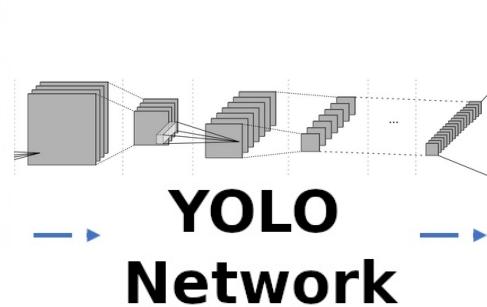
CAT, DOG, DUCK

*Image from Stanford Deep Learning course cs224

You Only Look Once - YOLO !

Originally introduced in Redmon et al. 2015 (V1), 2016 (V2), 2018 (V3)

*Images from [blog post](#) and Redmon papers



Fully Convolutional Neural network architecture

- The grid size corresponds to the spatial dimension of the last convolutional layer
- Each grid element maps a sub-region** of the image that corresponds to the **spatial reduction factor** of the full network
- Example :** an image of 224x224 reduced to a 7x7 correspond to a spatial reduction factor of 32
- Each grid element then map a region of 32x32 in the input image

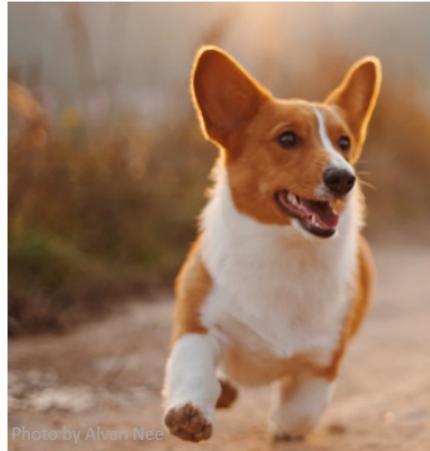
Bounding box encoding

- Instead of $xmin, ymin, xmax, ymax$ the box is encoded as **central position and size**
- Each box is subscribed to a specific grid element**
- The box center is encoded as its relative position inside the grid element
- The size of the box is encoded as an offset from a given **size prior**

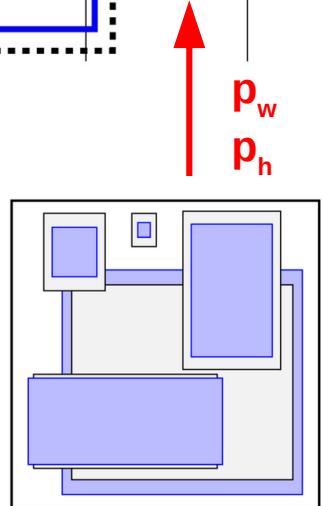
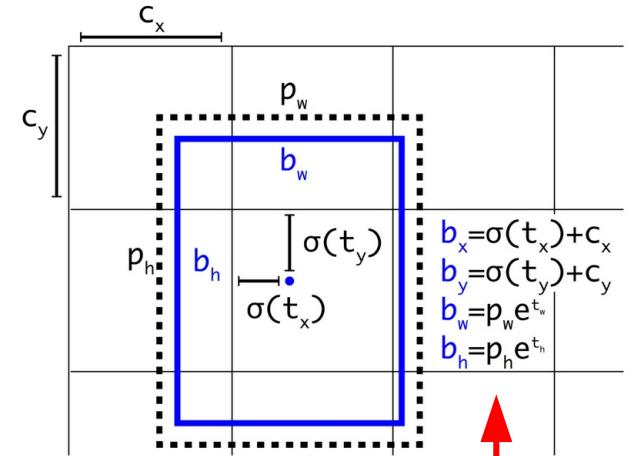
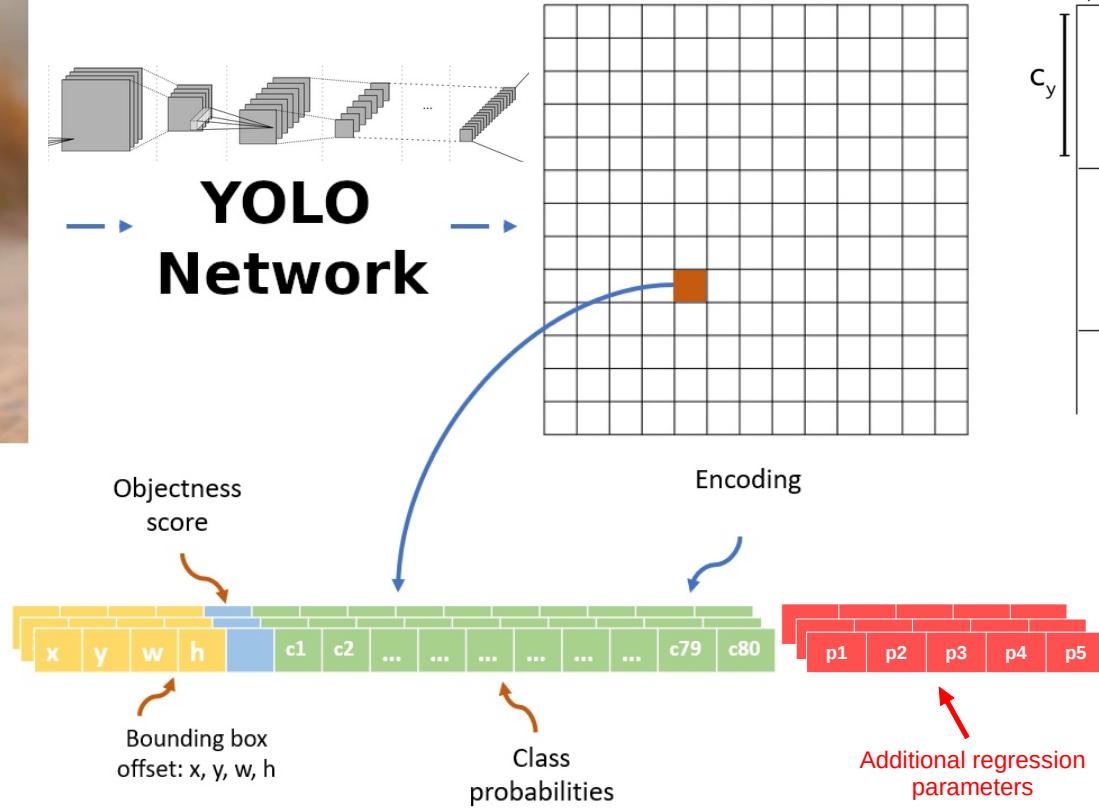
You Only Look Once - YOLO !

