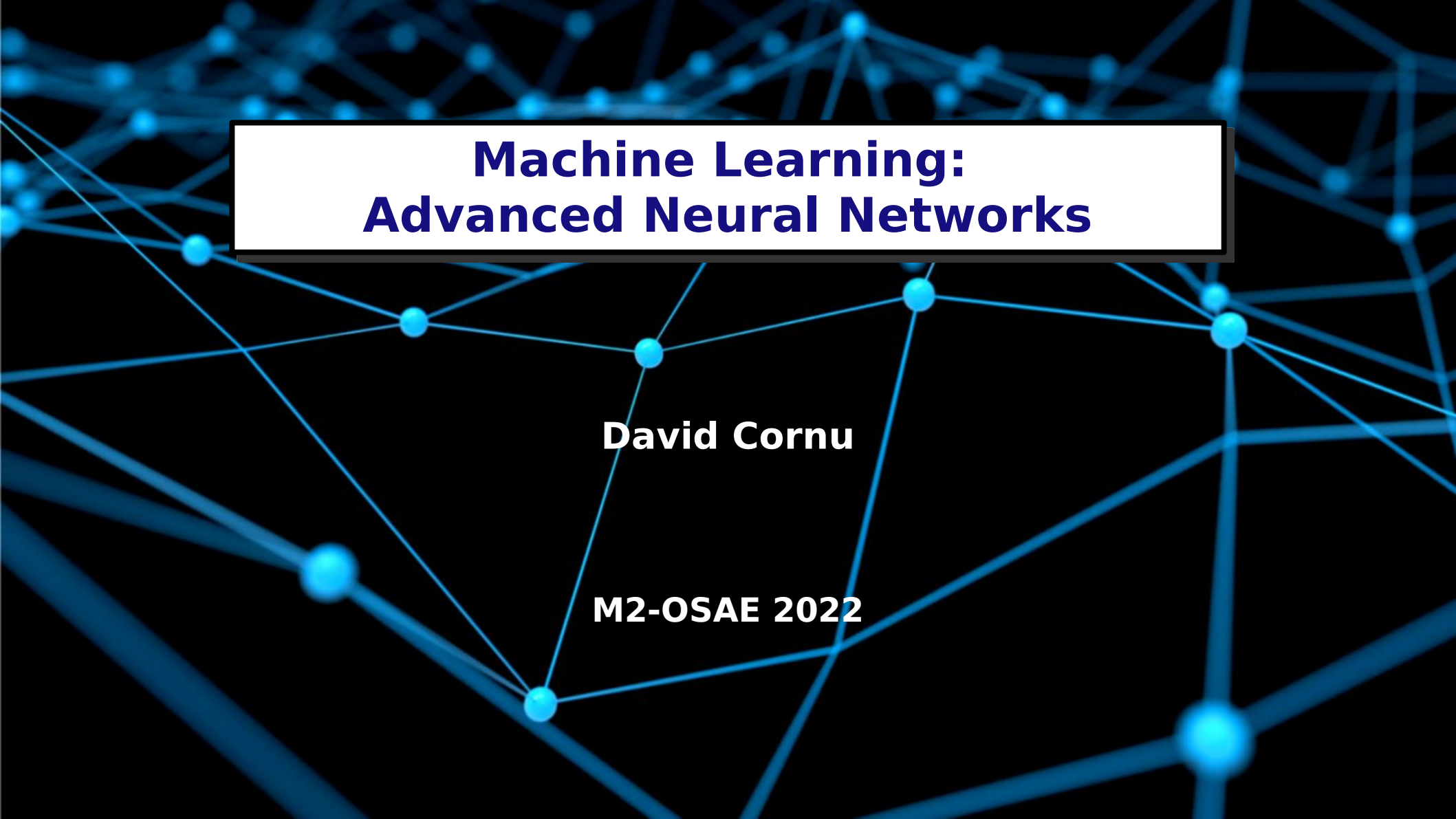


Machine Learning: Advanced Neural Networks

David Cornu

M2-OSAE 2022

Lessons materials

Slides, exercises, codes, corrections and datasets are **available on GitHub** and will be updated regularly:

http://github.com/Deyht/ML_OSAE_M2

git clone https://github.com/Deyht/ML_OSAE_M2
git pull

Or download the repository in zip file

Avoid losing your work on forced pull updates by copying all files from the cloned repository into a working directory!

Do not copy and past content from git-hub pages (lead to format errors).
Use python up to 3.10 but not more recent.

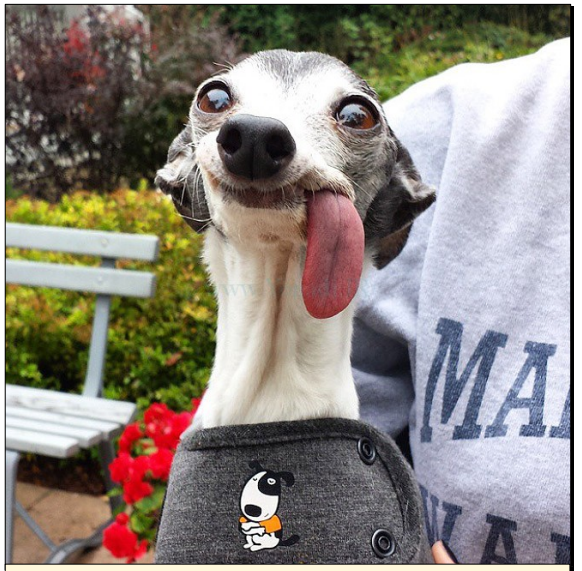
Neural Networks for images

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.



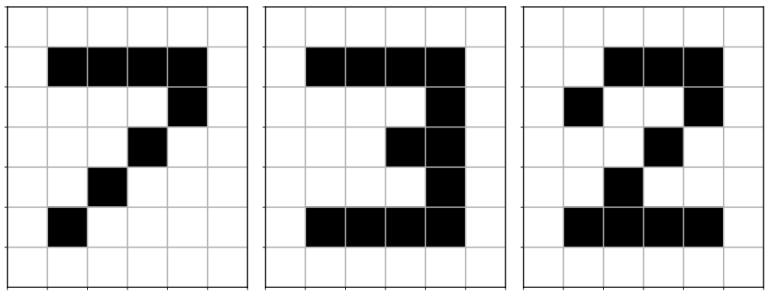
*A **highly dimensional** "dog" with ~0.5 Million pixels.*
Quite difficult to classify ...

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

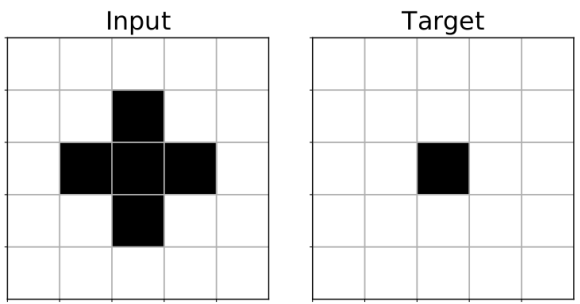
➡ **Convolutional Neural Networks** !

Spatially coherent information

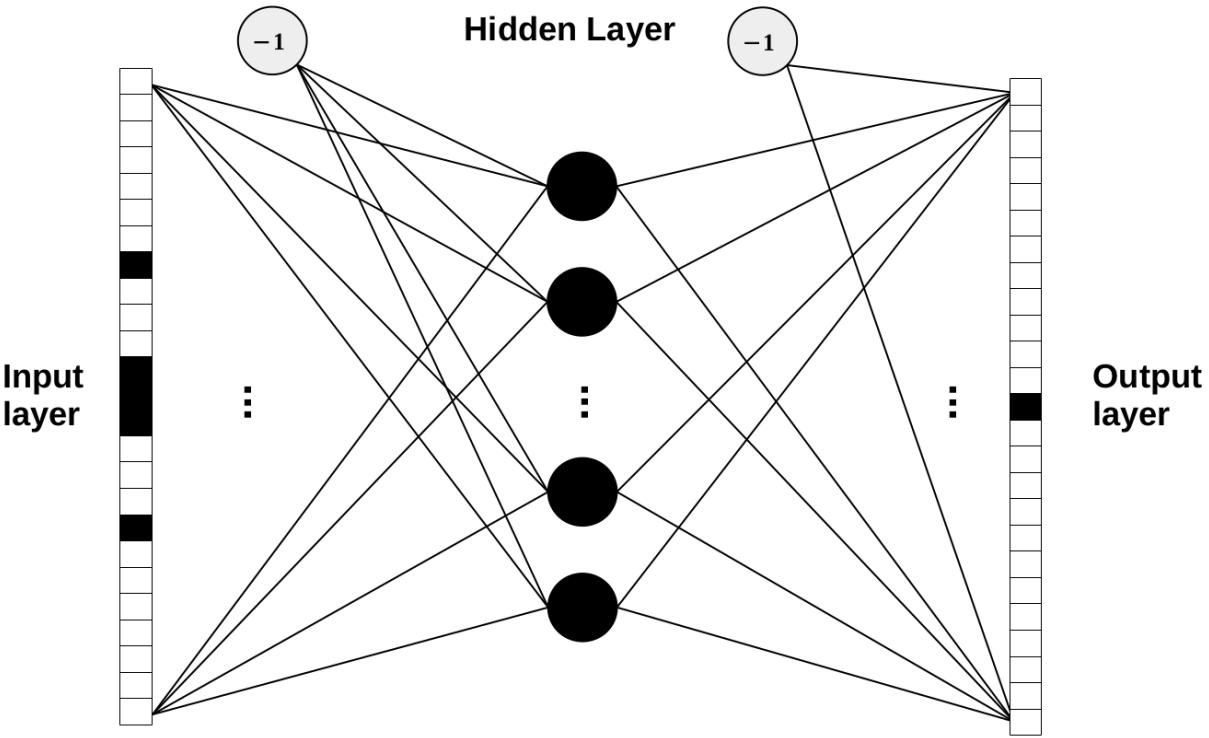
Classical ANN can deal with images by considering each pixel of an image as an individual input but it is ***STRONGLY inefficient****.*



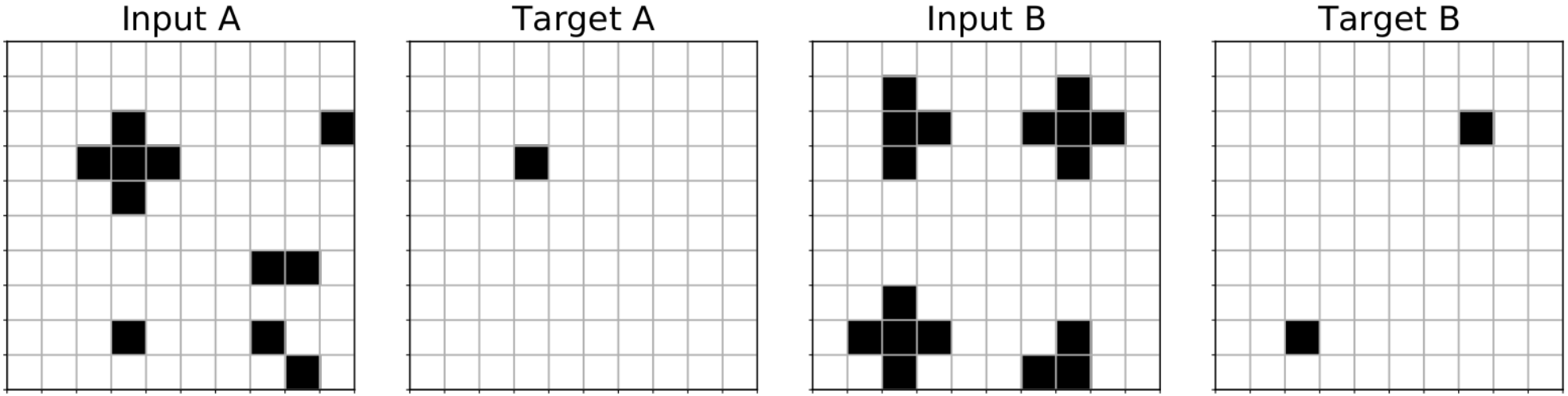
Simple digit representation as a 6x7 binary pixel image



Representation of a simple cross pattern on a 5 × 5 image as input, and the corresponding localization prediction on an equivalent size output image



Spatially coherent information : Pattern recognition

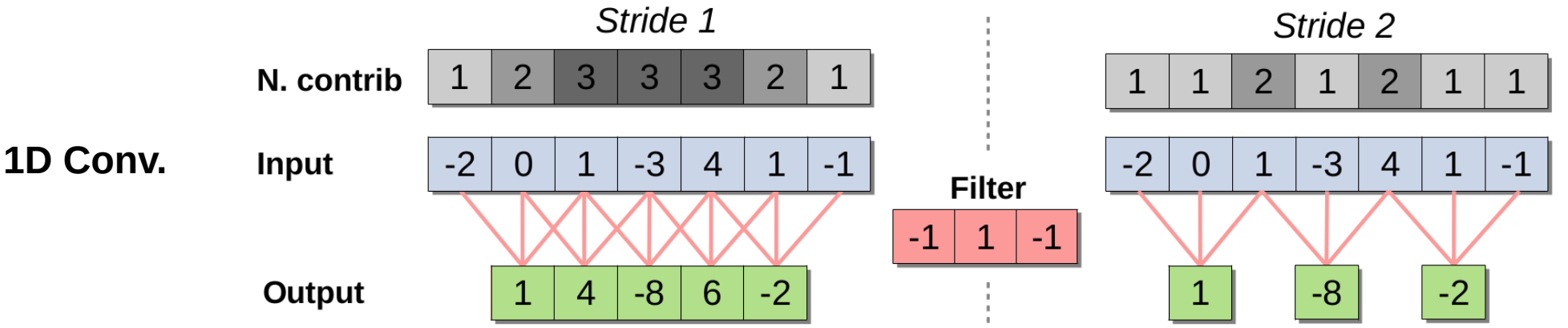


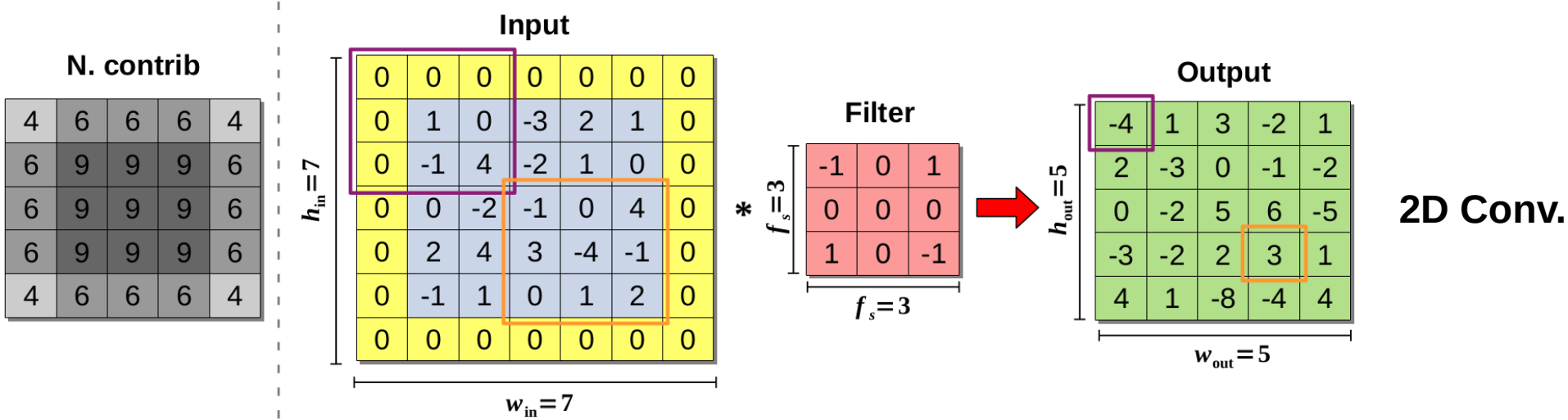
Looking for specific patterns can be automatized by scanning all the possible positions in the image.

