AI for Astrophysics: Advanced Neural Networks

David Cornu

LERMA, Observatoire de Paris, PSL

Doctoral course – ED AAIF
2024/2025

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.

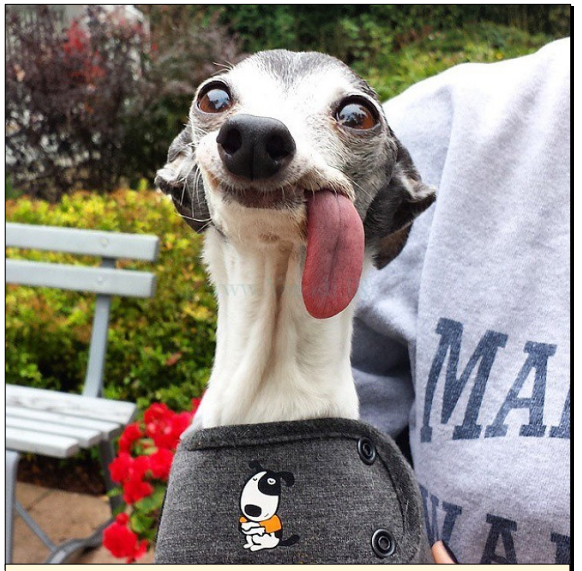
Neural Networks for images

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.



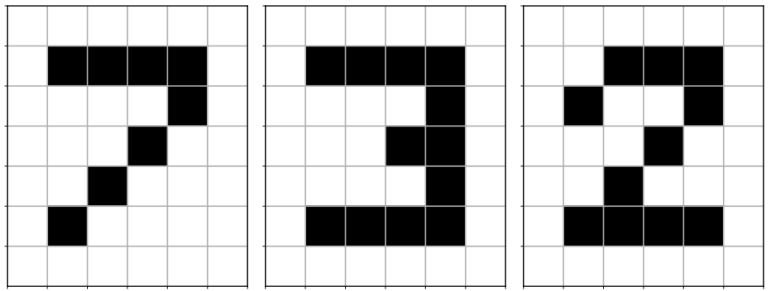
*A **highly dimensional** "dog" with ~0.5 Million pixels.*
Quite difficult to classify ...

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

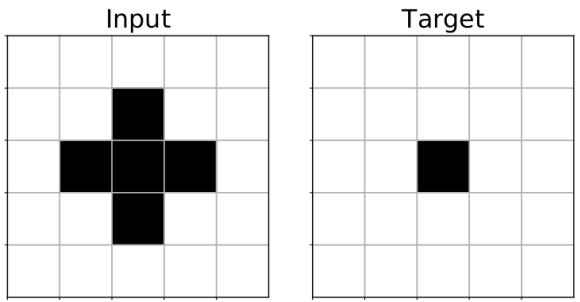
➡ **Convolutional Neural Networks** !

Spatially coherent information

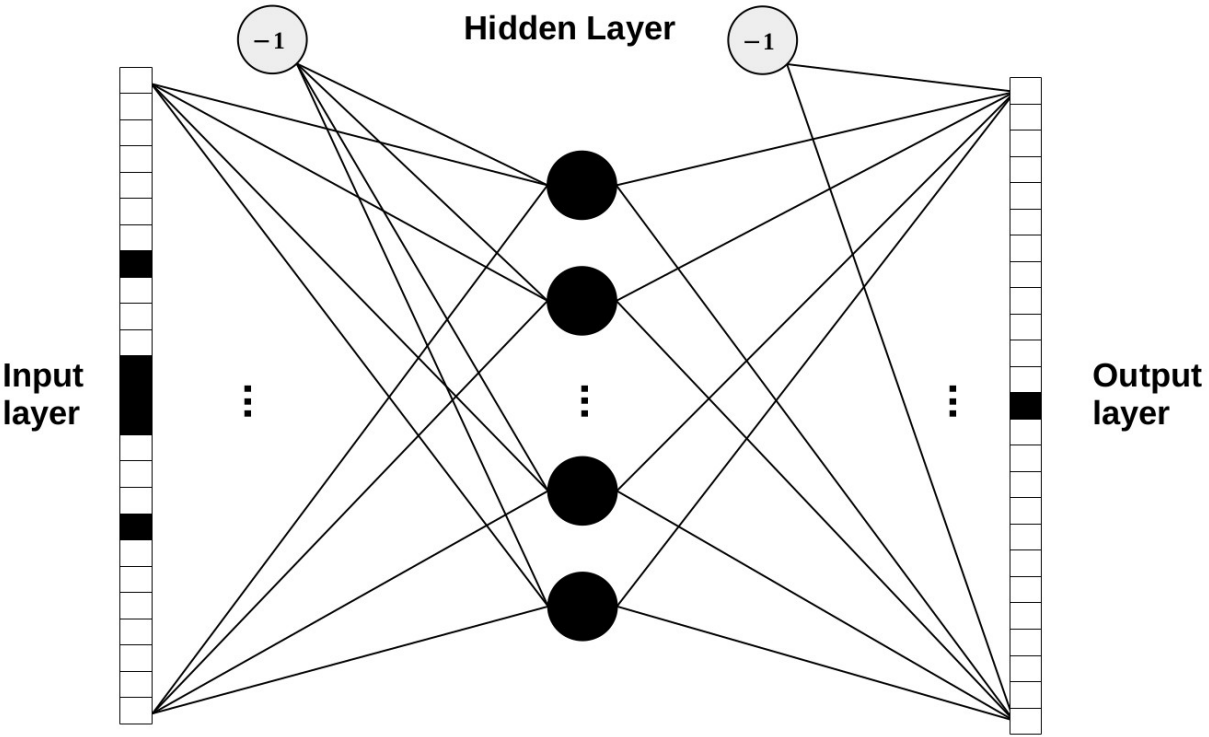
*Classical ANN can deal with images by considering each pixel of an image as an individual input but it is **STRONGLY inefficient**.*



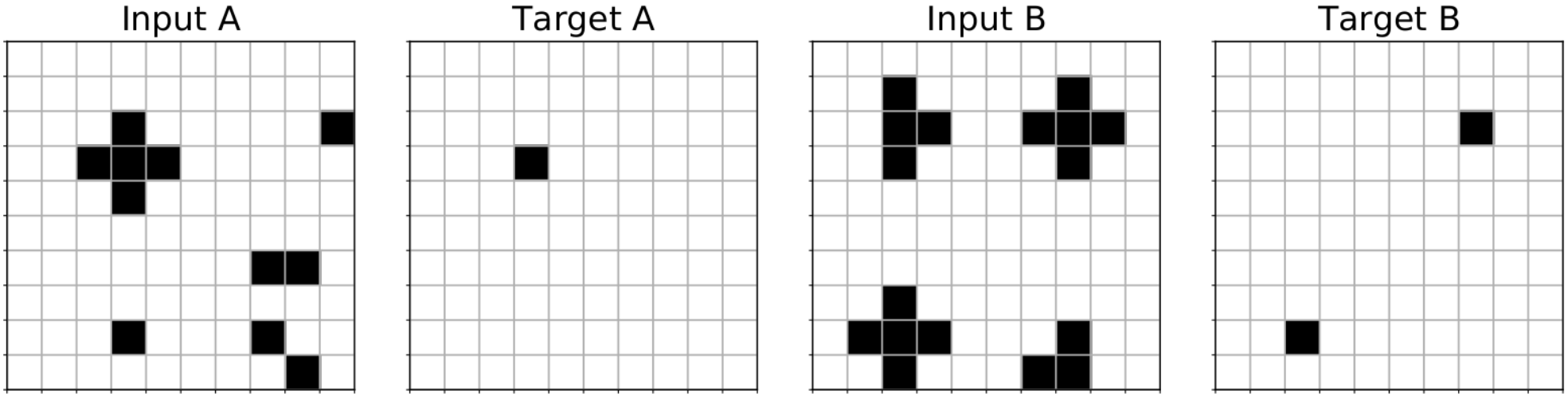
Simple digit representation as a 6x7 binary pixel image



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



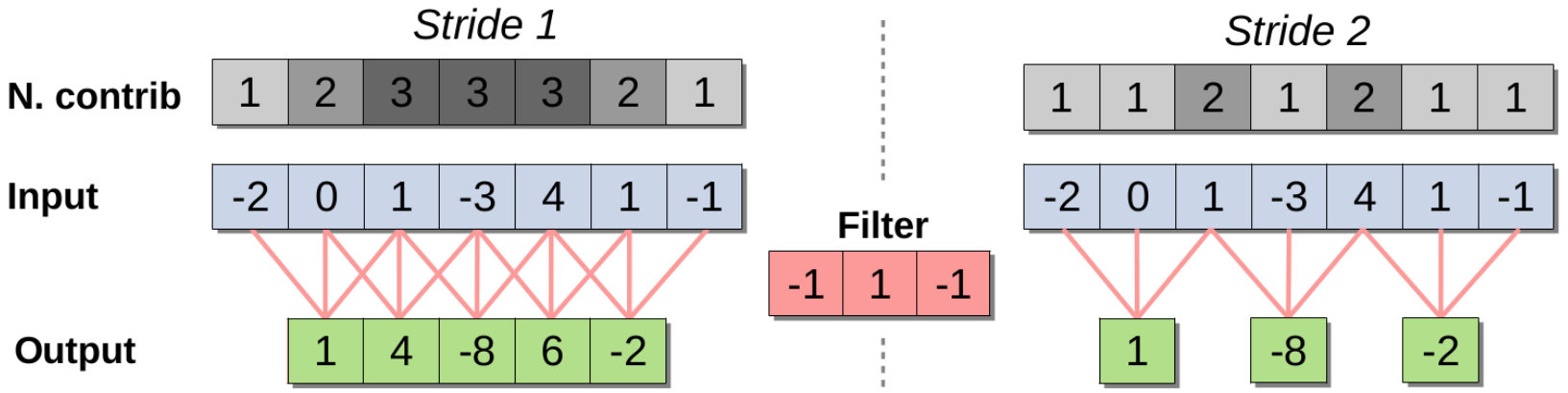
Looking for specific patterns can be automatized by scanning all the possible positions in the image.

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.

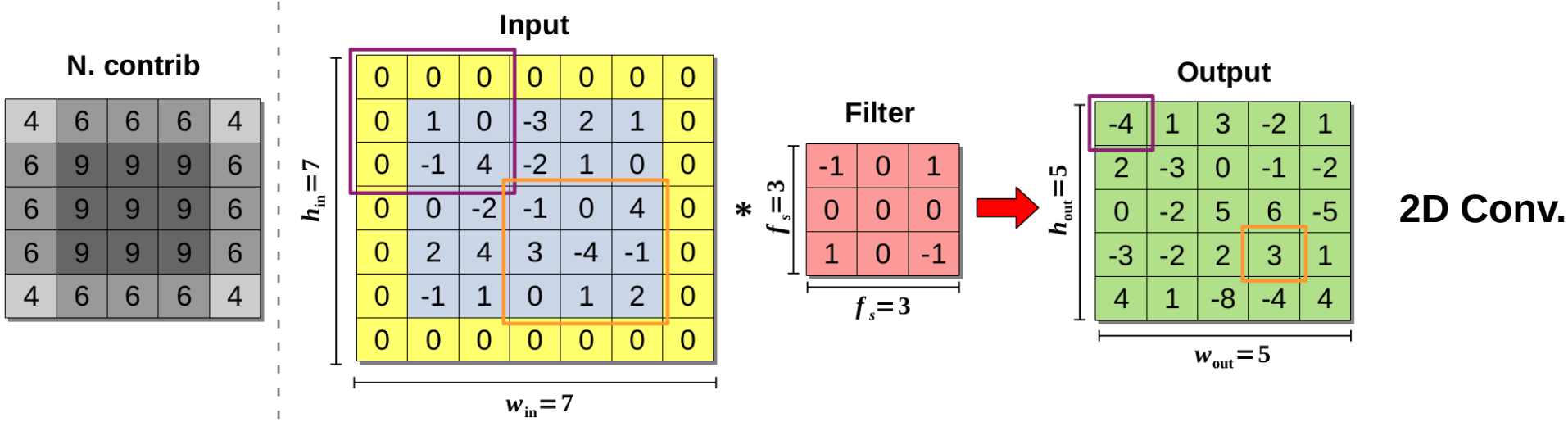
How to circumvent this behavior ? ➡ Use **Convolutional layers** !

Convolution filter

1D Conv.



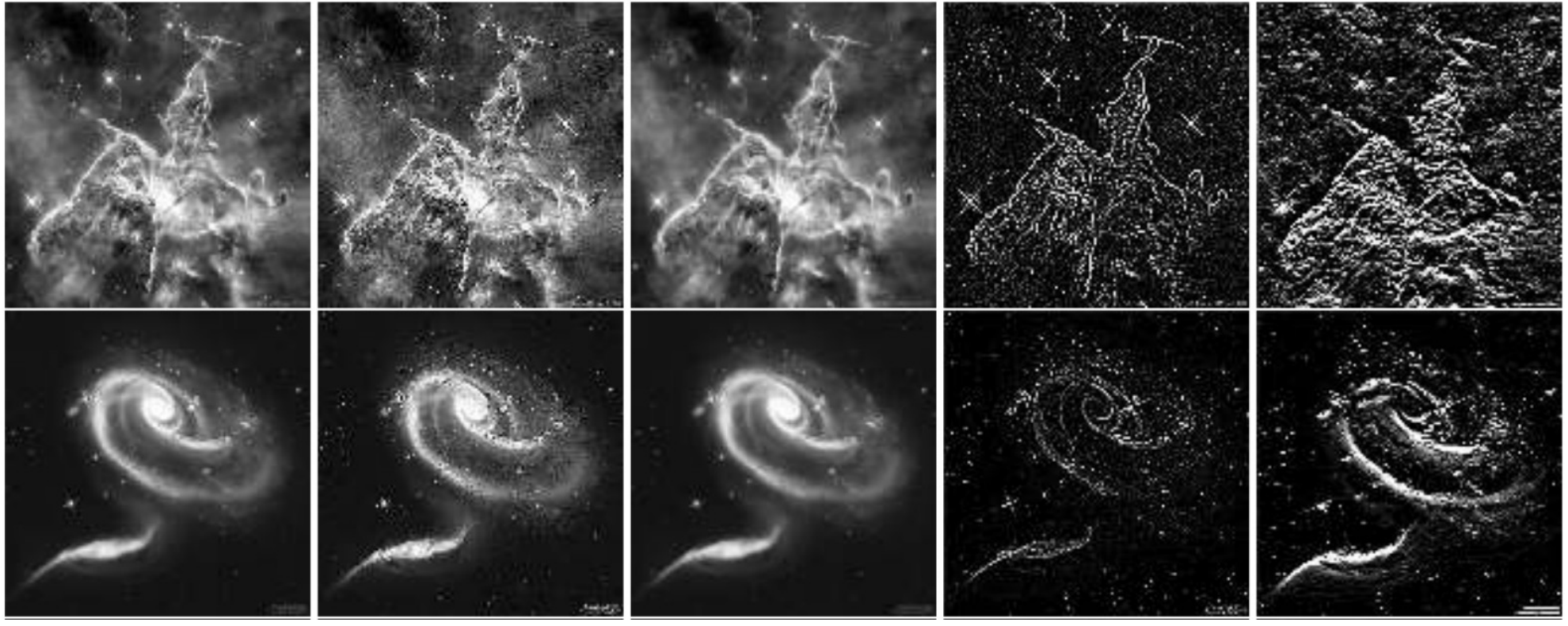
2D Conv.



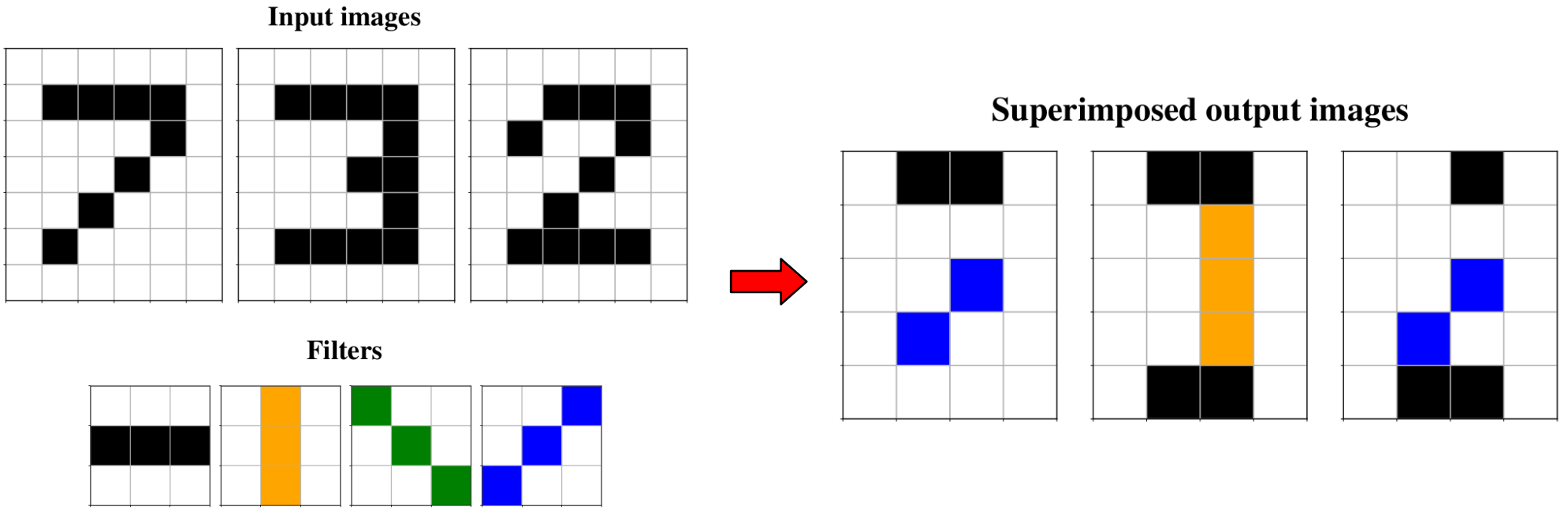
Stride 1 and Padding 1

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

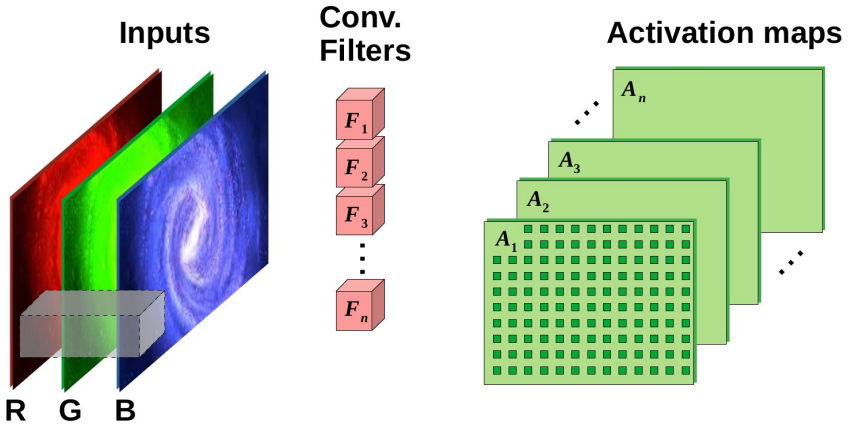


Each filter will produce its own activation map.

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

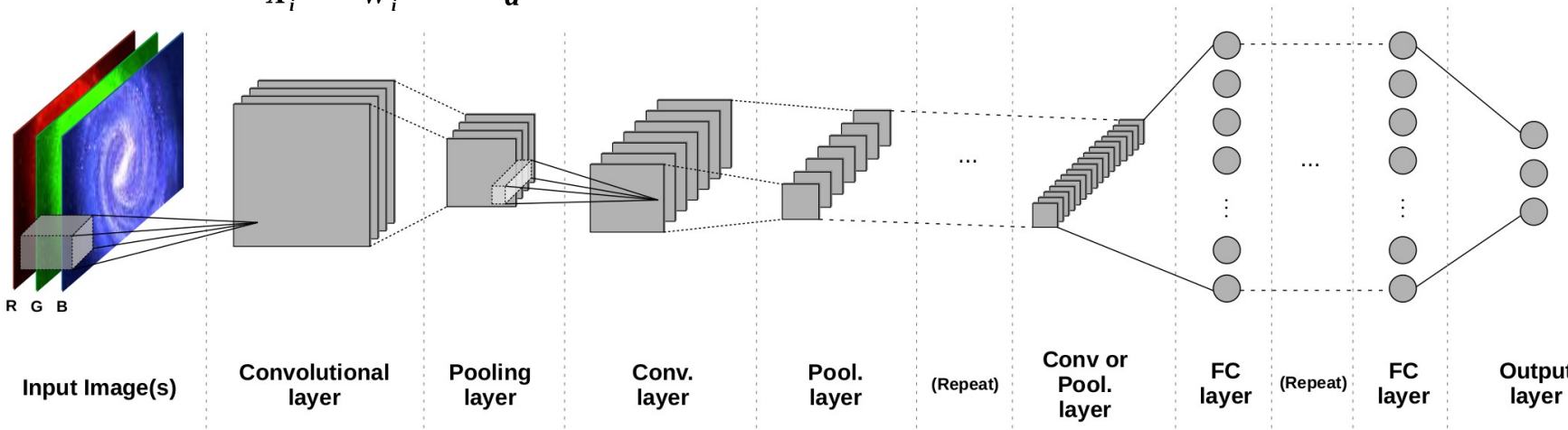


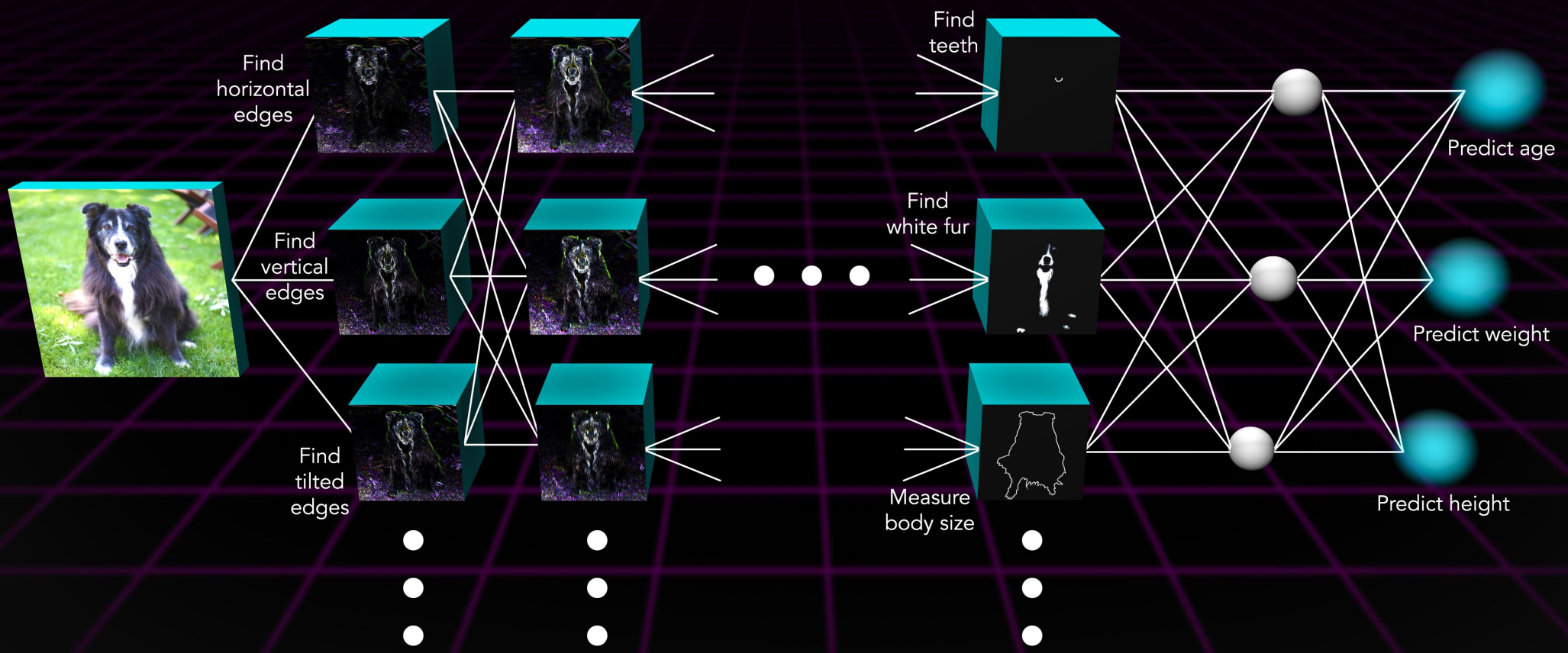
Each filter can be seen as a **single neuron** with one weight per input dimension in the filter.