Originally introduced in Redmon et al. 2015 (V1), 2016 (V2), 2018 (V3)

*Images from [blog post](#) and Redmon papers



Pre-processing Image



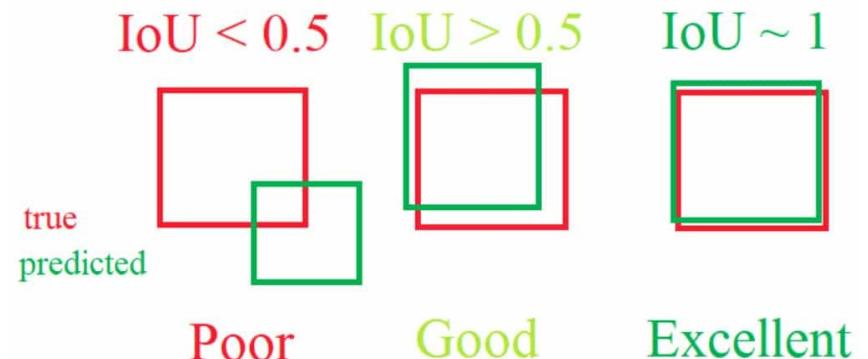
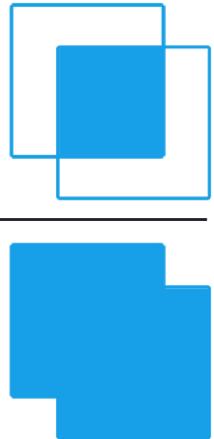
The last layer is conv. = the boxes « share » weights spatially.

The output is a 3D cube encoding all possible boxes on the output grid.

How to estimate box « correctness » ?

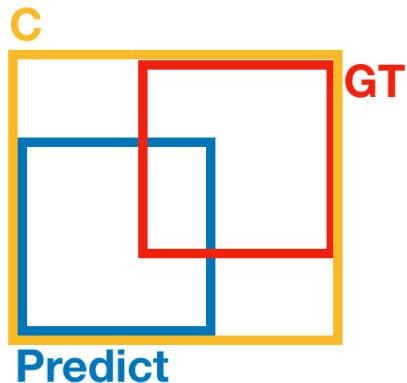
Classical IoU [0,1]

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



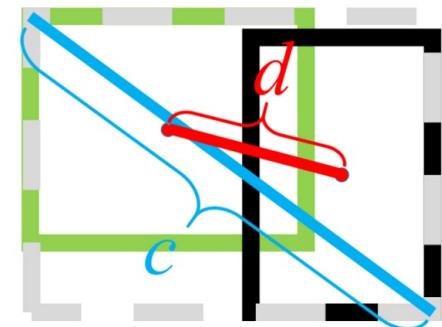
Generalized IoU [-1,1]