In contrast, training a fully connected network to do the same task would require learning the presence or non-presence of the pattern at every possible position instead of learning the pattern once and only checking its presence at every position.

How to circumvent this behavior ? ➡ Use **Convolutional layers** !

Convolution filter

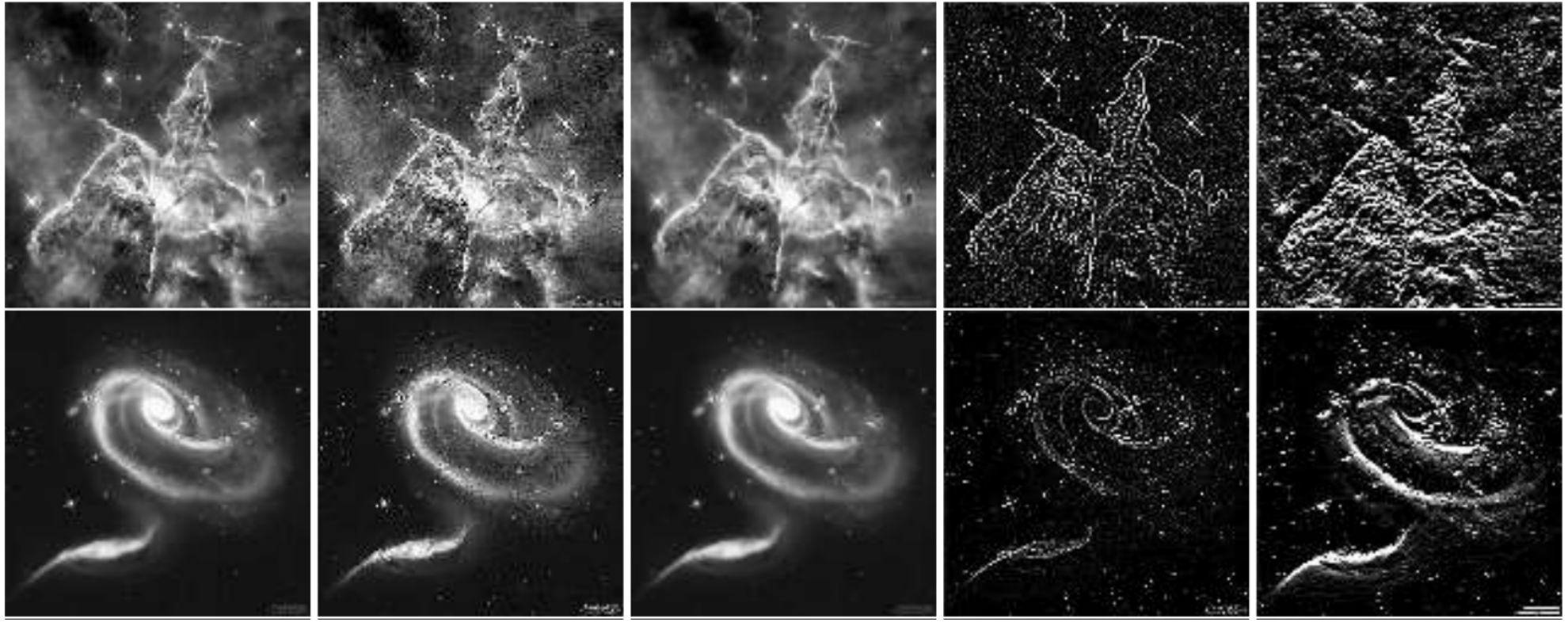




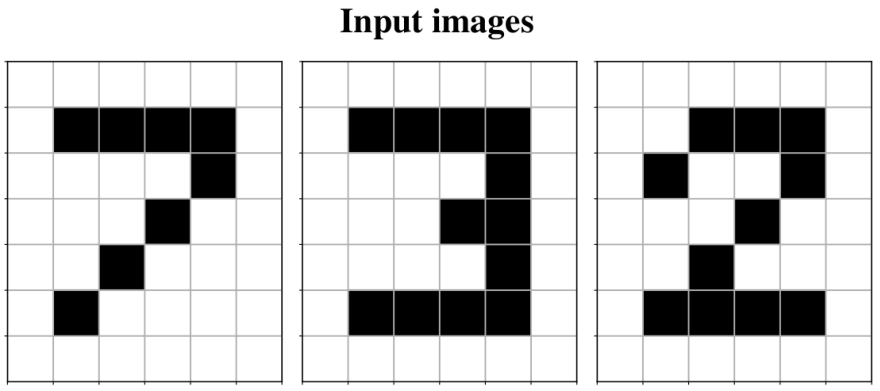
Stride 1 and Padding 1

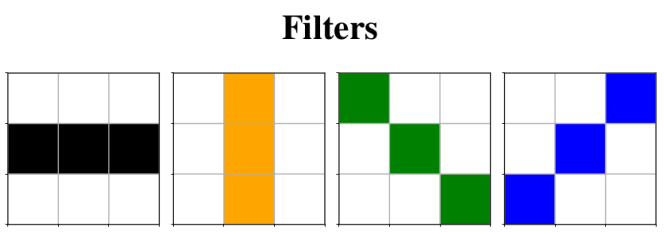
Filter effect examples

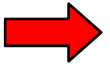
No filter	Sharpen	Gaussian blur	Edge detector	Axis elevation
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix}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

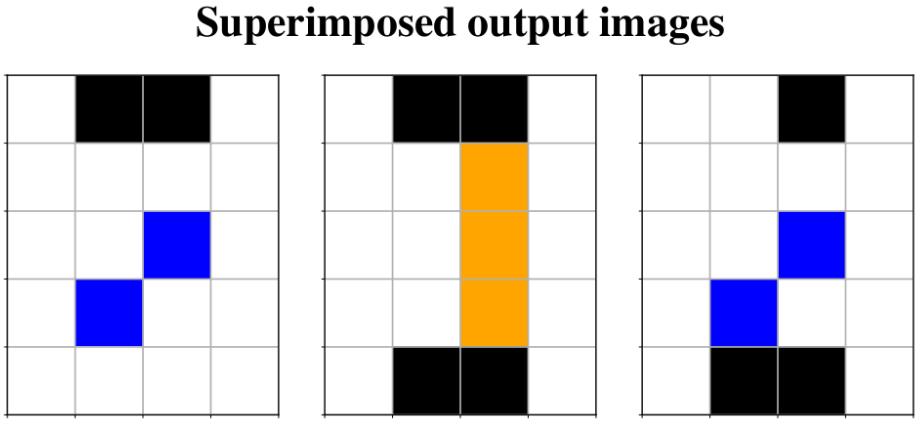


Pattern recognition with several filters







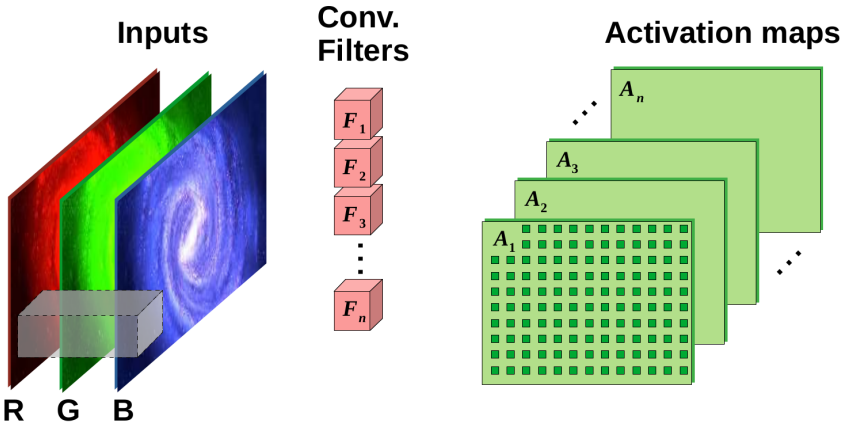


Each filter will produce its own activation map.

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

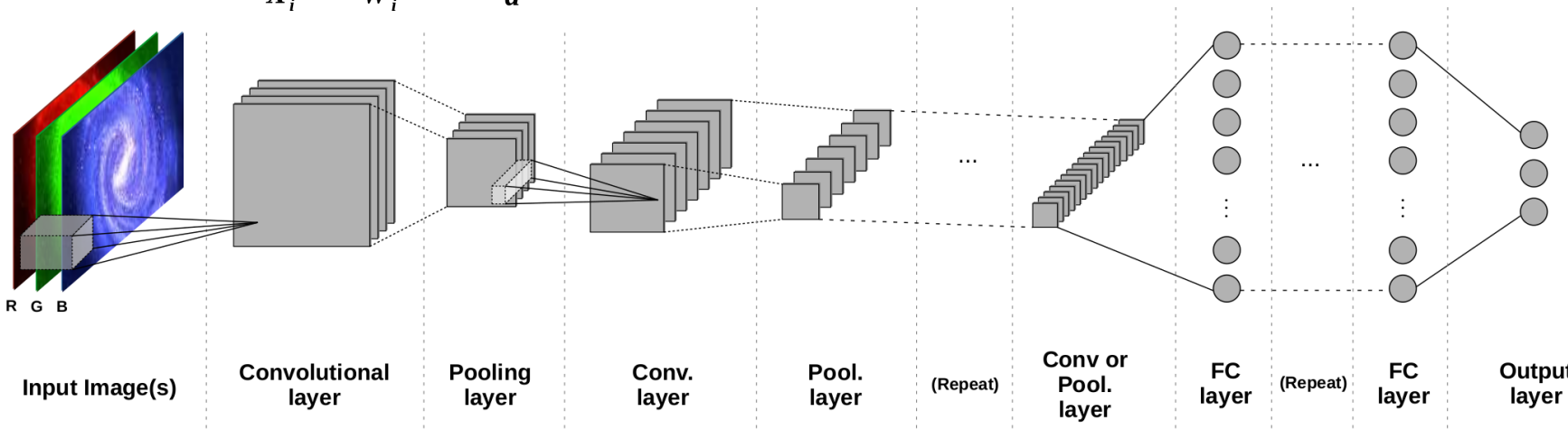


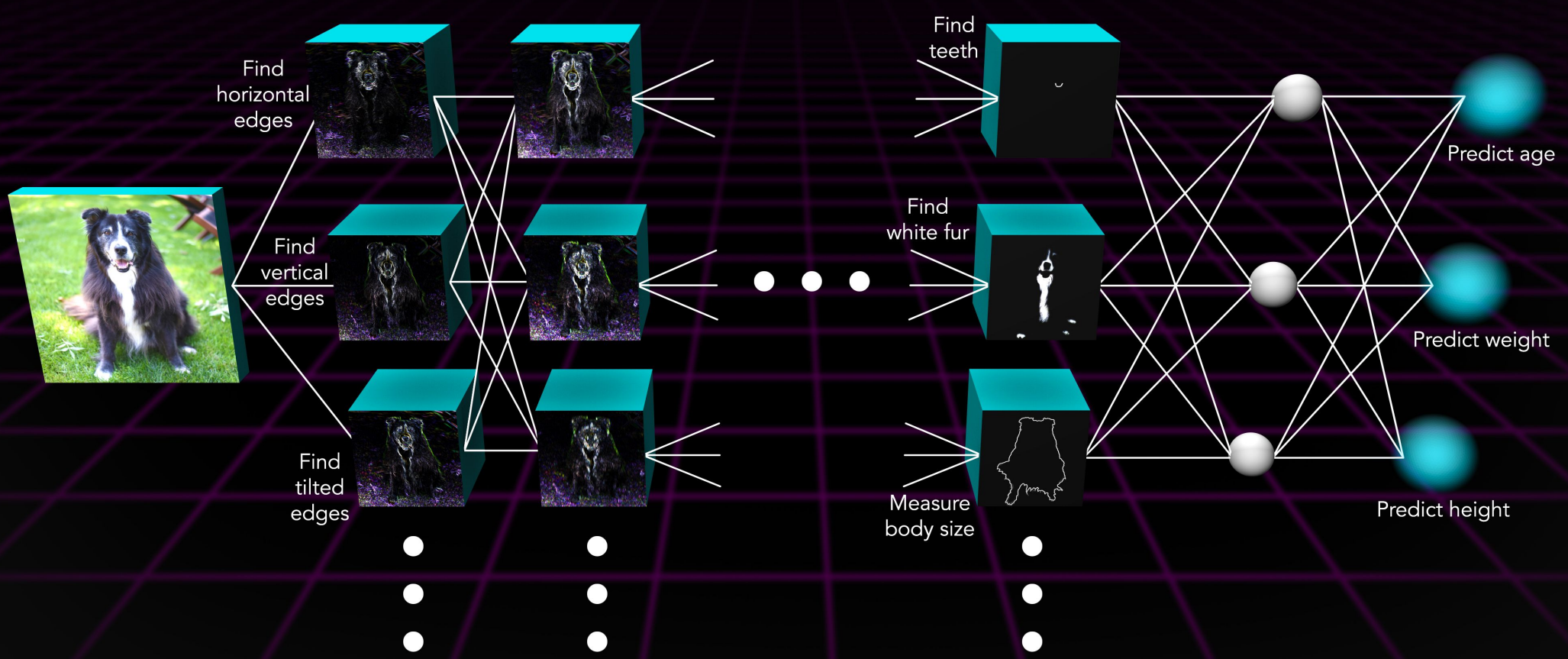
Each filter can be seen as a **single neuron** with one weight per input dimension in the filter.