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)

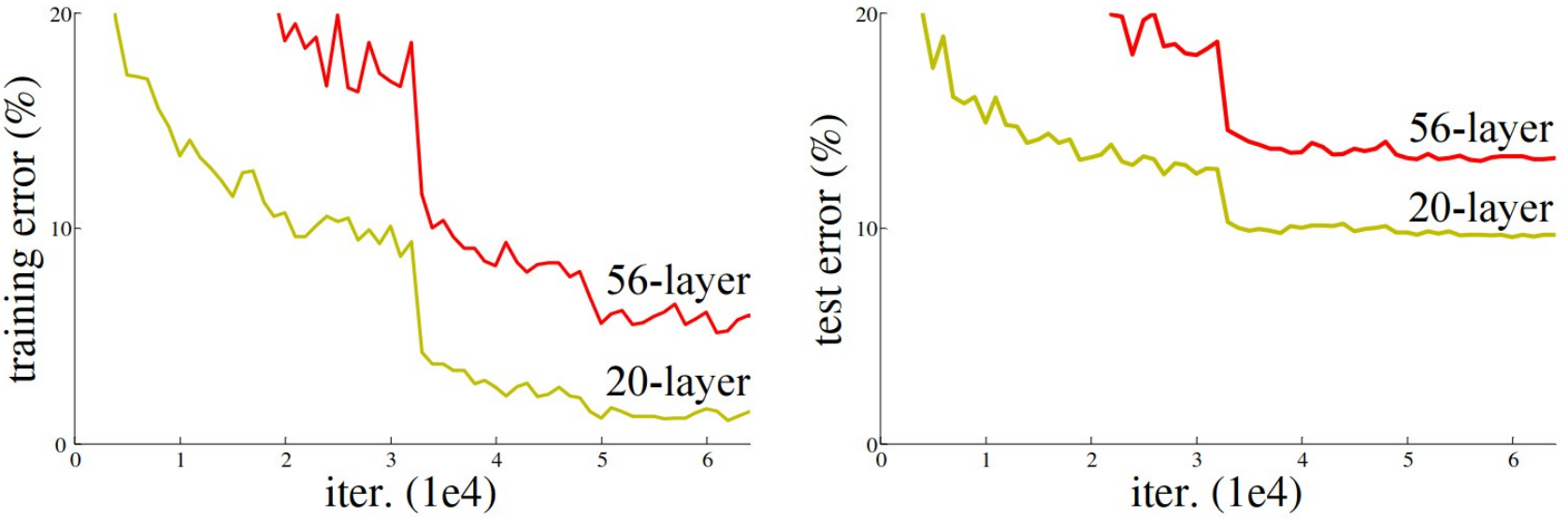
Patterns (layer mixed4a)

Objects (layers mixed4d & mixed4e)

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

Vanishing Gradient Issue

From He et al. 2015



In principle, the deeper the network, the higher its expressivity should be as long as it is trained with enough data. However, it is nos the case in practice due to the gradient slowly getting smaller and smaller as it goes through more layers.

Still many approaches can mitigate this issue to construct network with hundreds of layers (e.g., changing the activation or having pass through connections between layers that are far away in the network).

The Rectified Linear Unit (ReLU)

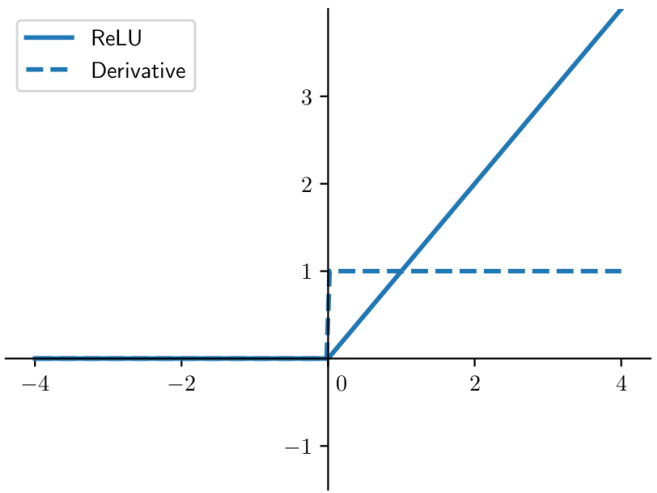
The ReLU (or its variance, the leaky-ReLU) has proven more efficient for CNN. It preserves a form of non-linearity and its constant derivative reduces **vanishing gradient problems**.

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

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

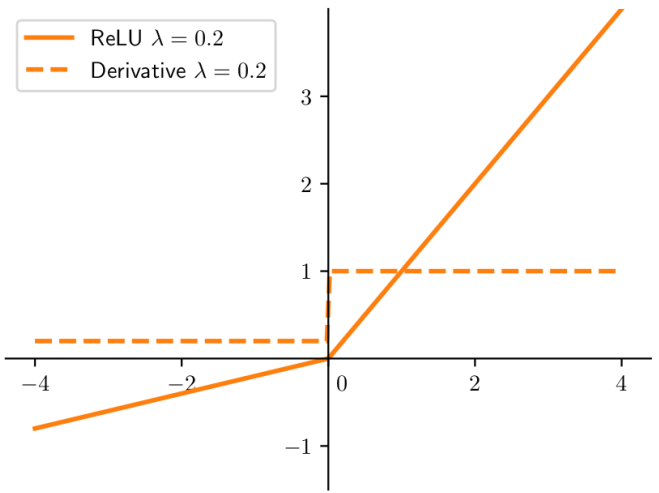
$$a_j = g(h_j) = \begin{cases} h_j & \text{if} \quad h_j \geq 0 \\ 0 & \text{if} \quad h_j < 0 \end{cases}$$

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

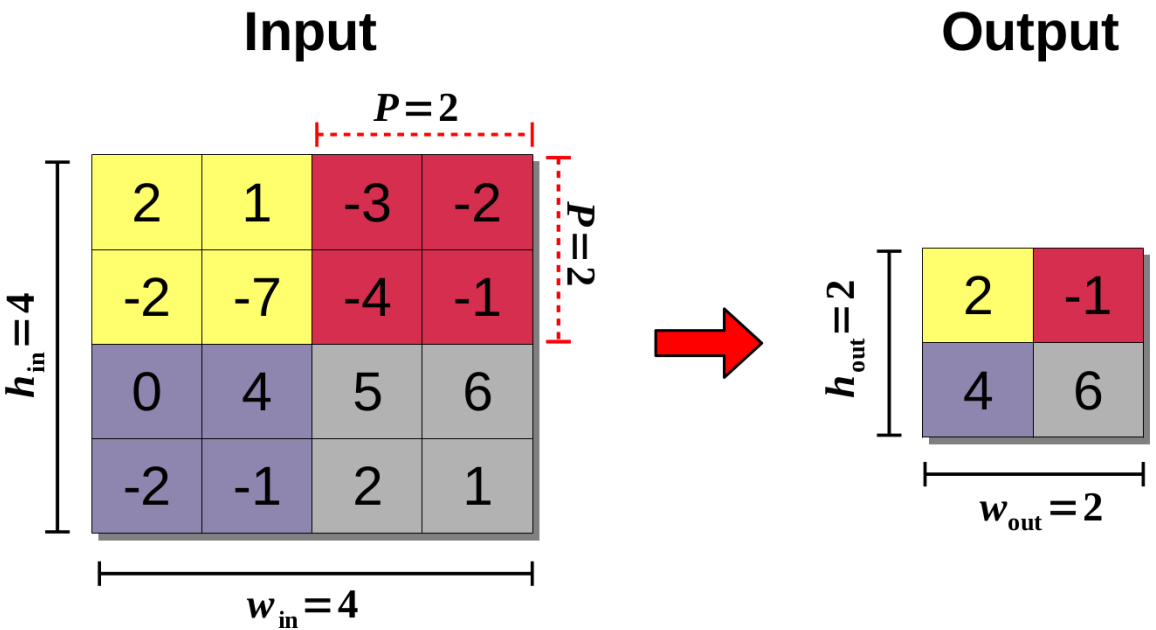


Dimensionality reduction: Pooling

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** !

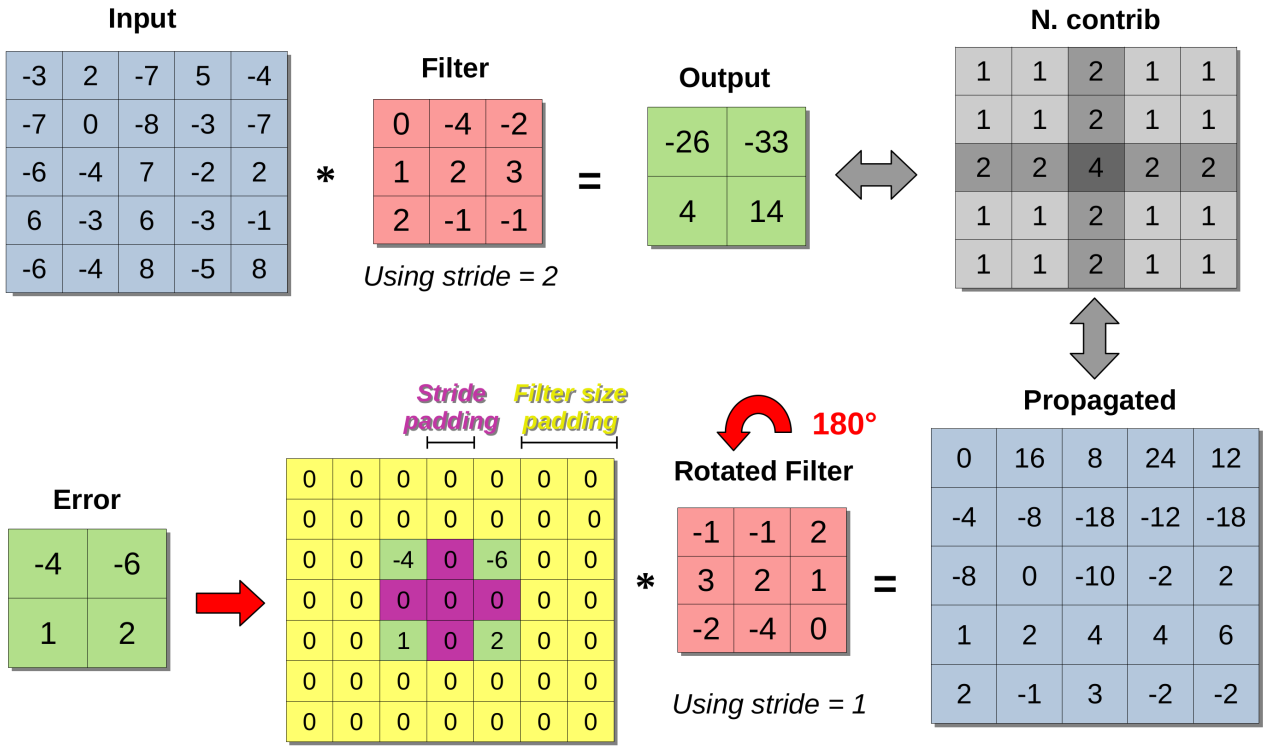


Different pooling methods exist, the most common being Max-Pooling and Average-Pooling. The pooling size can be modified, but most of the network architectures reduce each spatial dimension by a factor of two.

Learning the filters

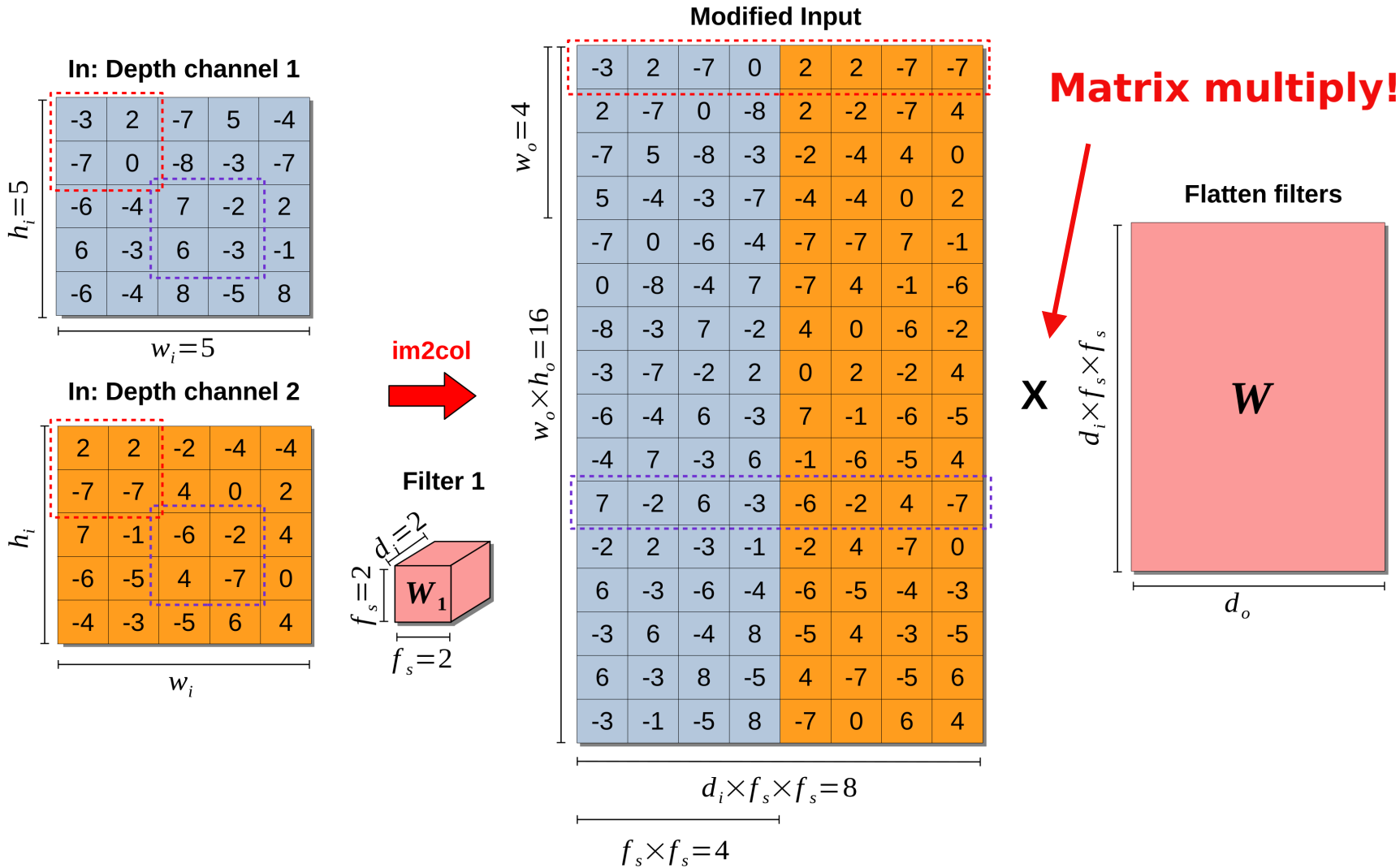
Like fully connected layers, the **convolutional filters** can be learned using **backpropagation** of the error measured at the output layer.

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.

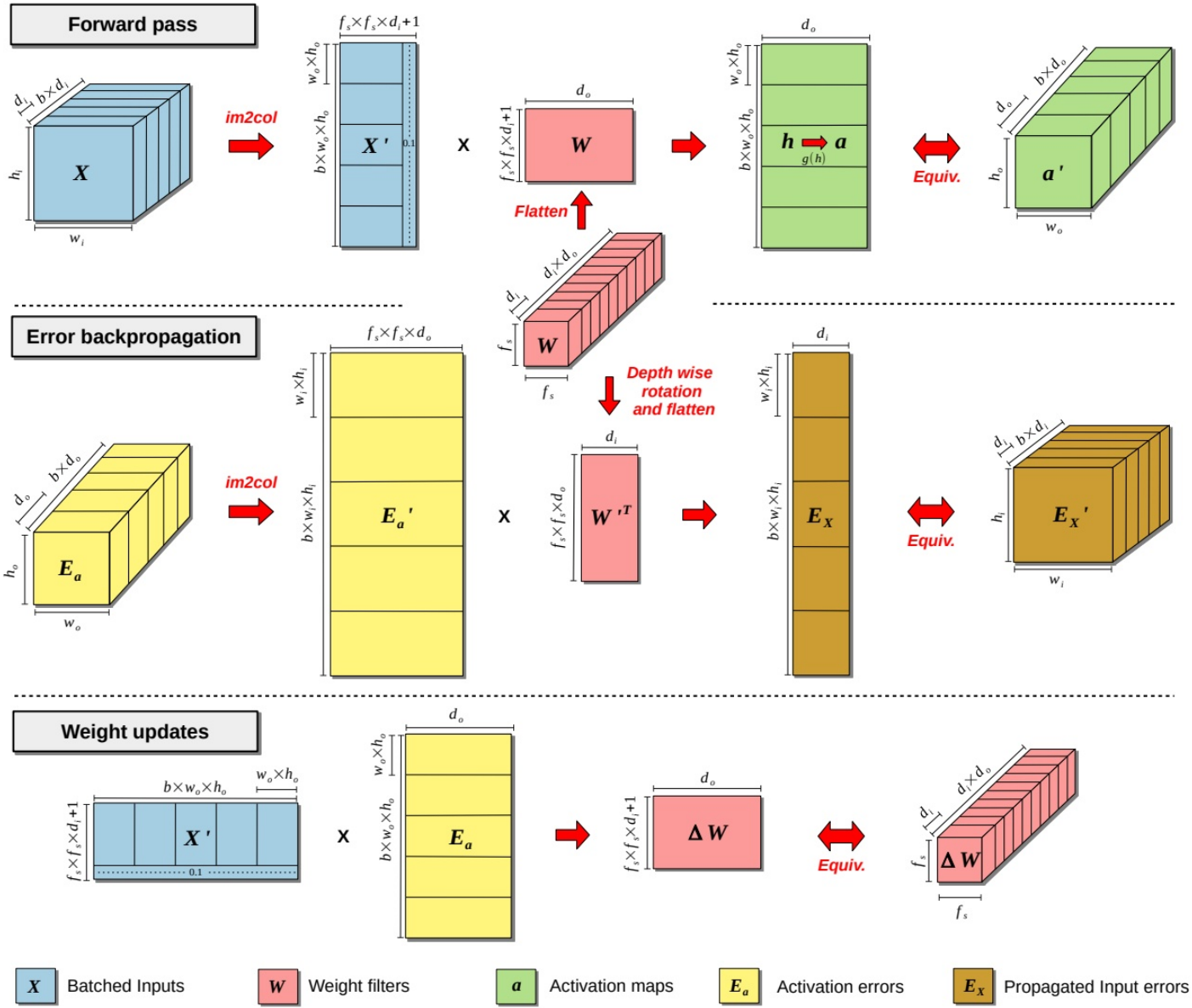


*Learning the convolutional filters is often considered to be the definition of "**Deep Learning**".*

The Im2col transformation



Convolutional layer complete matrix formalism

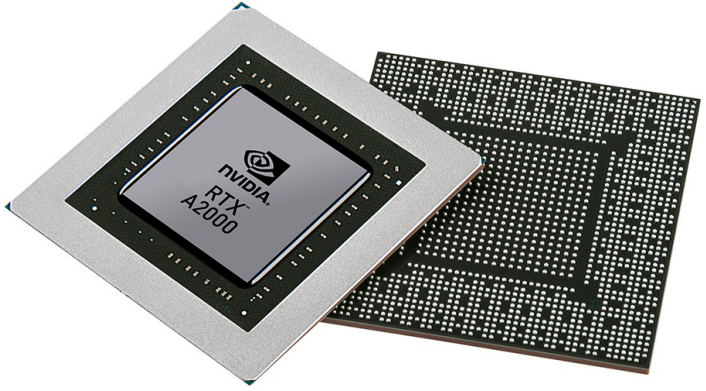


What about computing on GPU?