$$GIoU = IoU - \frac{A^c - \mathcal{U}}{A^c}$$

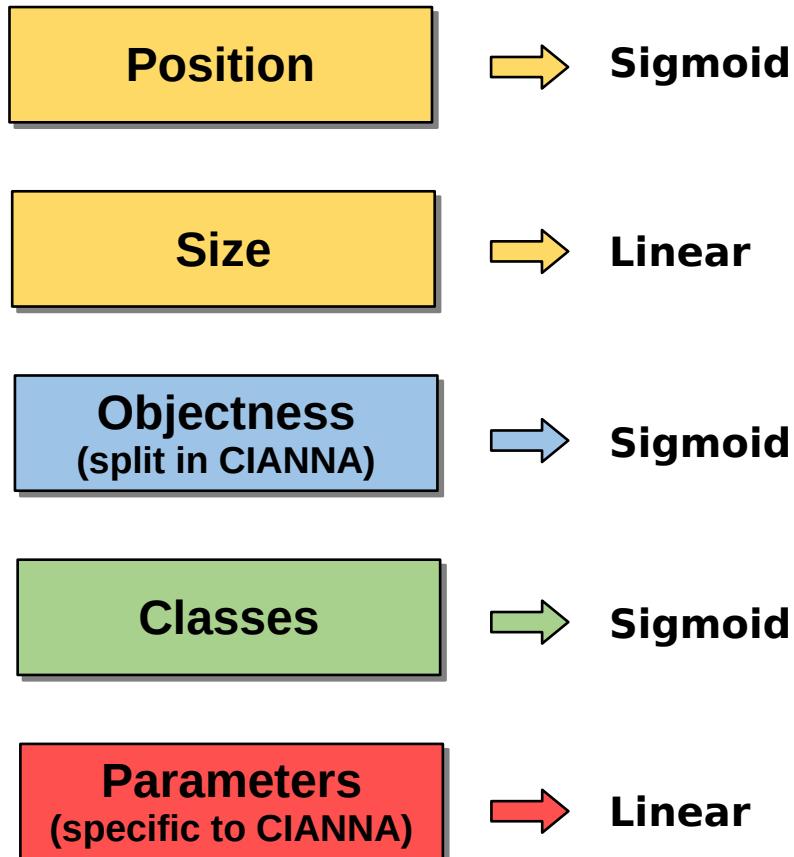


Distance IoU [-1,1]

$$DIoU = IoU - \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}$$



The YOLO activation / loss function

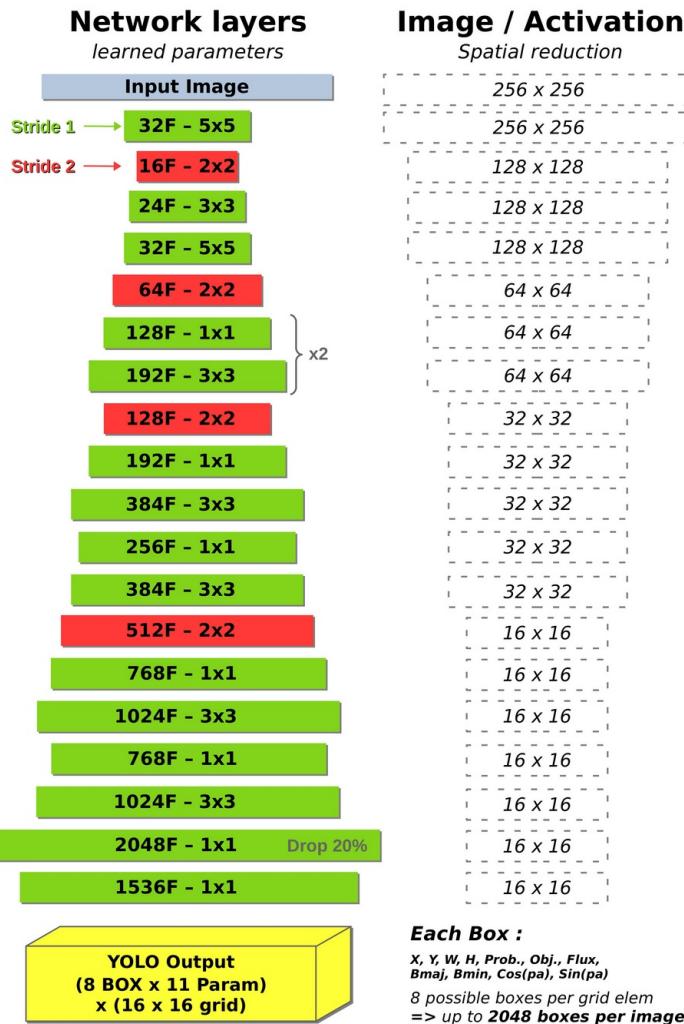


All error terms are using MSE

Training procedure :

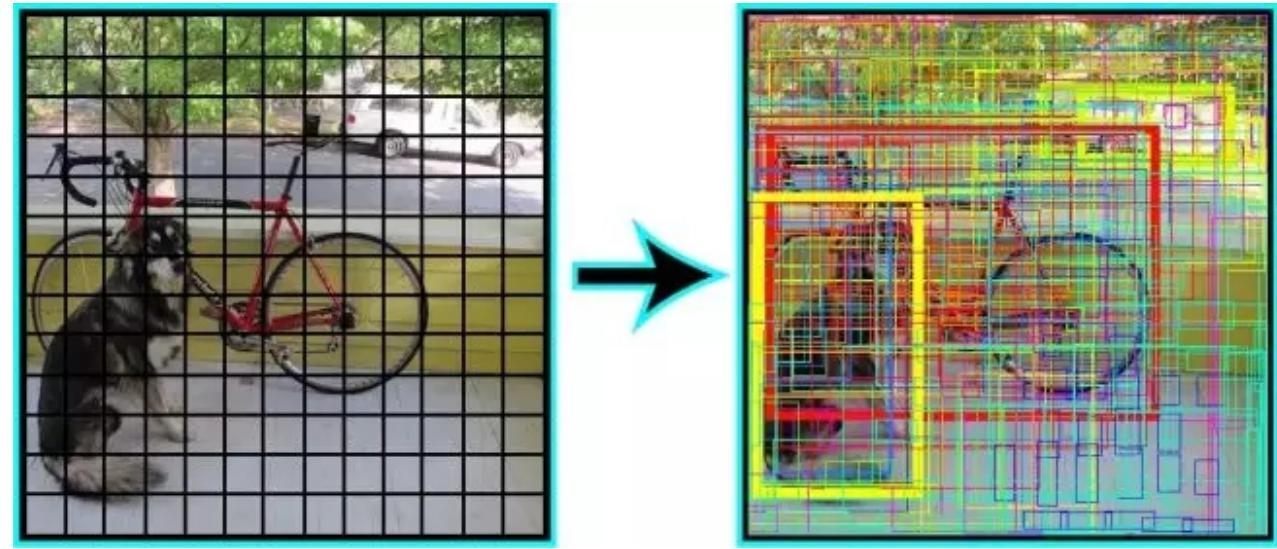
- For each target, find the corresponding grid element
- Search for the **best predicted box** in this grid element
- If other boxes are “**good but not best**”, (based on an IoU threshold) flag them so they are note penalized at all
- **For the best box only :**
 - Set the target objectness to the IoU value
 - Update classes according to target
 - Update parameters according to targets
- For all the predictions with no associated target, update only the objectness with a **target 0** using a smaller error scaling