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

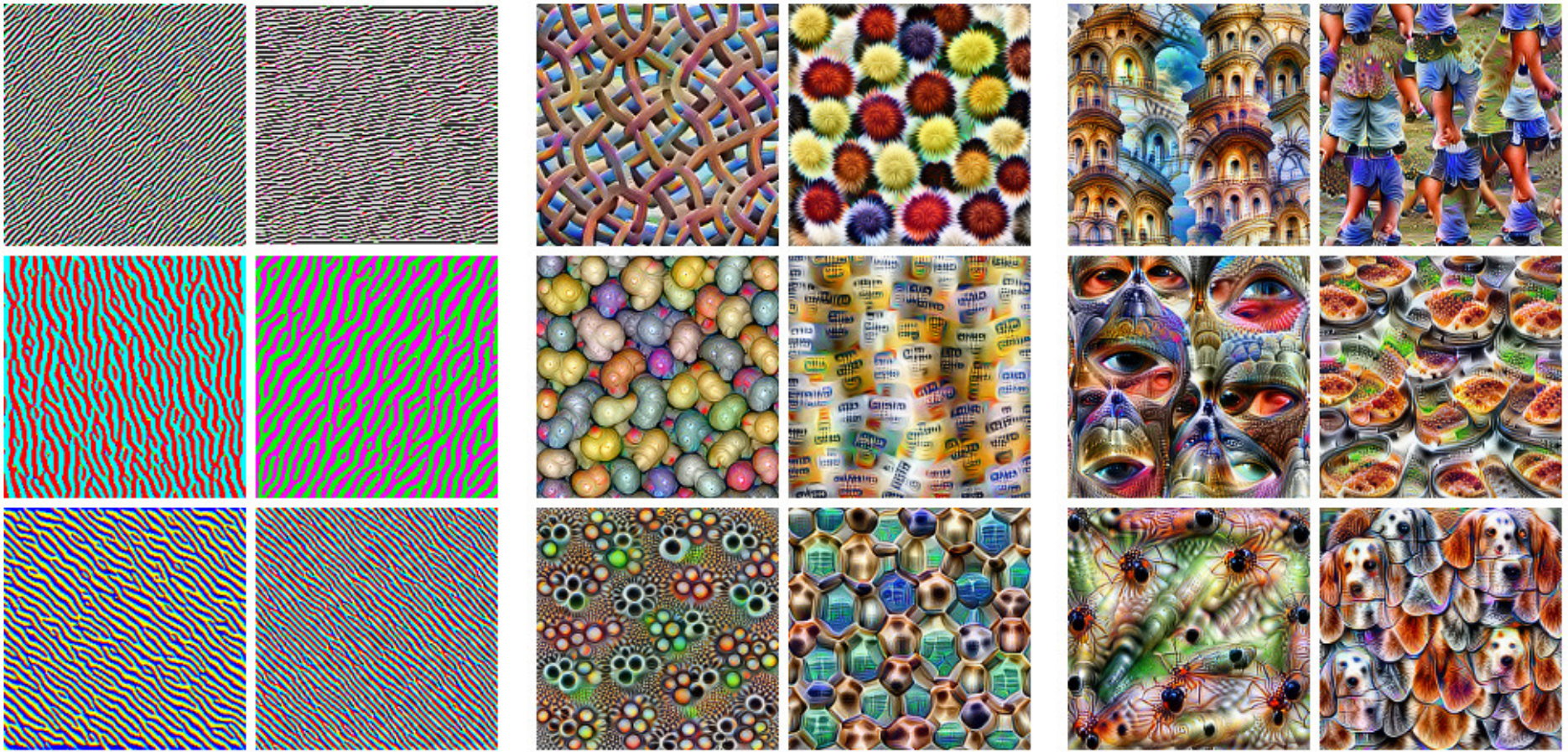
➡ **Translational equivariance !**

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



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.

The Rectified Linear Unit (ReLU)

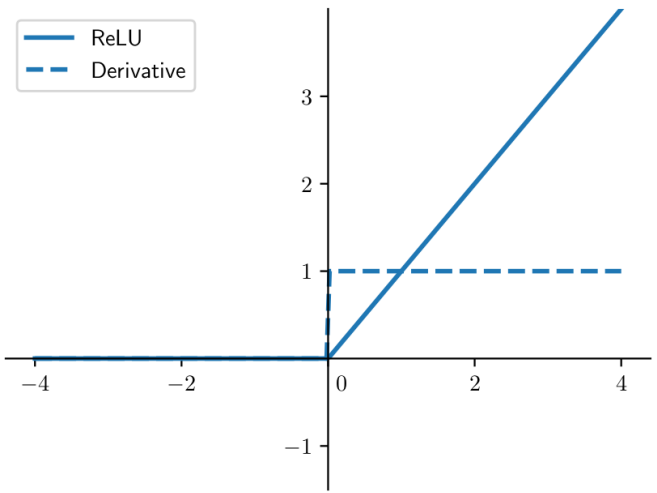
The ReLU (or its variance, the leaky-ReLU) has proven more efficient for CNN. It preserves a form of non-linearity and its constant derivative reduces **vanishing gradient problems**.

It is scale-invariant, and much faster to compute than other activation functions.

Using this activation, it becomes possible to construct much deeper networks.

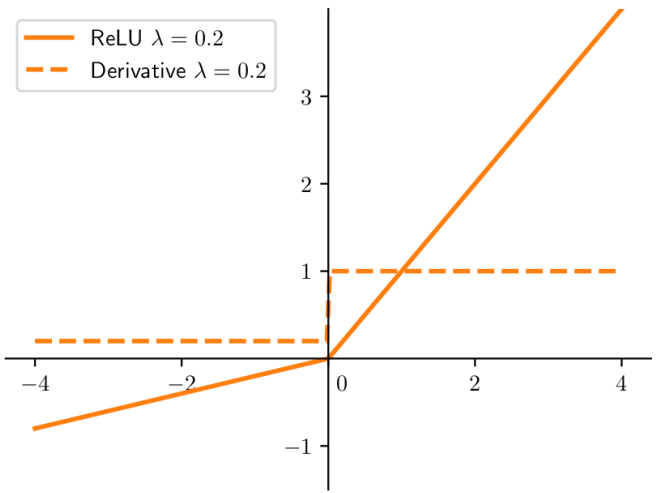
$$a_j = g(h_j) = \begin{cases} h_j & \text{if} \quad h_j \geq 0 \\ 0 & \text{if} \quad h_j < 0 \end{cases}$$

or $$a_j = g(h_j) = \max(0, h_j)$$



$$a_j = g(h_j) = \begin{cases} h_j & \text{if} \quad h_j \geq 0 \\ \lambda h_j & \text{if} \quad h_j < 0 \end{cases}$$

or $$a_j = g(h_j) = \max(0, h_j) + \min(0, \lambda h_j)$$

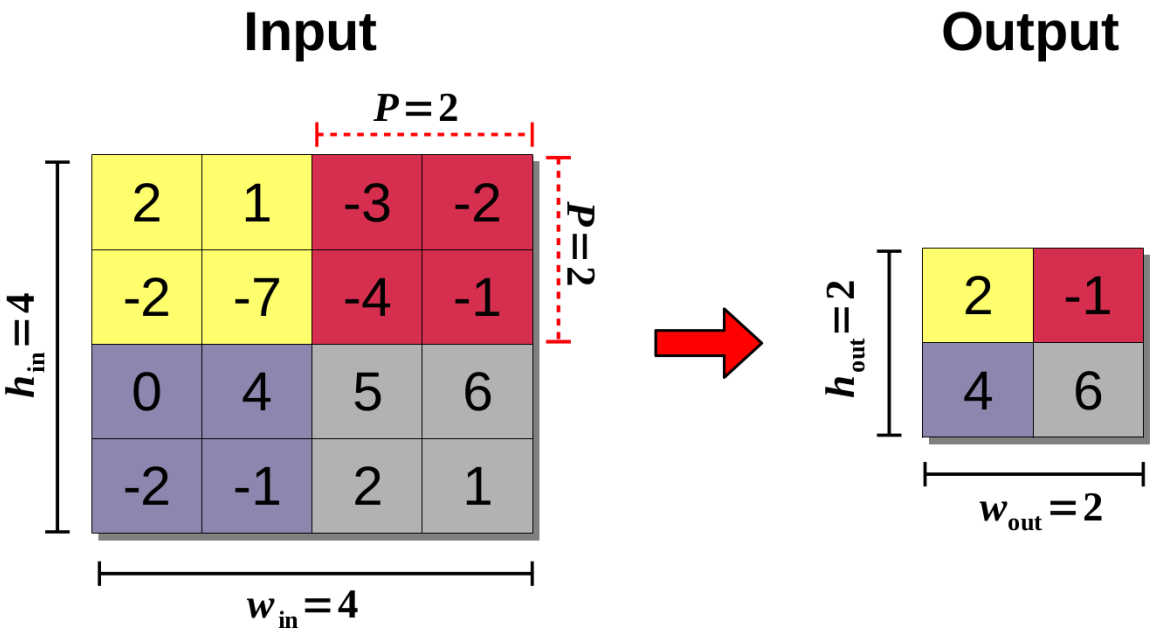


Dimensionality reduction: Pooling

A classical convolution operation is tuned to preserve the spatial dimensionality.
Still, it is most of the time necessary to **reduce the “image” size progressively**.

For classification tasks, the output layer is often reduced to a dense layer with a few neurons.

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

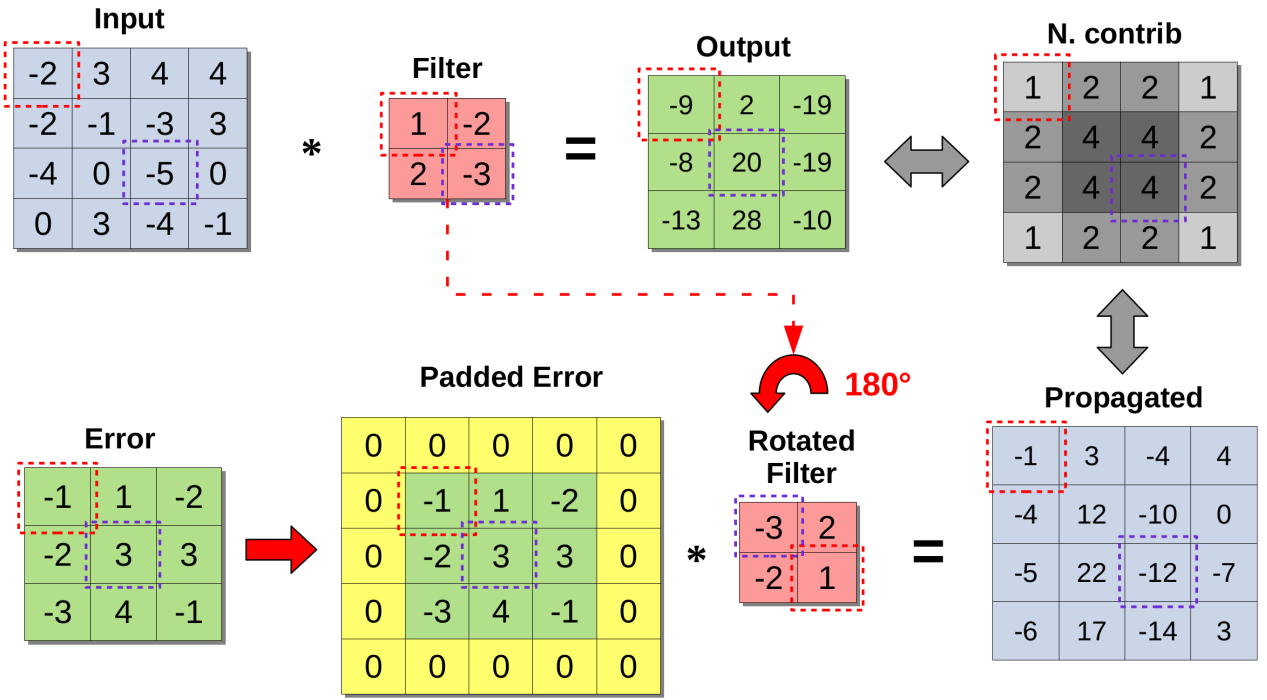


Different pooling methods exist, the most common being Max-Pooling and Average-Pooling. The pooling size can be modified, but most of the network architectures reduce each spatial dimension by a factor of two.

Learning the filters

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

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

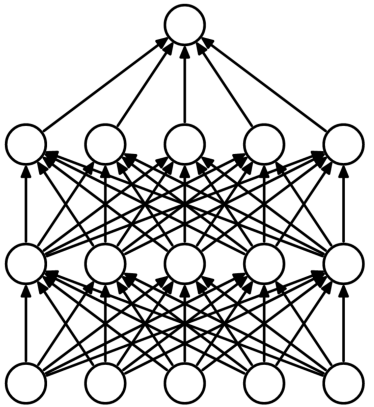


*Learning the convolutional filters is the boundary at which most researchers define "**Deep Learning**".*

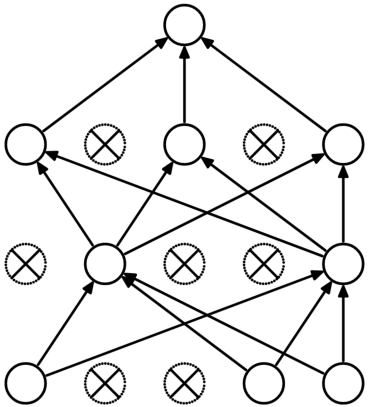
Additional regularization: dropout

One common regularization techniques is called **dropout**, which consists in randomly deactivating some neurons for each batch during the training phase.

This approach usually allow better results through different effects on the network's inner working (more salient features, increased sparsity, etc.).



(a) Standard Neural Net



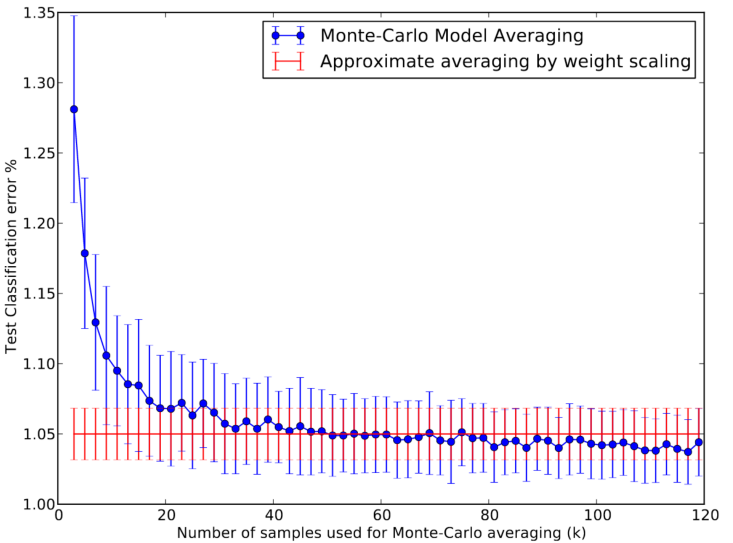
(b) After applying dropout.