GPU (Graphical Processing Unit) are massively parallel computing chips dedicated to SIMD like operations (thousands of cores). Most image processing algorithm apply the same transformation to millions of pixels, hence the SIMD formalism.

GPU have the same form factor than CPU but usually come as a dedicated daughter board with their own large cooling system as they can have a much higher power draw than CPU!

GPUs are not suited for all tasks, but for those they where designed for, they pack a huge amount of computing power, which include matrix multiplication!



Nvidia H100 GPU spec-sheet (AI dedicated)

Graphics Processor	
GPU Name:	GH100
Architecture:	Hopper
Foundry:	TSMC
Process Size:	4 nm
Transistors:	80,000 million
Density:	98.3M / mm²
Die Size:	814 mm²

Graphics Features	
DirectX:	N/A
OpenGL:	N/A
OpenCL:	3.0
Vulkan:	N/A
CUDA:	9.0
Shader Model:	N/A

Graphics Card	
Release Date:	Mar 21st, 2023
Generation:	Tesla Hopper (Hxx)
Predecessor:	Tesla Ada
Production:	Active
Bus Interface:	PCIe 5.0 x16

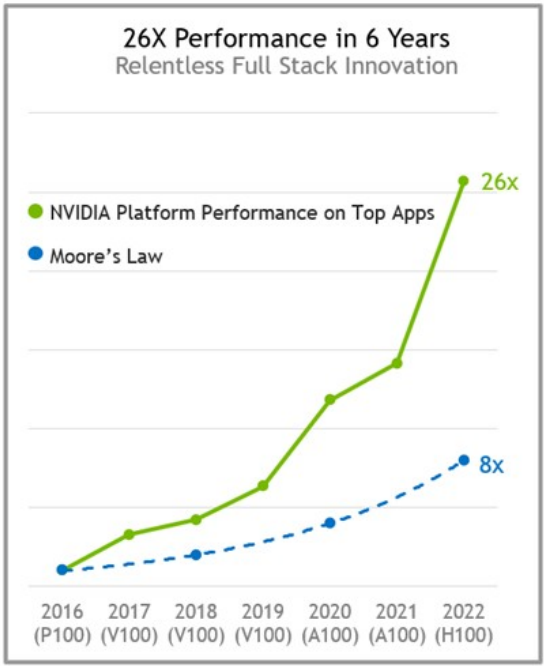
Board Design	
Slot Width:	Dual-slot
Length:	268 mm 10.6 inches
Width:	111 mm 4.4 inches
TDP:	350 W
Suggested PSU:	750 W
Outputs:	No outputs
Power Connectors:	1x 16-pin
Board Number:	P1010 SKU 200

Clock Speeds	
Base Clock:	1095 MHz
Boost Clock:	1755 MHz
Memory Clock:	1593 MHz 3.2 Gbps effective

Render Config	
Shading Units:	14592
TMUs:	456
ROPs:	24
SM Count:	114
Tensor Cores:	456
L1 Cache:	256 KB (per SM)
L2 Cache:	50 MB

Memory	
Memory Size:	80 GB
Memory Type:	HBM2e
Memory Bus:	5120 bit
Bandwidth:	2,039 GB/s

Theoretical Performance	
Pixel Rate:	42.12 GPixel/s
Texture Rate:	800.3 GTexel/s
FP16 (half):	204.9 TFLOPS (4:1)
FP32 (float):	51.22 TFLOPS
FP64 (double):	25.61 TFLOPS (1:2)



From Nvidia

From Techpowerup

GPU architecture

The GPU is specialized for highly parallel computations and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control. The schematic Figure 1 shows an example distribution of chip resources for a CPU versus a GPU.

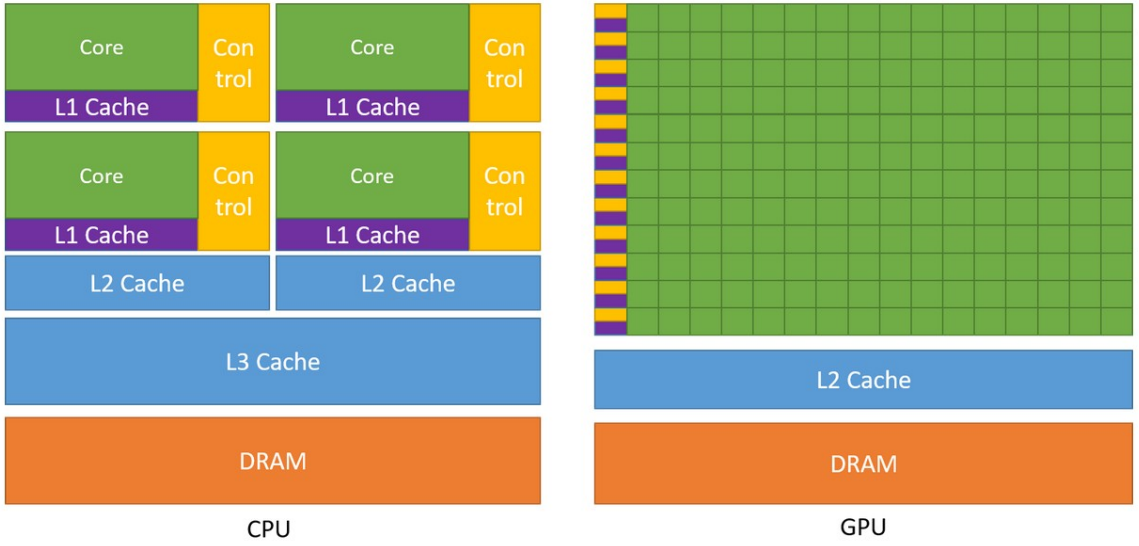


Figure 1: *The GPU Devotes More Transistors to Data Processing*

GPU architecture

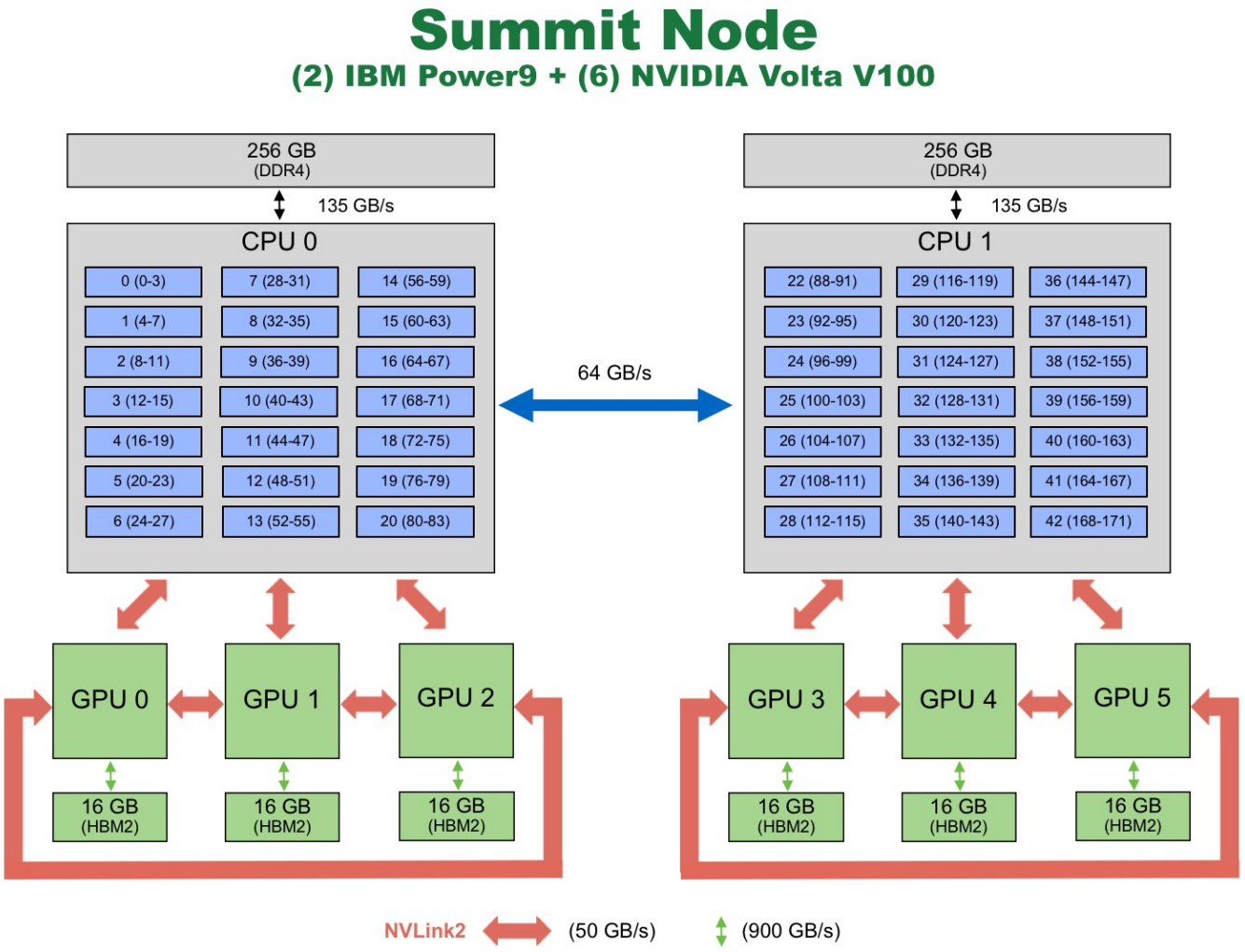
Ex. of a GP104 – Tesla P100

The GPU is equipped with a shared “on board” memory. All SM share the same large L2 cache.

Each SM has a dedicated cache and can launch multiple instruction blocs through multiple warp schedulers.

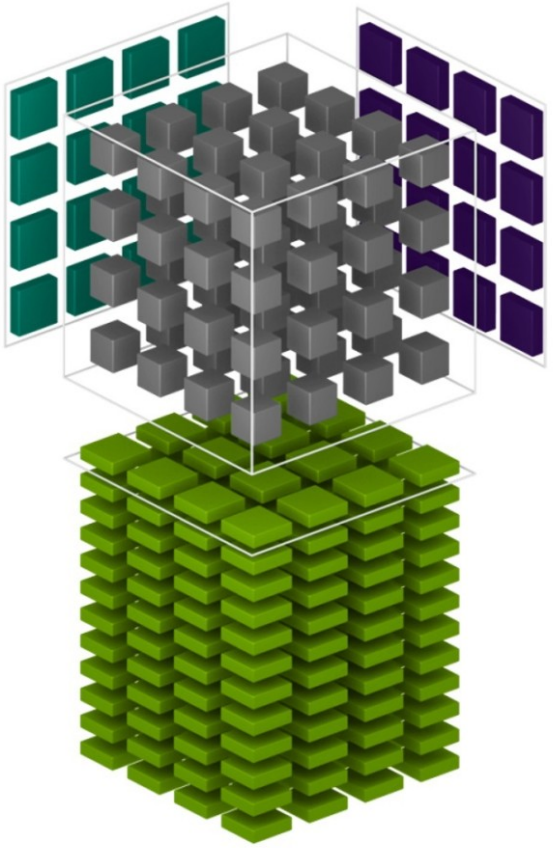
Inside each SM, there are several CUDA cores and other dedicated compute units.

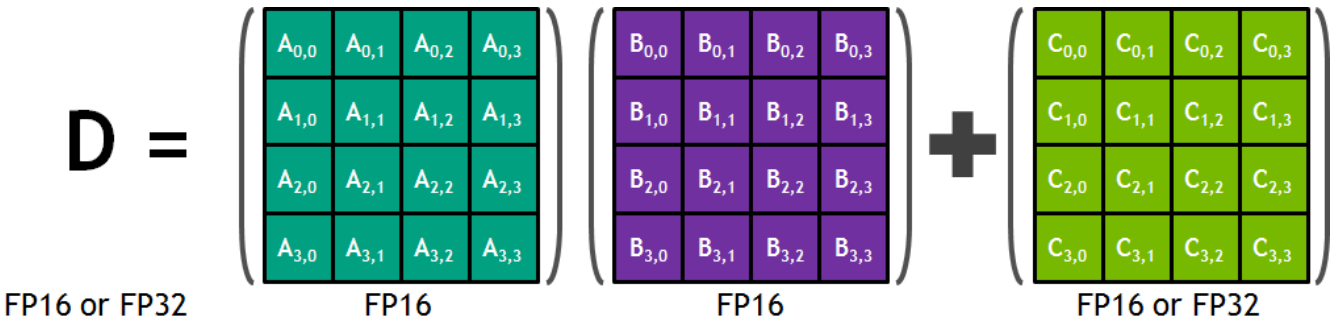
Distribution in GPU clusters

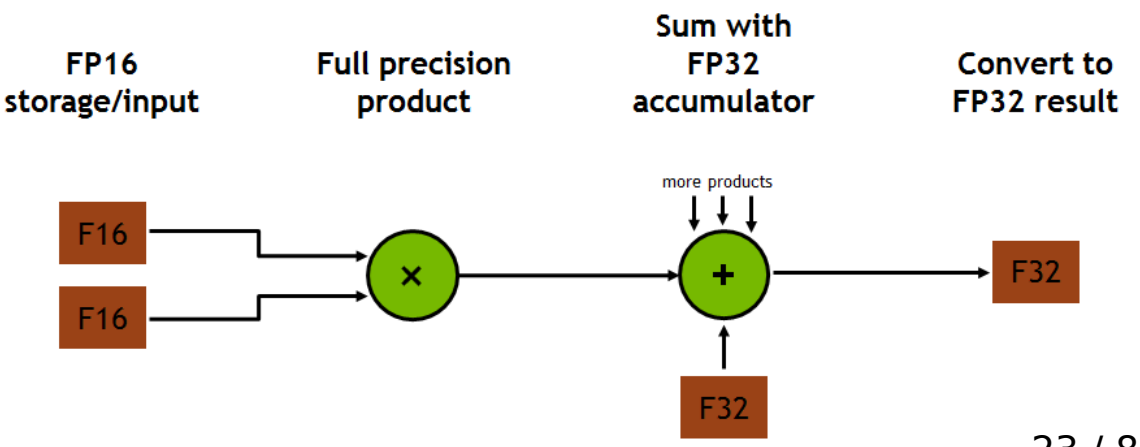


Nvidia Tensor Cores

Optimized Warp Matrix Multiply Add (WMMA) instructions !
CuBLAS can be set to used tensor core through the gemmEX function.

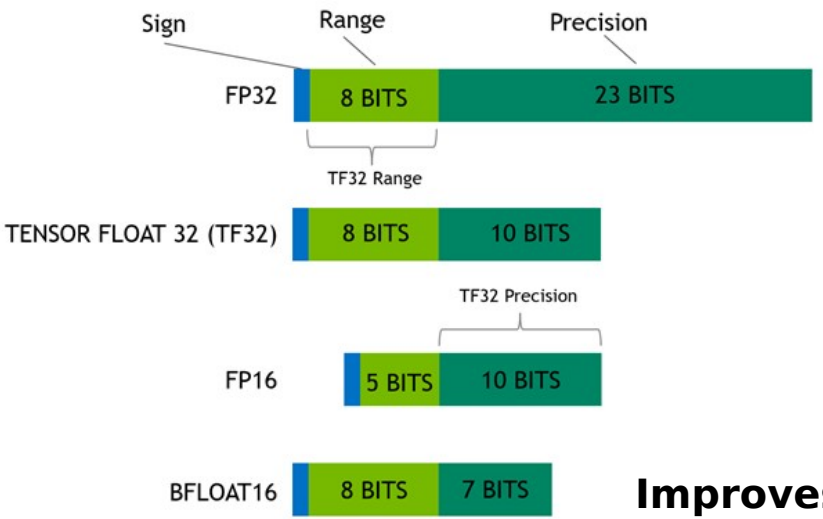


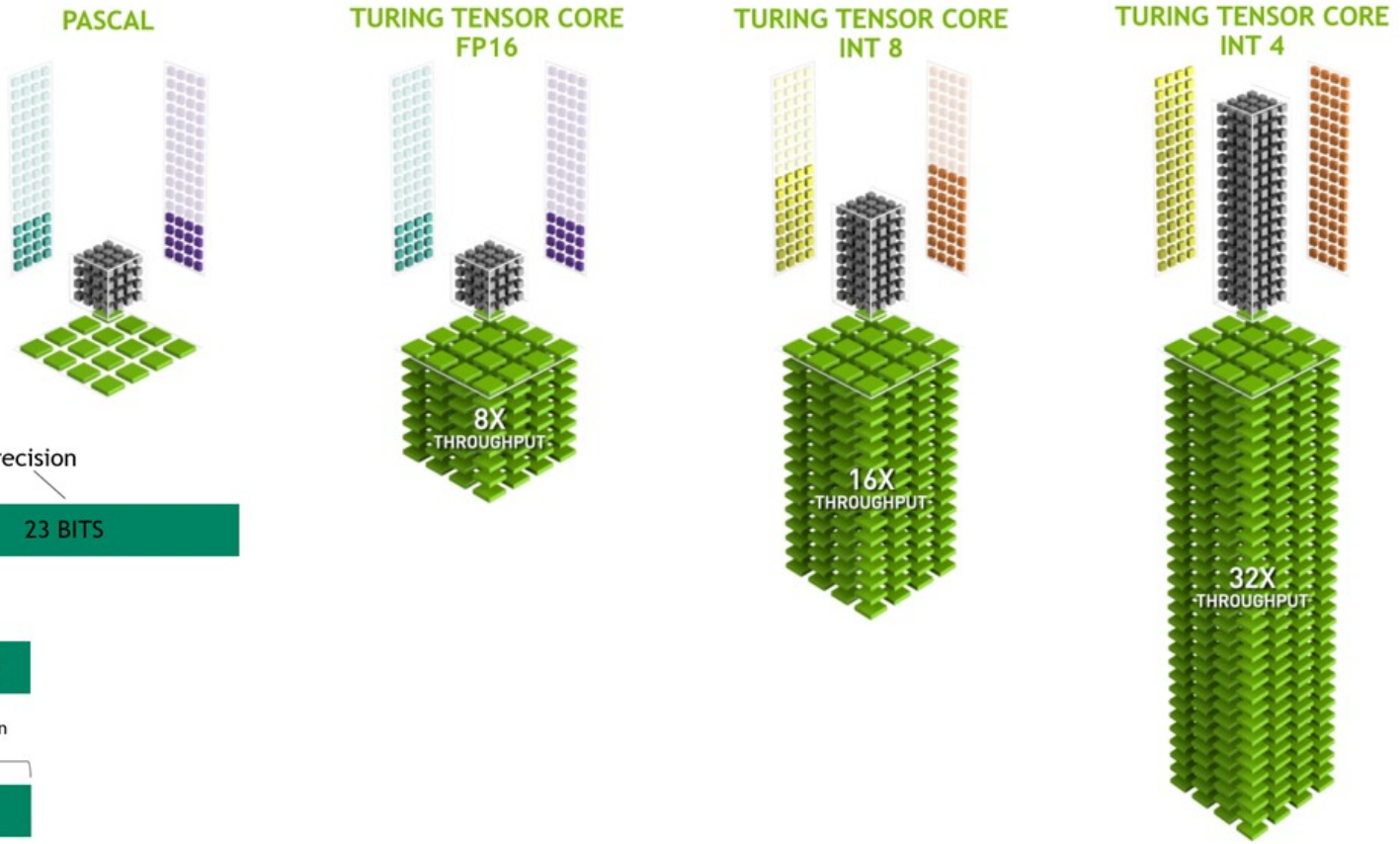




Nvidia Tensor Cores reduced quantization

$$1.2345 = \underbrace{12345}_{\text{mantissa}} \times 10^{\overbrace{-4}^{\text{exponent}}}$$

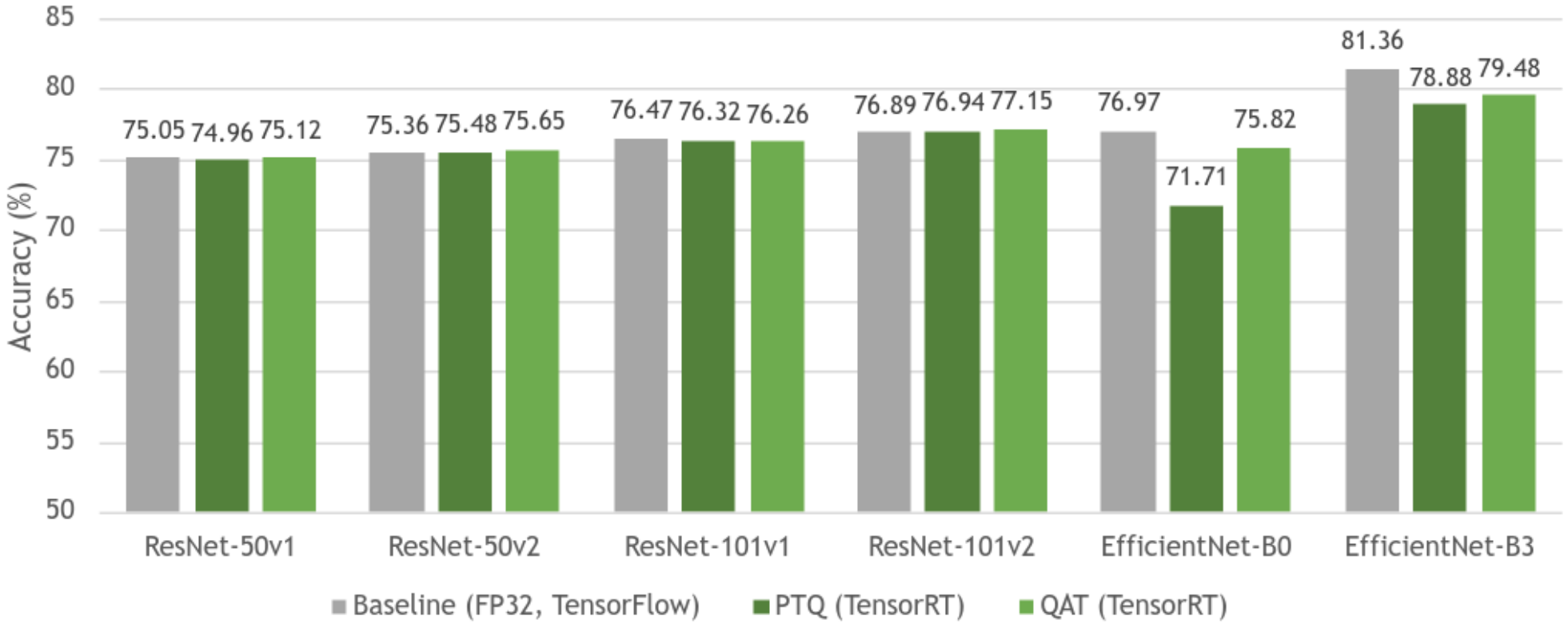




Improves compute throughput AND reduces data size,
→ reduce memory footprint and required bandwidth

