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

David Cornu

IRMIA Deep Learning Summer School 2022

MINERVA

l'Observatoire de Paris | PSL

Lessons materials

Slides and a 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

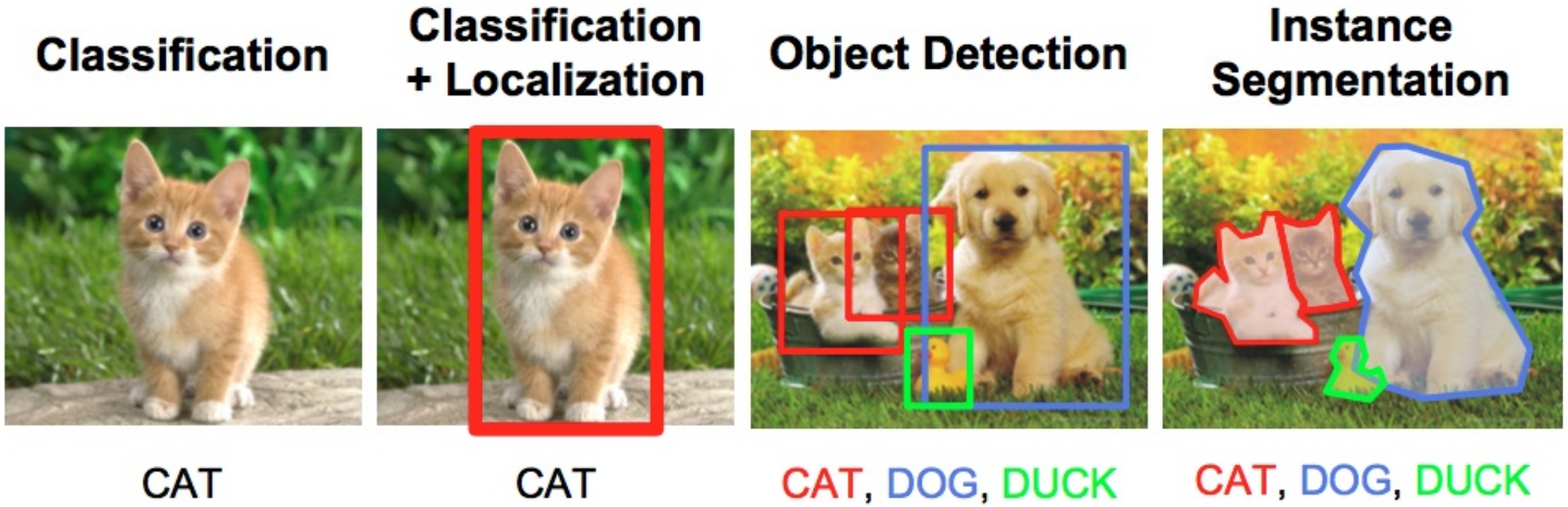
The Jupyter notebook is to be open in Google Colab with a GPU reservation.
(Should work on other environments with some adjustments, untested)

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)

Types of object detection in images

**Image from Stanford Deep Learning course cs224*

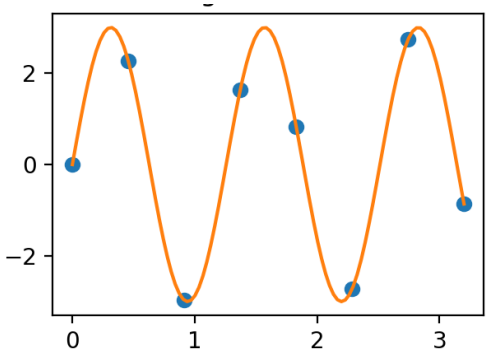


How to solve these tasks with Machine Learning and especially Artificial Neural Networks ?

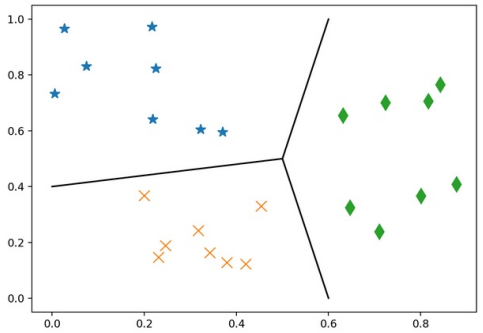
What is Machine Learning ?

Core concept: any method that **extract a statistical information** about a dataset by adapting a response through a **learning process**

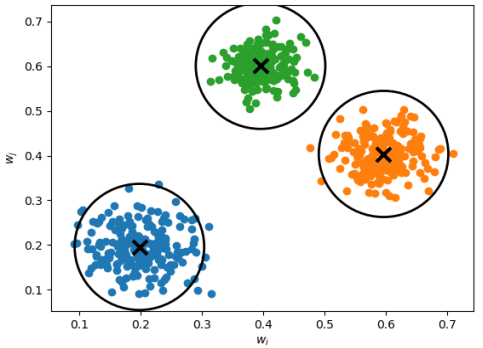
Very generic methods: Artificial Intelligence, computer vision, numerical science, ...



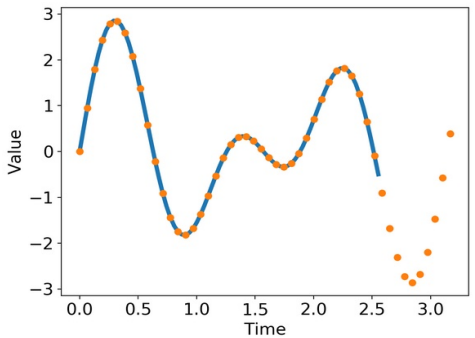
Regression



Classification



Clustering



Time series prediction

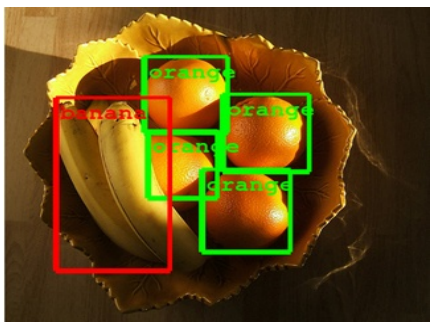


Image recognition

And much more ...

Methods diversity

Machine Learning has a broad definition, therefore it **regroups a lot of methods …**

Two main families of methods

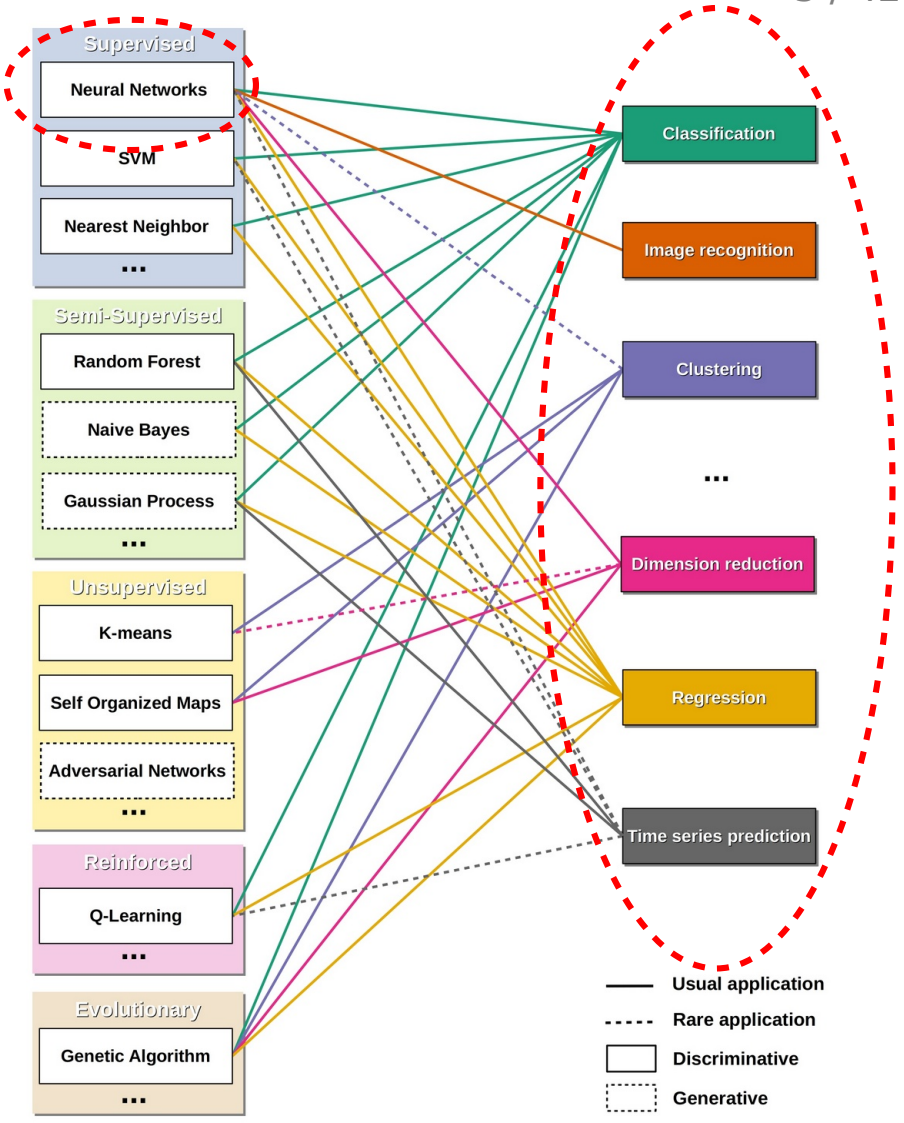
Supervised

Learn from labeled examples. Try to find a generalization to get correct prediction most of the time.

Un-Supervised

Dataset with examples is provided but with no labels. Try to find similarities in the input to create groups.

In this presentation we will only talk about **supervised Neural Networks**.



Artificial Neuron model

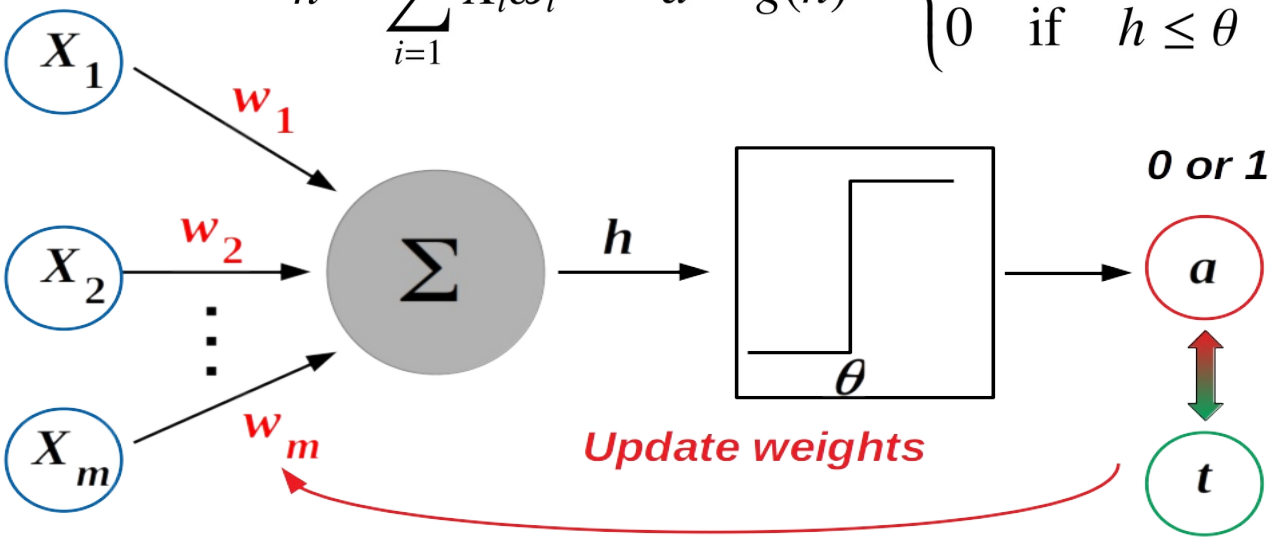
Mathematical model
McCulloch and Pitts (1943)

Weighted sum

$$h = \sum_{i=1}^{m} X_i \omega_i$$

Activation function

$$a = g(h) = \begin{cases} 1 & \text{if} \quad h > \theta \\ 0 & \text{if} \quad h \leq \theta \end{cases}$$