Raw YOLO prediction



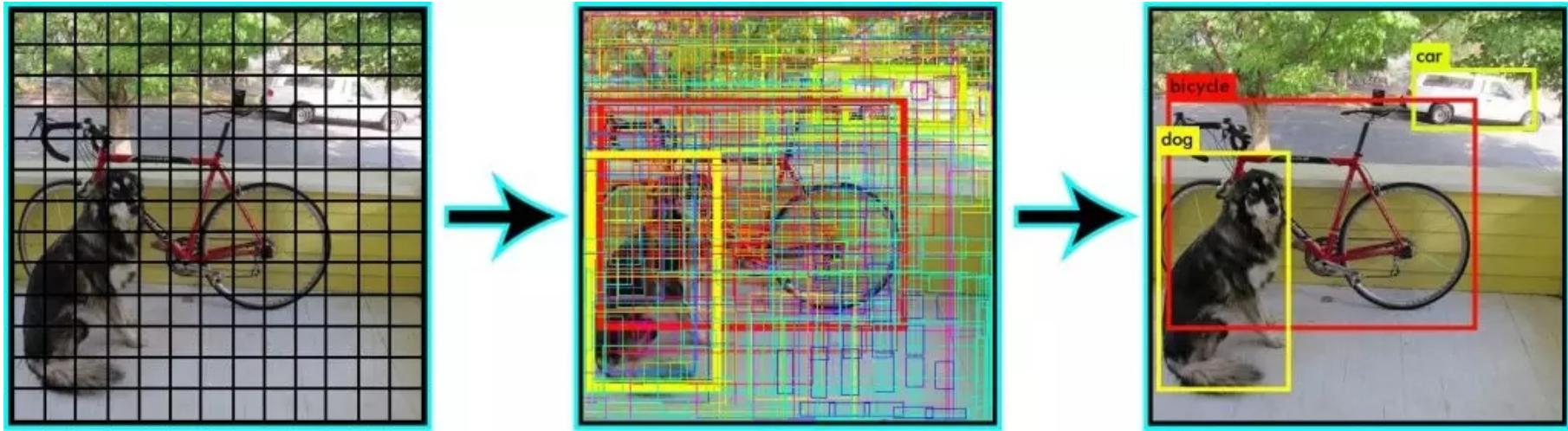
Example of a practical YOLO architecture

Due to the fixed size nature of its output layer,
a YOLO network always predict all the possible boxes

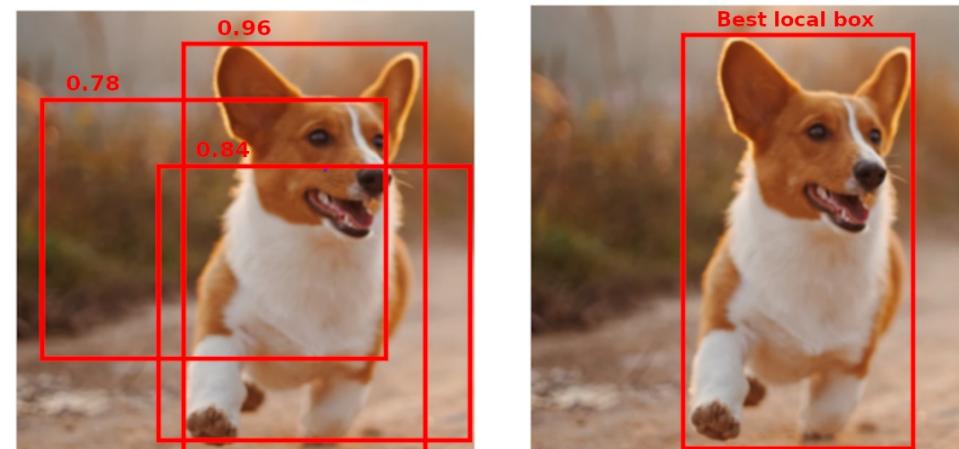


Necessary step : post-processing of the output !