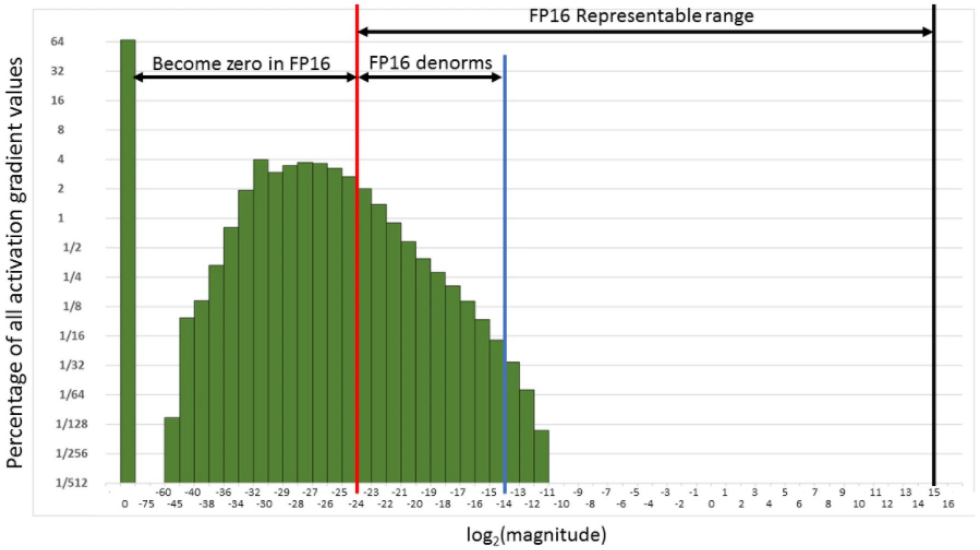
Quantization effect on AI model accuracy

All models quantized to INT8, accuracy for ImageNET-2012



PTQ = Post training quantization
QAT = Quantization aware training

Mixed precision training

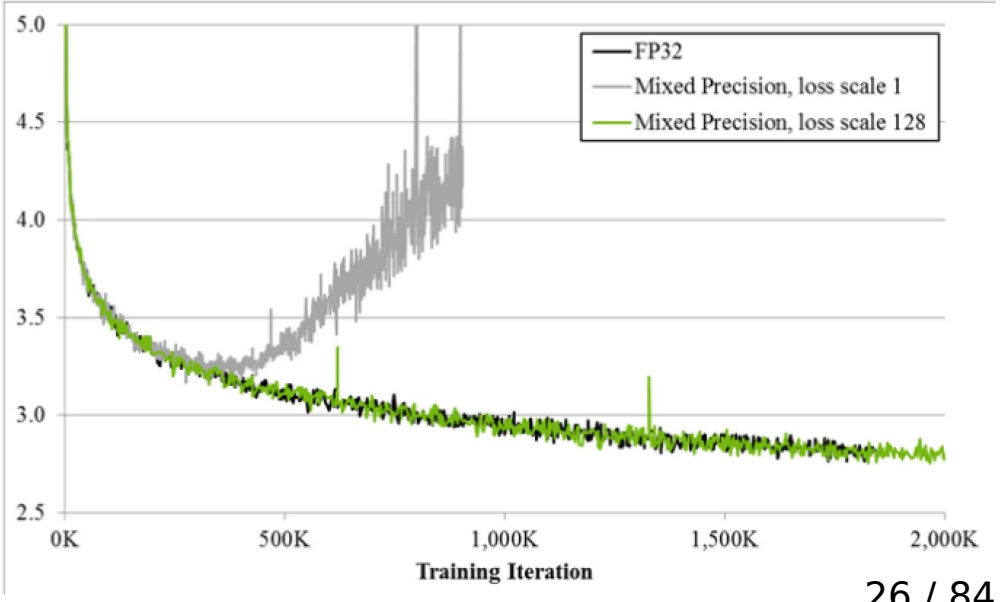


Reduced bit count variables have smaller representable ranges. This can lead to strong gradient vanishing problems.

This problem can first be mitigated by preserving an FP32 copy of the weights for accumulating the updates.

To further reduce this issue, a scaling is applied on the output loss. All propagated values are naturally scaled so they are more likely to be in the proper range.

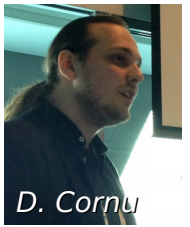
At weight update time the correction is scaled down by the same factor.

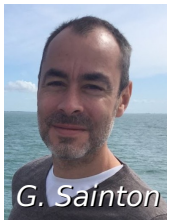


From Nvidia « Mixed precision training »

CIANNA

Development team





Convolutional Interactive Artificial Neural Networks by/for Astrophysicists

General purpose framework BUT developed for **astronomical applications**

x86 Intel/AMD & ARM / **Supports modern TC acceleration** / **Python interface**

Work on a wide variety of hardware from IoT to super-computing facilities

github.com/Deyht/CIANNA

Open source – Apache 2 license

July 24, 2024 (V-1.0.0.0) | Software | Open

Deyht/CIANNA: CIANNA V-1.0

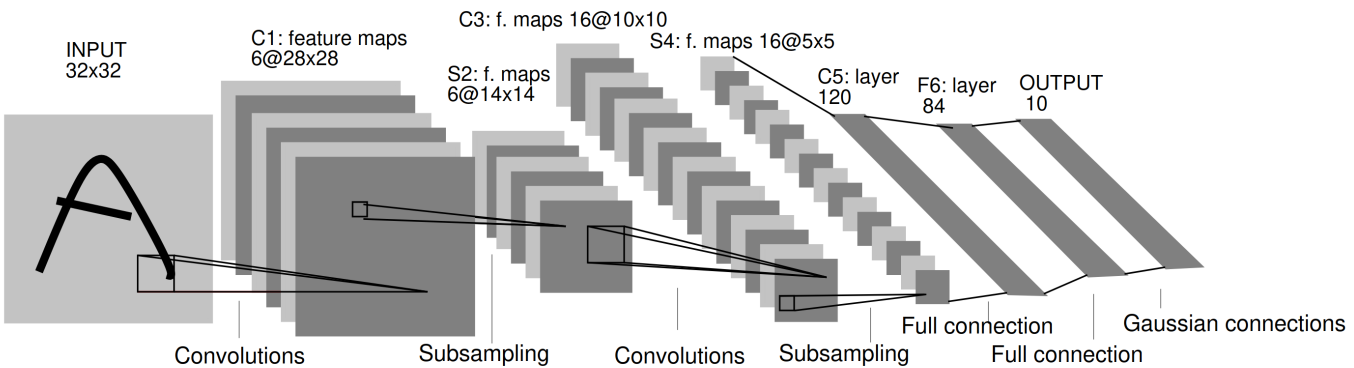
Simple examples: MNIST

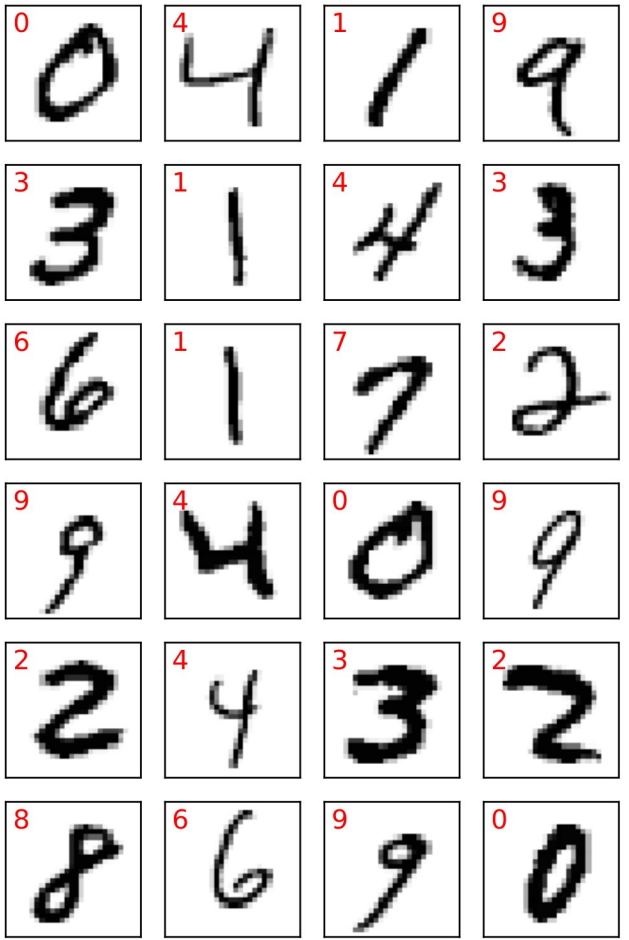
The well-known **MNIST** (Modified NIST's special dataset) dataset consists of handwritten digits from 500 different writers expressed as 28x28 grayscale images.

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):





How to use CIANNA for MNIST ?

Example script over MNIST, with an LeNet5 *like* network.

Interface is similar to widely adopted frameworks.

The **full interface documentation** is available on the github repo as a Wiki page listing all available functions with their descriptions.

Several example scripts are provided as Google Colab notebooks.

```
#CIANNA initialization
cnn.init(in_dim=i_ar([28,28]), in_nb_ch=1, out_dim=10,
    bias=0.1, b_size=16, comp_meth="C_CUDA",
    dynamic_load=1, mixed_precision="FP32C_FP32A")

#Create data subsets (from numpy arrays)
cnn.create_dataset("TRAIN", size=60000, input=data_train, target=target_train)
cnn.create_dataset("VALID", size=10000, input=data_valid, target=target_valid)
cnn.create_dataset("TEST", size=10000, input=data_test, target=target_test)

#Define the network structure sequentially
cnn.conv(f_size=i_ar([5,5]), nb_filters=8 , padding=i_ar([2,2]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.conv(f_size=i_ar([5,5]), nb_filters=16, padding=i_ar([2,2]), activation="RELU")
cnn.pool(p_size=i_ar([2,2]), p_type="MAX")
cnn.dense(nb_neurons=256, activation="RELU", drop_rate=0.5)
cnn.dense(nb_neurons=128, activation="RELU", drop_rate=0.2)
cnn.dense(nb_neurons=10, strict_size=1, activation="SMAX")

#Training loop configuration and launch
cnn.train(nb_iter=20, learning_rate=0.004, momentum=0.8, confmat=1, save_every=0)

#Evaluate network prediction after training
cnn.forward(repeat=1, drop_mode="AVG_MODEL")
```

Example results on MNIST

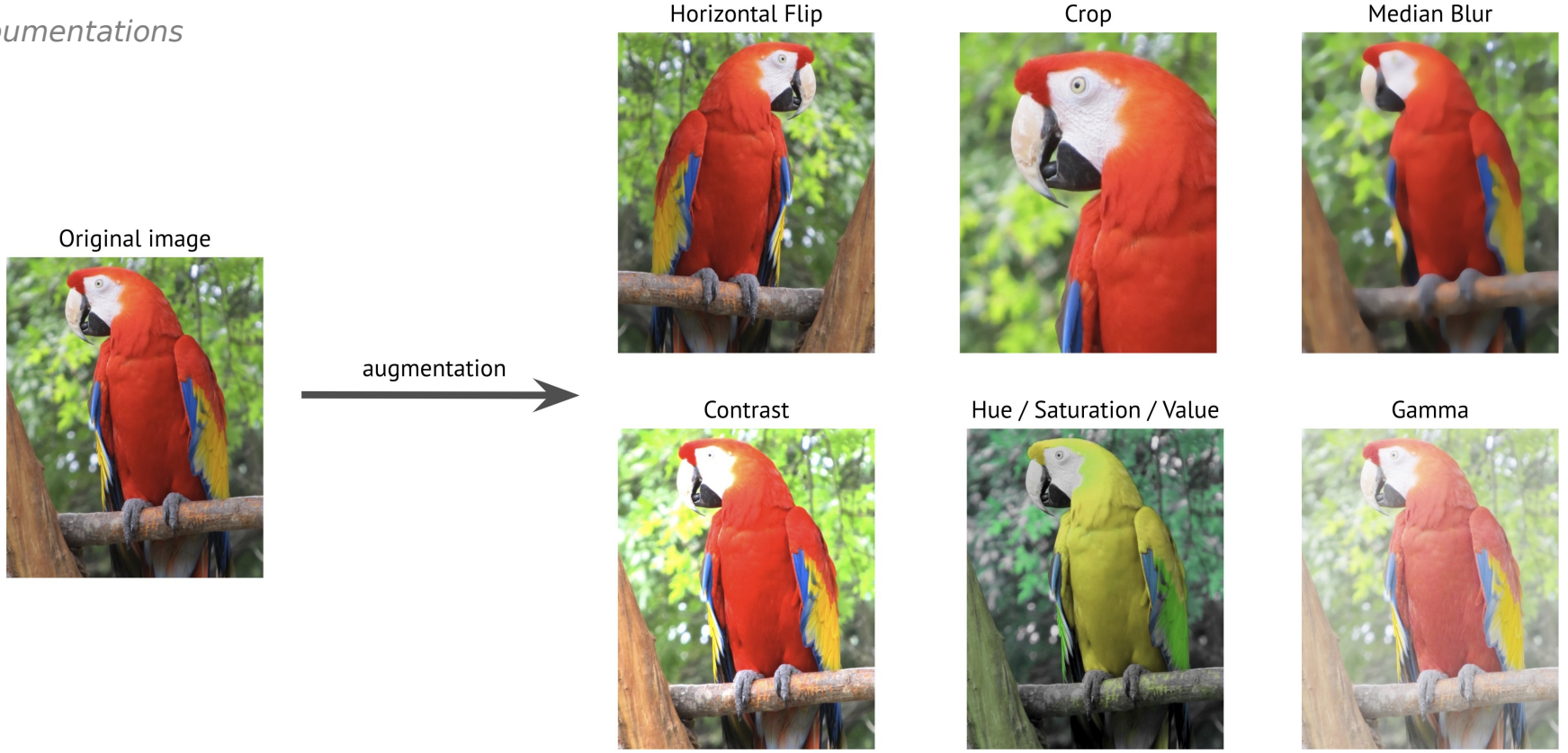
Actual \ Predicted	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	Recall
C0	976	0	1	0	0	0	1	1	1	0	99.6%
C1	0	1132	1	0	1	0	0	0	1	0	99.7%
C2	1	1	1027	0	1	0	0	1	1	0	99.5%
C3	0	0	1	1004	0	3	0	1	1	0	99.4%
C4	0	0	1	0	972	0	1	0	1	7	99.0%
C5	0	0	0	4	0	886	1	0	0	1	99.3%
C6	3	2	0	0	1	2	949	0	1	0	99.1%
C7	0	2	3	0	0	0	0	1020	1	2	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%

Practical work:

Reproduce this result using the provided notebooks. Try to modify the architecture *(filter size, #filters, #neurons, batch size, learning rate, etc.)* and estimate the impact of individual changes on the training / inference time and final accuracy.

Data augmentation

From Albumentations



Depending on the task, one can choose a set of transforms that do not alter the labeling corresponding to the image (here the class). This allow to increase the coverage of the feature space without the need of new data.

Augmentation does not solve all the dataset size issues, it usually **cannot create new contexts** nor it can generate features that are simply missing in the original dataset.

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 !

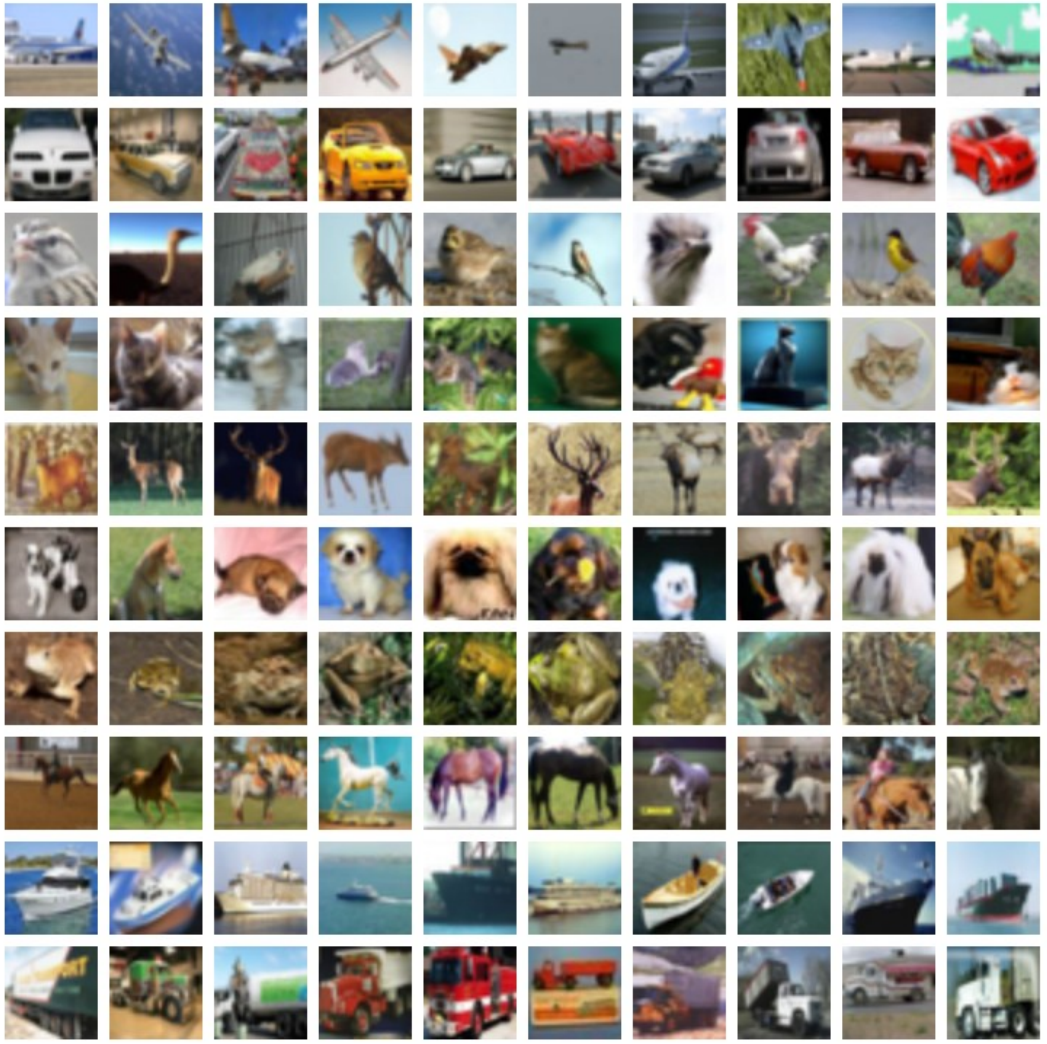
We can generate standardize and augmented images from the dataset using dynamic image augmentation.

Practical work:

Using the provided dataset and data loading script, implement a CNN (with either framework) for this classification. Explore the architectures, hyper-parameters, and augmentation setups by yourself and try to optimize the prediction result.

A more complex task, the CIFAR-10 dataset

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



60000 images of 32x32 pixels labeled into 10 classes.

50000 images for training and 10000 for testing.

Modern methods can reach 97% accuracy on this dataset.

The lower resolution combined to the increase in number of classes and image context make it a more difficult task to solve that for ASIRRA.

Practical work:

Do the same work you've done for ASIRRA on this new dataset. Try to identify the key differences between the dataset and explore how it affect the optimal architecture.