Error function

$$E = 0.5 \times (a - t)^2$$

$$\omega_i \leftarrow \omega_i - \eta\,(a - t)\,X_i$$

learning rate η

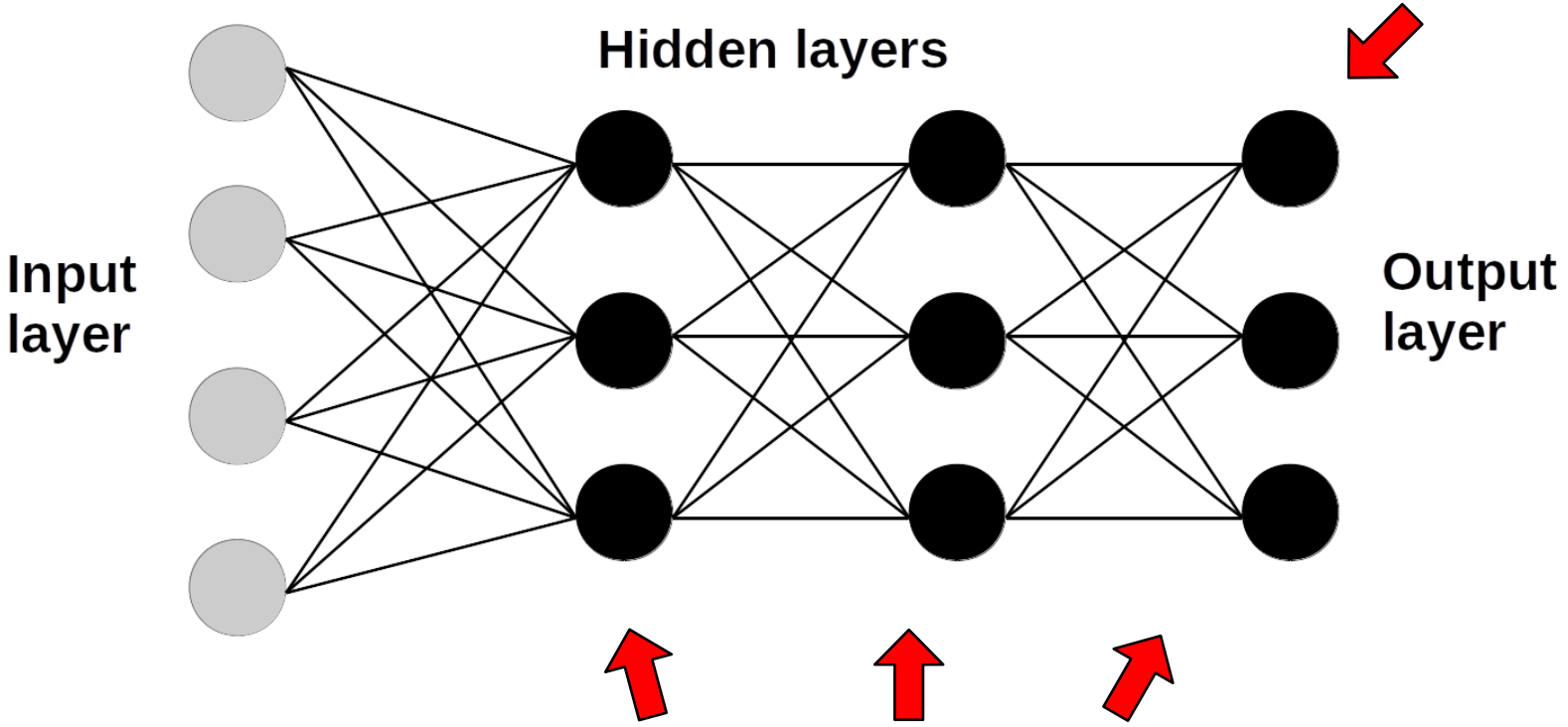
derivative of the error

input vector

Building a Neural Network

For complex problems several neurons must be used

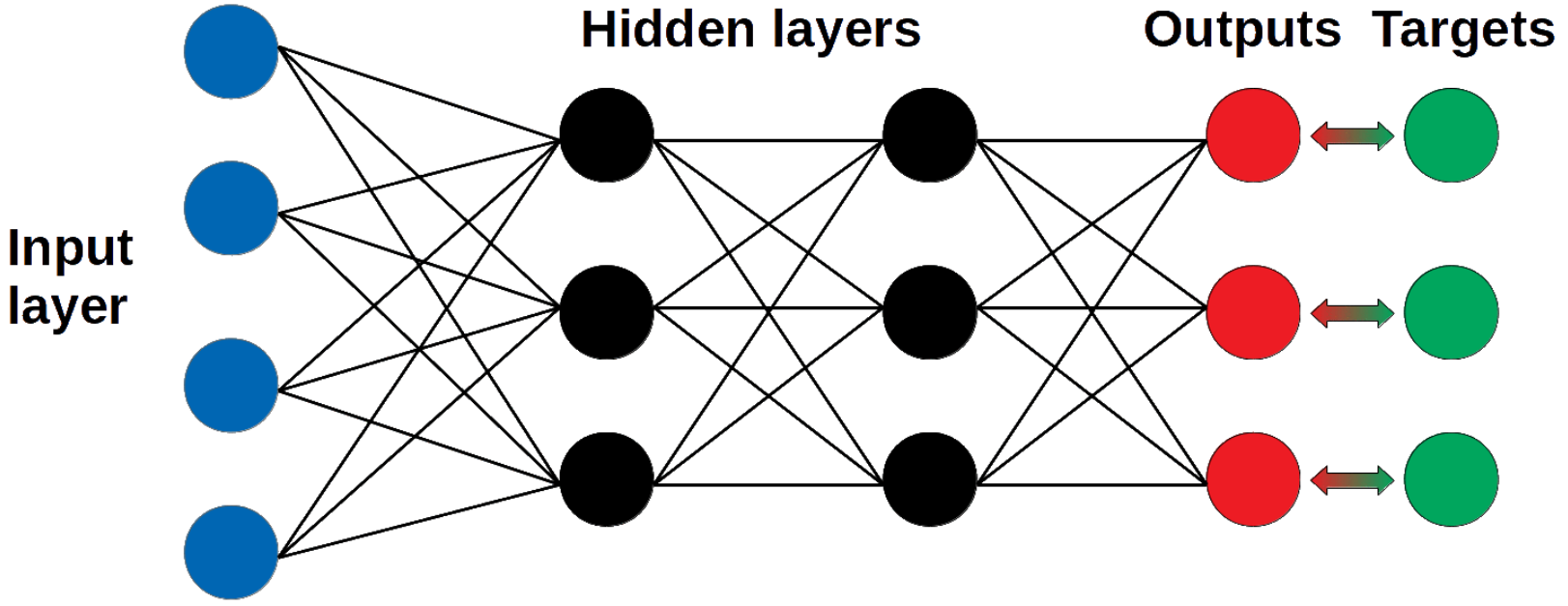
First: stacked independently in a layer (Perceptron)



Then: stack several layers
(Multi Layer Perceptron)
→ Non linear combination

Gradient propagation

The **error function (or loss)** provides a way to update the weights of the **« output »** layer, but how to have an error for the **« hidden »** layers ?



General error gradient using backpropagation :

$$\frac{\partial E}{\partial \omega_{ij}} = \delta_l(j)\frac{\partial h_j}{\partial \omega_{ij}} \quad \text{with} \quad \delta_l(j) \equiv \frac{\partial E}{\partial h_j} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial h_j} = \frac{\partial a_j}{\partial h_j}\sum_j \omega_{kj}\delta_{l+1}(k).$$

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.

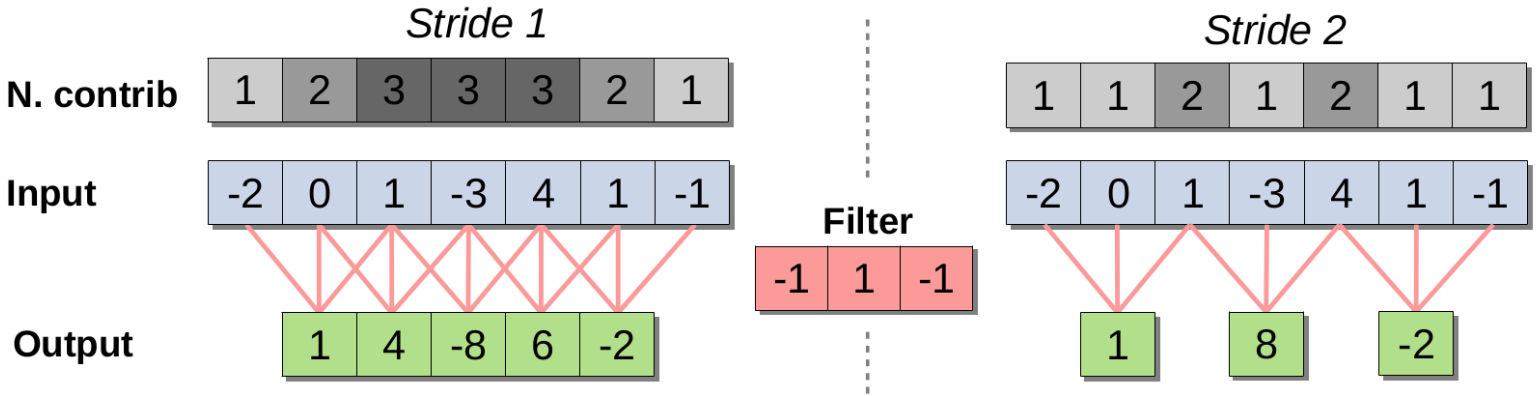
*A **highly dimensional** "dog" with ~0.5 Million pixels.*
Quite difficult to classify ...

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

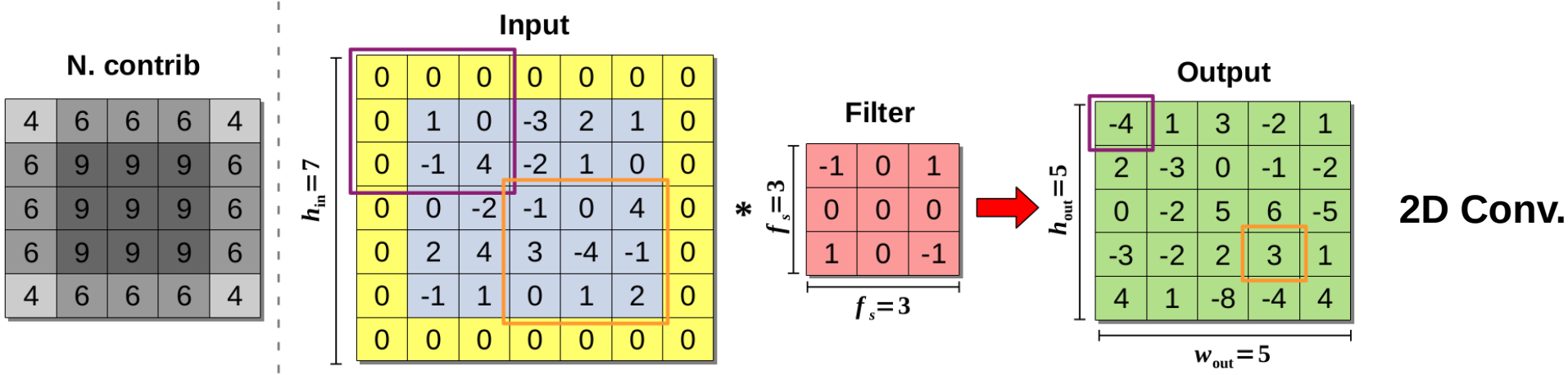
➡ **Convolutional Neural Networks** !

Convolution filter

1D Conv.



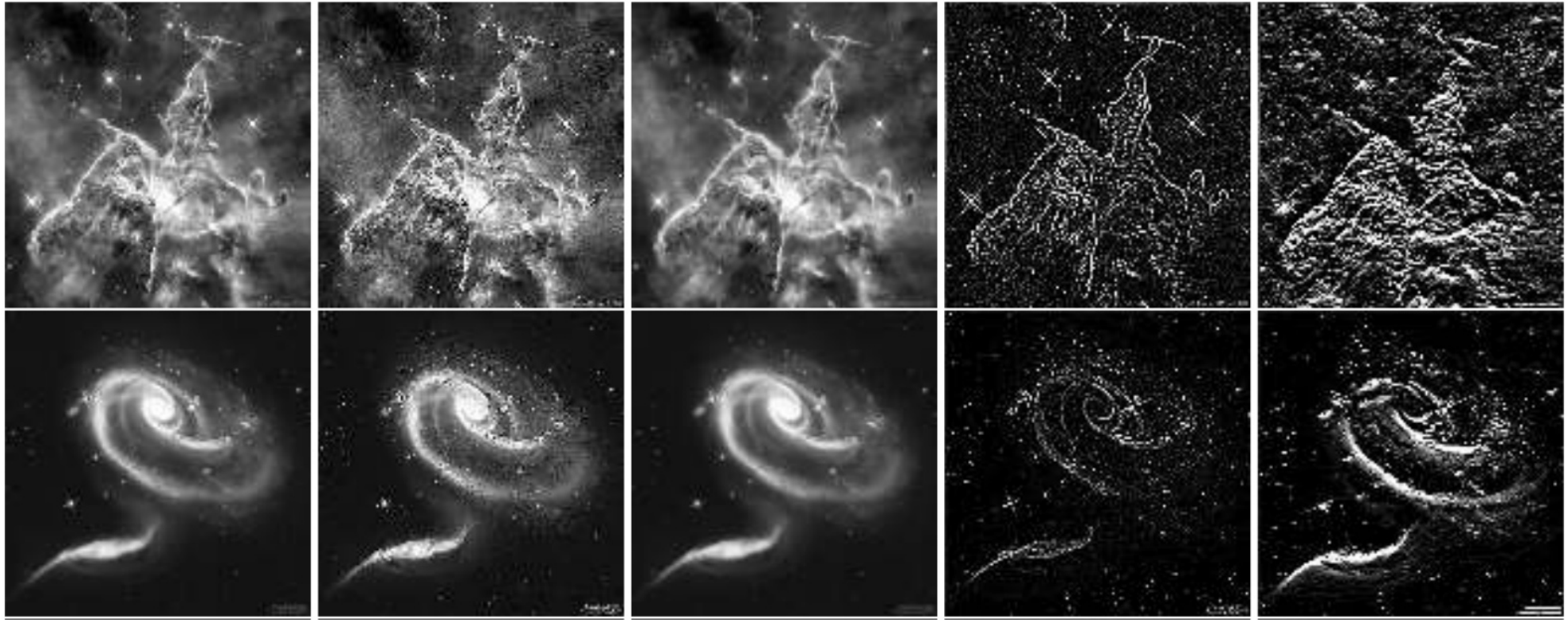
2D Conv.