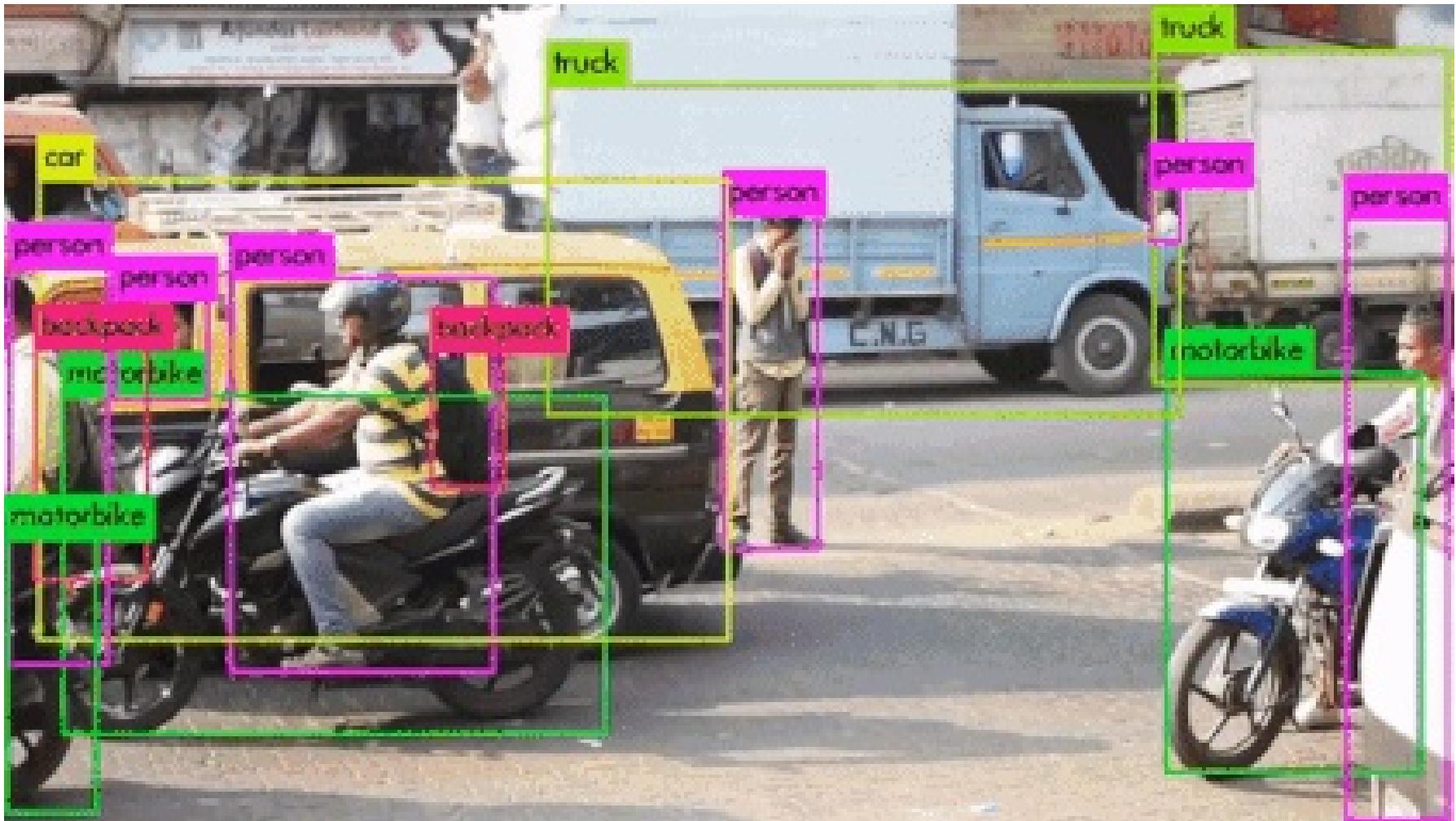
Non Max Suppression



- 1) Most probable boxes are kept using a threshold in objectness**
- 2) NMS takes the most probable box and removes overlapping ones based on an IoU threshold. Repeat.**



“Real-time” example



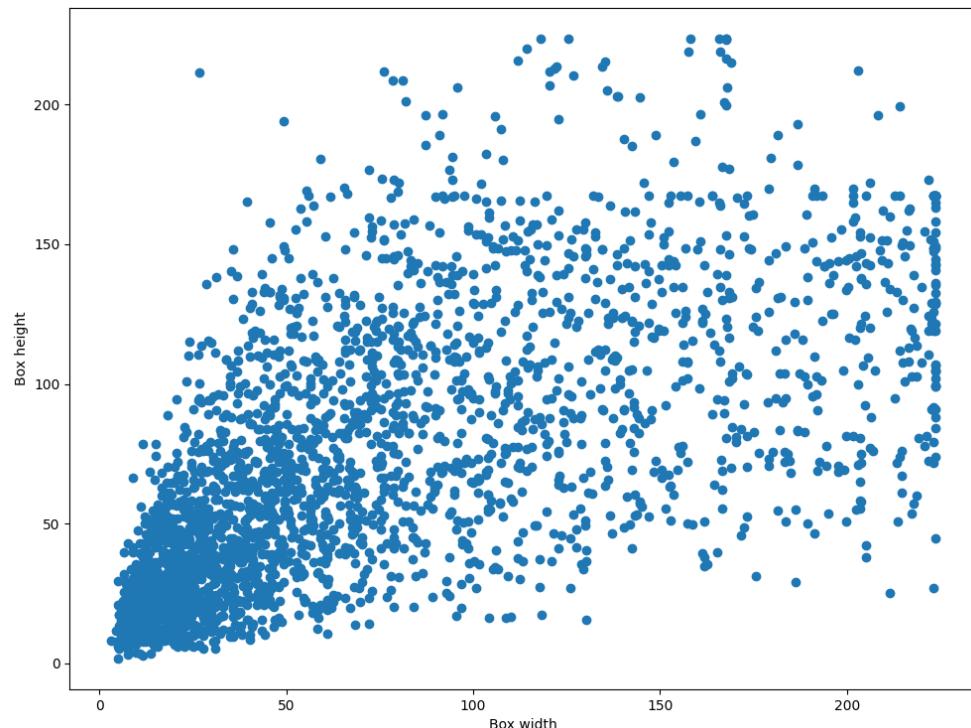
C - Training a YOLO network on PASCAL VOC

For the proposed network, we set the input dimension to **224x224**.