1D CNN for spectra

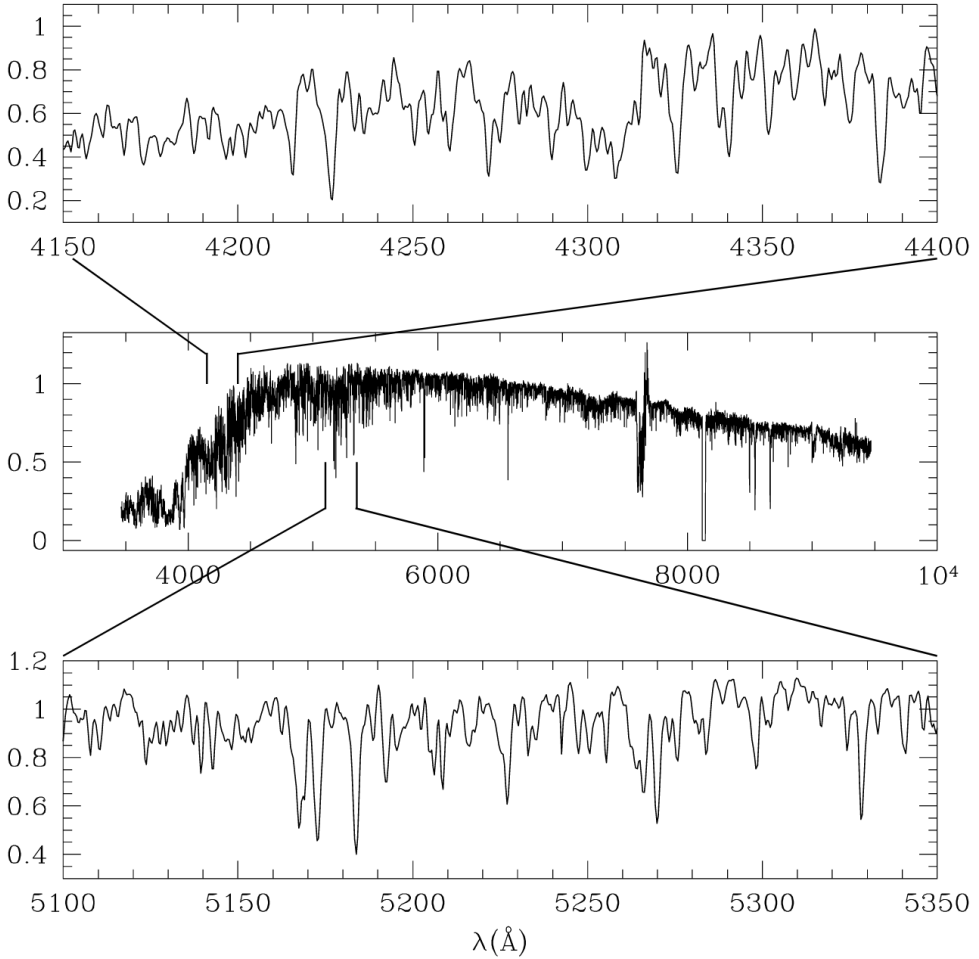
A spectra is a continuous 1D data structure decomposed into many frequency bins.

Like an image it can exhibit recurrent patterns that could be search in a **translation equivariant** way.

Practical work:

Using the stellar spectra dataset from the Perceptron and MLP parts of the course, build a 1D CNN that performs the classification. As before, a notebook is provided as a starting point.

As the number of frequency bins is much higher than in a typical image, you might need to use larger filters and to reduce the convoluted dimensionality more aggressively.



1D CNN for spectra

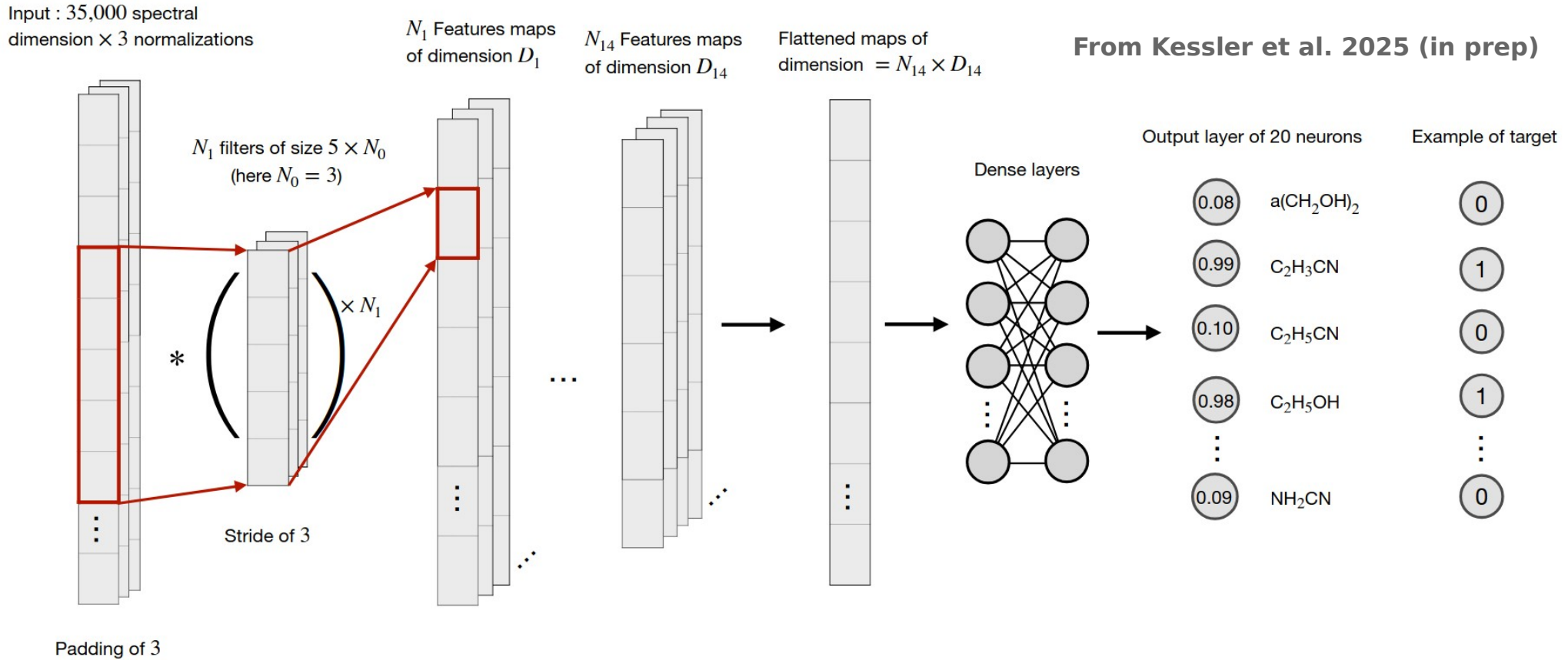
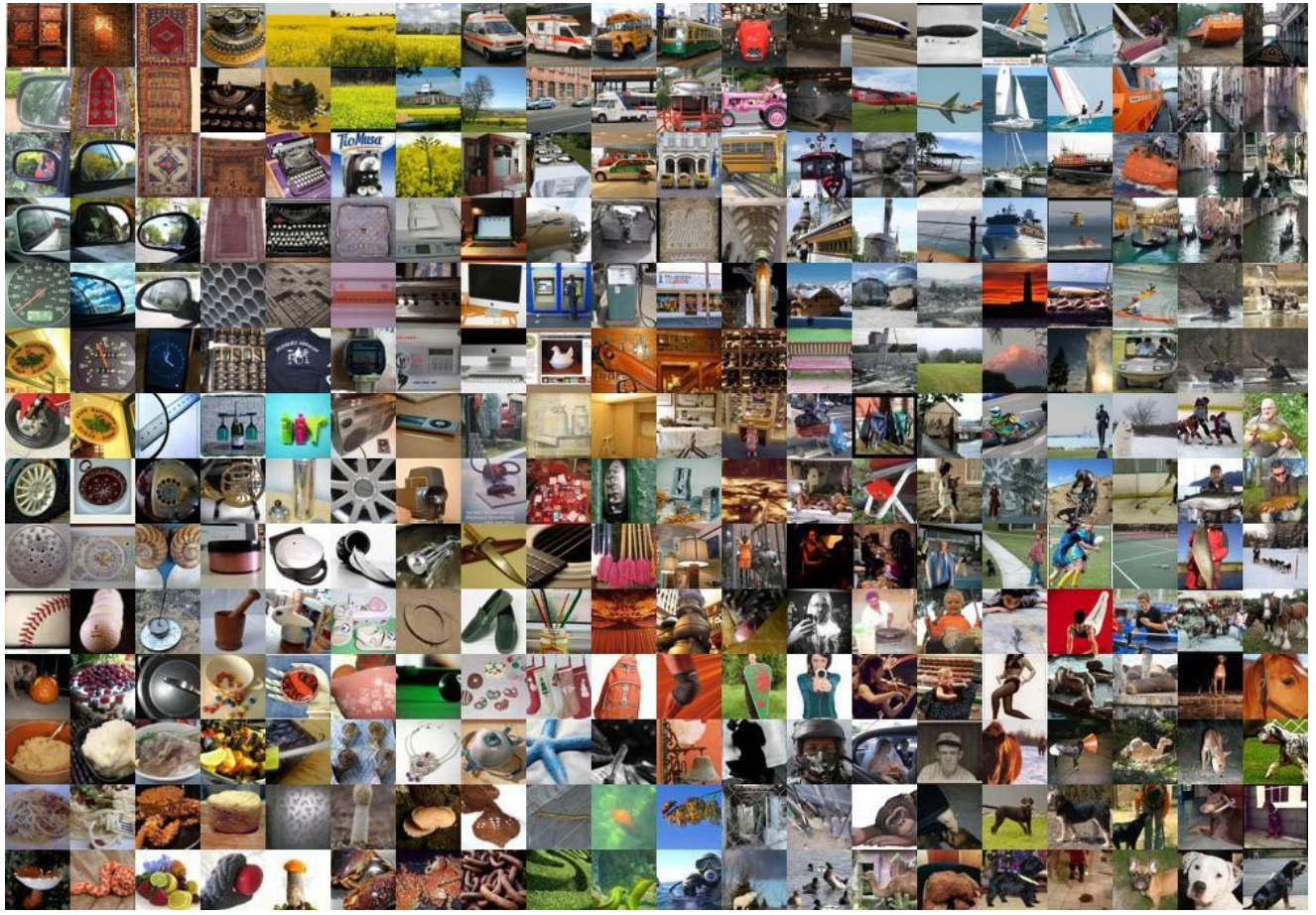


Fig. 5. Scheme of the ANN architecture. Filters are applied to the input data to convolve the information and produce features maps. This operation is done for each of the convolutional layers. Dense layers then combine the extracted features and learn how to label the spectra depending on the provided target. The output layer is composed of one neuron per class giving a score between 0 and 1 independent between each other.

The canonical ImageNet-2012 dataset



ImageNet is a famous dataset as it is large and diverse enough to pre-train large models for a large variety of applications (classification, generation, detection, etc.)

- 1,281,167 training images (150 GB)
- 50,000 validation images
- 100,000 test images,
- Each image is associated with one of 1000 possible classes.

Images are of variable resolution with an average around 500x400.

Practical work:

This dataset is way too heavy for training a model during the course. Use the provided script to run a pre-trained model, identify its architecture and visualize some results. You can try apply it to your own images.

The canonical ImageNet-2012 dataset

Result over ImageNet using a darknet-19 backbone, **91.7 Top5 Accuracy** over the 1000 classes, at a 448p resolution. The network run at 740 ips on an RTX 4090.

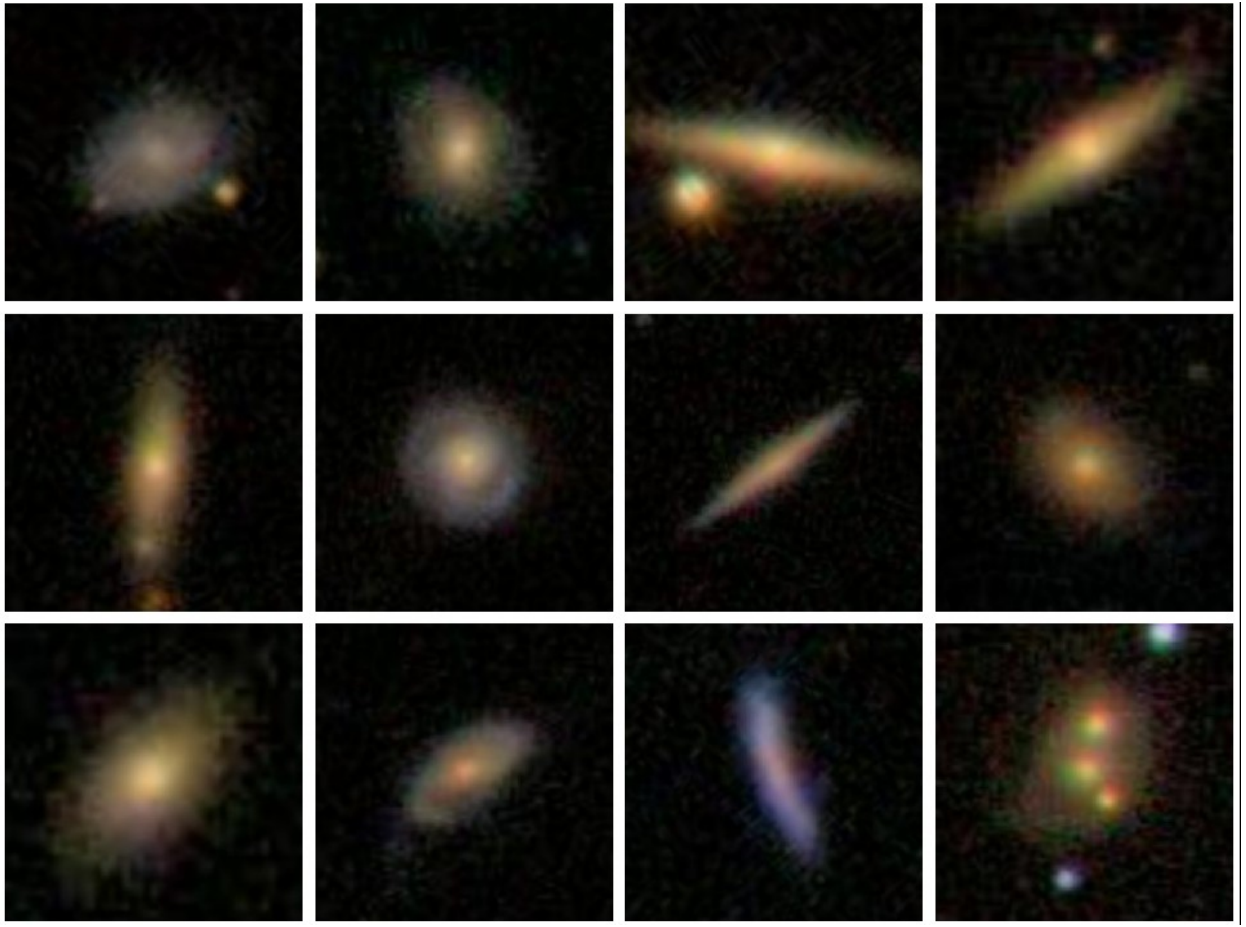
Explaining a model decision with occlusion analysis

An occlusion analysis is done by replacing a portion of the image by noise and measuring the impact it has on the model prediction. An occlusion map can be created by repeating moving the occlusion window over the original image.

Allow to identify image features that contribute positively or negatively to a given prediction.

Here the occlusion map is done with the ImageNet trained model for the « malamute » class.

The Galaxy Zoo 2 dataset
Morphological classification of galaxies

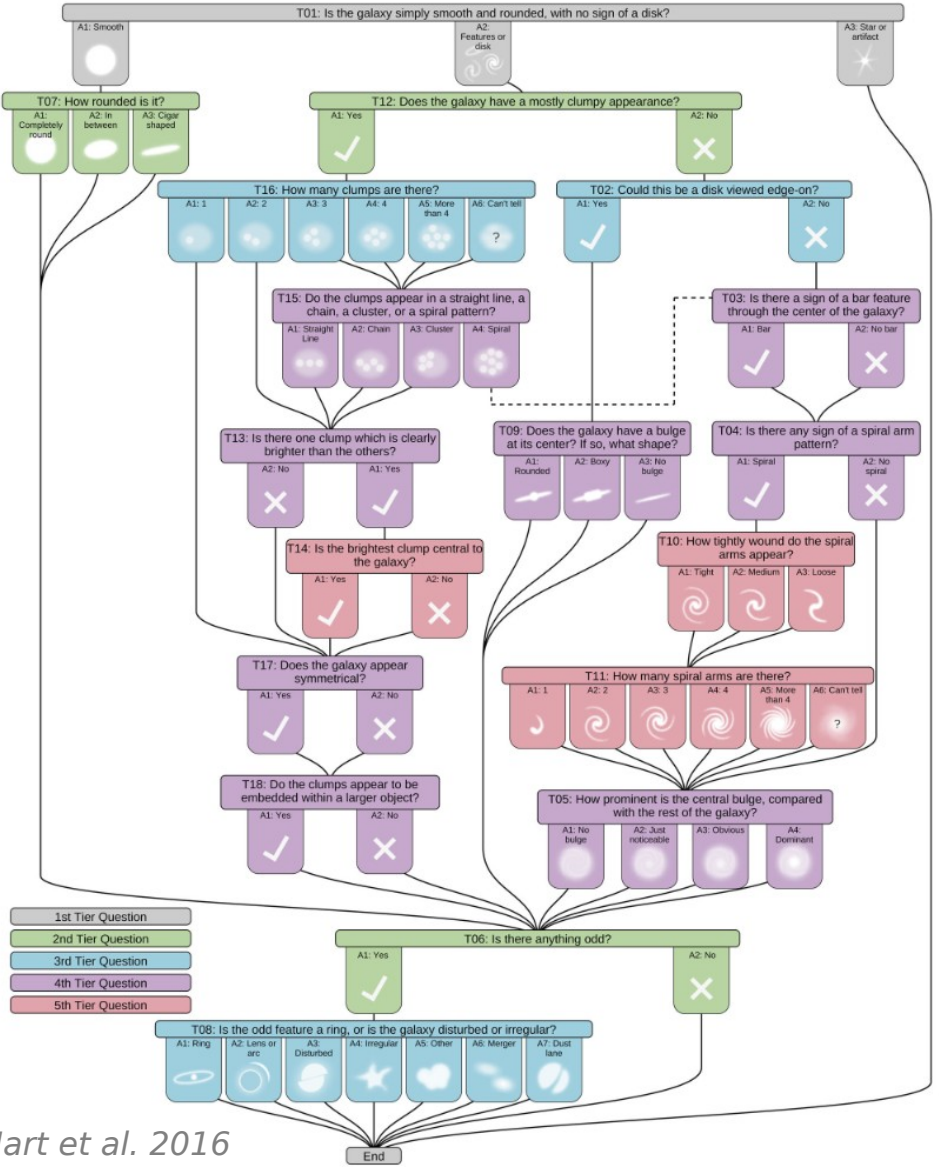


In GZ1 Volunteers classified images of SDSS galaxies as belonging to one of six categories - elliptical, clockwise spiral, anticlockwise spiral, edge-on, star/don't know, or merger. GZ2 extends the original classifications for a subsample of the brightest and largest galaxies, measuring more detailed morphological features. This includes galactic bars, spiral arm and pitch angle, bulges, edge-on galaxies, relative ellipticities, and many others.

There are 243434 images in total, all resized to a 424x424 resolution. Images are composed so the main object is centered and a part of the environment is visible. This implies that the FoV of each image is different. For simplicity, we will use cropped and resized images that are more zoomed in toward the object and resized to a either a 64x64 or a 128x128 image resolution.

Details on the classification process can be found in Hart et al. 2016

GZ2 DL classification



Hart et al. 2016

The catalog contains the debiased answer to each question for every object in the dataset.

Two approaches:

1) Try to reproduce only the E, S, SB classification
  → it is relatively balance
  → low class count
  → Not very representative of the task

2) Try to reproduce the result of all the Txx steps
  → Some classes will have a low example count
  → Output can't be mutually exclusive, logistic output required, but we can re-normalize mutually exclusive answers "a posteriori".
  → more representative of the initial labeling process
  → The debiased probability could be used as a non binary target!

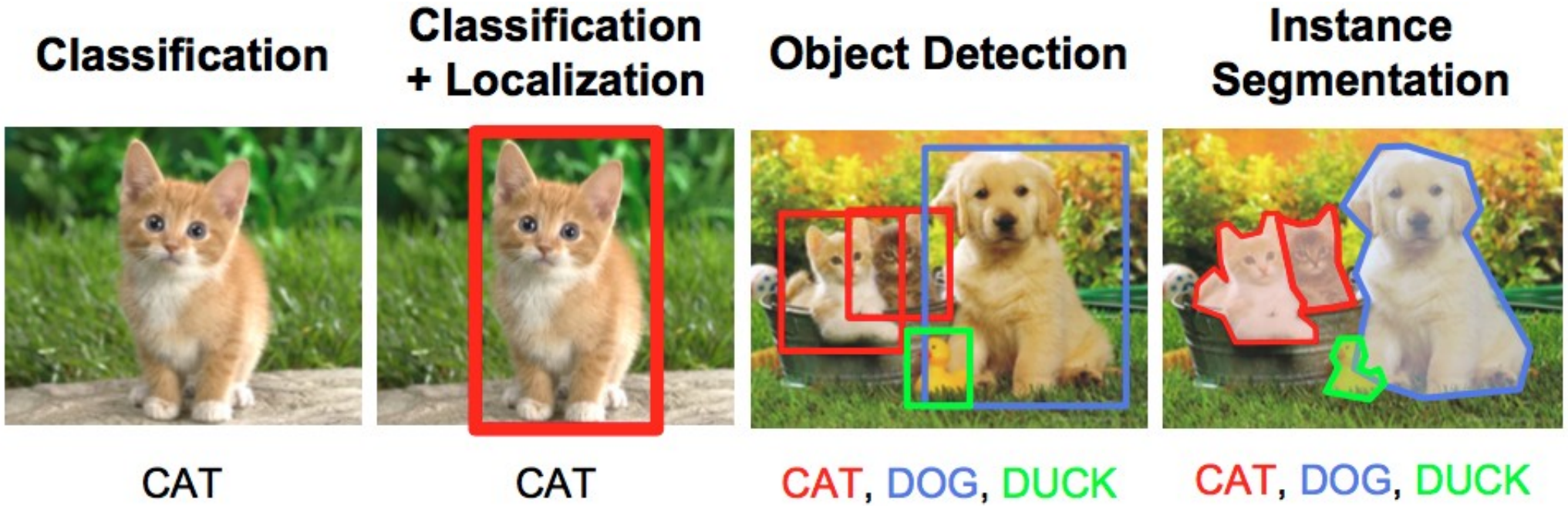
Practical work:

Two notebooks are provided corresponding to both strategies. Try to adapt the ASIRRA architecture to this problem.