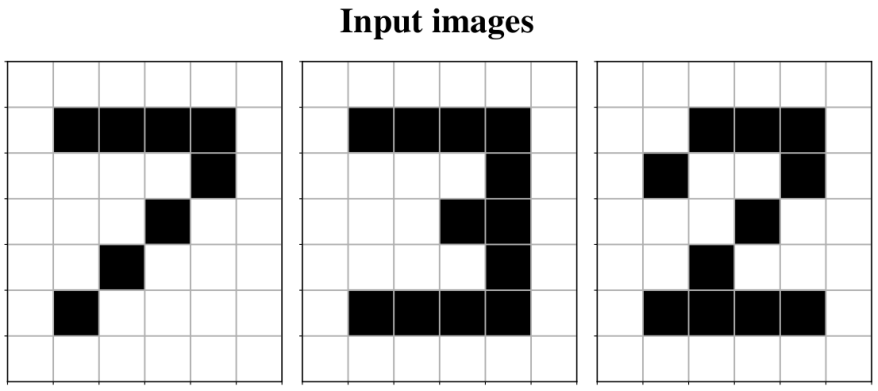
Stride 1 and Padding 1

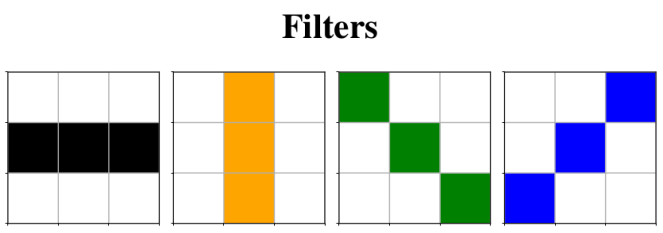
Filter effect examples

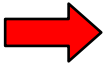
No filter	Sharpen	Gaussian blur	Edge detector	Axis elevation
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

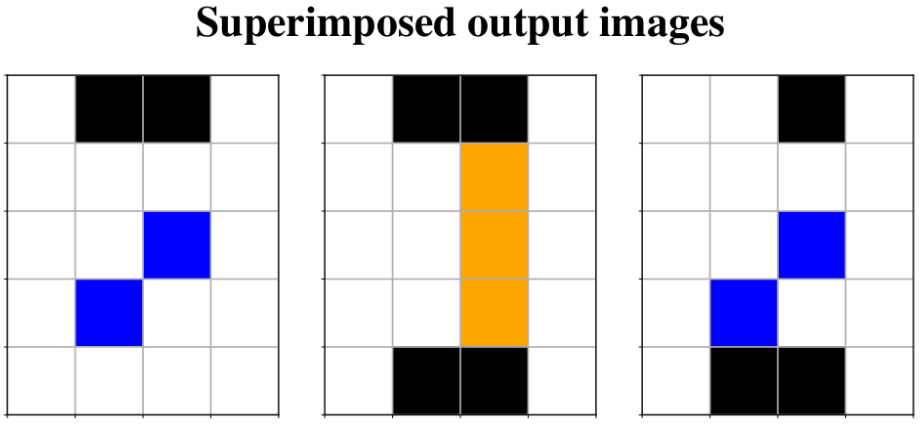


Pattern recognition with several filters





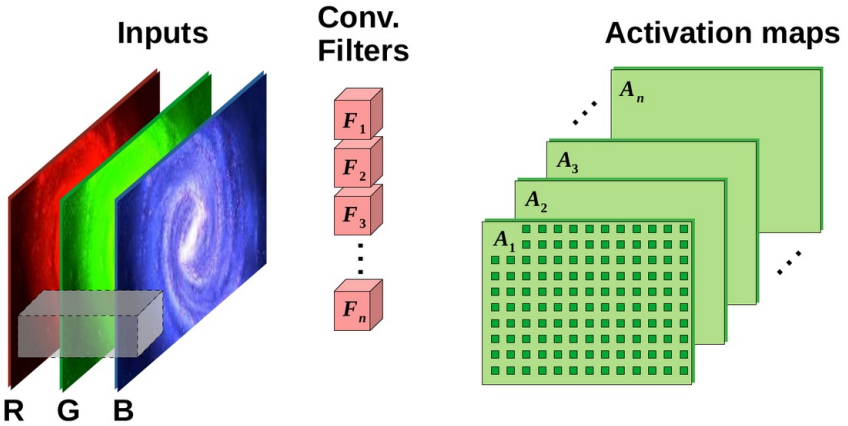




Each filter will produce its own activation map. Combining the information from different activation maps allows to construct more complex patterns.

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

Convolutional Neural Networks



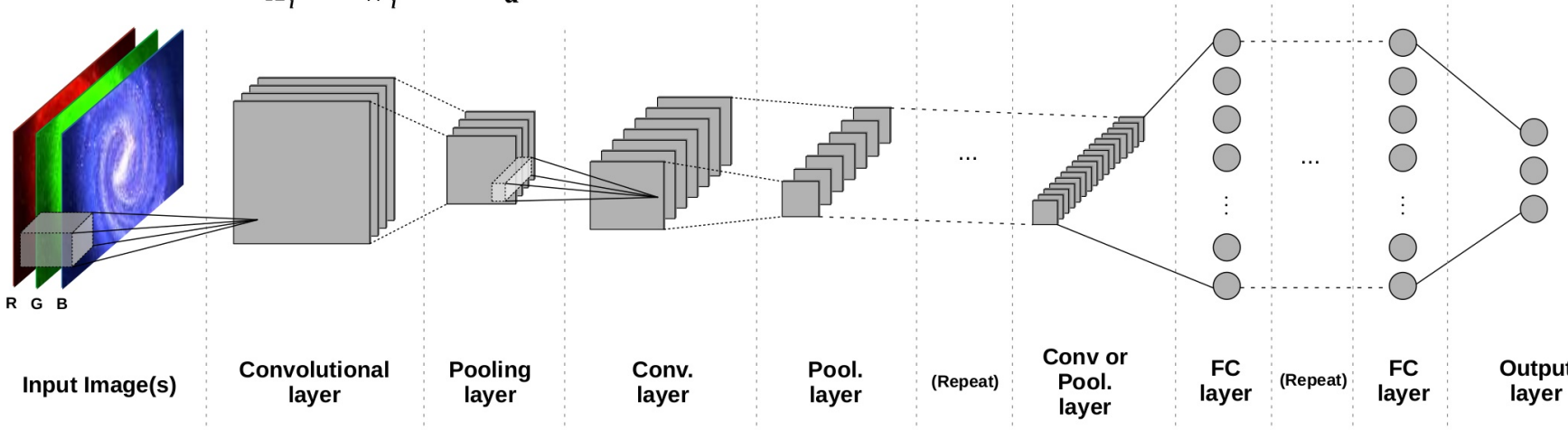
$$g\left(\sum_i X_i \circ W_i\right) = a$$

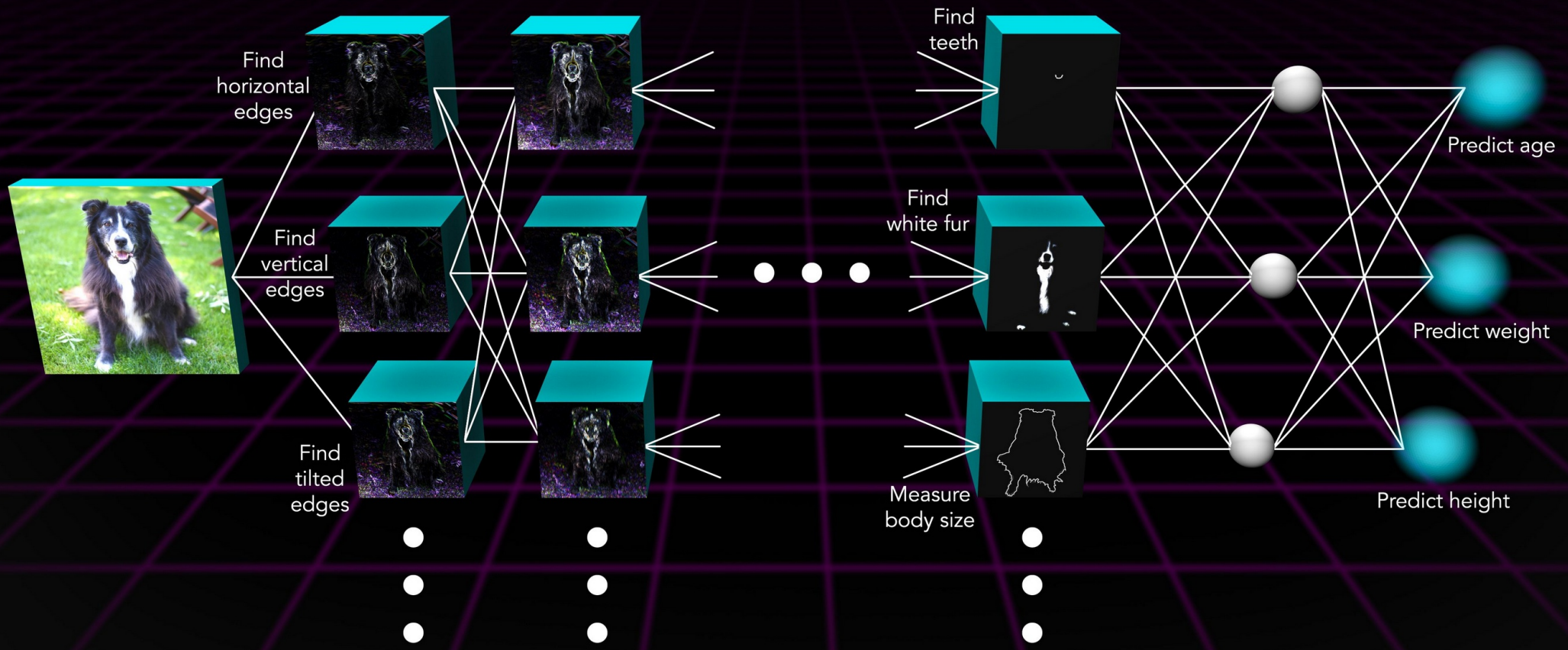
Each filter can be seen as a **single neuron** with one weight per input dimension in the filter.

BUT, the same weights are used at every position and the outputs are independent.

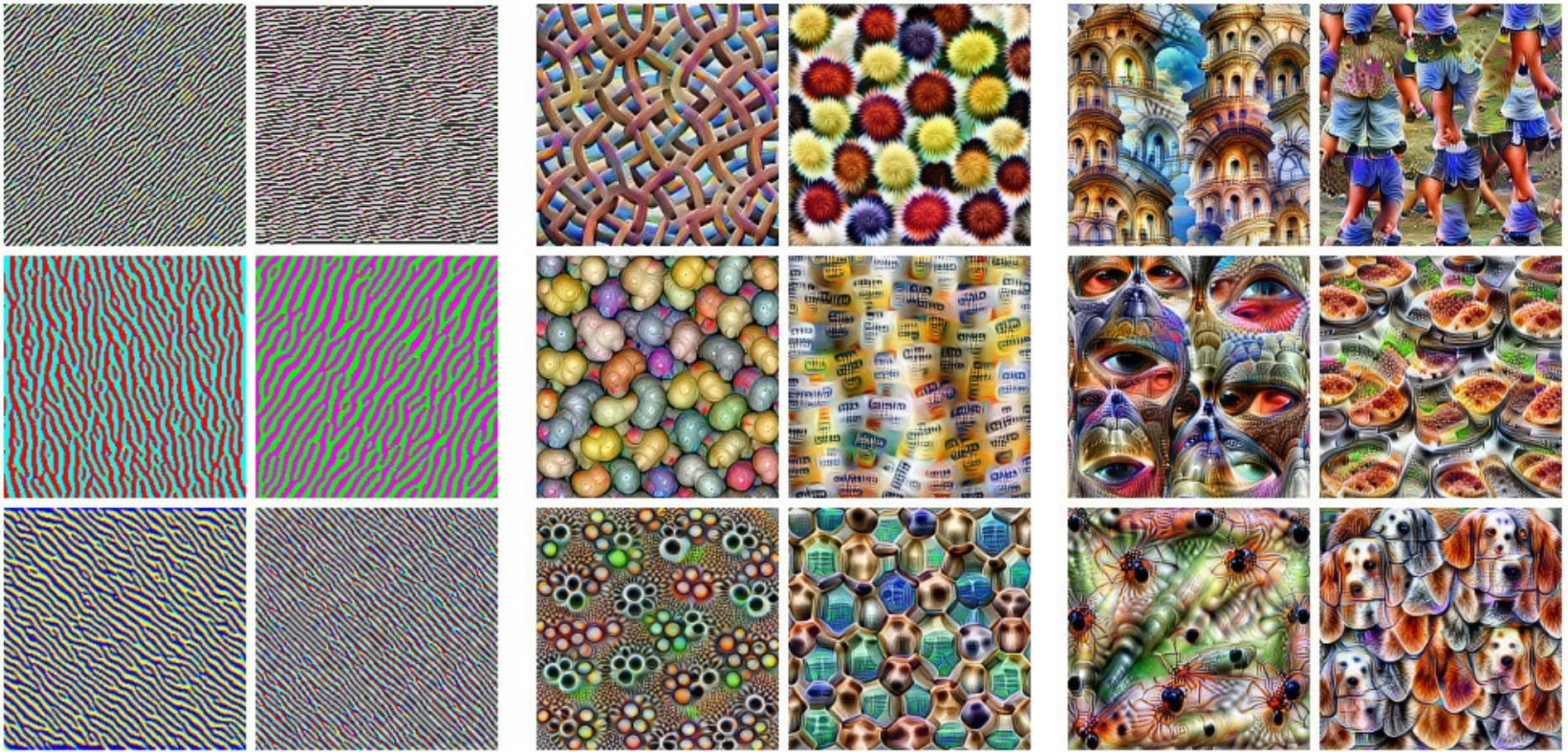
➡ **Translational equivariance !**

A network made of stacked convolutional layers can be tuned for Translation invariance.