Having 5 pooling layers in the proposed network architecture we reduce the **spatial dimension by a factor of 32**, ending with a **7x7 output grid**.

Based on the observed distribution of box sizes and on the observed crowding, we chose the following square priors for our boxes
[12.,16.,32.,64.,128.,192.]

In some cases, the optimal boxes prior can be determined by using **clustering on the shape distribution**.



C - Training a YOLO network on PASCAL VOC

Suggested network architecture :

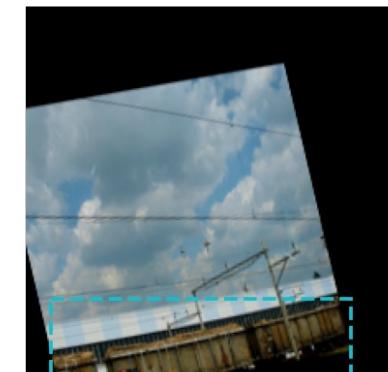
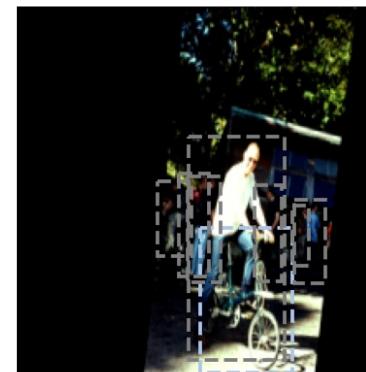
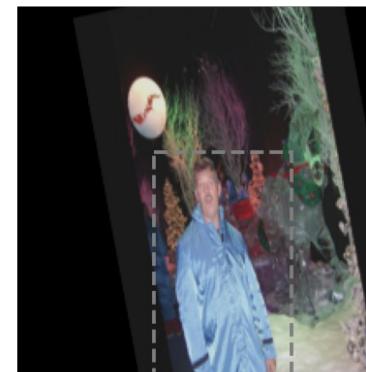
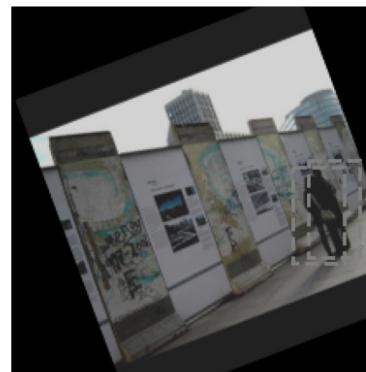
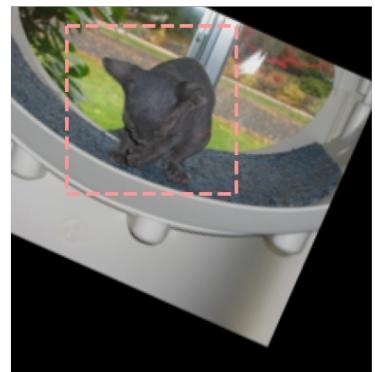
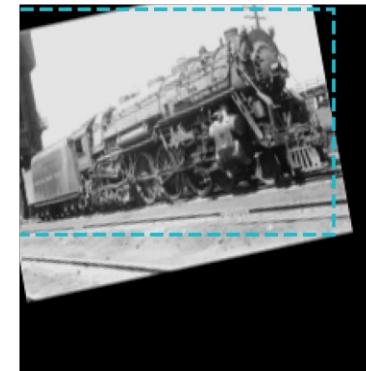
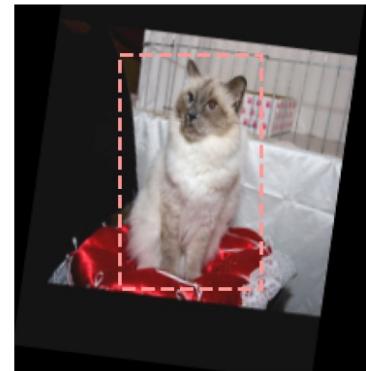
```
cnn.conv(f_size=i_ar([3,3]), nb_filters=24, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=48, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=128, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=256, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=256, padding=i_ar([1,1]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([3,3]), nb_filters=512, padding=i_ar([1,1]), activation="RELU")
cnn.conv(f_size=i_ar([3,3]), nb_filters=512, padding=i_ar([1,1]), activation="RELU")

cnn.conv(f_size=i_ar([1,1]), nb_filters=1024, padding=i_ar([0,0]), activation="RELU", drop_rate=0.3)
cnn.conv(f_size=i_ar([1,1]), nb_filters=512, padding=i_ar([0,0]), activation="RELU")
cnn.conv(f_size=i_ar([1,1]), nb_filters=nb_yolo_filters, padding=i_ar([0,0]), activation="YOLO")
```