Object detection

“Object detection” types

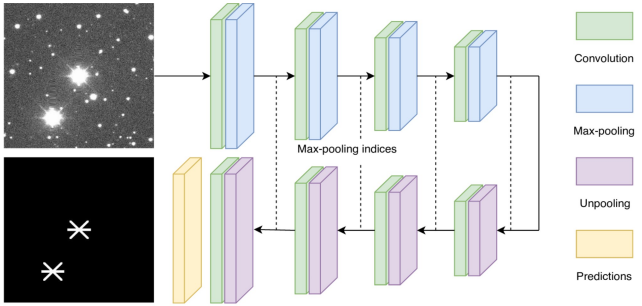
*Image from Stanford Deep Learning course cs224



CNN architectures for object detection

U-Nets

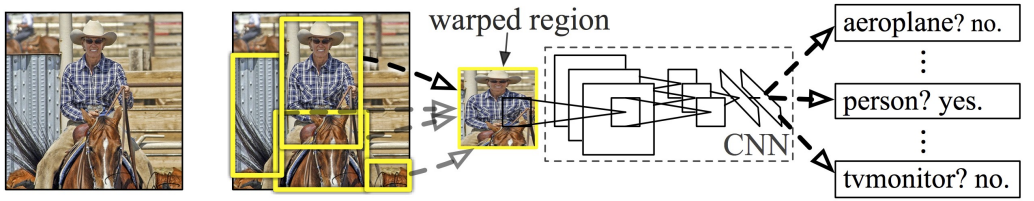
Pros: segmentation maps, shallow latent space, ...



Region-based

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

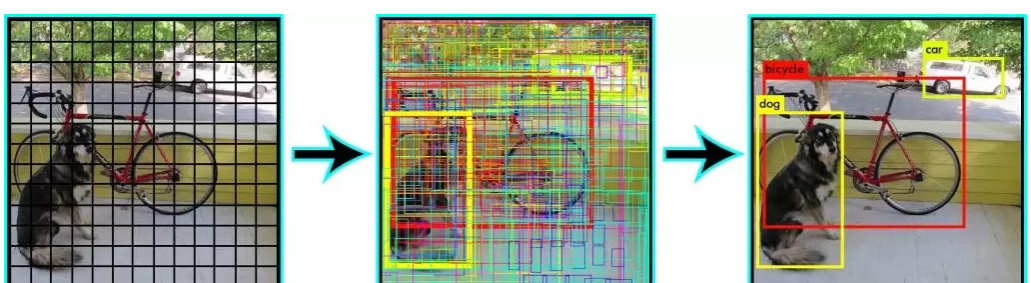
Pros: Best accuracy, ...



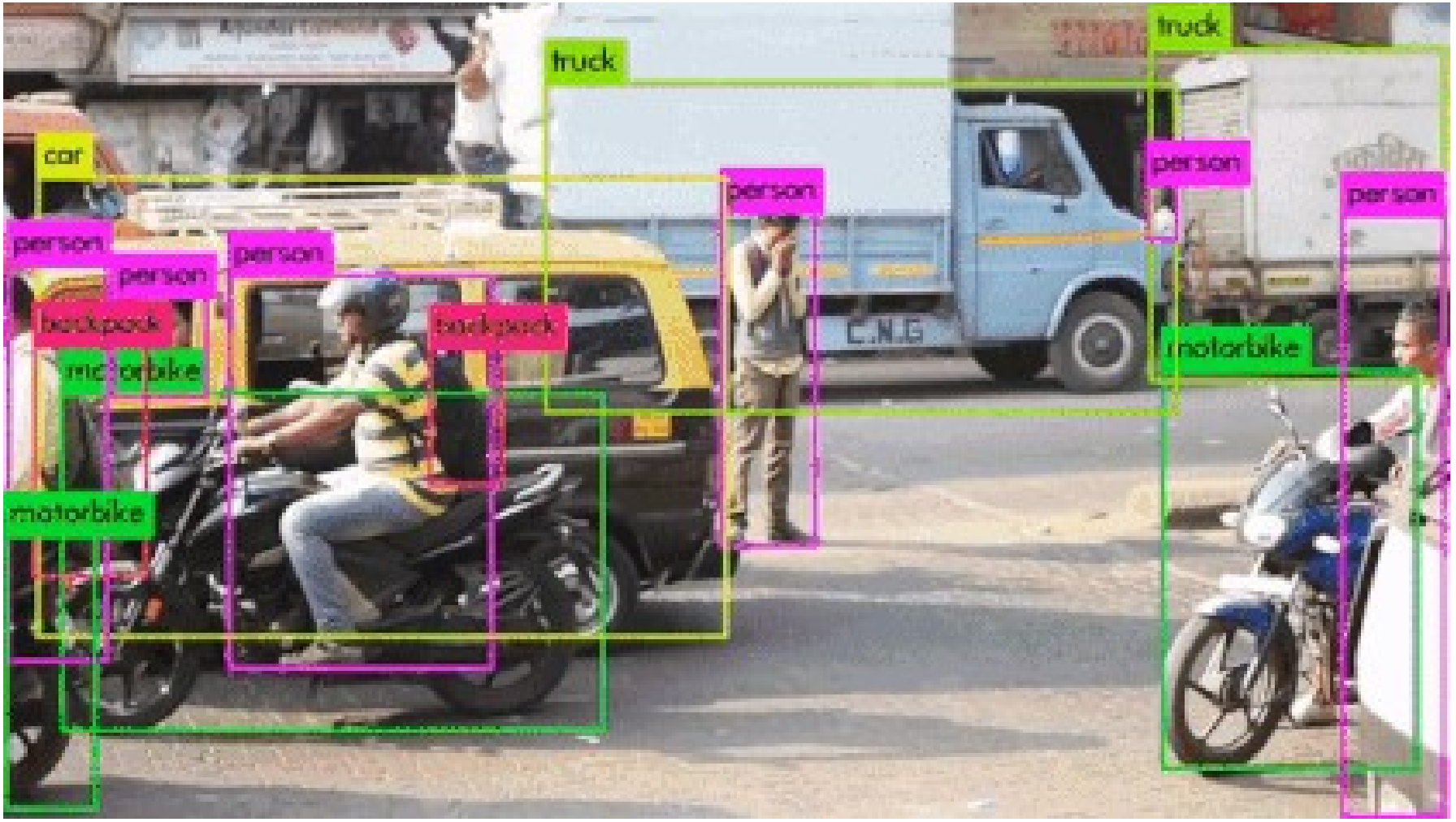
Regression-based

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

Pros: Very Fast, straightforward architecture,...



Real time 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” :

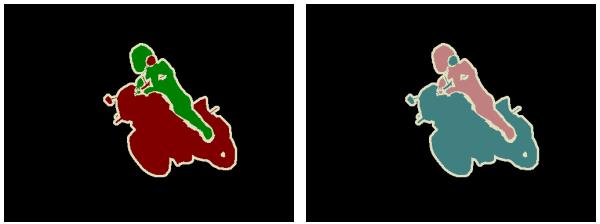
Classification

[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)

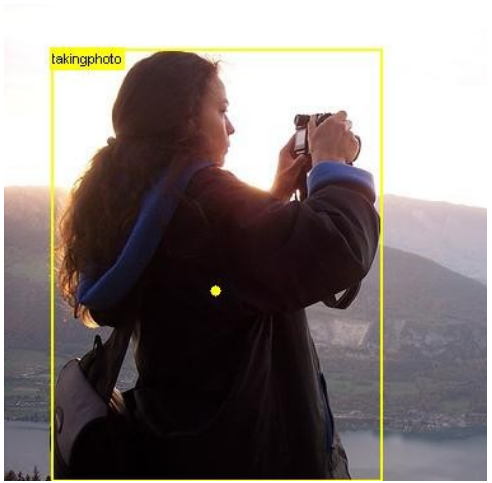
Standardized image datasets for object class recognition organized in the form of annual challenges from 2005 to 2012

PASCAL2
Pattern Analysis, Statistical Modelling and Computational Learning

Targets for various “tasks” :

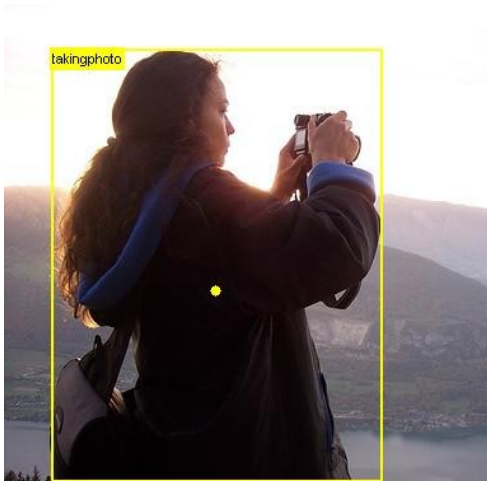
http://host.robots.ox.ac.uk/pascal/VOC/index.html

Boxless Action



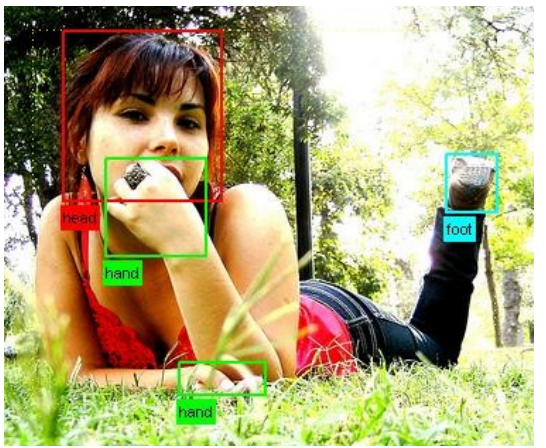
A single reference point and the associated action

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

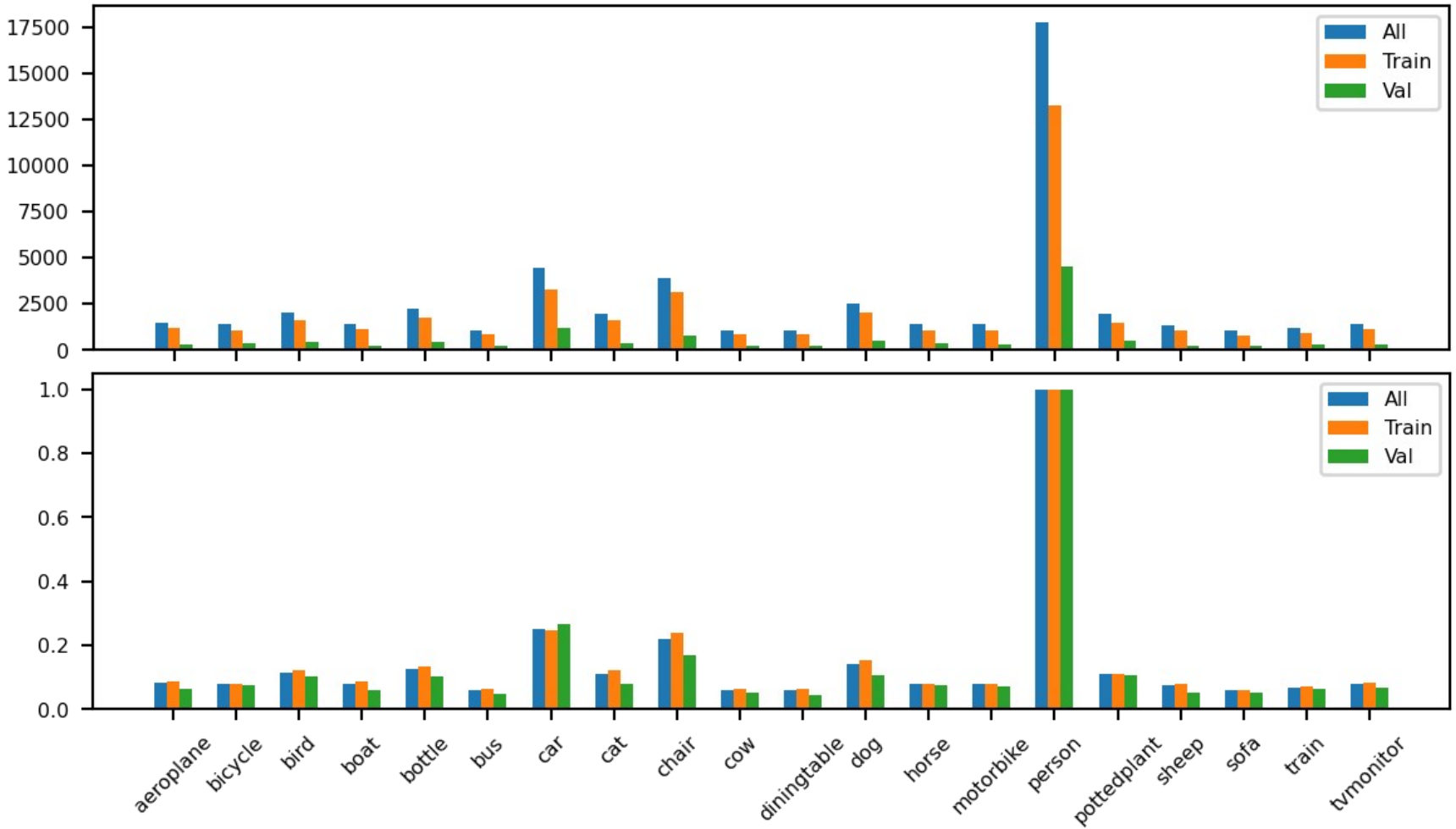
VOC - 2012 : incremental from 2008 to 2012
11540 Train annotated images (test set hidden), detection task with 20 classes

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

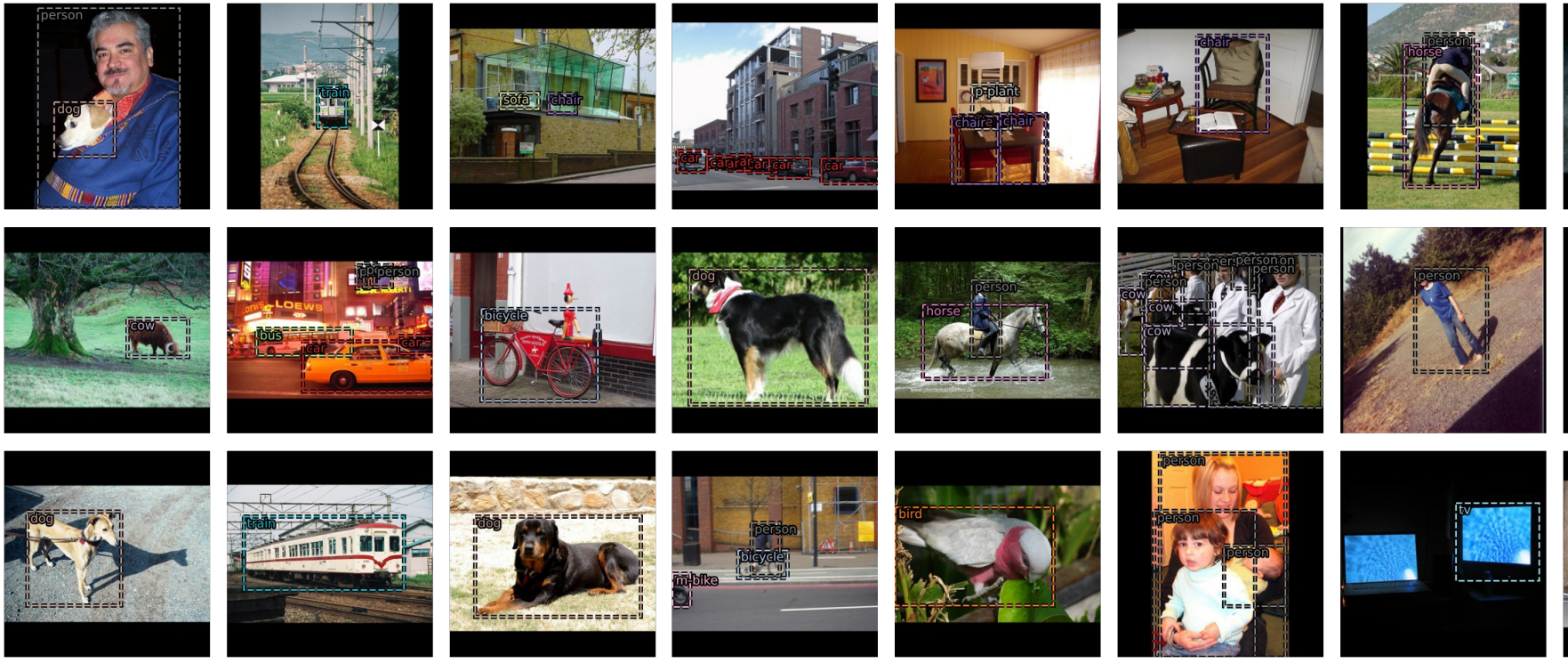
Training dataset summary statistics

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.



Simple classifier using object cutouts

We define a classifier with an **input dim. of 128x128**, and an **output dim. of 20** (number of classes) using **Softmax** activation. The 4952 Test 2007 images are kept untouched for validation.

The training images are dynamically generated using the Albumentations 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, color, 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:

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 :

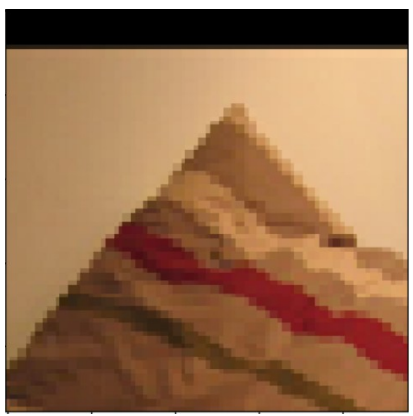
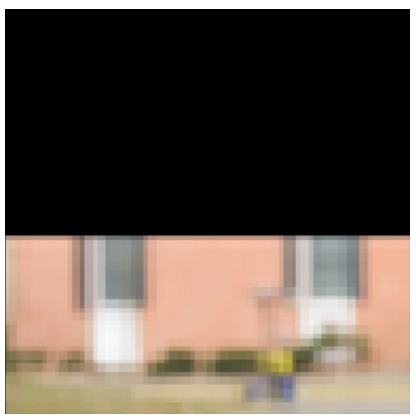
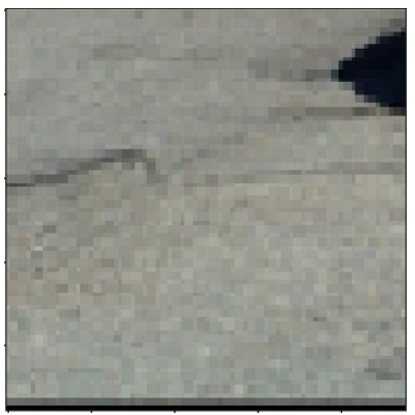
**********									ConfMat	**********											Recall
93	0	3	3	0	0	1	0	0	0	1	0	0	0	3	0	0	0	3	0		86.92%
0	133	0	1	1	1	0	0	4	0	0	0	0	8	8	0	0	0	0	0		85.26%
3	1	154	1	3	0	4	6	1	5	0	3	5	2	12	3	2	0	1	0		74.76%
4	2	2	92	0	0	2	0	0	0	1	0	2	1	1	1	0	0	4	0		82.14%
0	0	0	1	99	0	1	0	4	0	0	4	0	2	22	3	0	1	0	1		71.74%
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%

Adding a background class

To be used as a detector our classifier must be able to predict an empty case

- 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 too 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 ~94% recall on background examples

Using the new classifier as a sliding window detector

The sliding window principle

Classifier = identify individual objects relatively centered

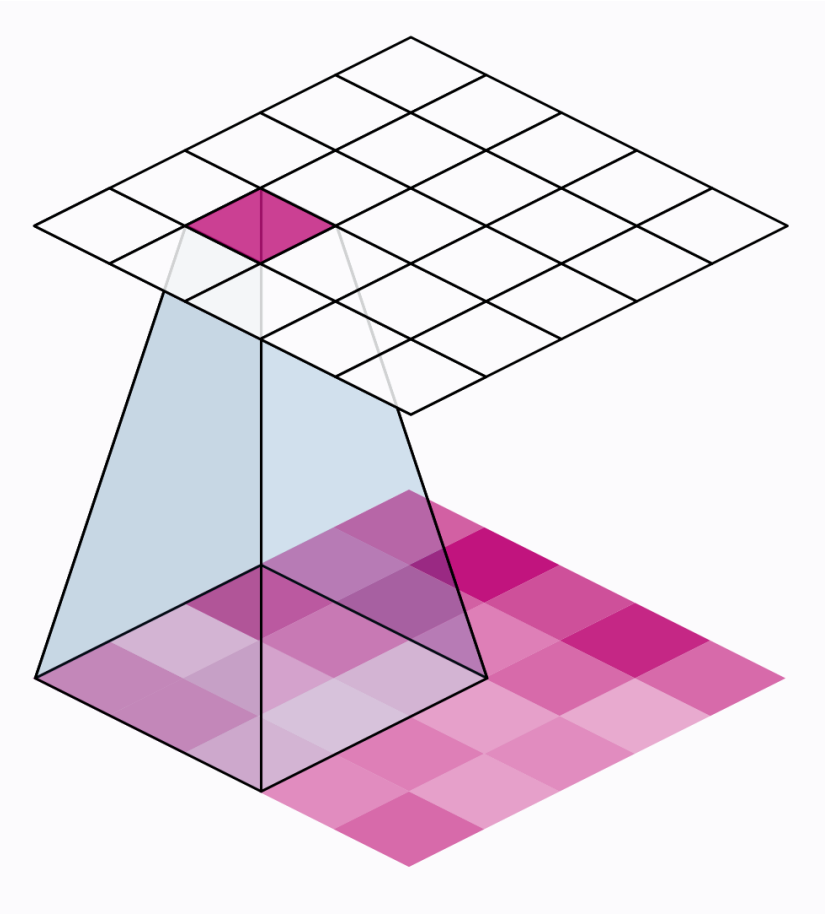
Can be used as a sliding window detector:

→ Split a larger image in “chunks”