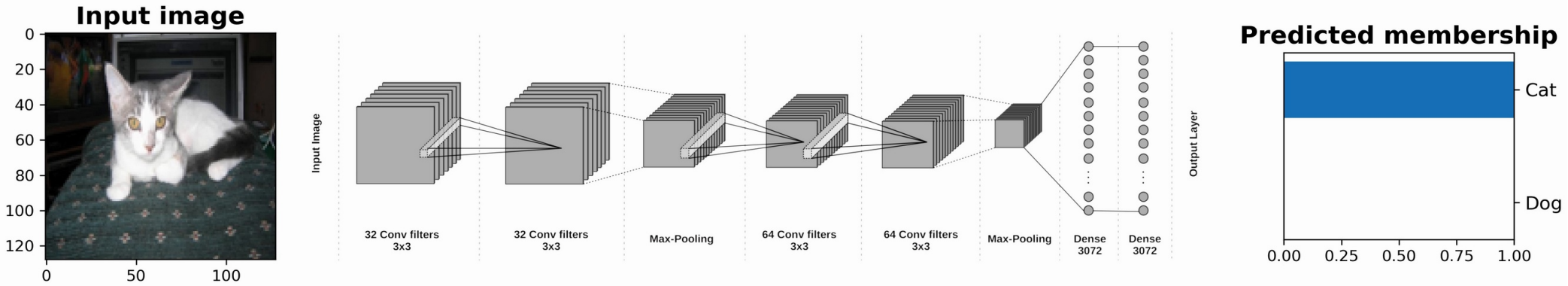

Find horizontal edges
Find vertical edges
Find tilted edges
Find teeth
Find white fur
Measure body size
Predict age
Predict weight
Predict height

Examples of filter maximization



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

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%
```

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

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

Nowadays modern CNN methods have more than 98% accuracy on ASIRRA !

“Object detection” types

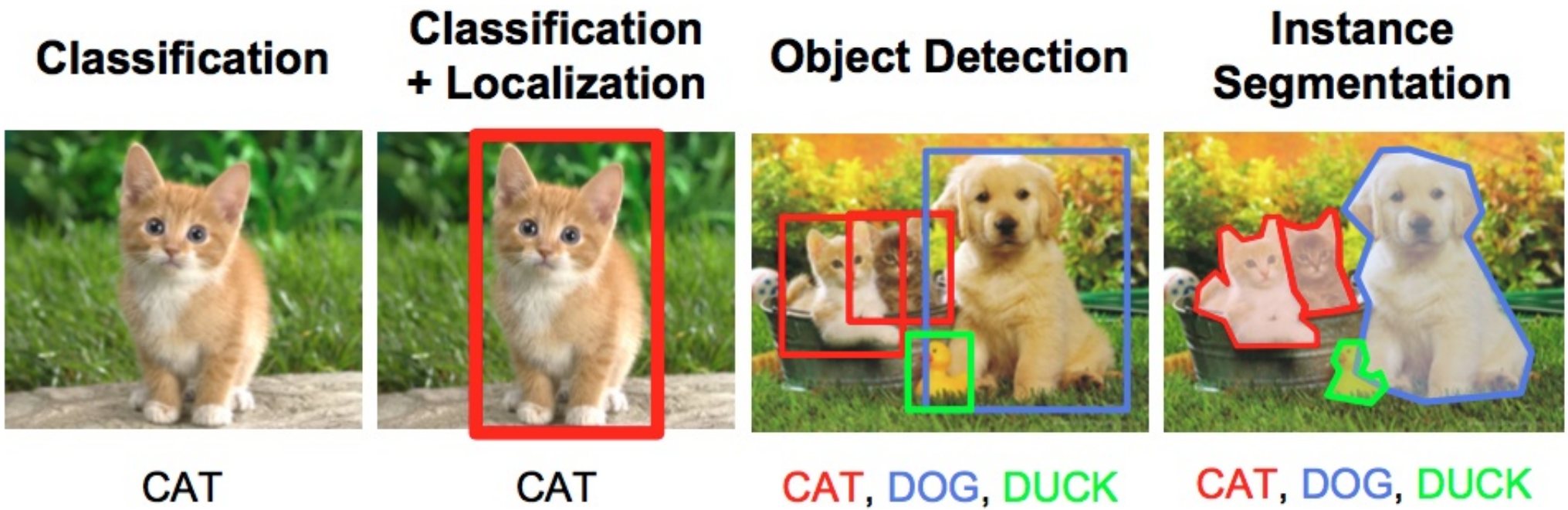


*Image from Stanford Deep Learning course cs224

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

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

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

Targets for various “tasks” :

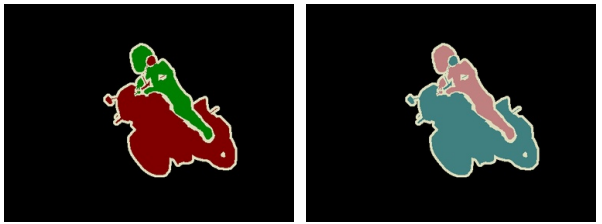
Classification

[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)

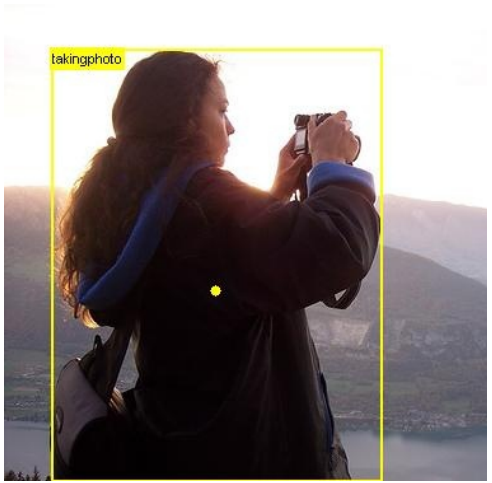
Standardized image datasets for object class recognition organized in the form of annual challenges from 2005 to 2012

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

Targets for various “tasks” :

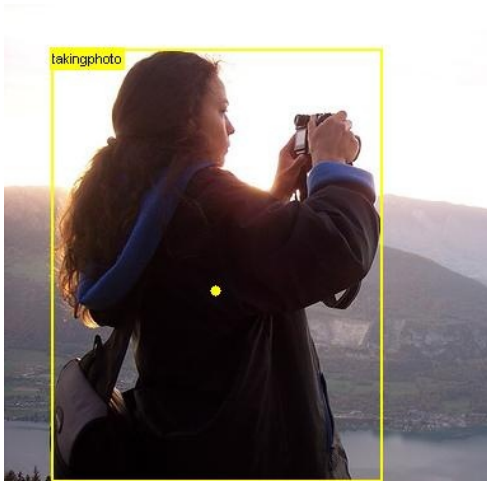
http://host.robots.ox.ac.uk/pascal/VOC/index.html

Boxless Action



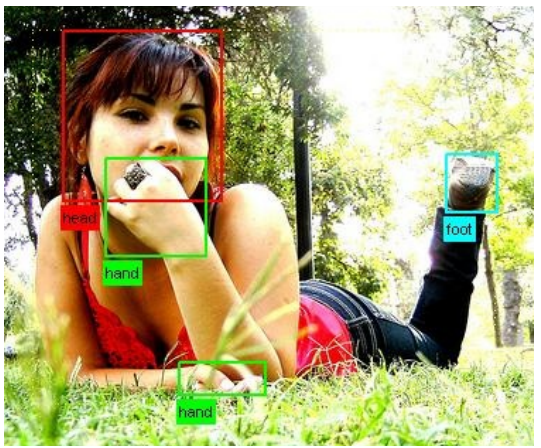
A single reference point and the associated action

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

The data from 2007 and 2012 can be combined to obtain a larger 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

CIANNA

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

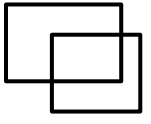
CPU OpenMP OpenBLAS
x86 Intel/AMD & ARM

GPU NVIDIA CUDA
5 dernière générations

Custom YOLO implementation


Activation


Cost


Association

Full user interface

Successfully deployed on

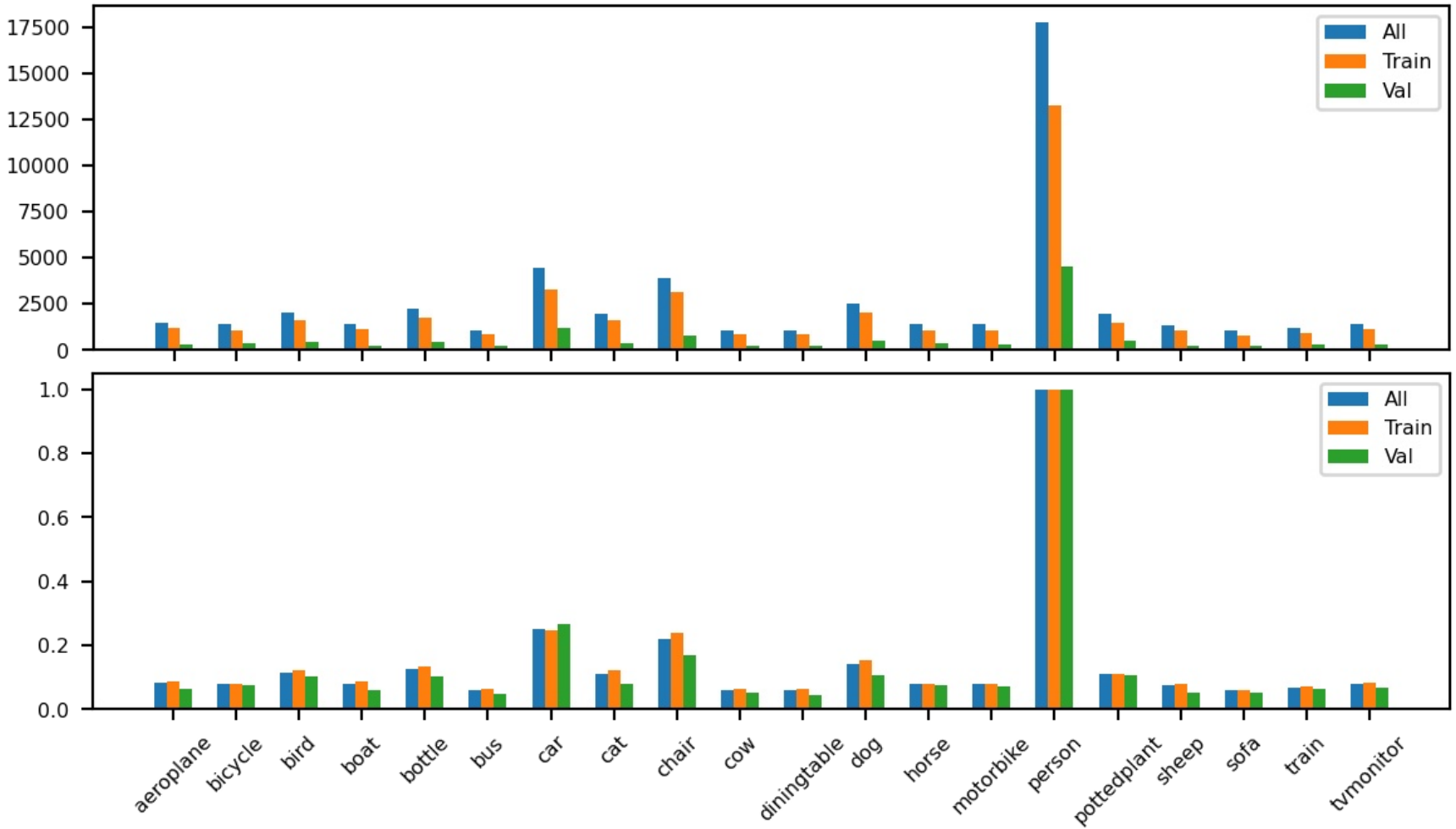
- **Laptops / Workstation**
- **Local compute serveurs**
- **Mesocenteurs**
- **Large computing facilities**

And specificities :

- **supplementary parameters per box**
- **Cascading loss**
- **Custom association process**
- **Custom NMS, and more ...**

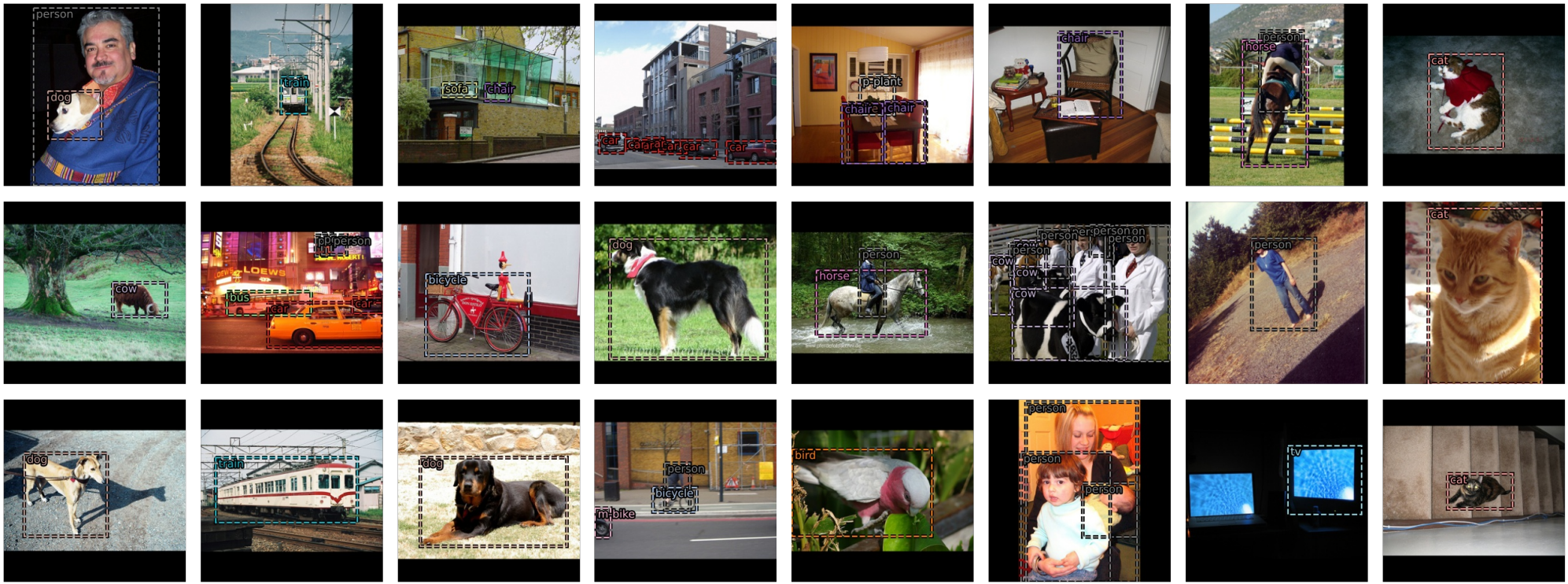
Training dataset summary statistics

Training dataset = Train 2007 + Train 2012 Valid/Test dataset = Test 2007



Training dataset summary statistics

All the images are made square and resized to 288x288 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.



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. All the 4952 Test 2007 images are kept untouched for validation.

The training images are dynamically generated using the imgaug library

- Pick a random image in the training set
- Pick a random target box in the image and perform a cutout and rescale
- Apply augmentation (Flip, blur, contrast, translate, rotate, etc.)

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

Some examples of training images:

A - Simple classifier using object cutouts

Suggested network architecture :