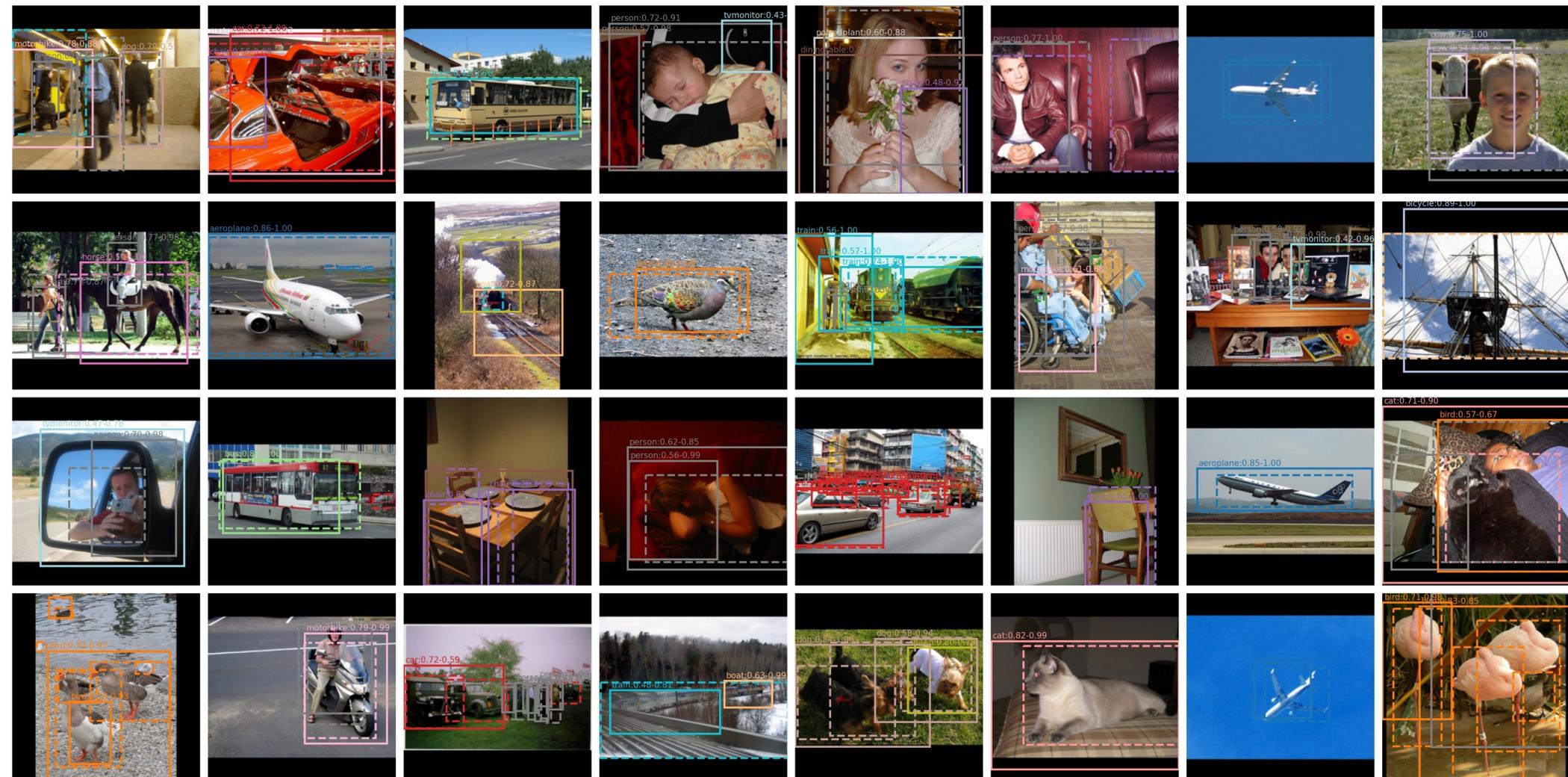
Inspired by the YOLO-Light V2 and Tiny-Yolo V2 architectures