→ Overlapping Chunks for more « centered » objects

Process similar to a convolution, with Classifier ≈ Filter

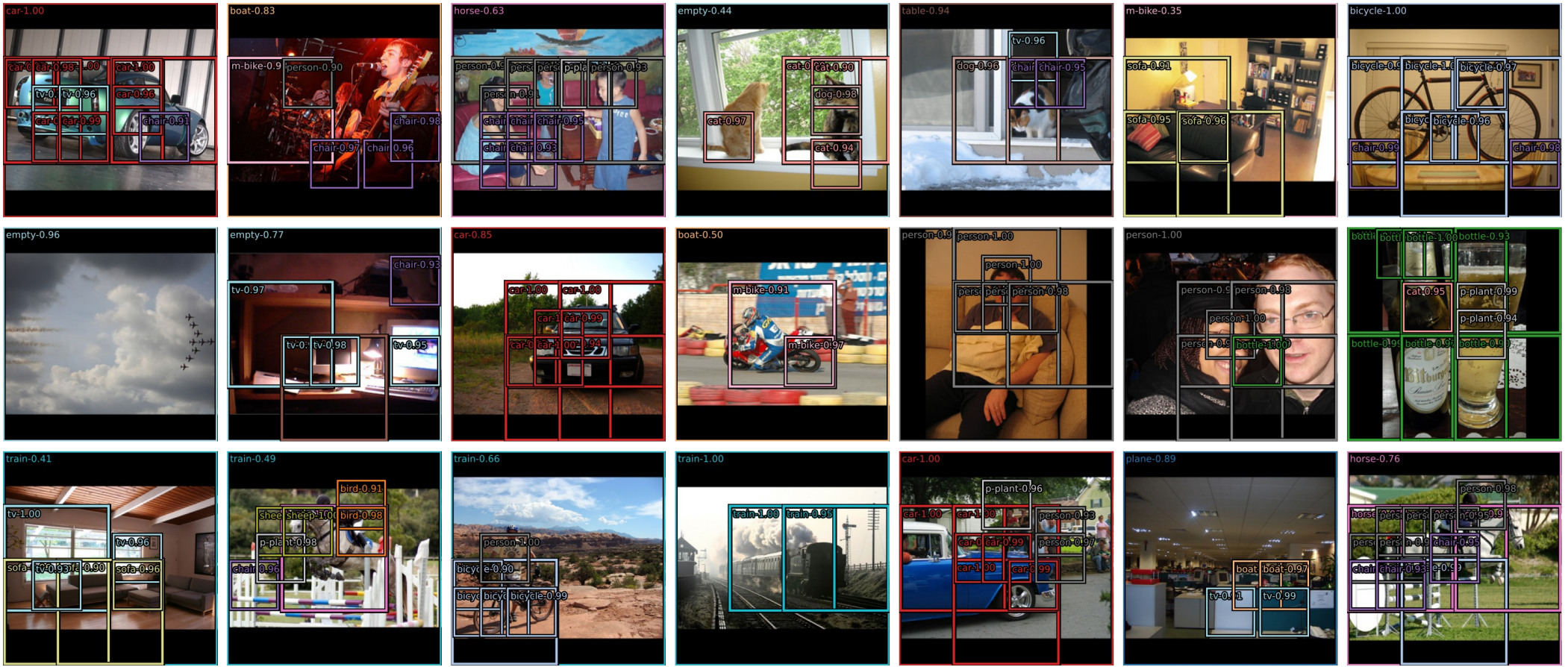
→ Using multiple window sizes increases the chance of a proper cutout for objects of various sizes



*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

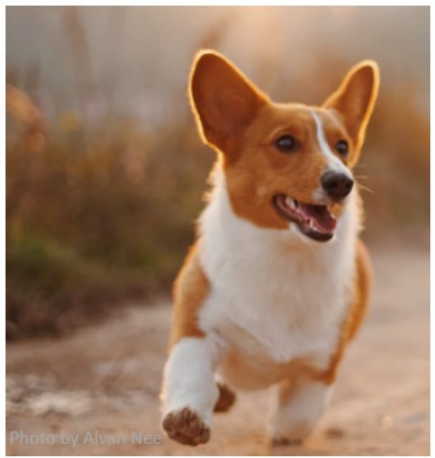


Questions : How to filter overlapping detection ? How to evaluate prediction performance ?

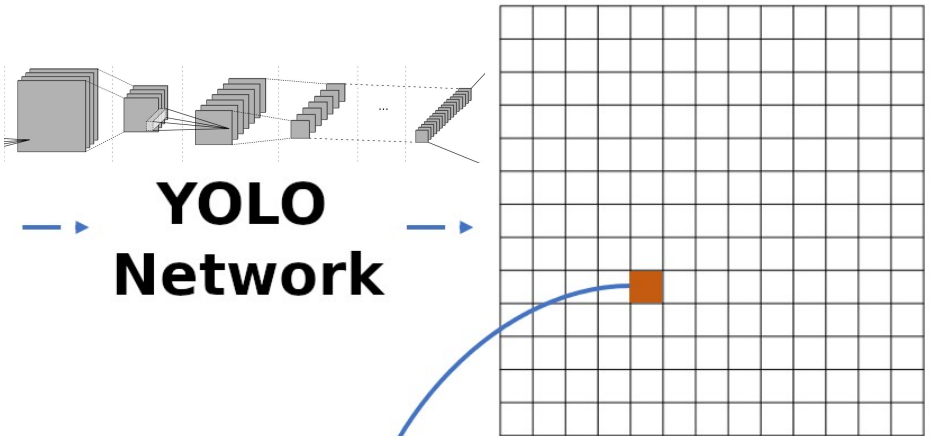
The You Only Look Once (YOLO) detector

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

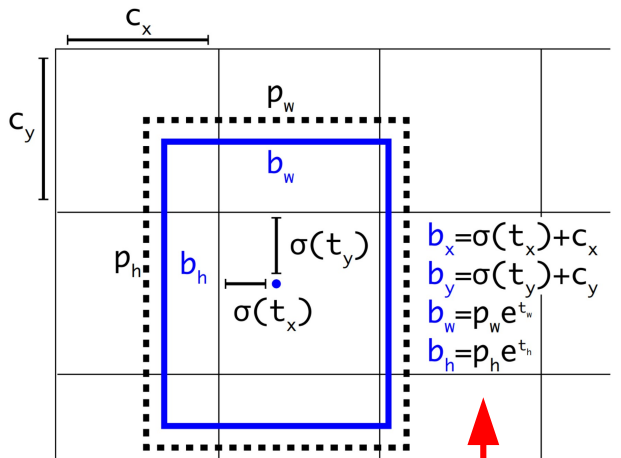
**Images from blog post and Redmon papers*

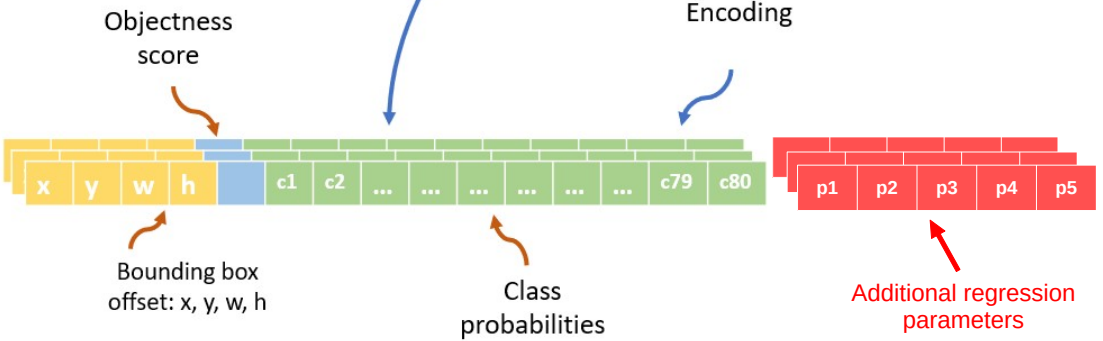


Pre-processing Image



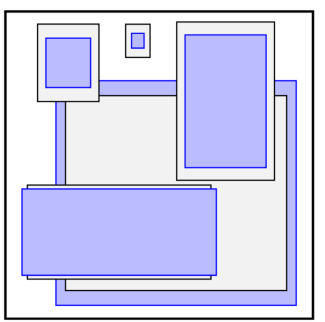
Encoding





p_w
p_h

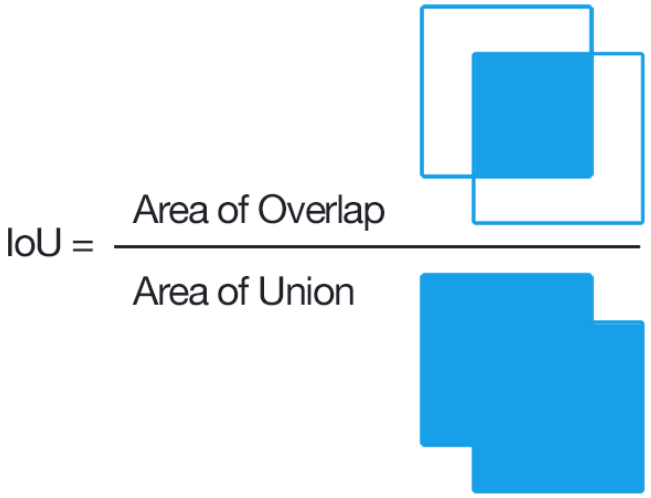
Box size priors

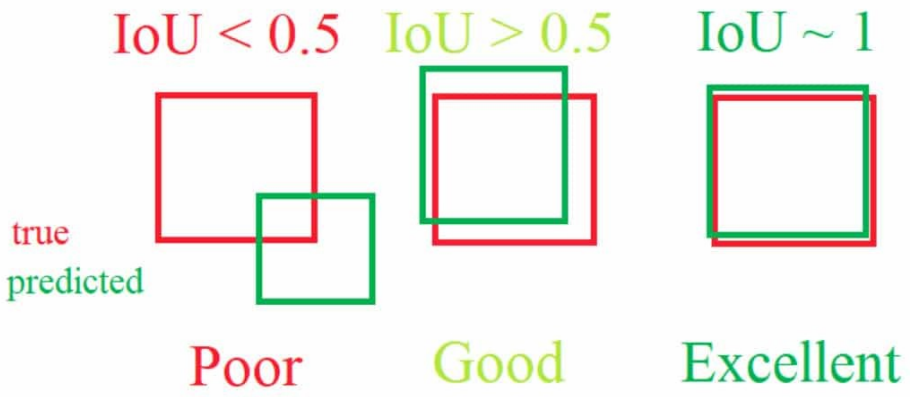


The last layer is conv. = the boxes « share » weights spatially.

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

How to estimate box « correctness » ?

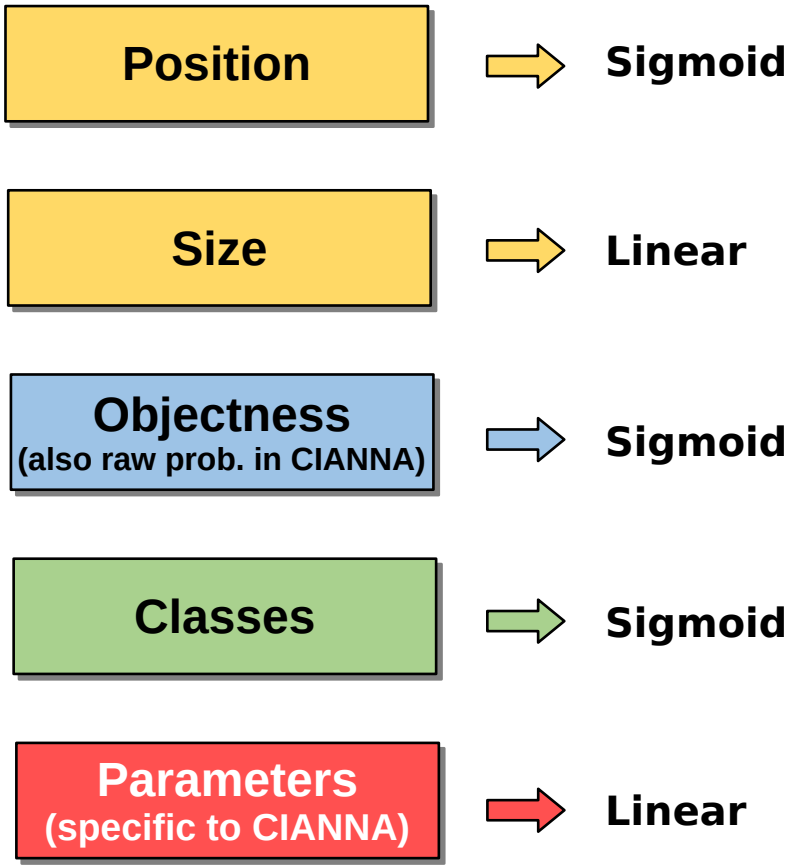




For each box the network will predict an “objectness” score defined as:

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

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

The YOLO activation / Loss function

$$
\begin{aligned}
\mathcal{L} = \sum_{i=0}^{N_g}\sum_{j=0}^{N_b} \mathbb{1}_{ij}^{match} \Big(& \lambda_{pos} && \left[(o_{ij}^{x} - \hat{o}_{ij}^{x})^2 + (o_{ij}^{y} - \hat{o}_{ij}^{y})^2\right] \\
& +\lambda_{size} && \left[(o_{ij}^{w} - \hat{o}_{ij}^{h})^2 + (o_{ij}^{h} - \hat{o}_{ij}^{h})^2\right] \\
& +\lambda_{class} && \mathbb{1}_{ij}^{C\_good} \sum_{k}^{N_C} \left(-\hat{C}_{ij}^{k} \log(C_{ij}^{k})\right) \\
& +\lambda_{param} && \mathbb{1}_{ij}^{p\_good} \sum_{k}^{N_p} \gamma^{k} \left(p_{ij}^{k} - \hat{p}_{ij}^{k}\right)^2 \\
& +\lambda_{prob} && \mathbb{1}_{ij}^{P\_good} \left(P_{ij} - 1\right)^2 \\
& +\lambda_{obj} && \mathbb{1}_{ij}^{O\_good} \left(O_{ij} - \mathrm{fIoU}_{ij}\right)^2 \Big)
\end{aligned}
$$

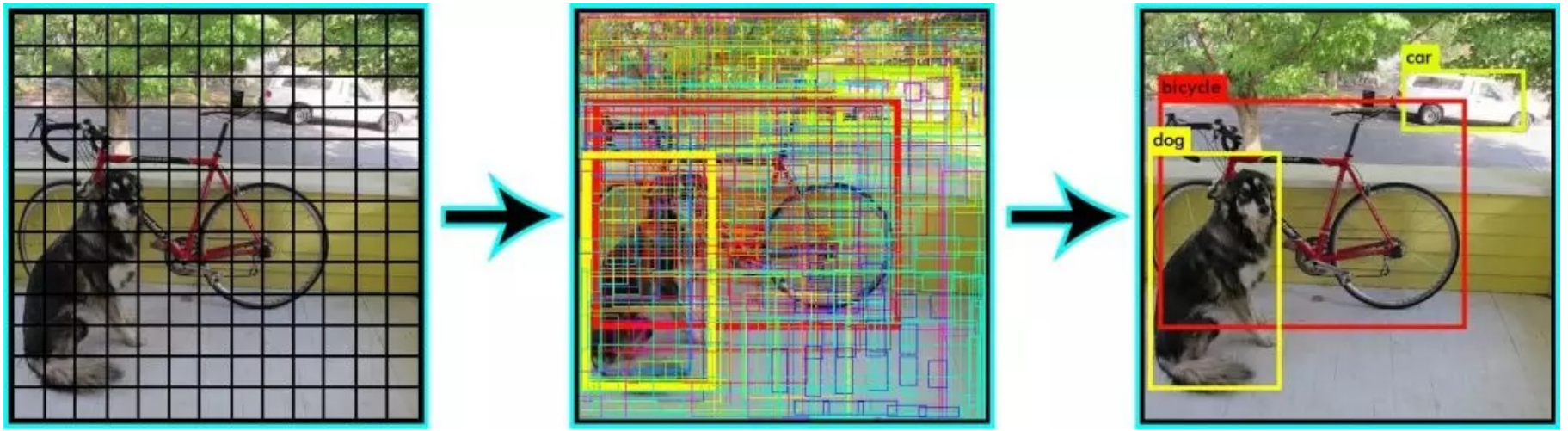
Only for the best box corresponding to each target

$$
+ \sum_{i=0}^{N_g}\sum_{j=0}^{N_b} \mathbb{1}_{ij}^{void} \lambda_{void}^{j} \left(\lambda_{prob} \left(P_{ij} - 0\right)^2 + \lambda_{obj} \left(O_{ij} - 0\right)^2\right). \qquad (14)
$$

Only for background predicted boxes

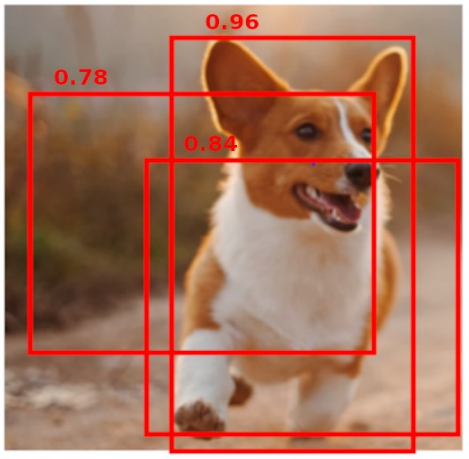
Post-processing: Non Max Suppression

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.



Post-processing: Non Max Suppression

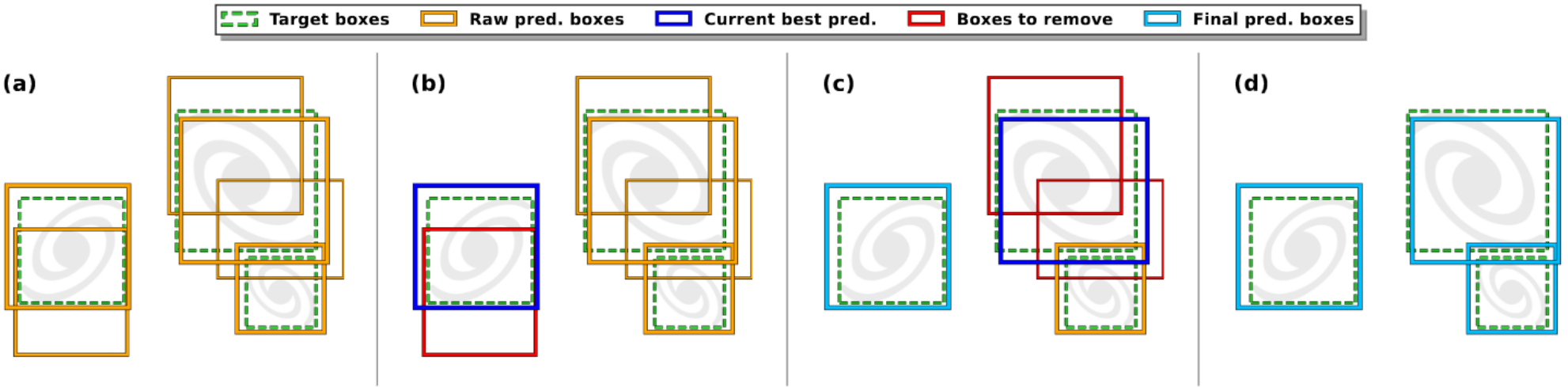
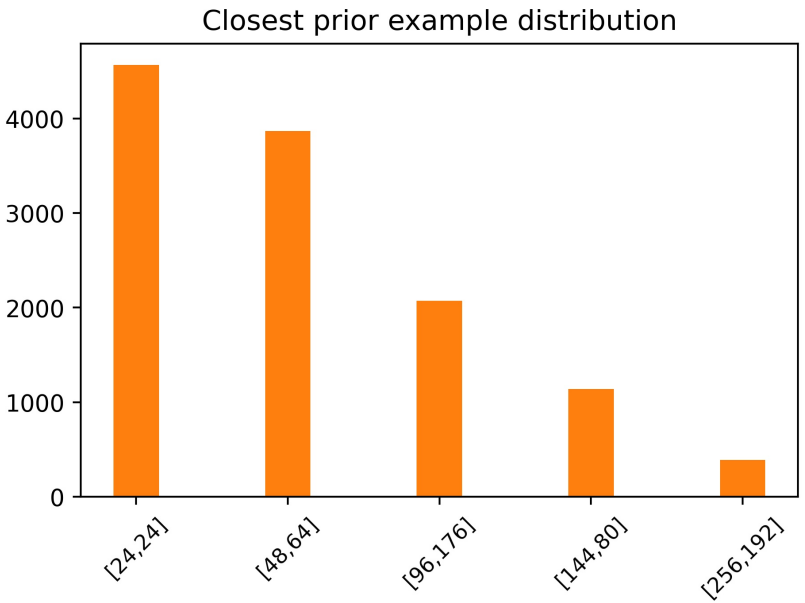


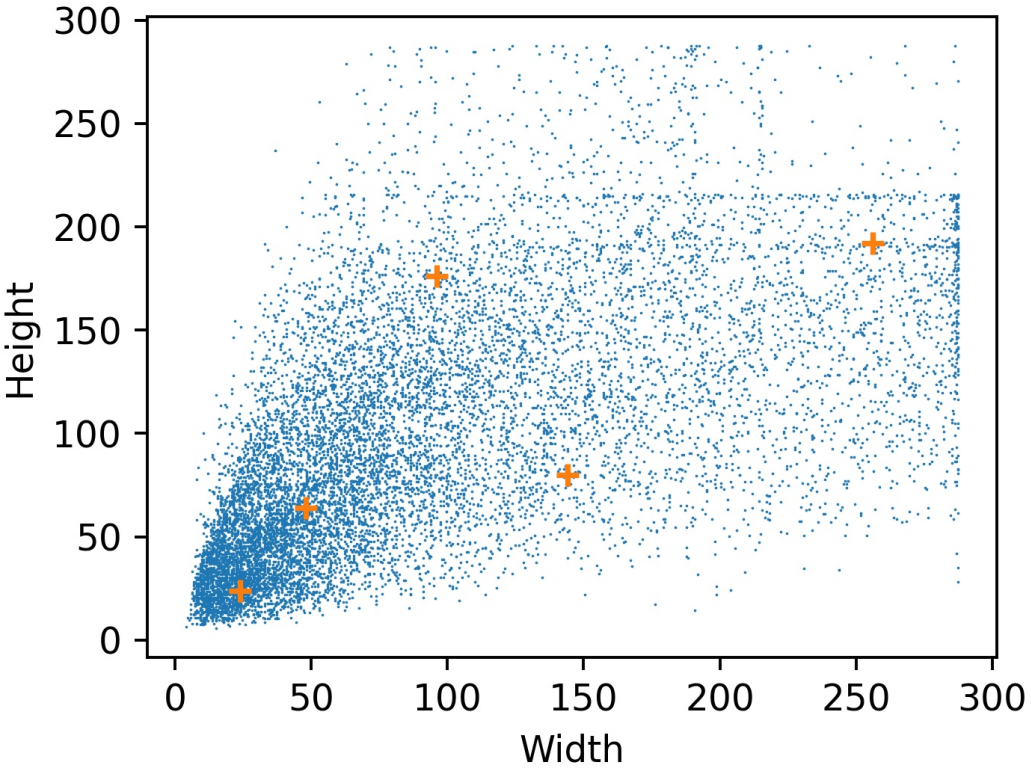
Fig. 5: Illustration of the NMS process in a given image. The dashed boxes are the target, and the solid boxes are the predictions. The line widths of the boxes are scaled on their respective objectness score. The colors indicate the state of the box in the NMS process at different steps. Frame (a) represents the targets and the raw network predictions that remain after the objectness filtering. Frames (b) and (c) represent two successive steps of the NMS process with a different best current box. Frame (d) represents the remaining boxes after completion of the NMS.