```
cnn.conv(f_size=i_ar([3,3]), nb_filters=16, 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=32, 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=64, 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=128, padding=i_ar([1,1]), activation="RELU")
cnn.dense(nb_neurons=256, activation="RELU", drop_rate=0.2)
cnn.dense(nb_neurons=nb_class, activation="SMAX")
```

Confusion matrix on the validation set for the pre-trained network at epoch 8000 :

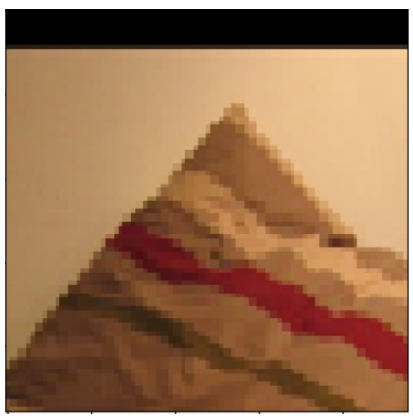
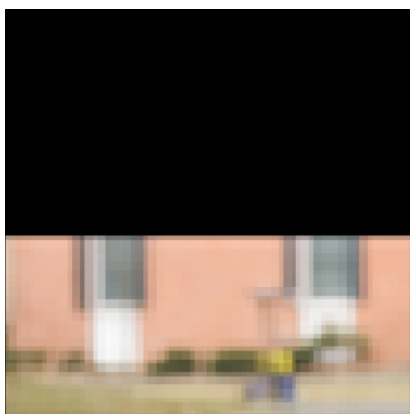
**********									ConfMat	**********											Recall
200	1	10	16	0	1	13	0	1	0	3	1	1	1	7	0	1	1	5	0		76.34%
3	264	3	2	1	0	1	0	5	1	3	3	1	19	27	3	0	0	1	0		78.34%
5	4	277	8	3	0	8	15	4	2	0	19	5	5	71	9	11	0	0	1		61.97%
18	1	4	184	2	4	15	0	4	1	2	1	1	1	6	3	0	1	12	1		70.50%
2	5	7	6	271	1	7	2	21	1	13	6	1	4	84	19	1	0	1	7		59.04%
1	0	0	5	0	155	24	0	5	0	1	1	0	2	1	0	0	0	17	1		72.77%
13	9	6	11	7	23	1029	0	6	0	1	1	0	18	29	3	0	4	12	7		87.28%
1	2	14	4	2	0	4	211	7	4	3	48	6	2	31	4	6	6	1	2		58.94%
1	9	12	3	16	4	10	2	514	1	20	5	2	7	101	10	0	17	12	10		67.99%
0	0	6	4	1	0	1	1	5	135	0	18	36	0	10	1	23	0	2	0		55.56%
2	4	4	10	1	1	5	2	19	0	110	0	2	2	25	3	0	10	6	0		53.40%
4	5	26	2	1	0	2	46	9	12	3	248	27	3	81	0	7	11	1	1		50.72%
0	5	6	0	0	0	1	3	2	15	0	14	244	3	41	1	9	2	1	1		70.11%
1	17	0	0	4	0	17	0	0	0	2	2	0	233	43	5	0	0	0	0		71.91%
3	31	37	6	47	4	29	22	68	9	13	44	26	60	3933	34	6	29	6	10		89.04%
1	3	14	4	6	3	2	5	15	0	3	4	1	4	25	376	0	1	3	2		79.66%
0	0	15	0	0	0	0	7	0	18	1	21	8	0	6	1	152	2	1	0		65.52%
4	3	1	7	0	1	8	2	36	1	6	4	1	2	27	3	1	125	6	1		52.30%
11	3	2	12	4	6	5	2	0	0	2	0	0	0	2	2	4	0	226	1		80.14%
0	1	1	2	4	0	6	1	17	0	0	0	0	0	8	2	2	1	3	259		84.36%
Prec.	74.07%	71.93%	62.25%	64.34%	73.24%	76.35%	86.69%	65.73%	69.65%	67.50%	59.14%	56.36%	67.40%	63.66%	86.29%	78.50%	68.16%	59.52%	71.52%	85.20%	Acc 77.31%