Average dropout

At inference time it, is possible to **average all the models** represented by the dropout selection by keeping all neurons activated but with a scaling of the weights proportional to the dropout rate at each layer.

Monte-Carlo dropout

Another approach is to keep dropping neurons but performing several predictions for each input to test.
This is a way to reconstruct a **posterior probability distribution** for each output dimension.
This approach allows to measure the network **prediction uncertainty**.



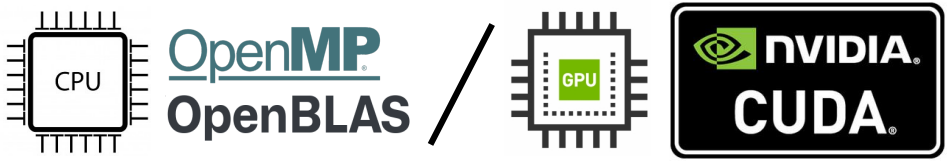
CIANNA

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



x86 Intel/AMD & ARM / **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:

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

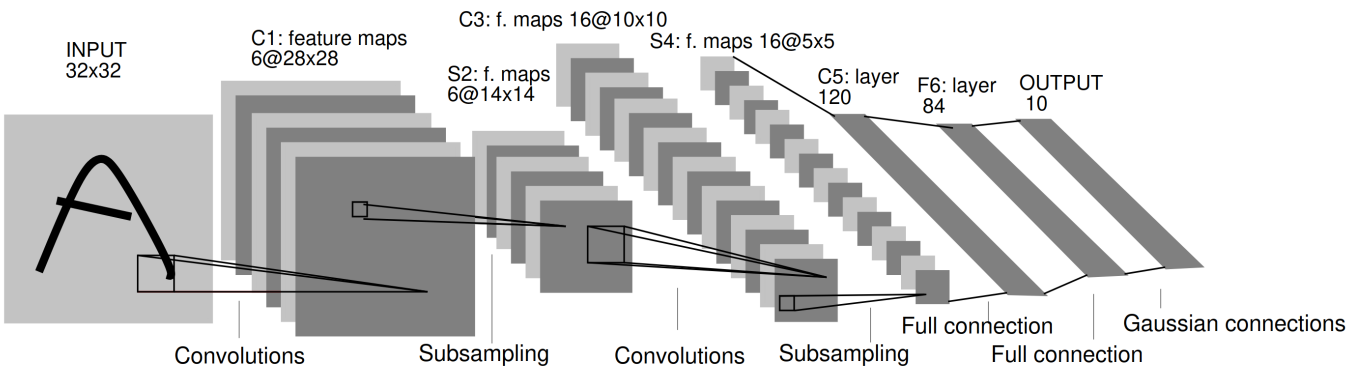
Simple examples: MNIST

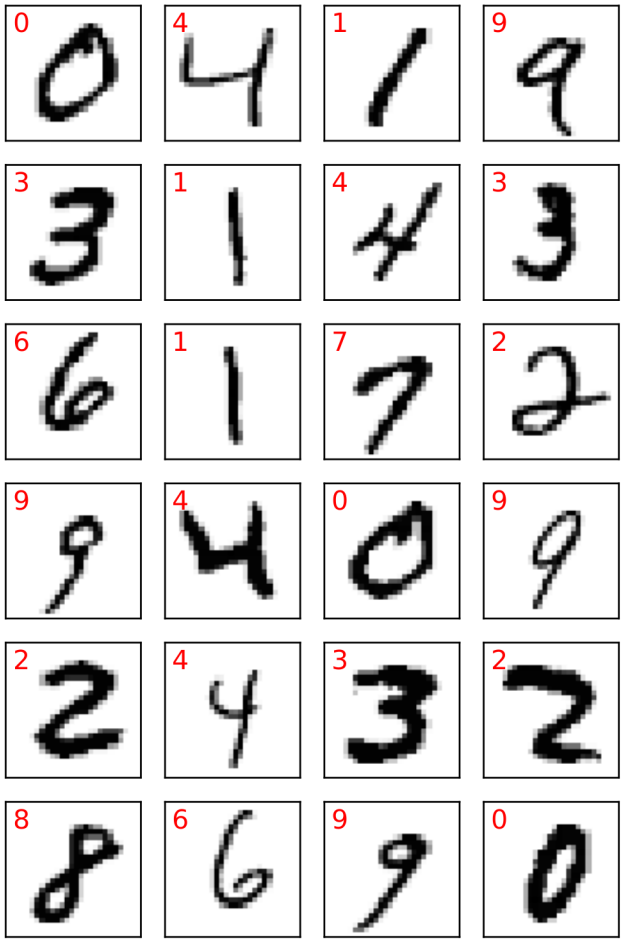
The well-known **MNIST** (Modified NIST's special dataset) dataset consists of handwritten digits from 500 different writers expressed as 28x28 grayscale images.

It is freely accessible in the form of a 60000 image training set, 10000 images validation, set and a 10000 image test set.

Very simple CNN architectures can achieve over **99.3 classification accuracy** on this dataset.

LeNet 5 network architecture (from LeCun et al. 1998):





Example results on MNIST

Actual \ Predicted	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	Recall
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 $\eta=2x10^{-4}$ with a small decay up to $\eta=1x10^{-4}$, and a momentum of $\alpha=0.9$.

The network is trained for 40 epochs.

More advanced image classification

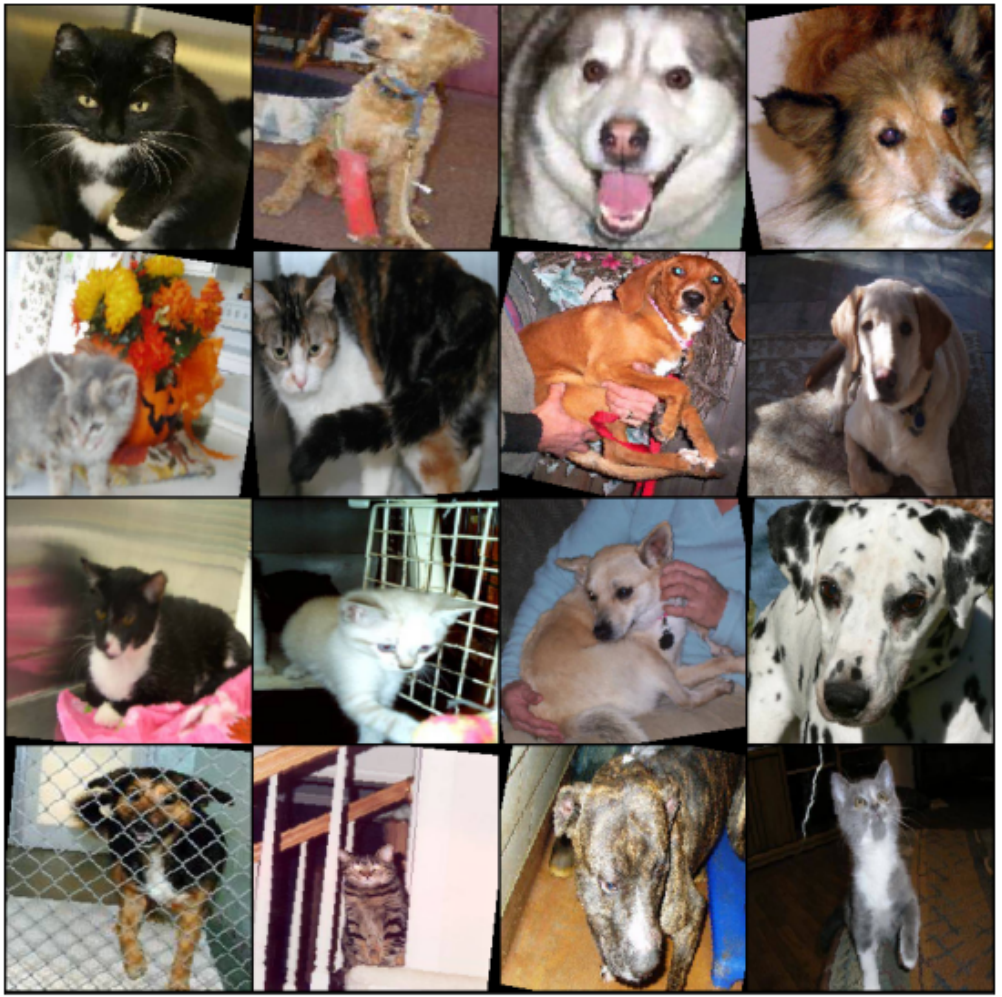
ASIRRA (Animal Species Image Recognition for Restricting Access): 25000 images, 50 % cats, 50 % dogs.

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

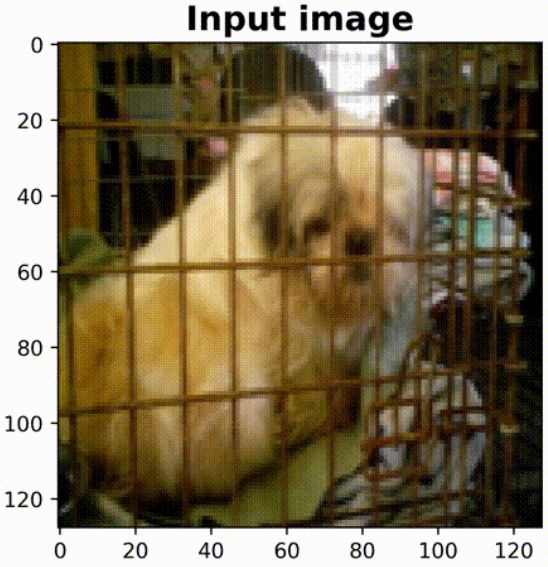
➡ Nowadays modern CNN tools have more than 90% accuracy on ASIRRA !

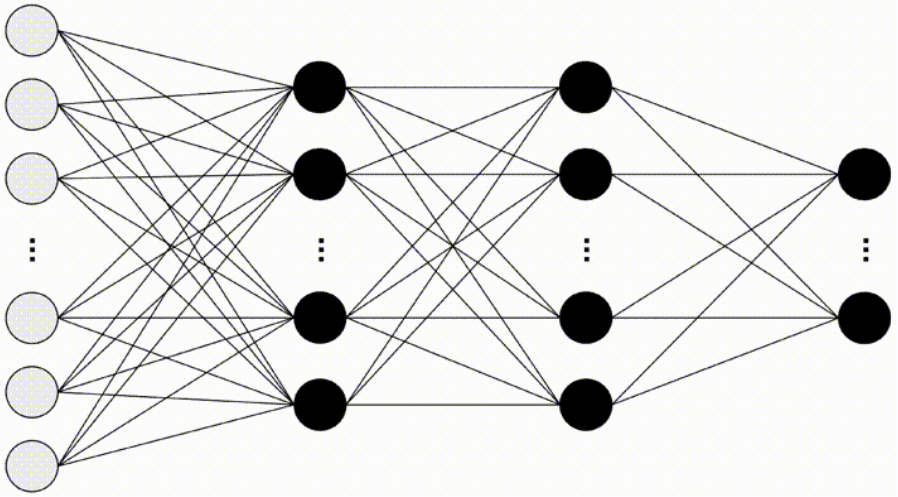
The dataset is made of .jpg images that vary in size and ratio. In order to homogenize we pad and rescale all images in a 128x128 pixels format.

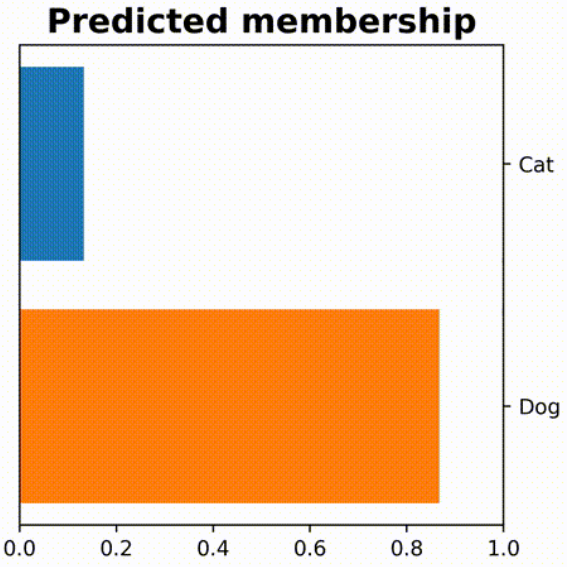
Using the provided dataset (augmented by mirroring) and data loading script, implement a CNN (with either framework) that performs this classification. Explore the architecture and parameters by yourself and try to optimize the prediction result.



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 with Deep Learning

Types of object detection in images

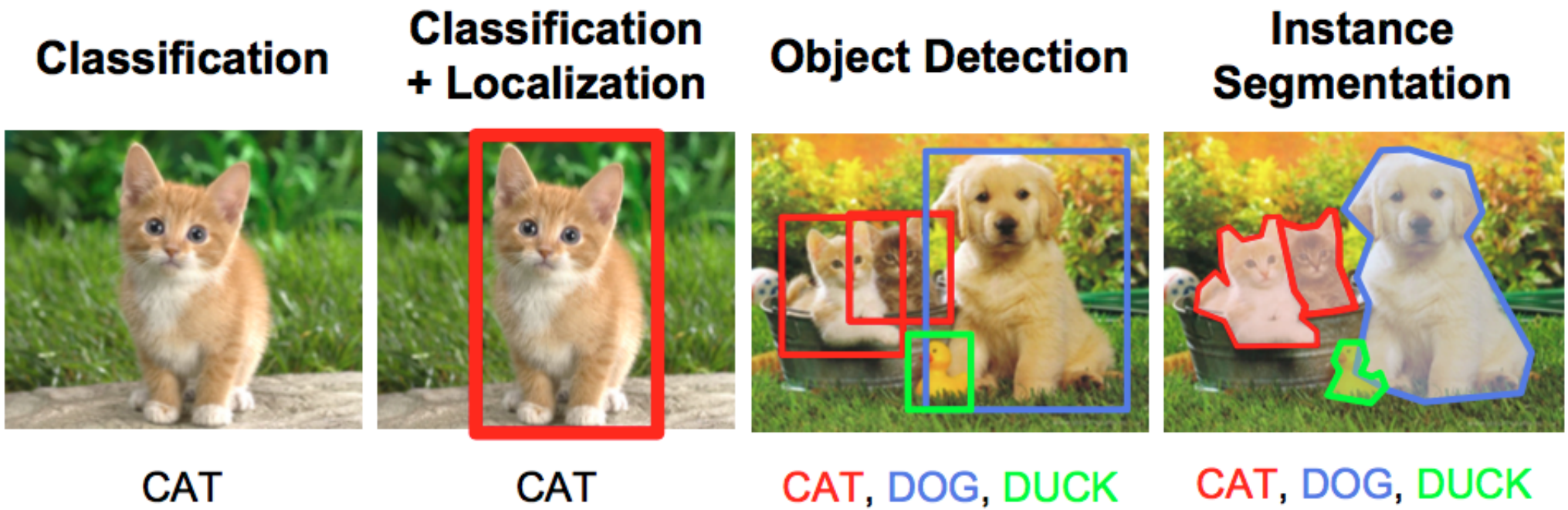


*Image from Stanford Deep Learning course cs224

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” :