C - Training a YOLO network on PASCAL VOC

Advanced data augmentation using `imgaug`



C - Training a YOLO network on PASCAL VOC

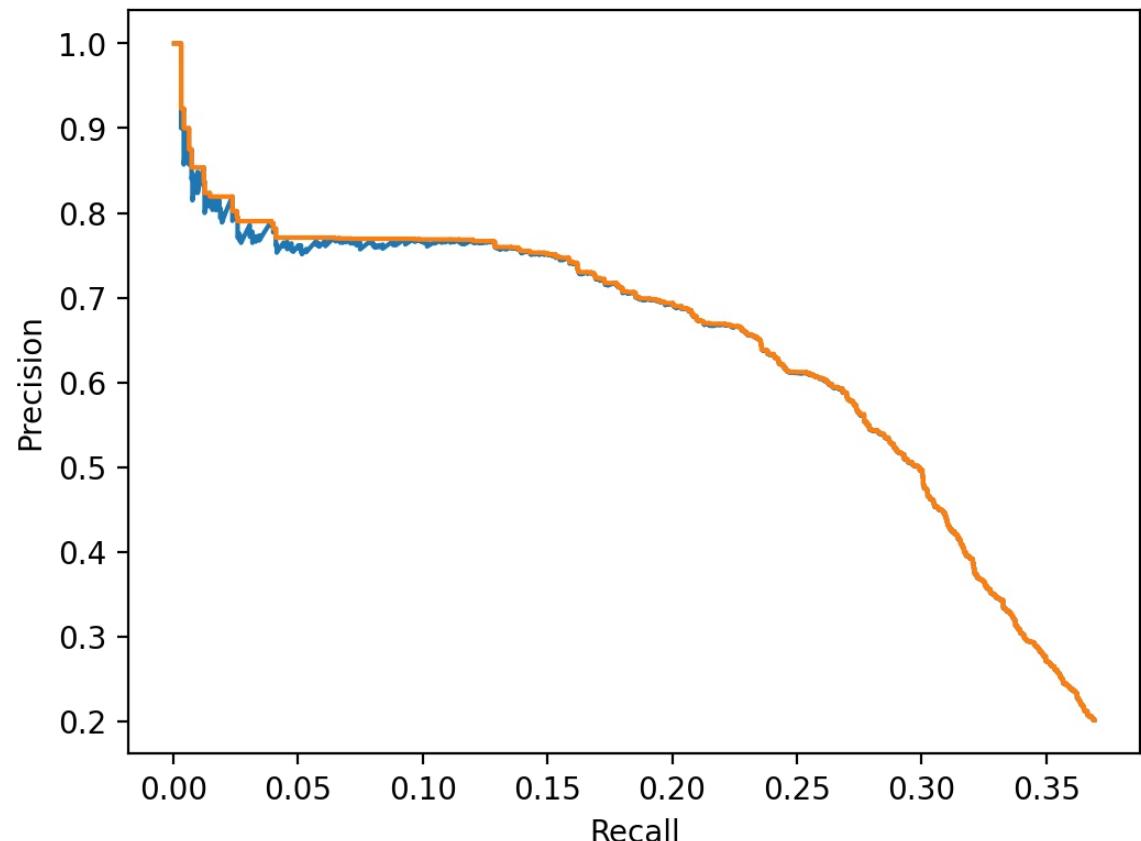


Detection performance metric

The match between a predicted and a target box can be determined by choosing a minimum threshold in IoU, most of the time 0.5

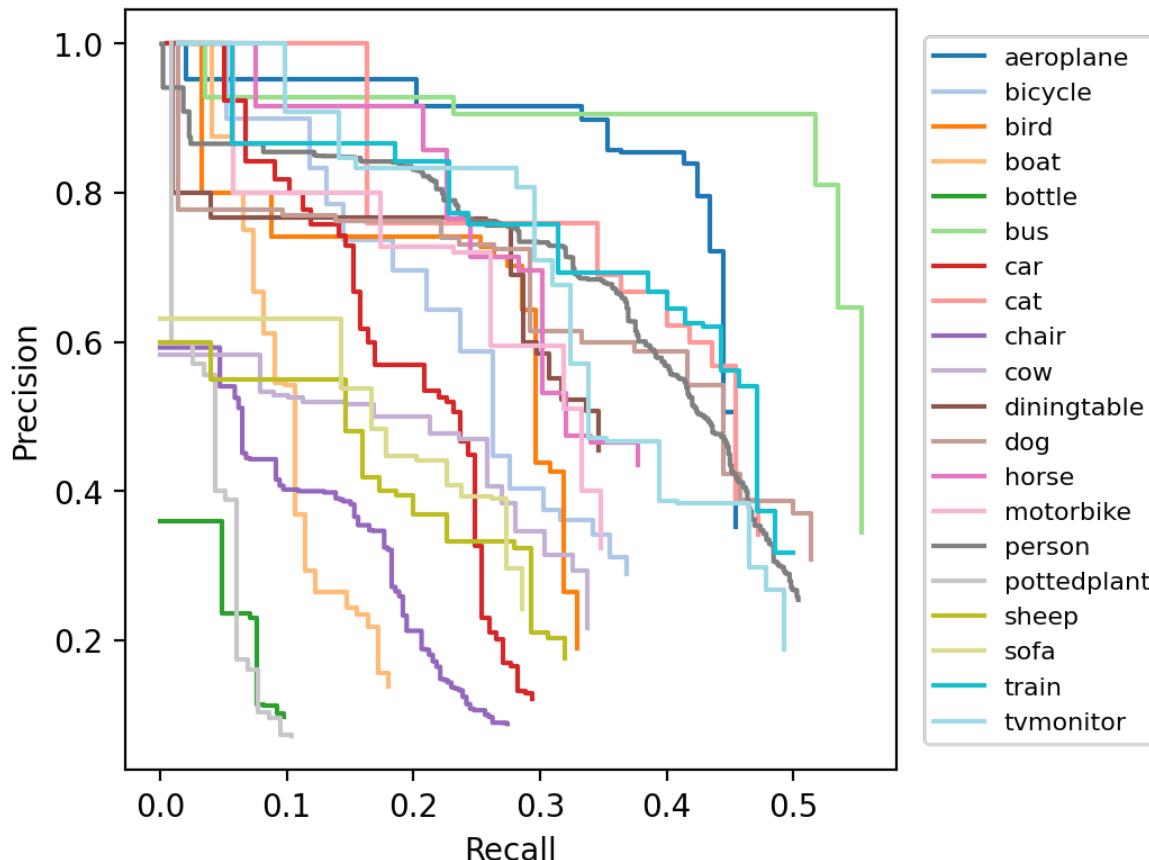
The precision-recall curve

- Predictions are sorted using a prediction score (e.g objectness)
- For each data point compute the precision and recall using all the prediction that are scored higher, using **IoU@0.5** and correct classification as “True” criteria
- The curve is then smoothed so it can only monotonically decrease (precision is always equal to its maximum value for any higher recall)
- The area under the curve define the global metric called the **Averaged-Precision AP@50**



Per class AP and Mean AP

Usually, the recall-precision precision curve and AP values are compute for each class individually. The final metric is the **mean of the AP values**



Warning: definition of AP and mAP might change depending on the challenge or dataset