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**
  - ➞ **20% of the time, 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” empty examples using the validation images

The performance of this new classifier should be similar with ~92% recall on background examples

B - Training a sliding window detector

The sliding window principle

Classifier = identify individual objects relatively centered

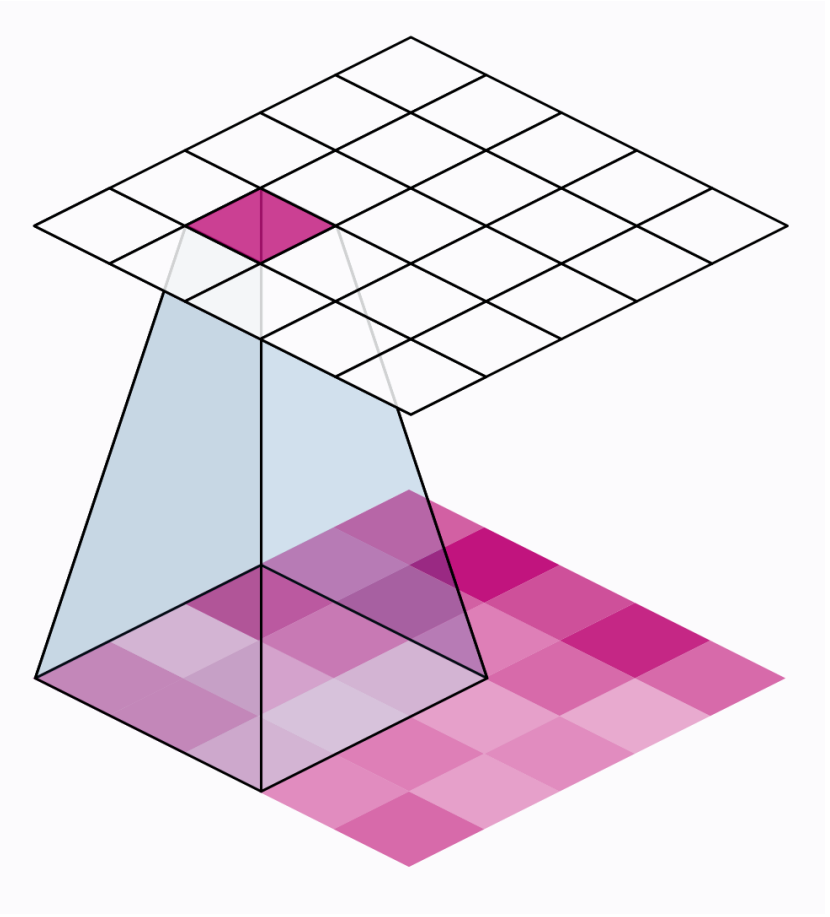
Can be used as a sliding window detector:

→ Split a larger image in “chunks”

→ Overlapping Chunks for more « centered » objects

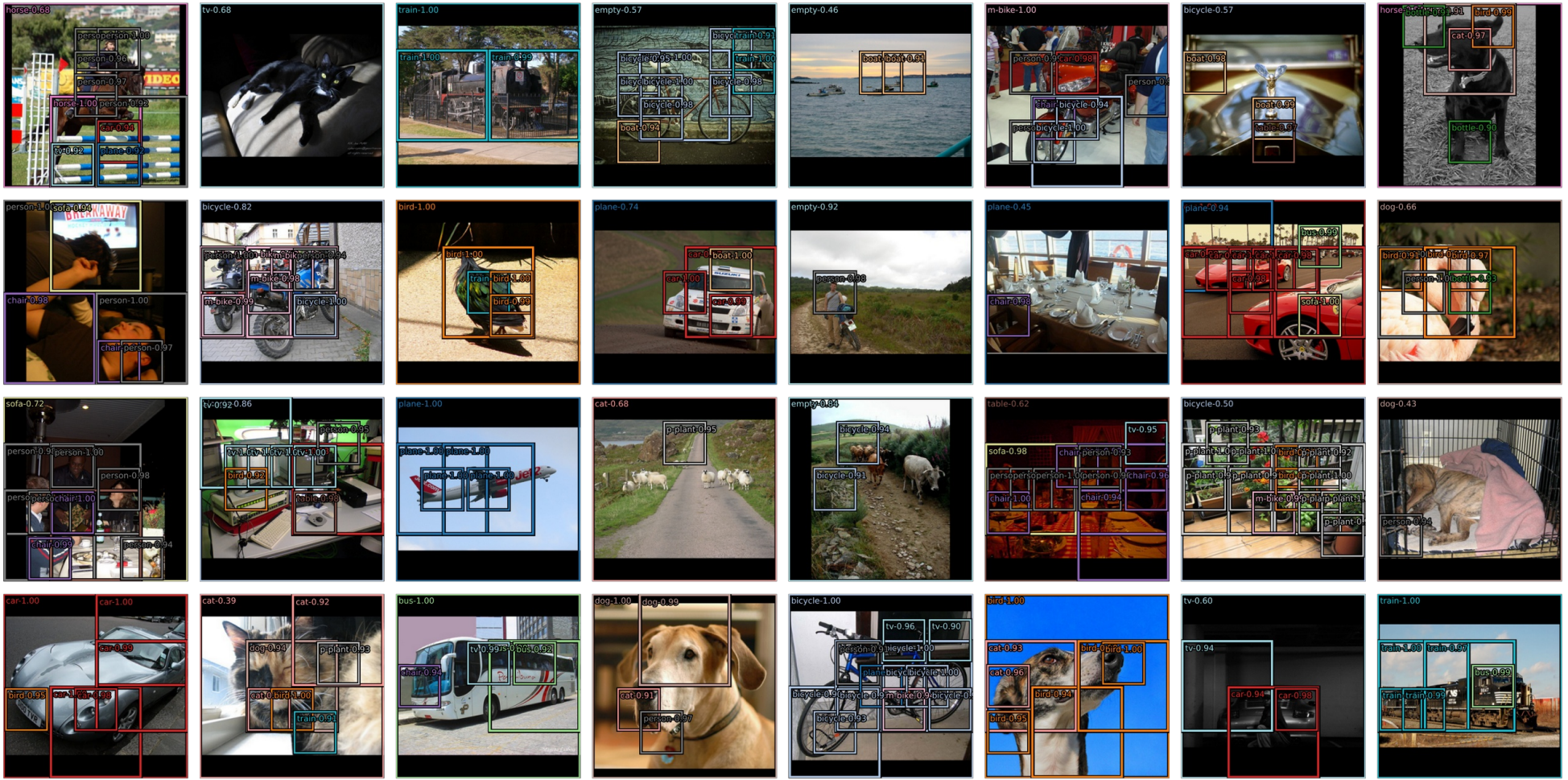
Process similar to a convolution,
with Classifier ≈ Filter

→ Using multiple window sizes: increase the chance of a proper cutout for objects of various sizes



*In the present case we use **3 window sizes** [288, 144,72] with a respective stride of [0, 72, 36].*
The number of « chunk » par size is [1, 9, 49], corresponding to grids of [1x1, 3x3, 7x7].

B - Training a sliding window detector

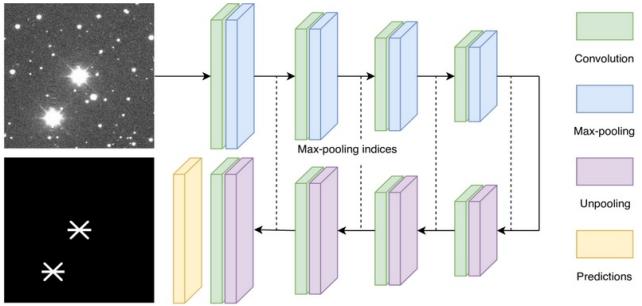


Questions : How to filter overlapping detection ? How to evaluate prediction performance ?

CNN architectures for object detection

U-Nets

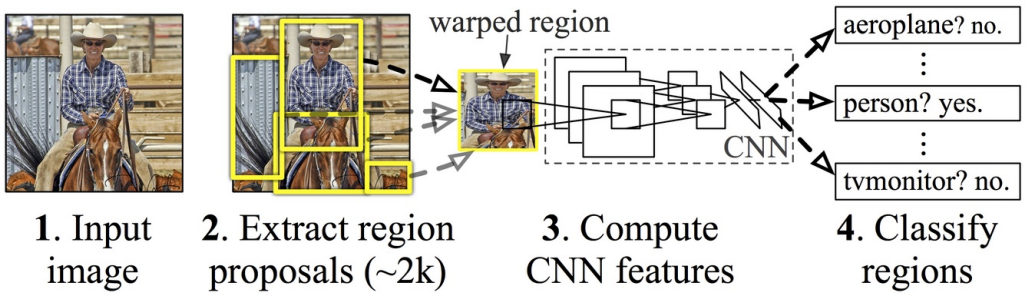
Pros: segmentation maps, shallow latent space, ...



Region-based

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

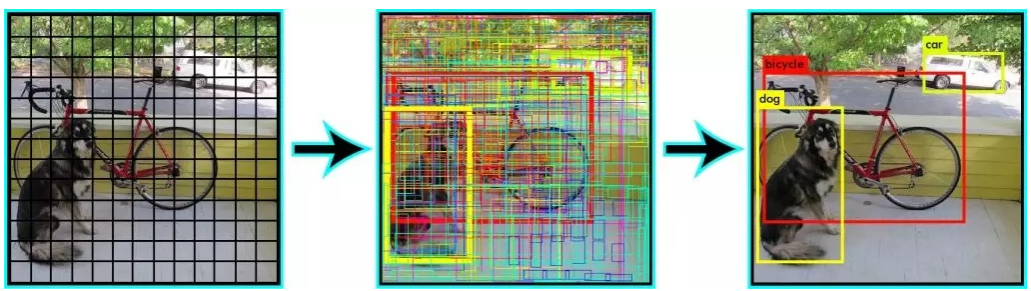
Pros: Best accuracy, ...



Regression-based

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

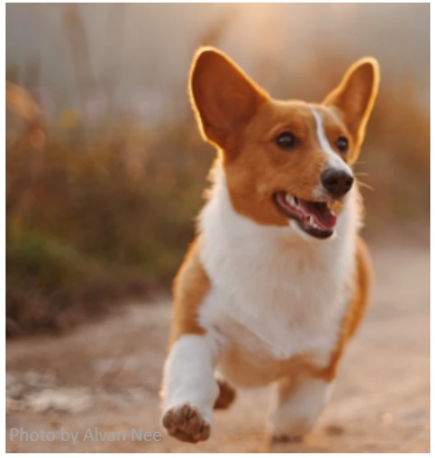
Pros: Very Fast, straightforward architecture,...



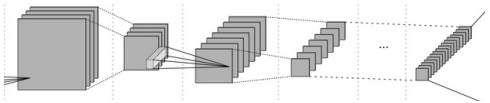
You Only Look Once – YOLO !