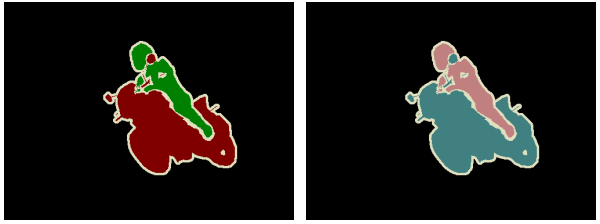
Classification

[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)

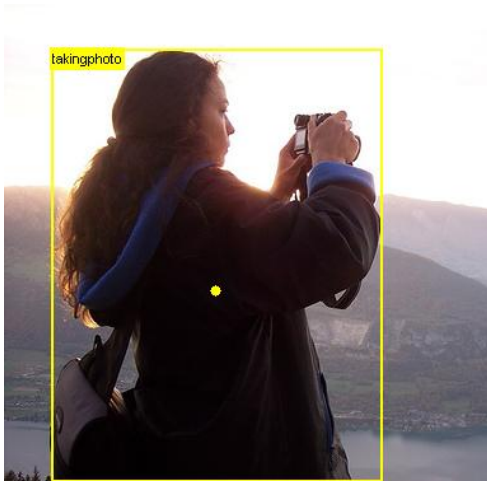
Standardized image datasets for object class recognition organized in the form of annual challenges from 2005 to 2012

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

Targets for various “tasks” :

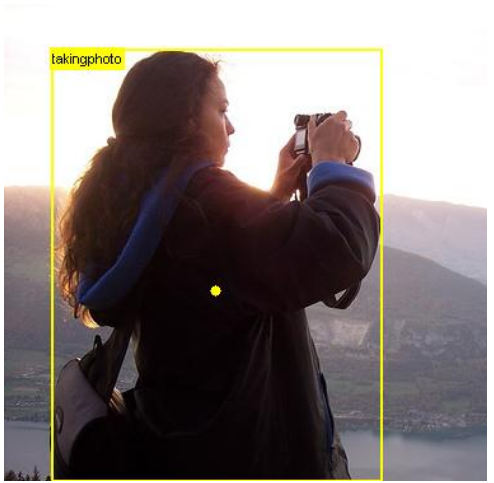
http://host.robots.ox.ac.uk/pascal/VOC/index.html

Boxless Action



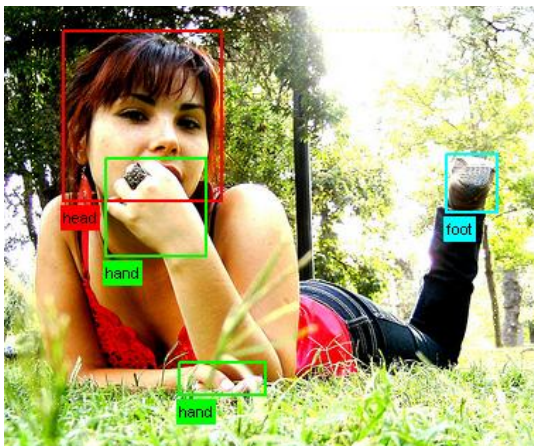
A single reference point and the associated action

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

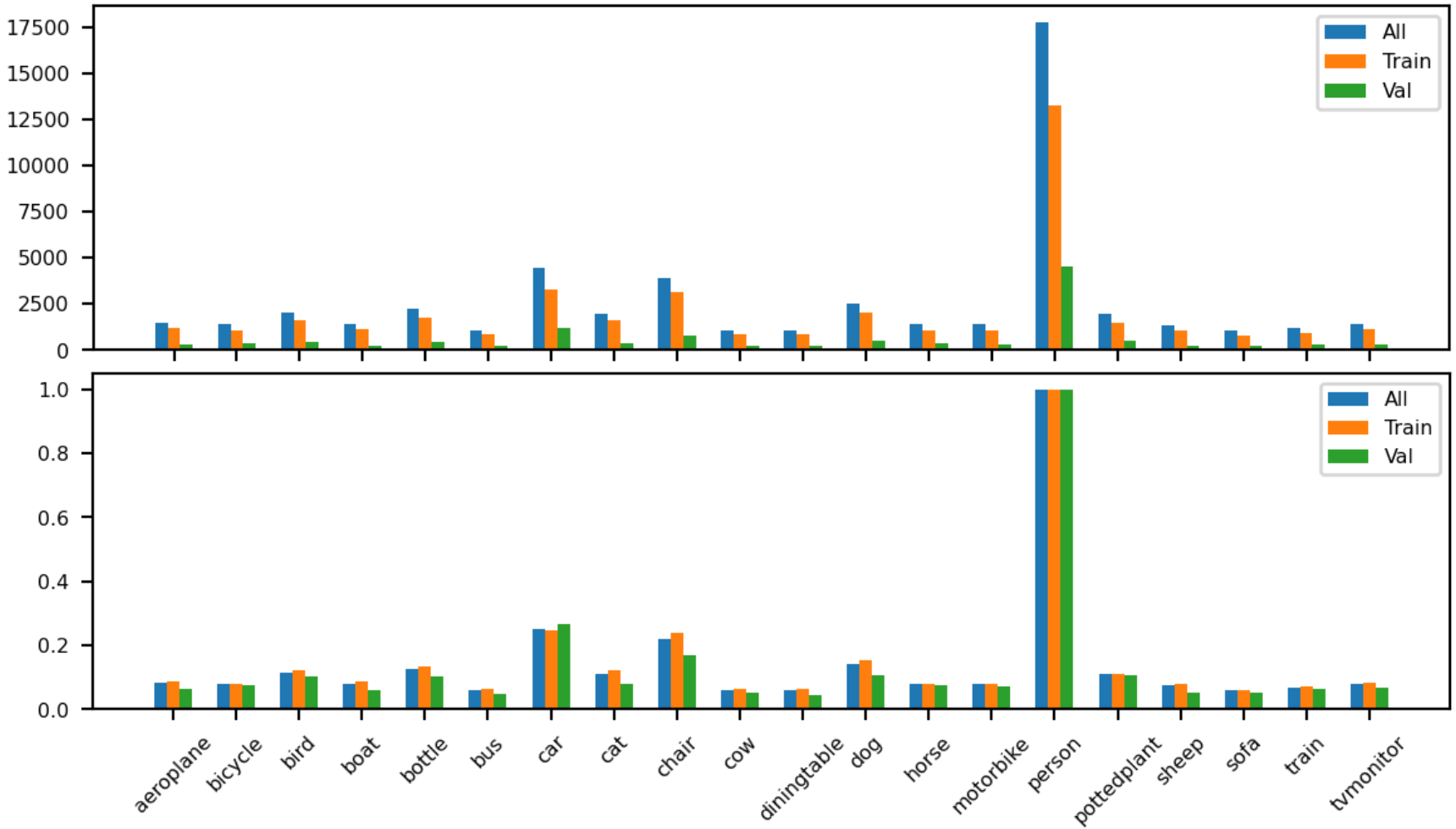
VOC - 2012 : incremental from 2008 to 2012
11540 Train annotated images (test set hidden), detection task with 20 classes

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

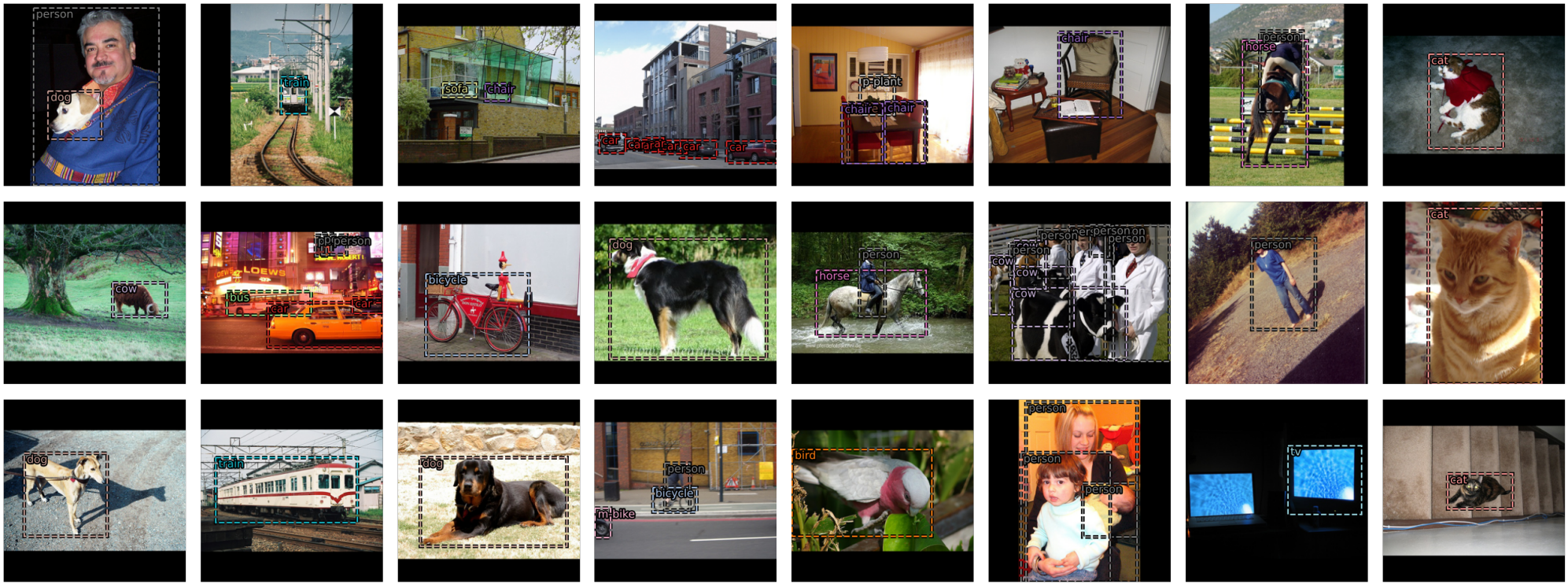
Training dataset summary statistics

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=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=1024, padding=i_ar([1,1]), activation="RELU")
cnn.conv(f_size=i_ar([3,3]), nb_filters=1024, padding=i_ar([1,1]), activation="RELU")
cnn.conv(f_size=i_ar([1,1]), nb_filters=nb_class, padding=i_ar([0,0]), activation="LIN")
cnn.pool(p_size=i_ar([1,1]), p_global=1, p_type="AVG", activation="SMAX")
```

Confusion matrix on the validation set for the pre-trained network at epoch 300 :

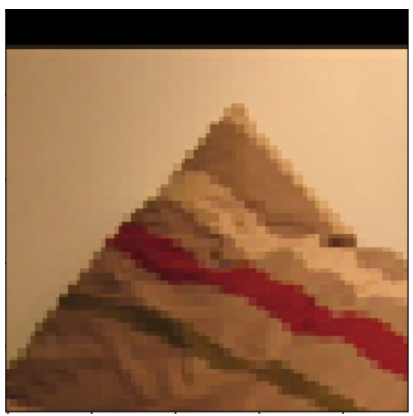
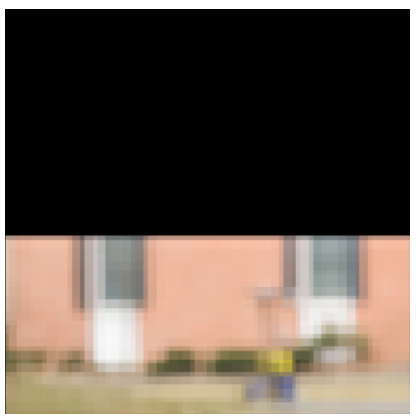
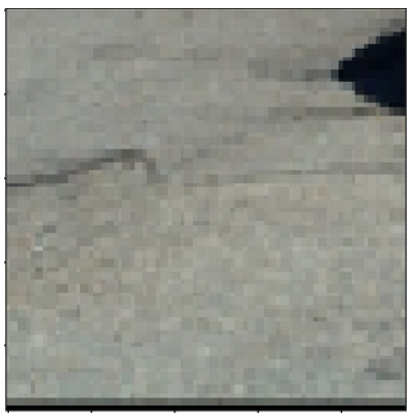
**********										ConfMat	**********										Recall
93	0	3	3	0	0	1	0	0	0	1	0	0	0	3	0	0	0	3	0		86.92%
0	133	0	1	1	1	0	0	4	0	0	0	0	8	8	0	0	0	0	0		85.26%
3	1	154	1	3	0	4	6	1	5	0	3	5	2	12	3	2	0	1	0		74.76%
4	2	2	92	0	0	2	0	0	0	1	0	2	1	1	1	0	0	4	0		82.14%
0	0	0	1	99	0	1	0	4	0	0	4	0	2	22	3	0	1	0	1		71.74%
2	1	0	4	0	77	8	0	0	0	0	0	0	0	0	0	0	0	7	0		77.78%
8	2	0	4	1	9	404	0	4	0	1	1	0	4	4	1	0	2	2	1		90.18%
0	0	5	0	1	0	1	117	4	0	0	19	2	0	13	1	4	2	1	0		68.82%
0	3	2	1	0	0	1	0	261	0	14	0	1	1	30	3	1	19	6	2		75.65%
0	0	1	0	0	0	0	1	2	49	0	9	11	0	2	0	8	0	0	0		59.04%
0	1	0	4	1	0	3	0	4	0	77	0	1	0	4	1	0	4	3	0		74.76%
0	1	4	0	0	0	0	23	1	9	0	159	16	0	24	1	3	3	0	0		65.16%
1	0	3	0	0	0	0	0	1	3	1	5	132	0	14	0	3	0	0	0		80.98%
1	4	0	0	1	0	5	0	0	0	1	1	0	114	14	0	0	0	0	0		80.85%
0	13	10	6	12	2	6	4	24	1	5	14	13	13	1685	0	0	11	4	1		92.38%
0	1	5	0	2	0	3	1	7	0	2	0	0	0	3	158	0	0	2	0		85.87%
0	0	2	0	0	0	0	3	1	12	0	14	2	0	1	0	73	0	0	0		67.59%
1	1	0	1	0	0	3	2	8	1	3	2	1	0	17	0	0	72	3	0		62.61%
1	2	1	3	0	1	0	0	0	0	0	0	0	0	2	0	0	0	125	1		91.91%
0	0	0	1	1	1	0	0	6	0	0	1	1	0	4	2	0	0	2	99		83.90%
Prec.	81.58%	80.61%	80.21%	75.41%	81.15%	84.62%	91.40%	74.52%	78.61%	61.25%	72.64%	68.53%	70.59%	78.62%	90.45%	90.80%	77.66%	63.16%	76.69%	94.29%	Acc 83.46%

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

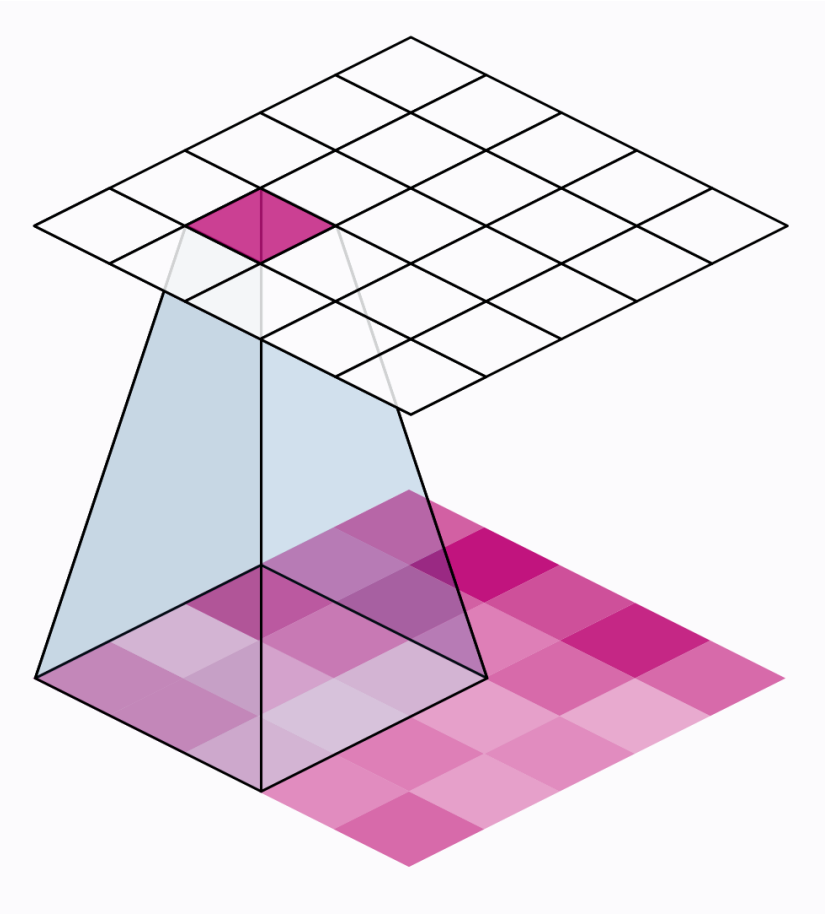
Can be used as a sliding window detector:

→ Split a larger image in “chunks”

→ Overlapping Chunks for more « centered » objects