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

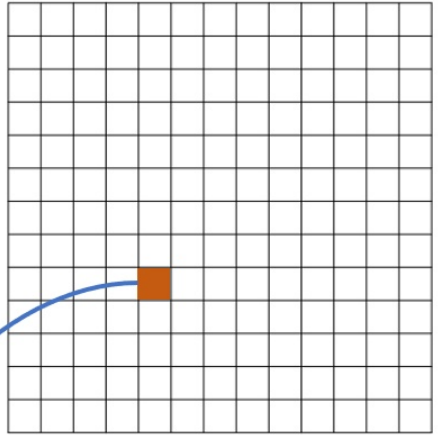
**Images from blog post and Redmon papers*



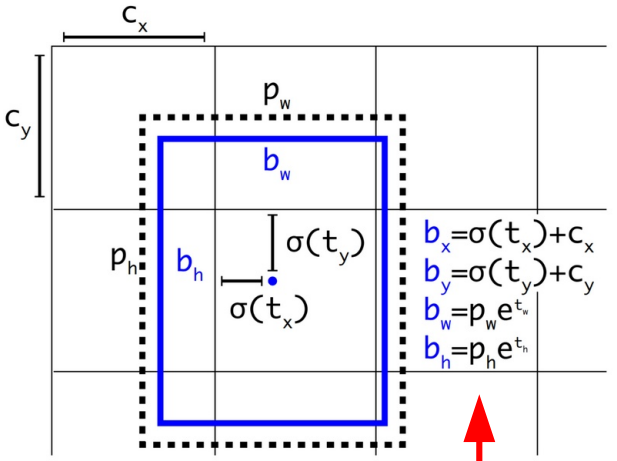
Pre-processing Image



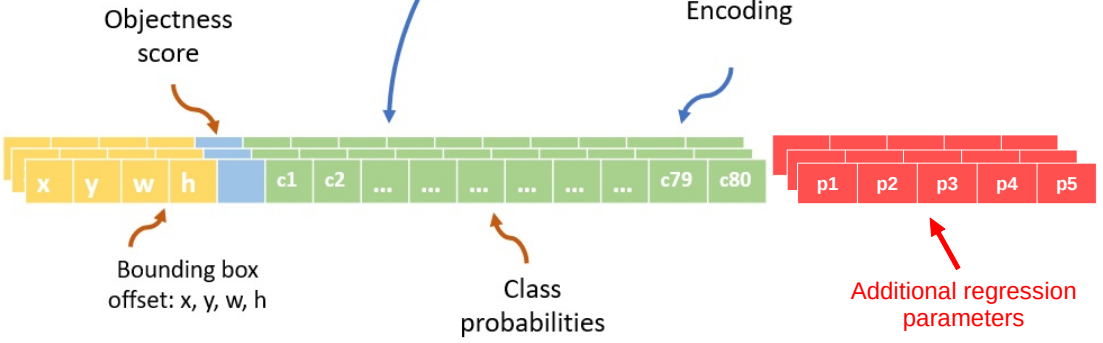
YOLO Network



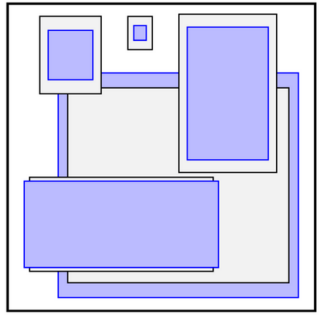
Encoding



p_w
p_h

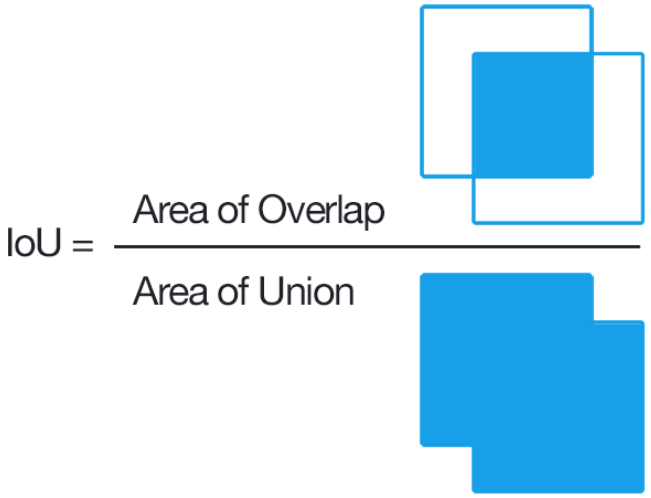


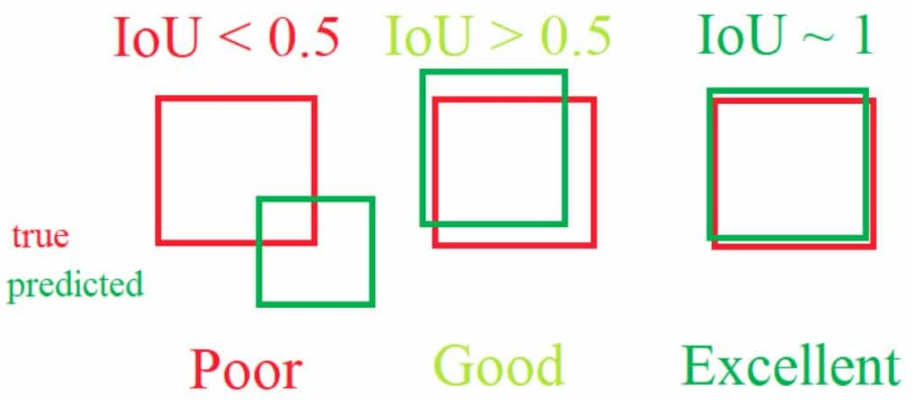
Box size priors



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 » ?

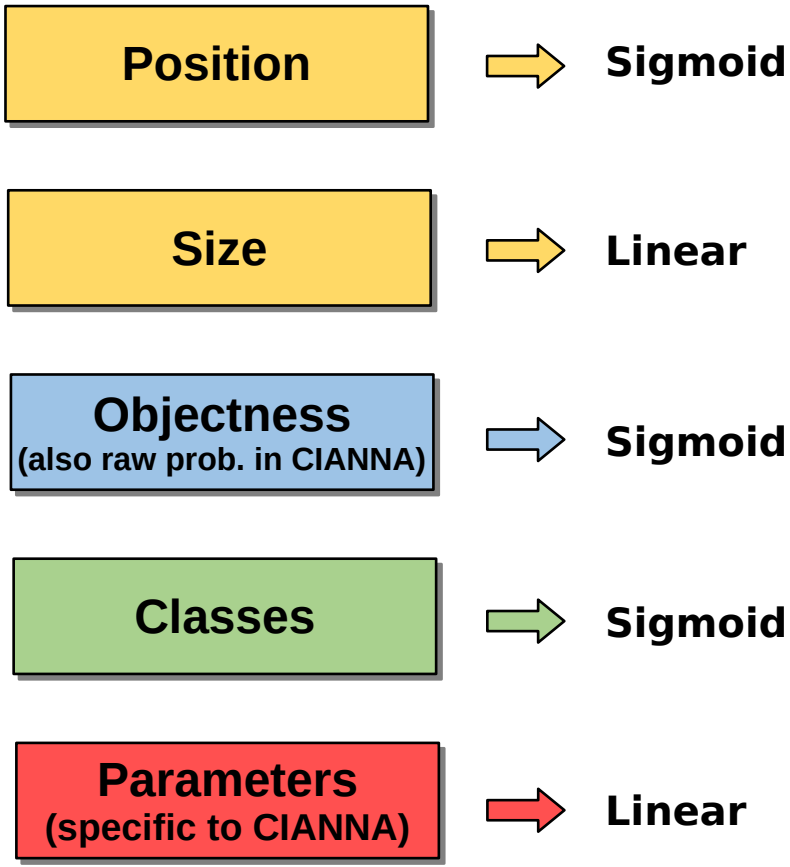




For each box the network will predict an “objectness” score defined as:

$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

The YOLO activation / Loss function



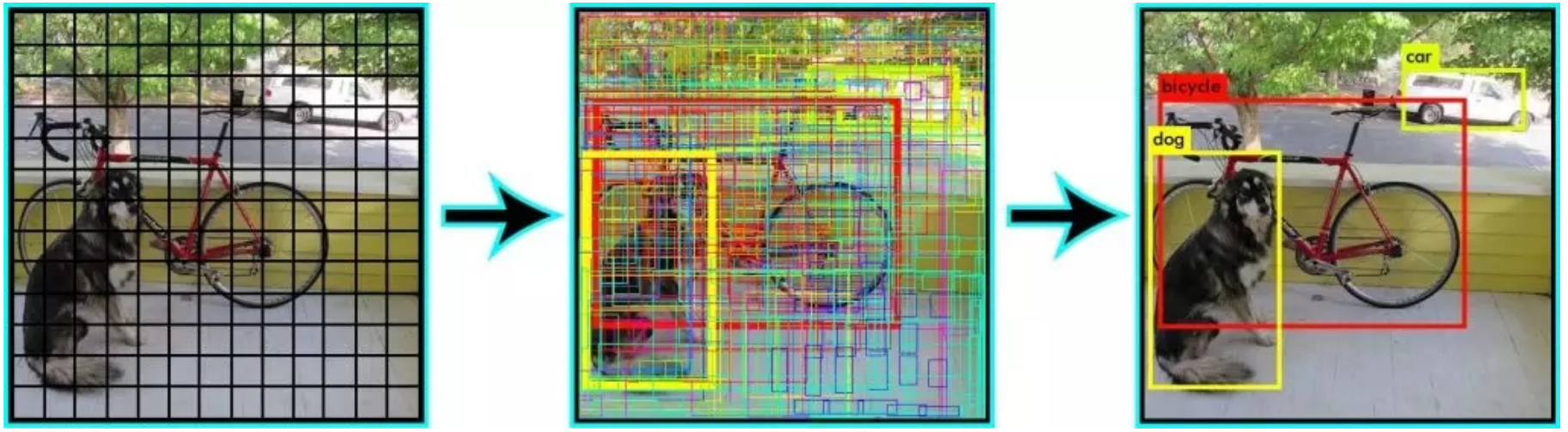
All error terms are using MSE

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 targets
  - 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

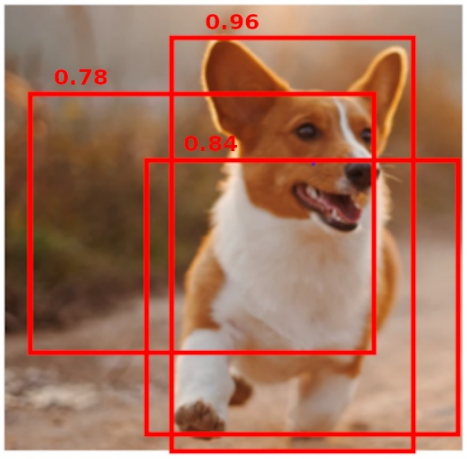
Post-processing: Non Max Suppression

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

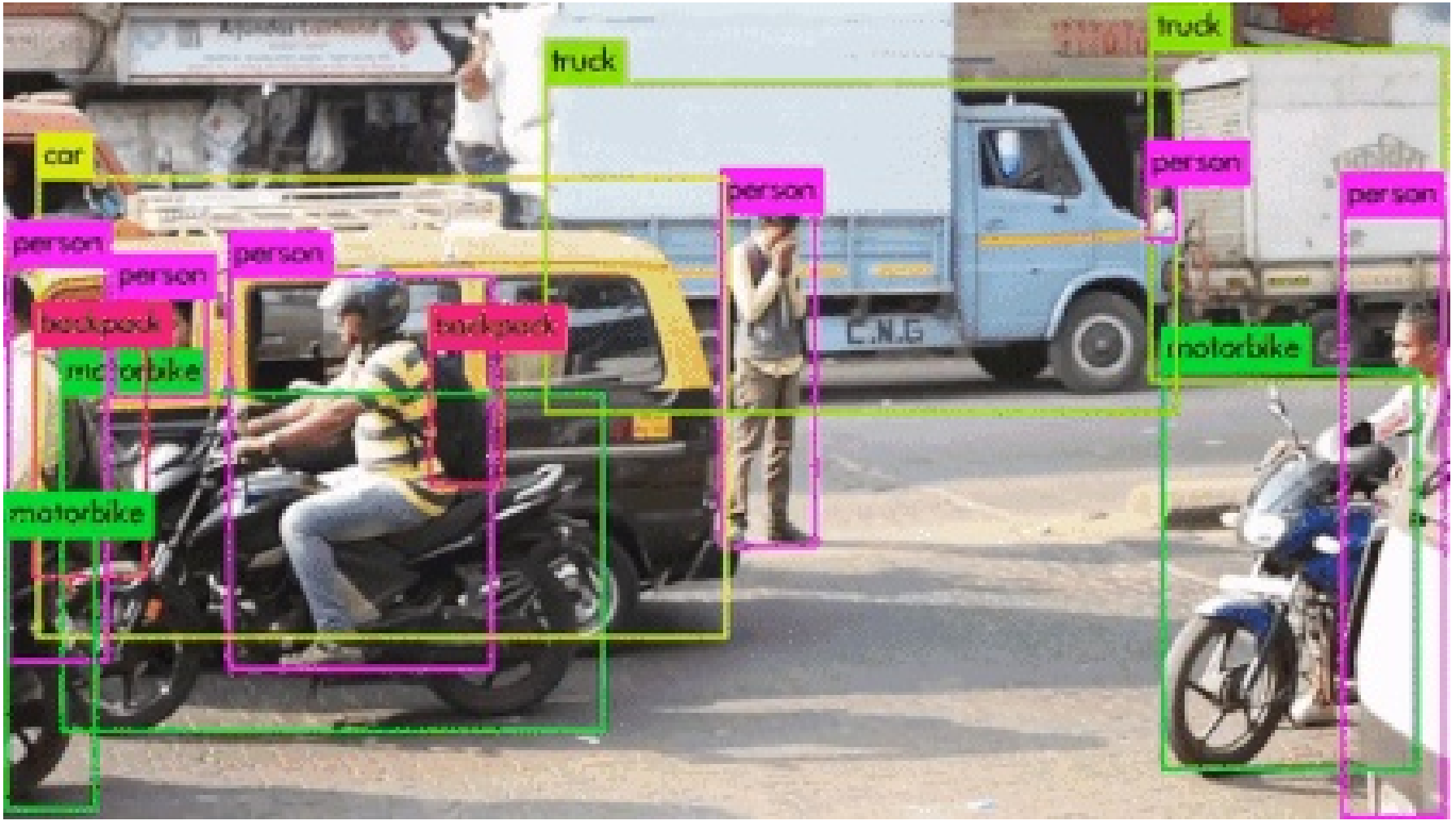


1) Most probable boxes are kept using a threshold in objectness

2) NMS takes the most probable box and removes overlapping ones based on an IoU threshold. Repeat.



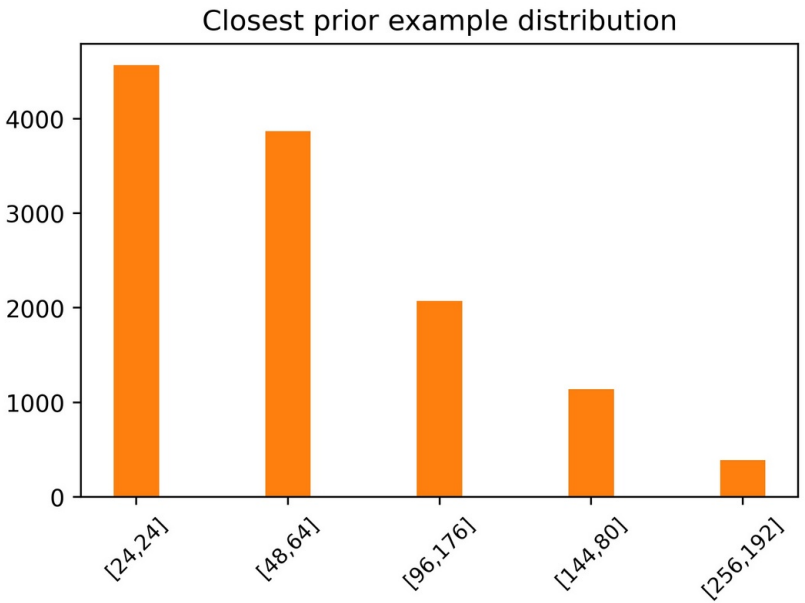
“Real-time” example

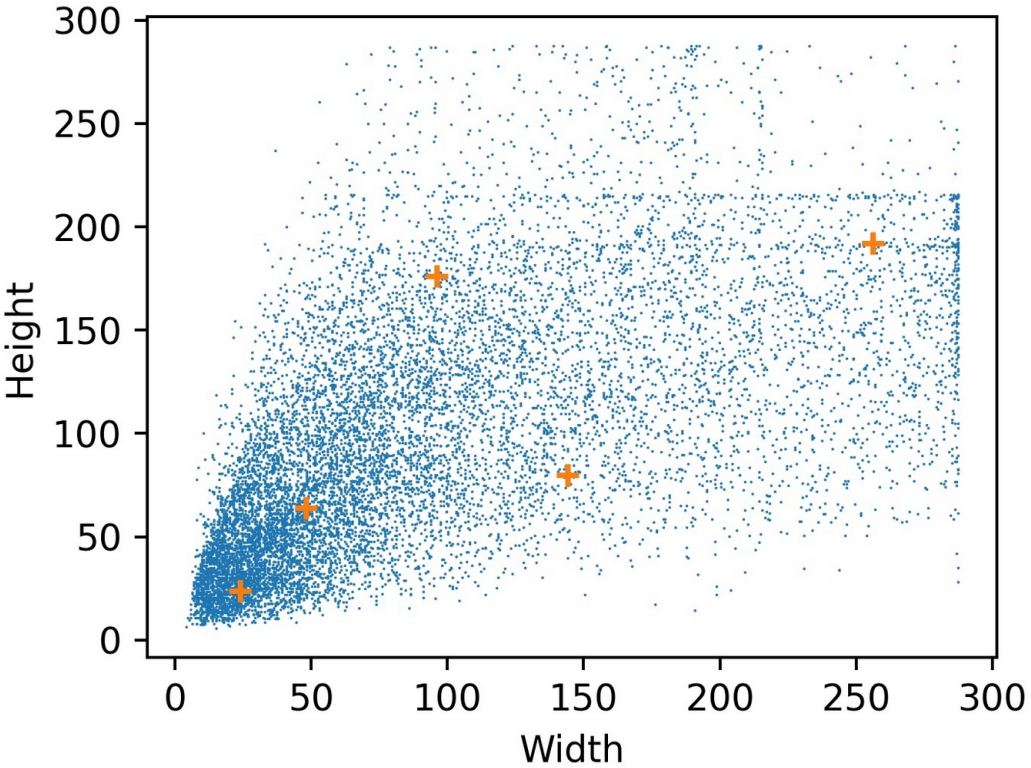


C – Training a YOLO network on PASCAL VOC

Provide a pre-trained network and a suggested architecture

Input dimension: 288x288
Spatial reduction factor: 32
→ Output grid: 9x9





Based on testset box size distribution
Nb priors: 5
[24,24];[48,64];[96,176];[144,80];[256,192]

C – Training a YOLO network on PASCAL VOC

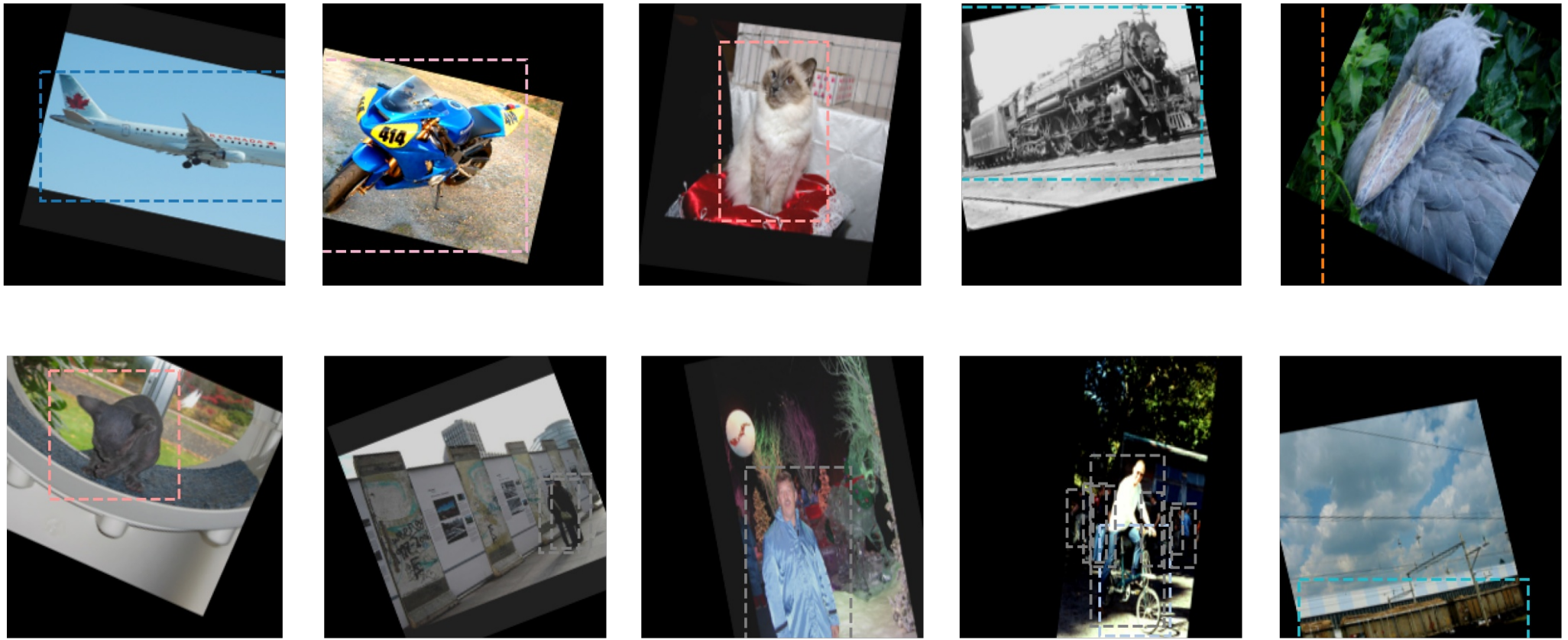
Suggested network architecture *(also the one of the provided pre-trained network)*:

```
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=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=256, padding=i_ar([1,1]), activation="RELU")
cnn.conv(f_size=i_ar([3,3]), nb_filters=256, padding=i_ar([1,1]), activation="RELU")
cnn.conv(f_size=i_ar([1,1]), nb_filters=512, padding=i_ar([0,0]), activation="RELU", drop_rate=0.1)
cnn.conv(f_size=i_ar([1,1]), nb_filters=nb_yolo_filters, padding=i_ar([0,0]), activation="YOLO")
```

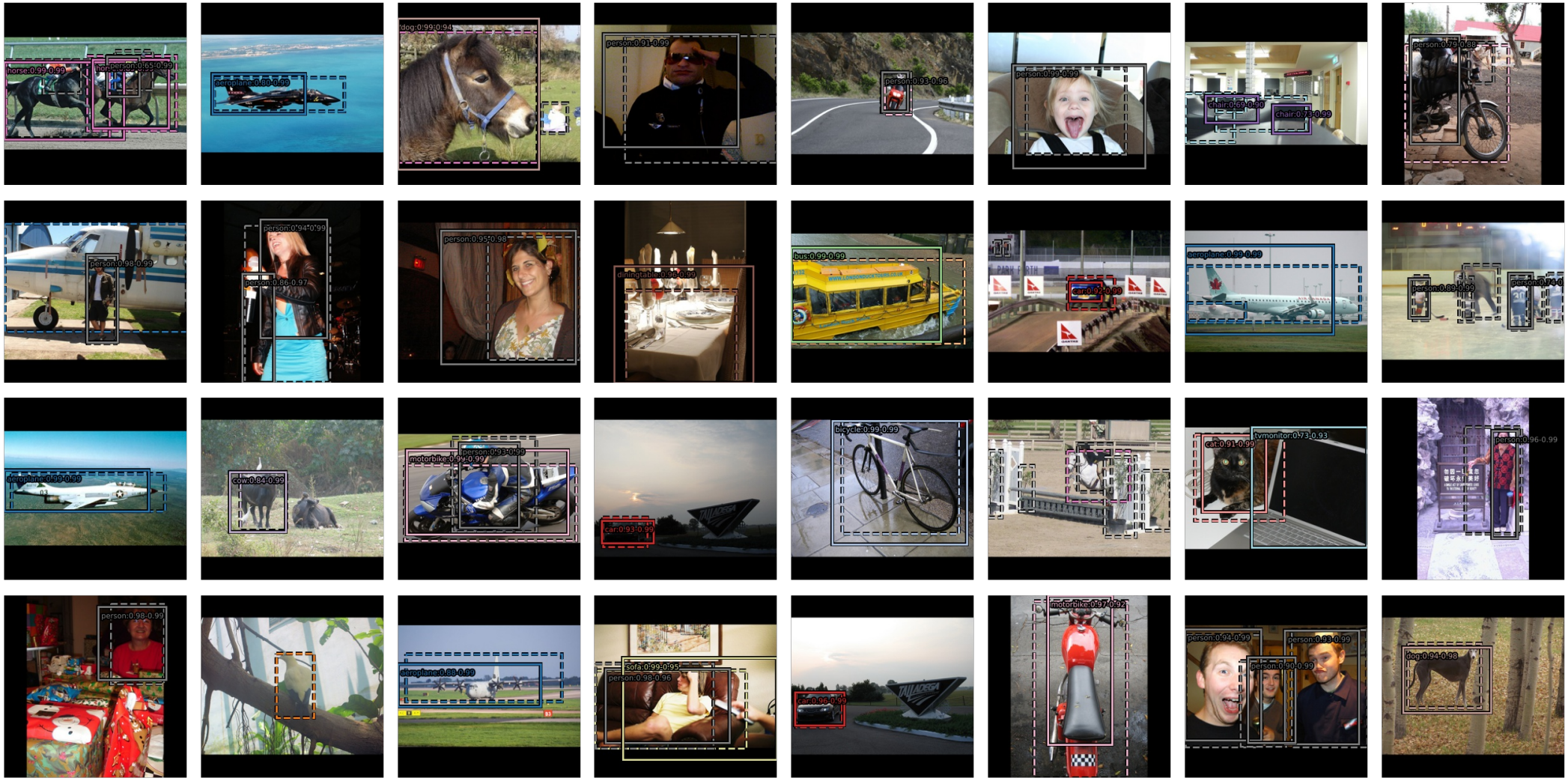
Inspired by the YOLO-Light V2 and Tiny-Yolo V2 architectures

C – Training a YOLO network on PASCAL VOC

Advanced data augmentation using imgaug



C – Training a YOLO network on PASCAL VOC

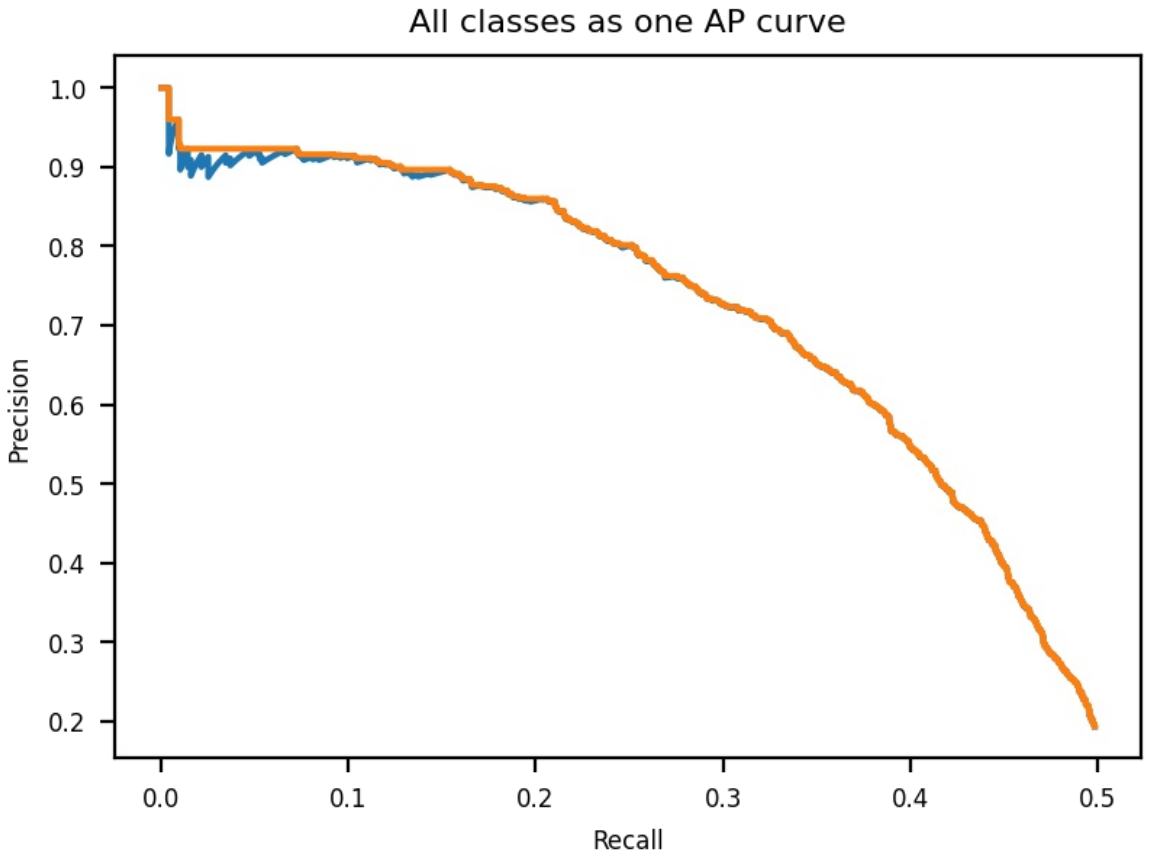


Detection performance metric

Match if IoU < V between prediction and target → Usually V = 0.5

The precision-recall curve

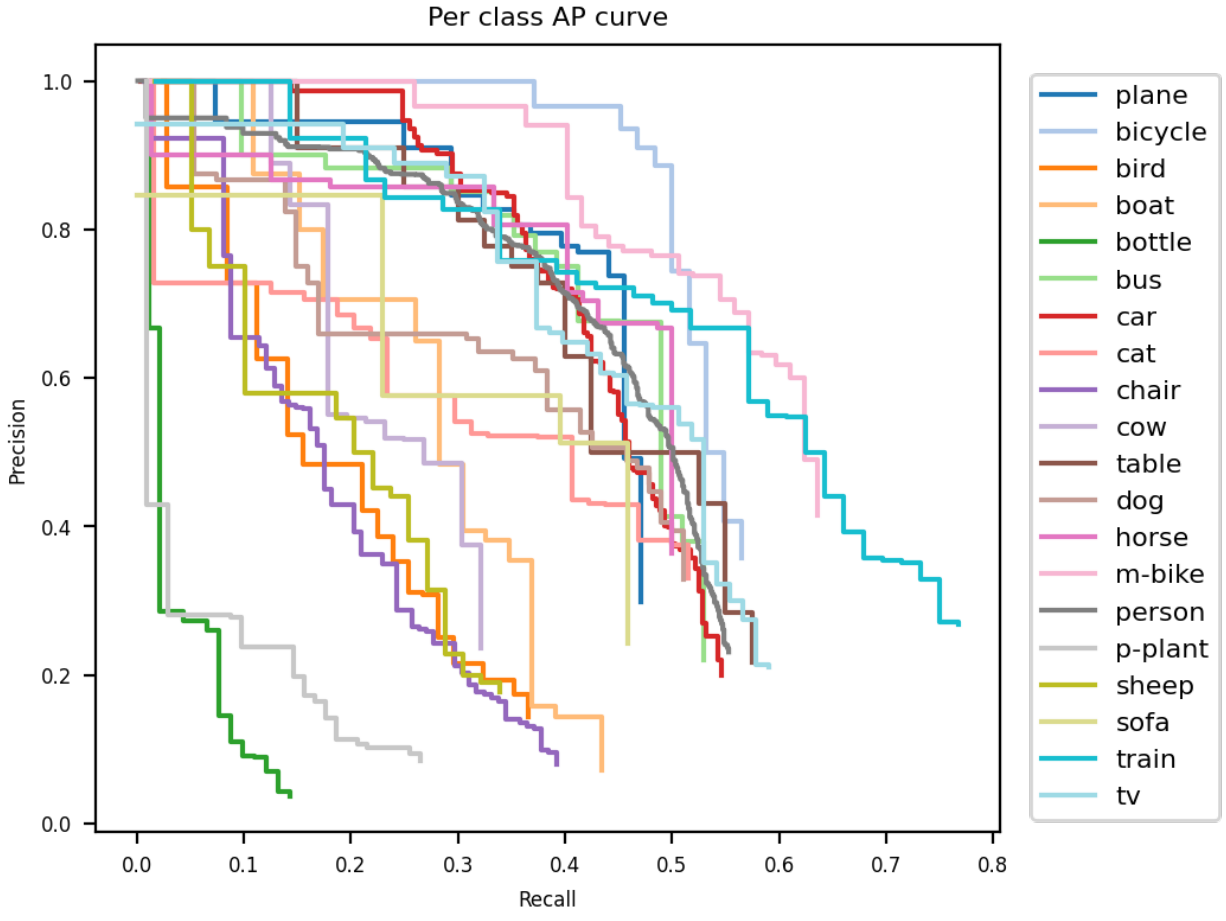
- Predictions are sorted by score
- For each data point, compute the precision and recall using all the predictions that are scored higher, using **IoU@0.5** and correct classification as “True” criteria
- The curve is then smoothed so it can only monotonically decrease (precision is always equal to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



Per class AP and Mean AP

Each class gets its own Recall-Precision curve and AP score.
The final metric is the mean of the AP values.

Here mAP@50 ~ 32%



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