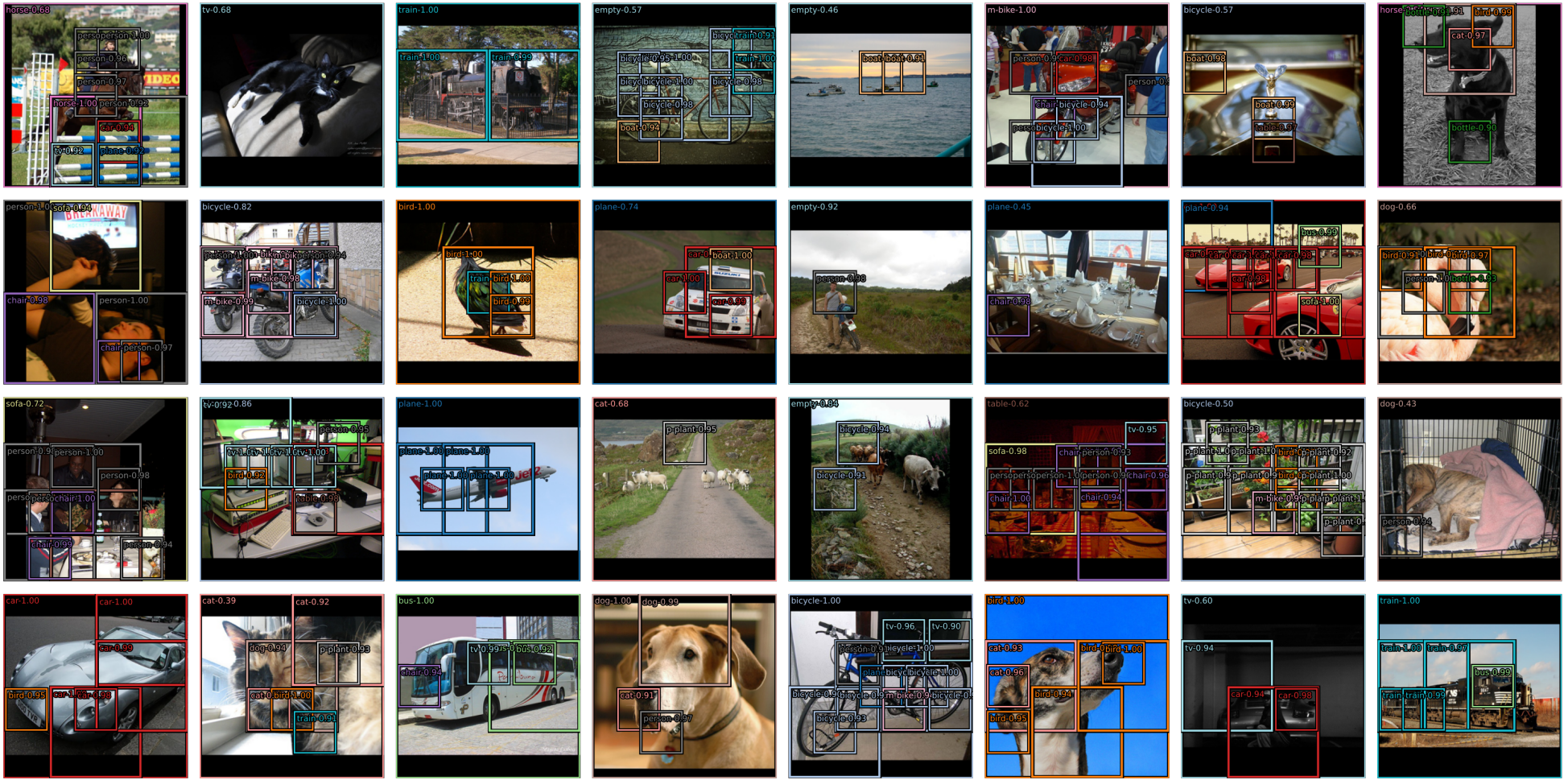
Process similar to a convolution,
with Classifier ≈ Filter

→ 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

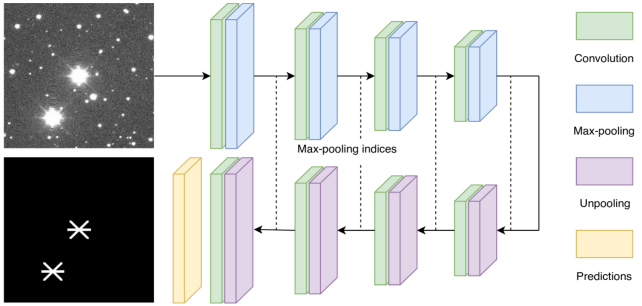


Questions : How to filter overlapping detection ? How to evaluate prediction performance ?

CNN architectures for object detection

U-Nets

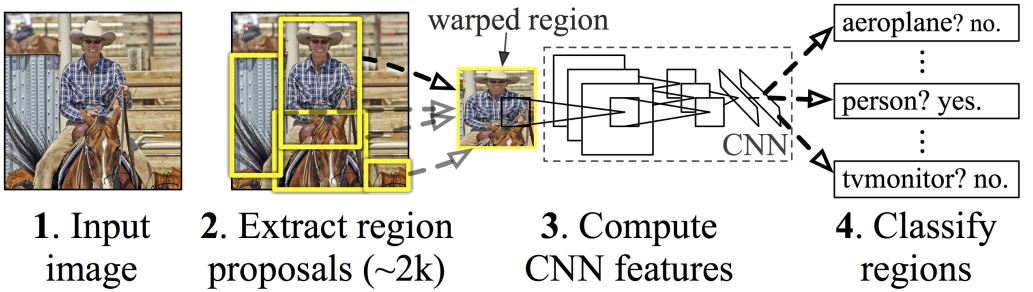
Pros: segmentation maps, shallow latent space, ...



Region-based

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

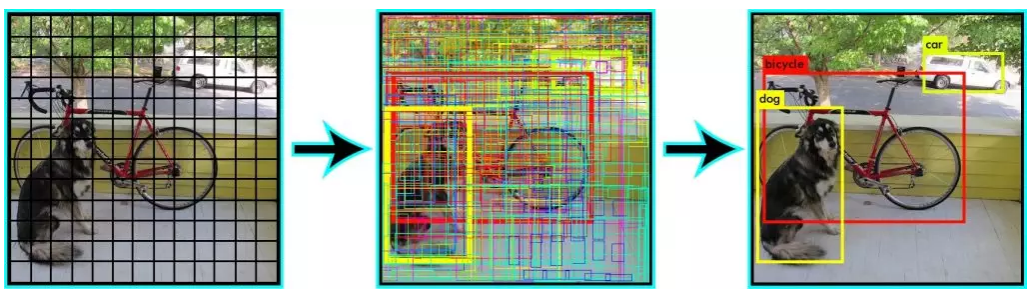
Pros: Best accuracy, ...



Regression-based

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

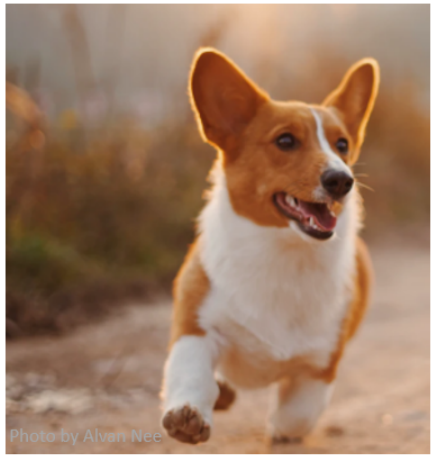
Pros: Very Fast, straightforward architecture,...



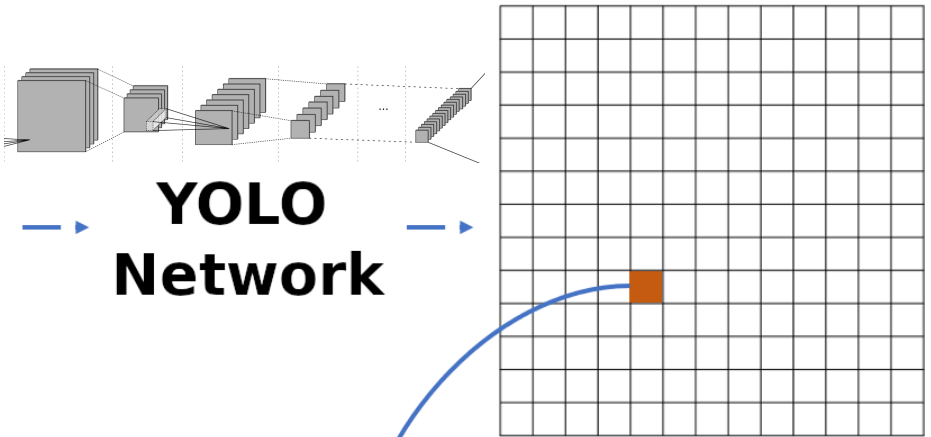
You Only Look Once – YOLO !

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

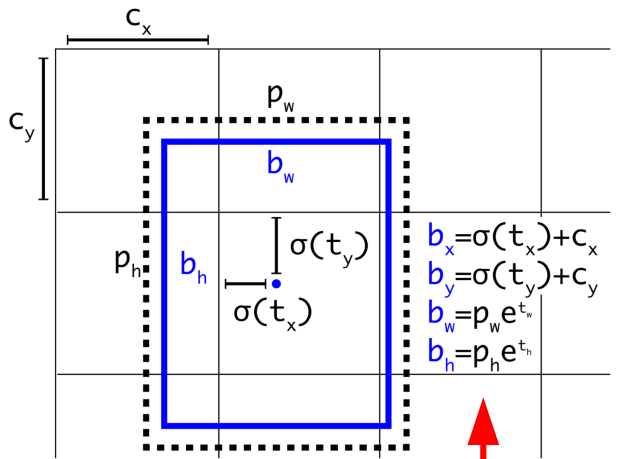
**Images from blog post and Redmon papers*

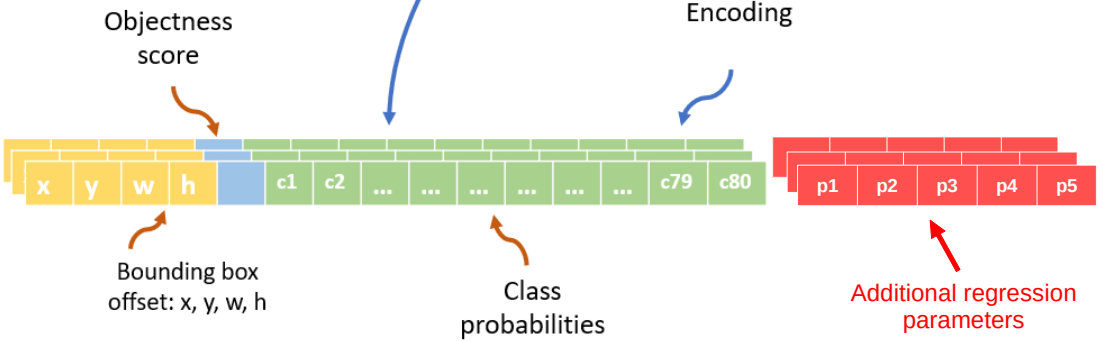


Pre-processing Image

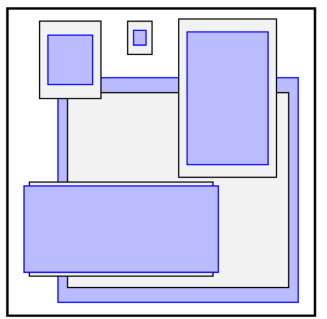


Encoding



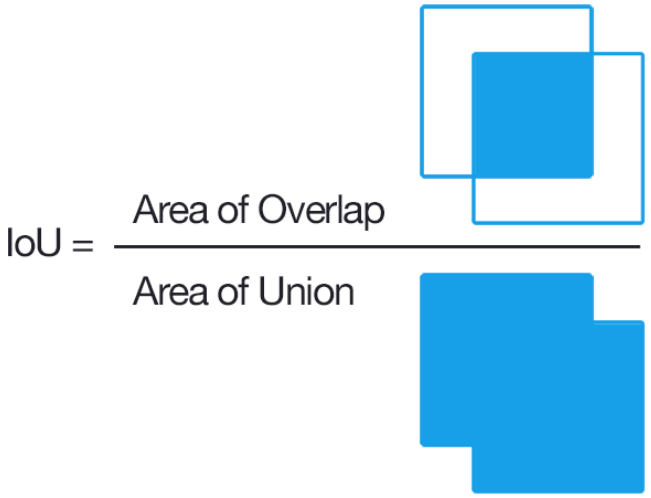


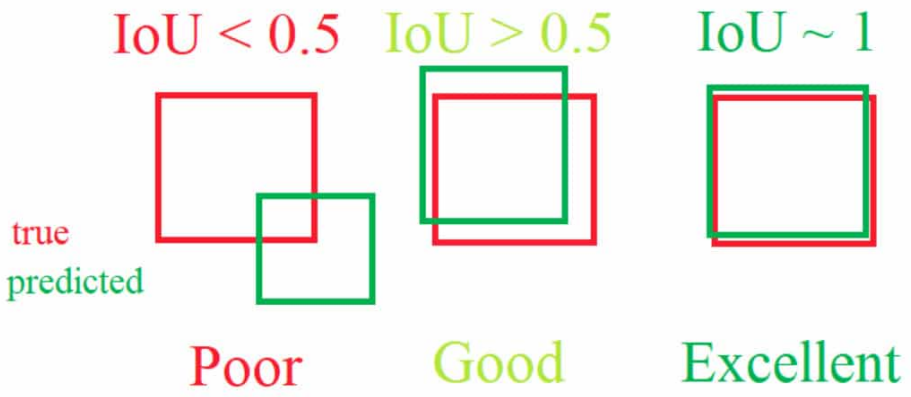
Box size priors



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 » ?

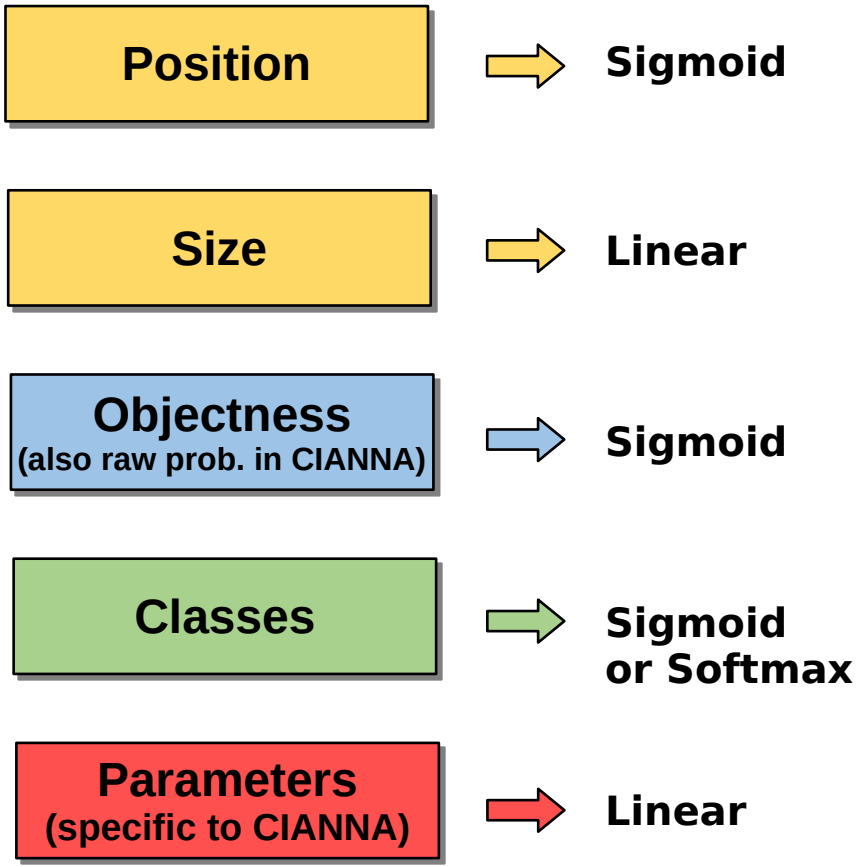




For each box the network will predict an “objectness” score defined as:

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

The YOLO activation / Loss function



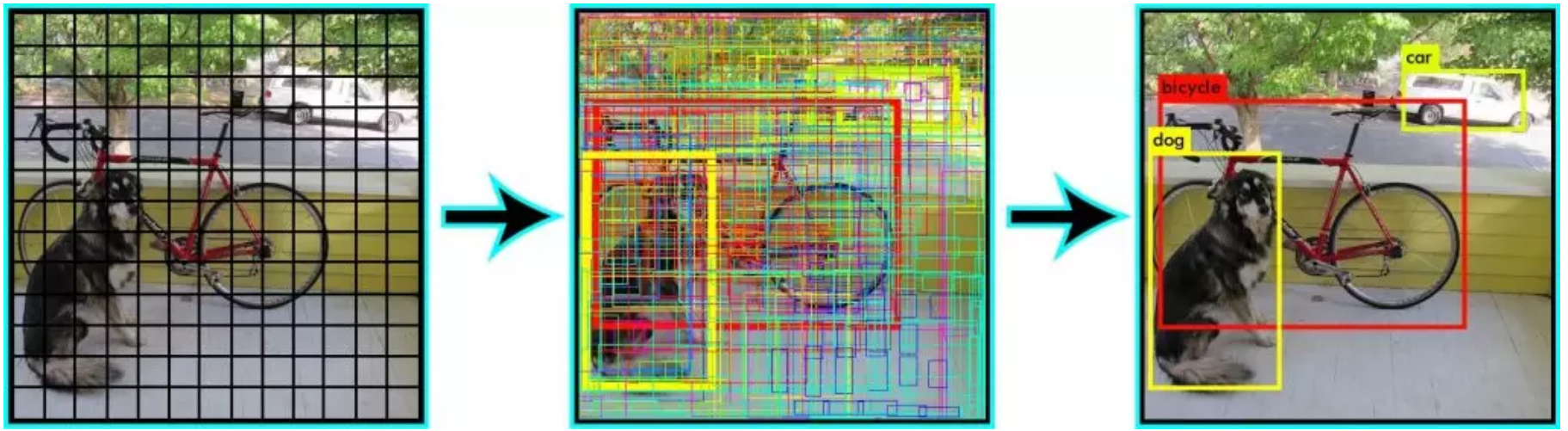
All error terms are using MSE

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

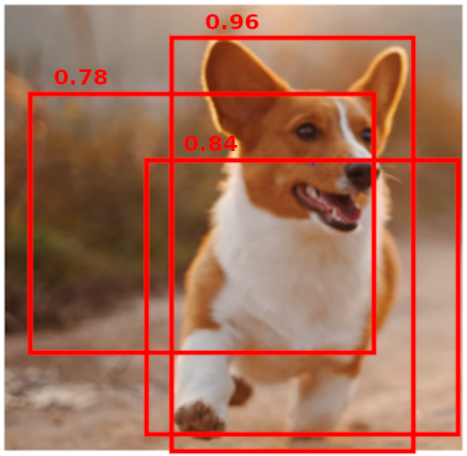
Post-processing: Non Max Suppression

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

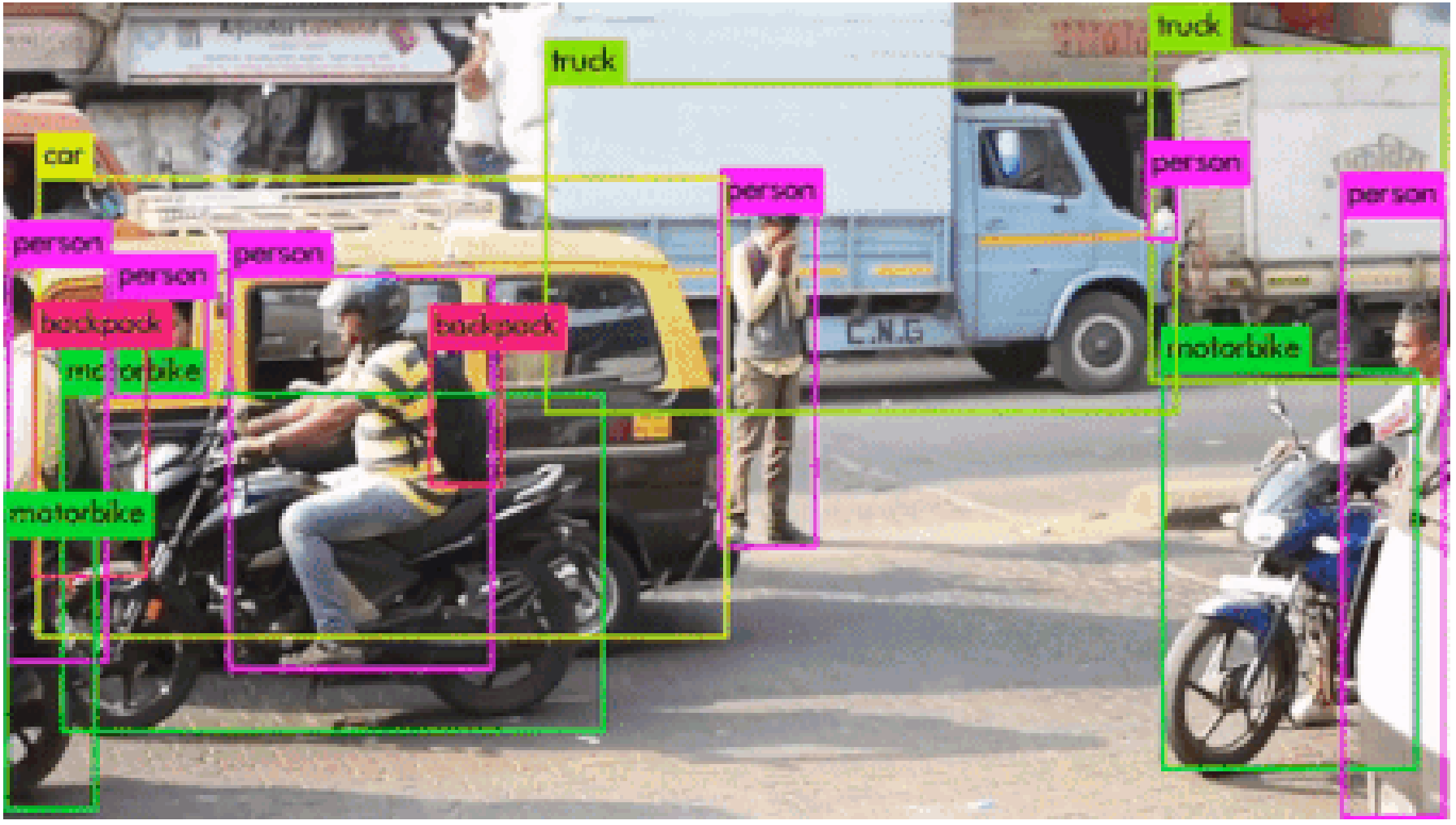


1) Most probable boxes are kept using a threshold in objectness

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



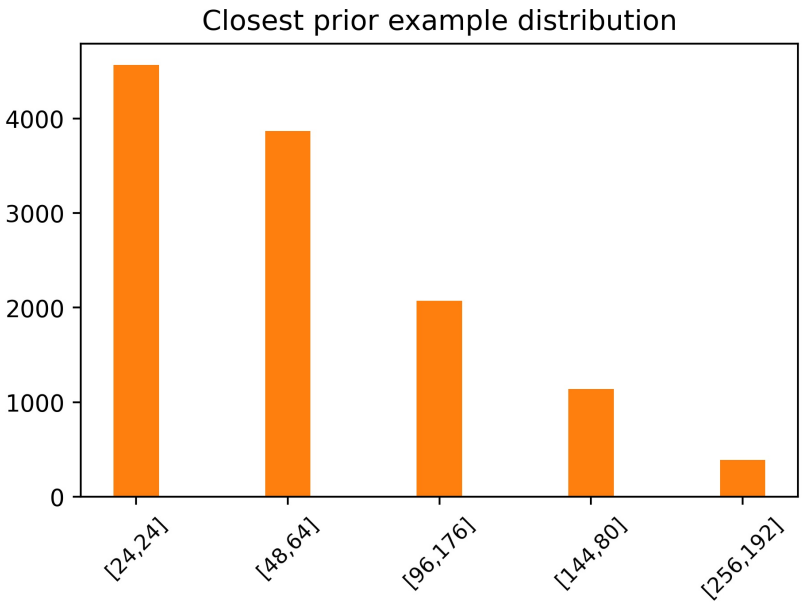
“Real-time” example

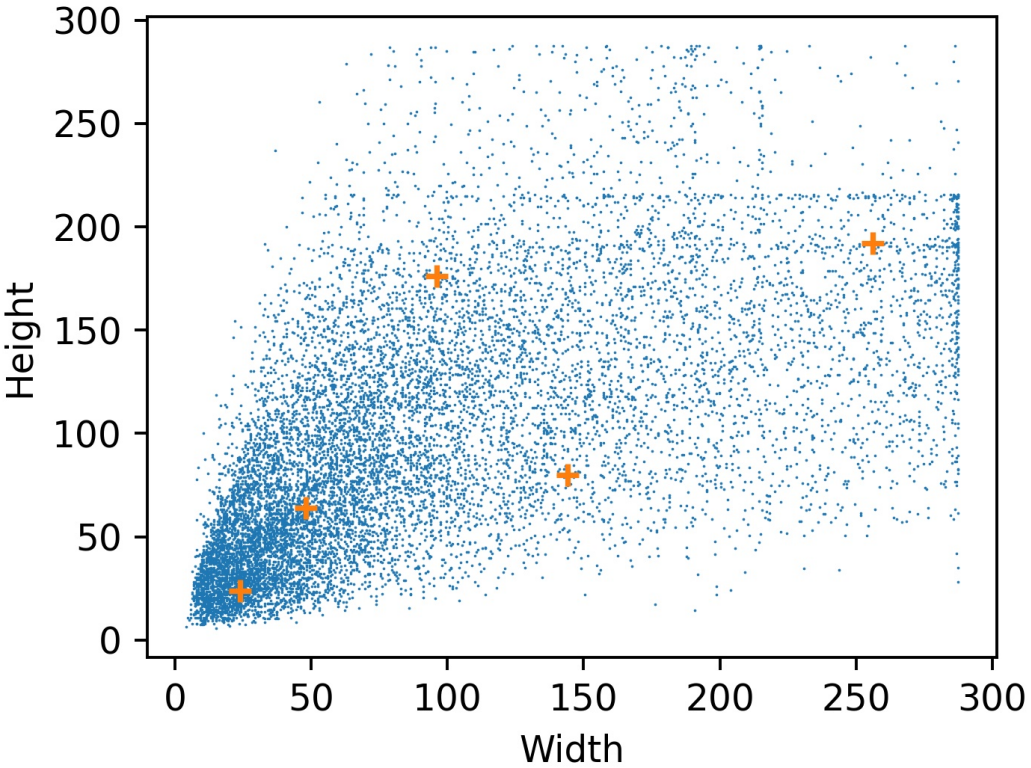


C – Training a YOLO network on PASCAL VOC

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

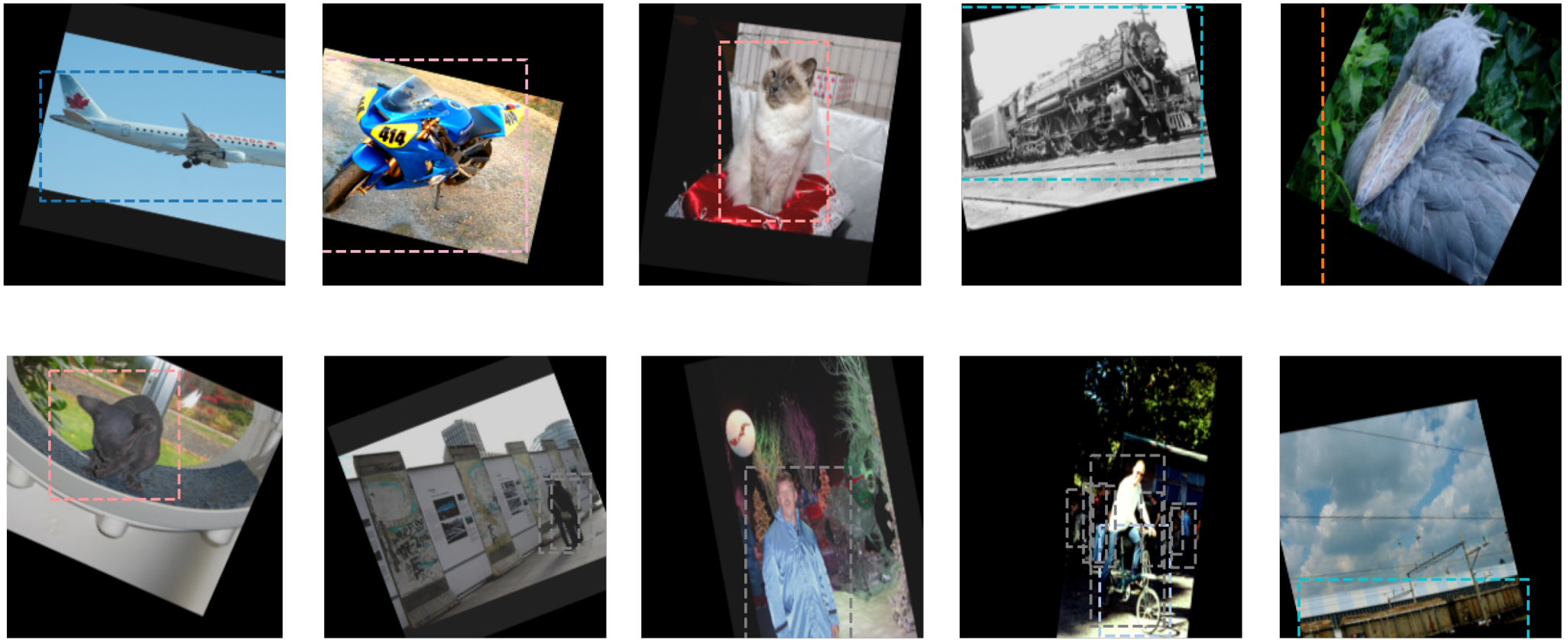
Suggested network architecture *(also the one of the provided pre-trained network)*:

```
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")
```

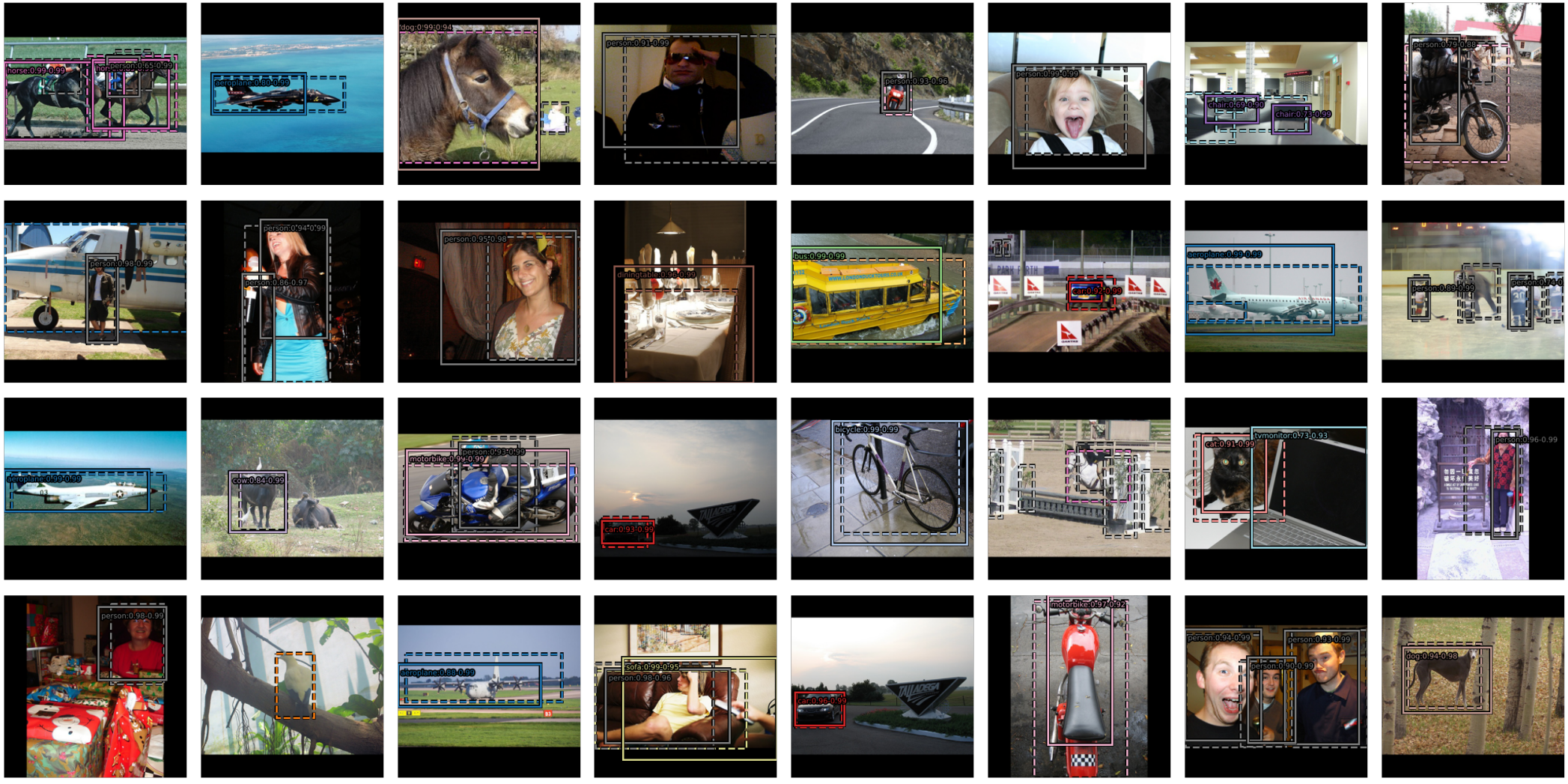
Inspired by the YOLO-Light V2 and Tiny-Yolo V2 architectures

C – Training a YOLO network on PASCAL VOC

Advanced data augmentation using imgaug



C – Training a YOLO network on PASCAL VOC

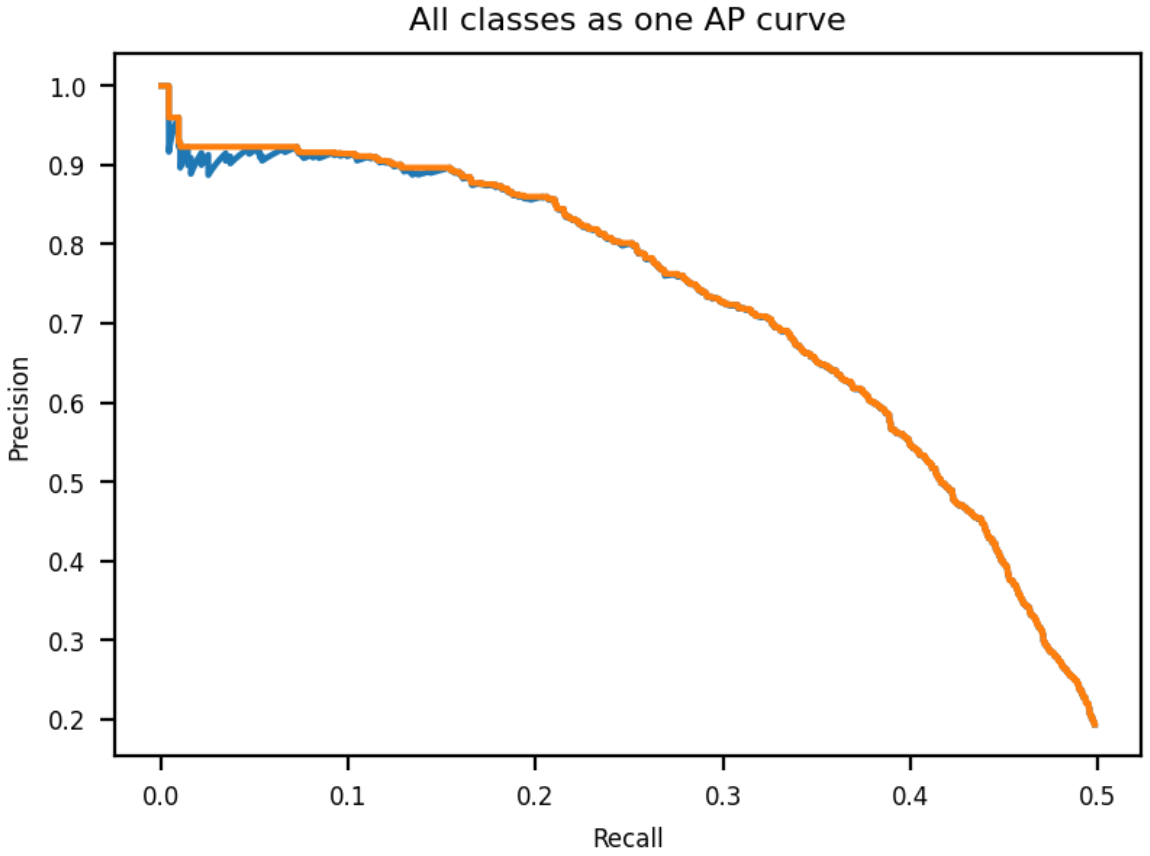


Detection performance metric

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

The precision-recall curve

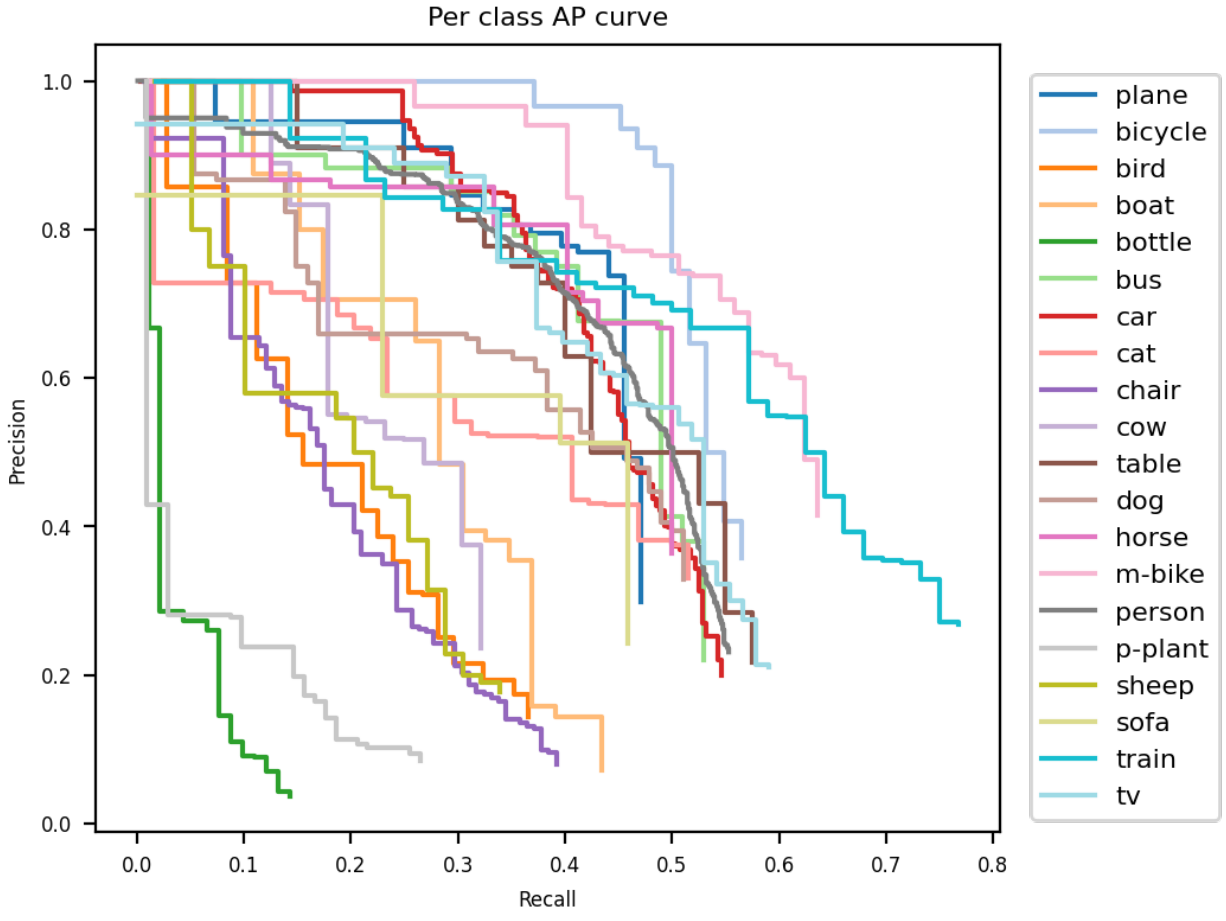
- Predictions are sorted by score
- For each data point, compute the precision and recall using all the predictions that are scored higher, using **IoU@0.5** and correct classification as “True” criteria
- The curve is then smoothed so it can only monotonically decrease (precision is always equal to 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