From Cornu et al. 2024

Training a YOLO network on PASCAL VOC

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

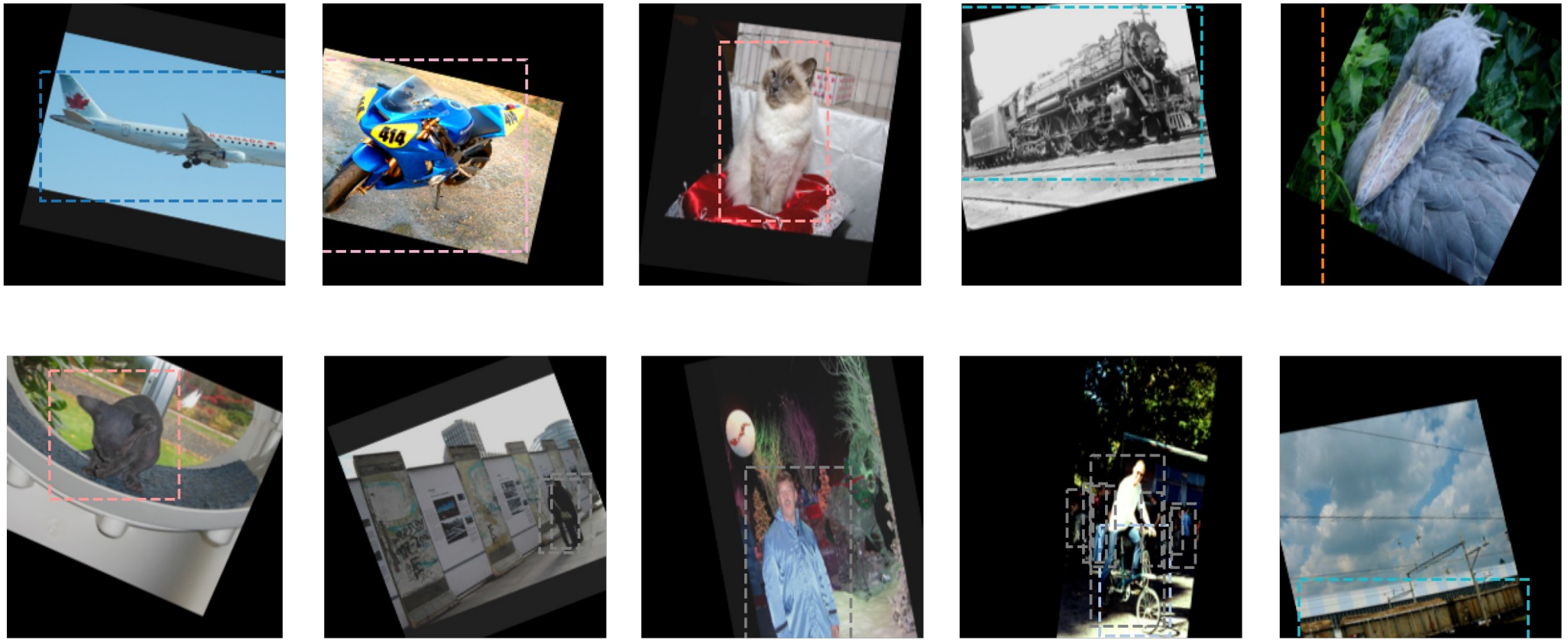
Suggested network architecture *(also the one of the provided pre-trained network)*:

```
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")
```

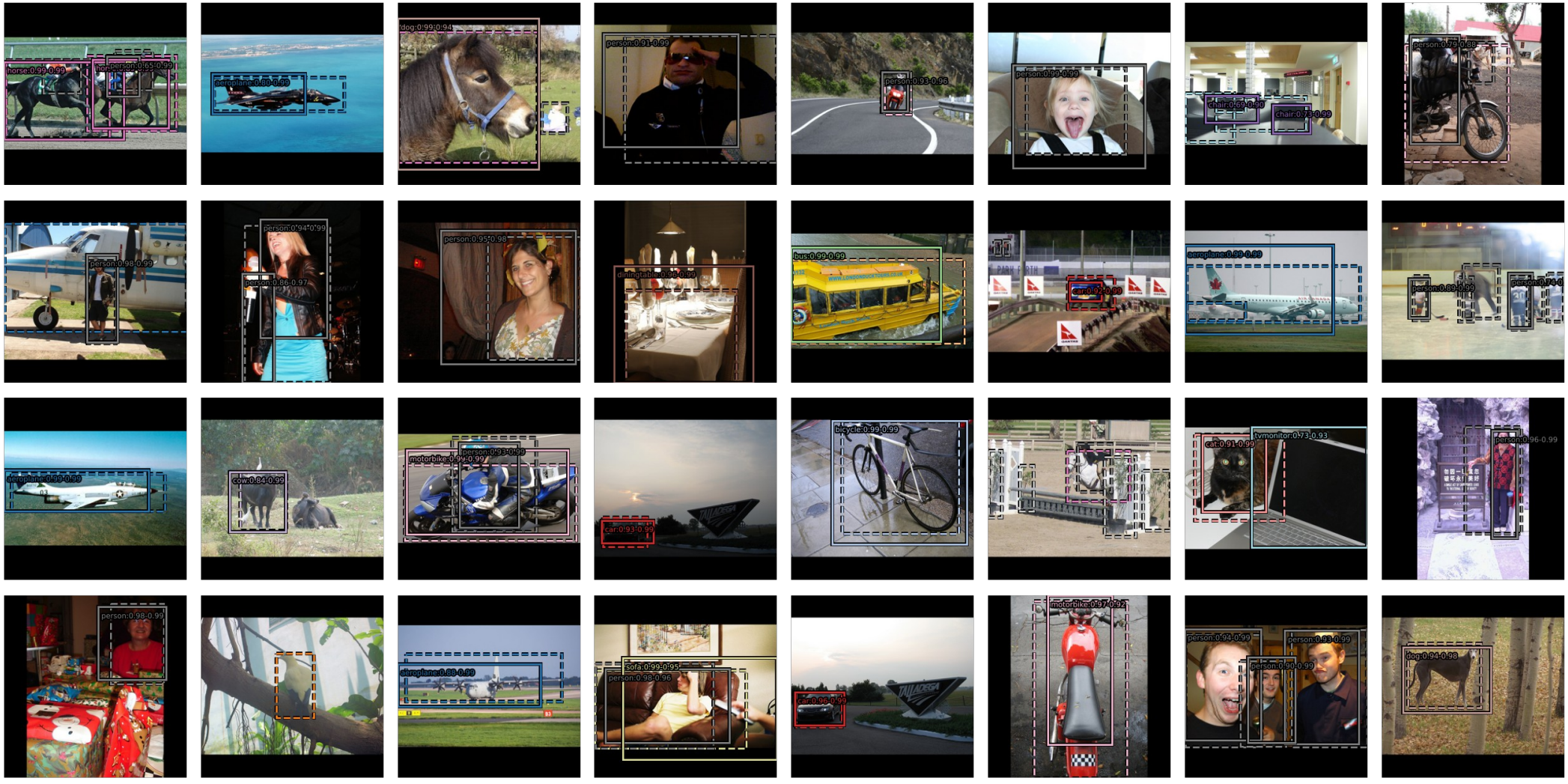
Inspired by the YOLO-Light V2 and Tiny-Yolo V2 architectures

C – Training a YOLO network on PASCAL VOC

Advanced data augmentation



C – Training a YOLO network on PASCAL VOC

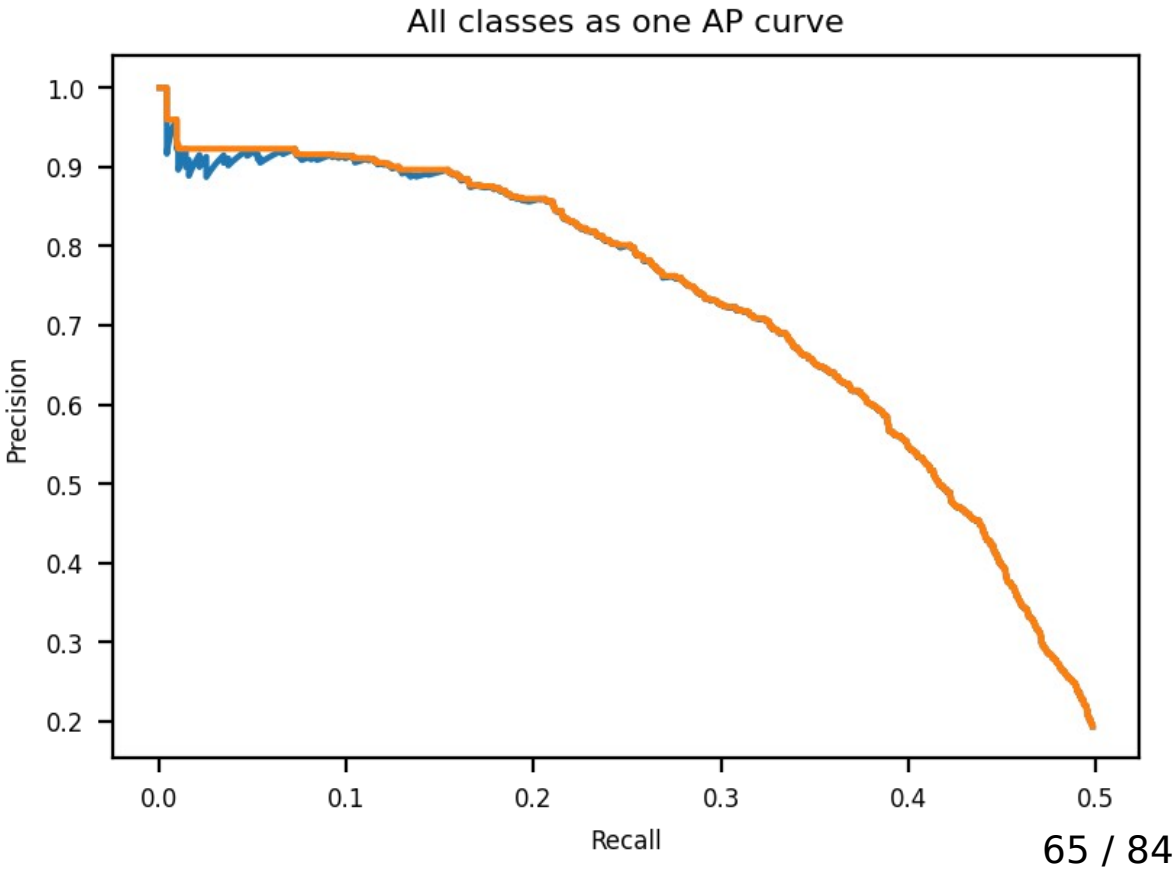


Detection performance metric

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

The precision-recall curve

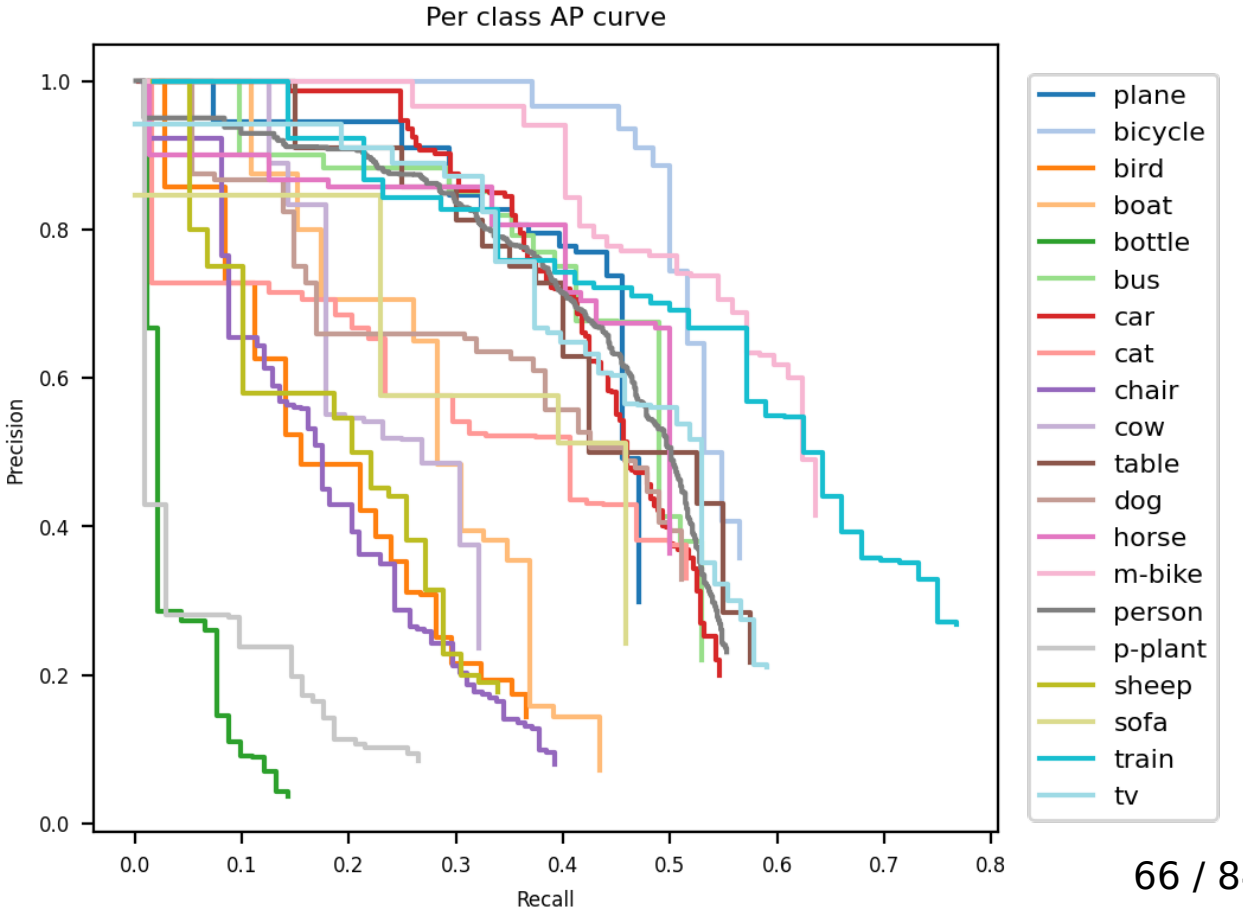
- Predictions are sorted by score
- For each data point, compute the precision and recall using all the predictions that are scored higher, using **IoU@0.5** and correct classification as “True” criteria
- The curve is then smoothed so it can only monotonically decrease (precision is always equal to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

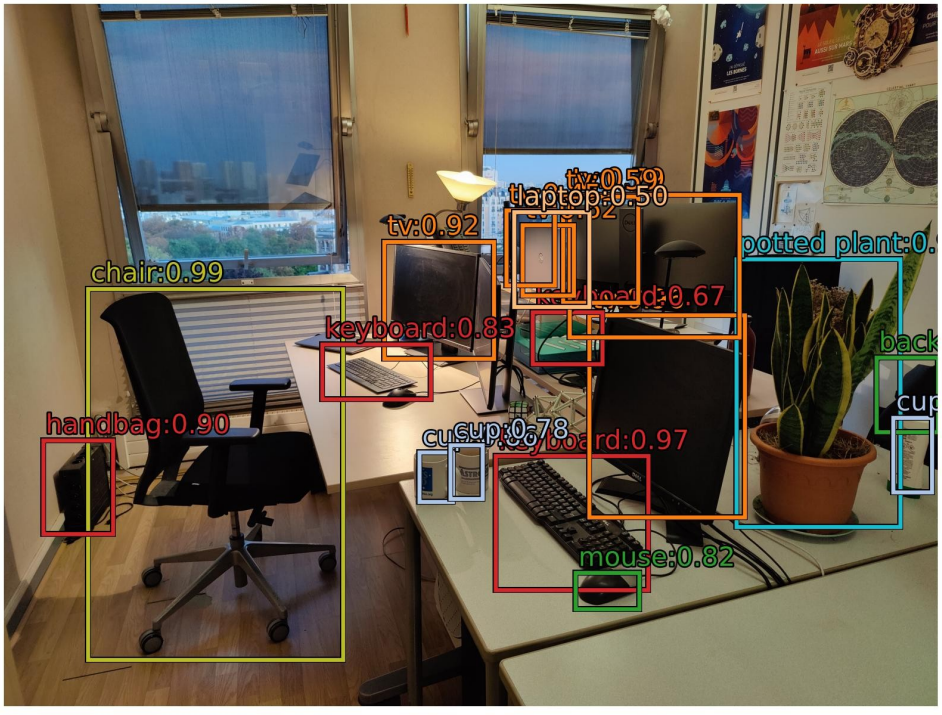


Warning: definition of AP and mAP might change depending on the challenge or dataset

More advanced YOLO detection

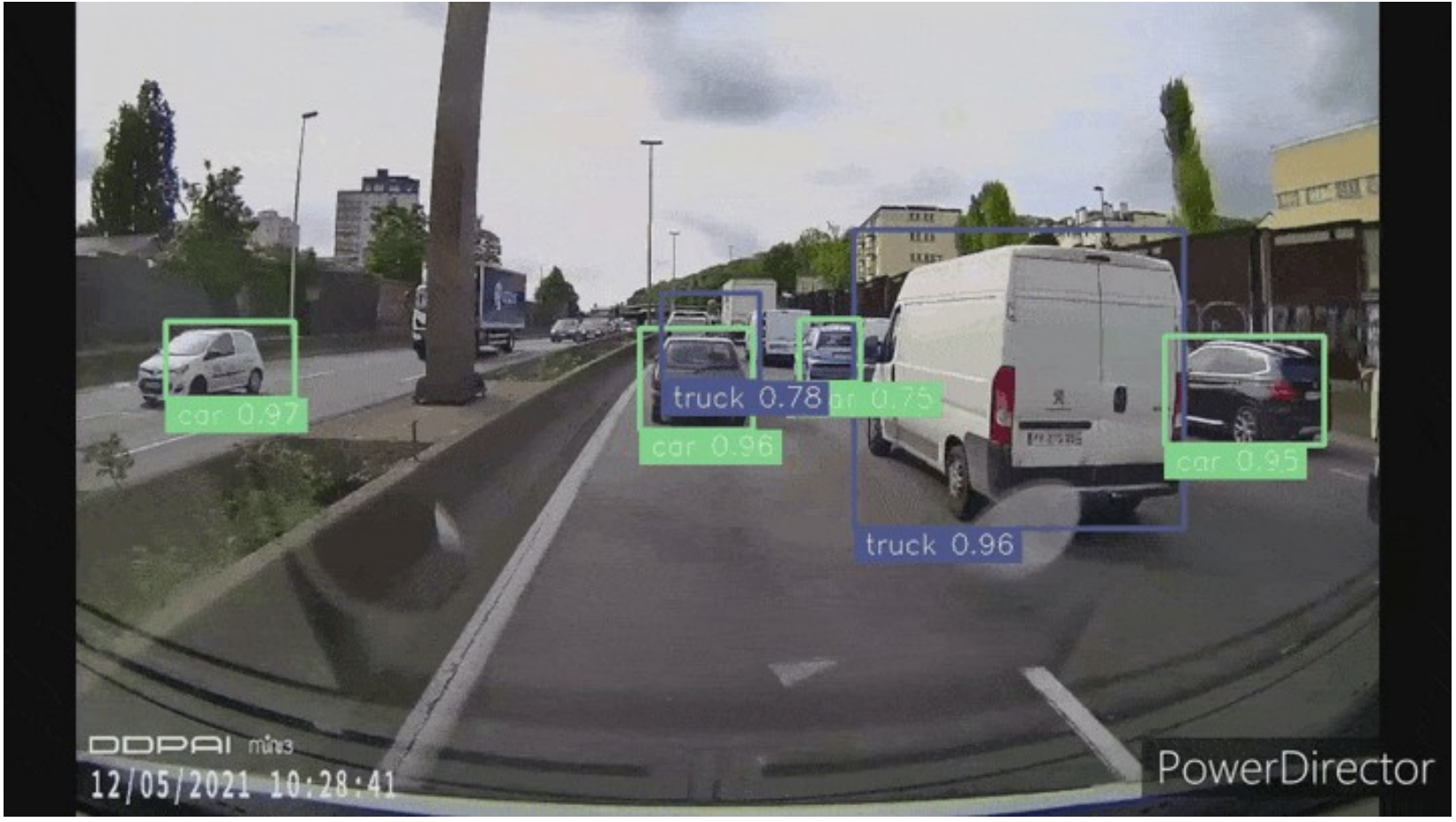
In the CIANNA git repository (https://github.com/Deyht/CIANNA), we provide several detection models using the custom YOLO-CIANNA method with backbone very similar to the darknet-19 architecture and applied to several datasets.

The model is pre-trained on ImageNet and then trained on the PASCAL and COCO datasets independently.



Results from a YOLO network trained on COCO

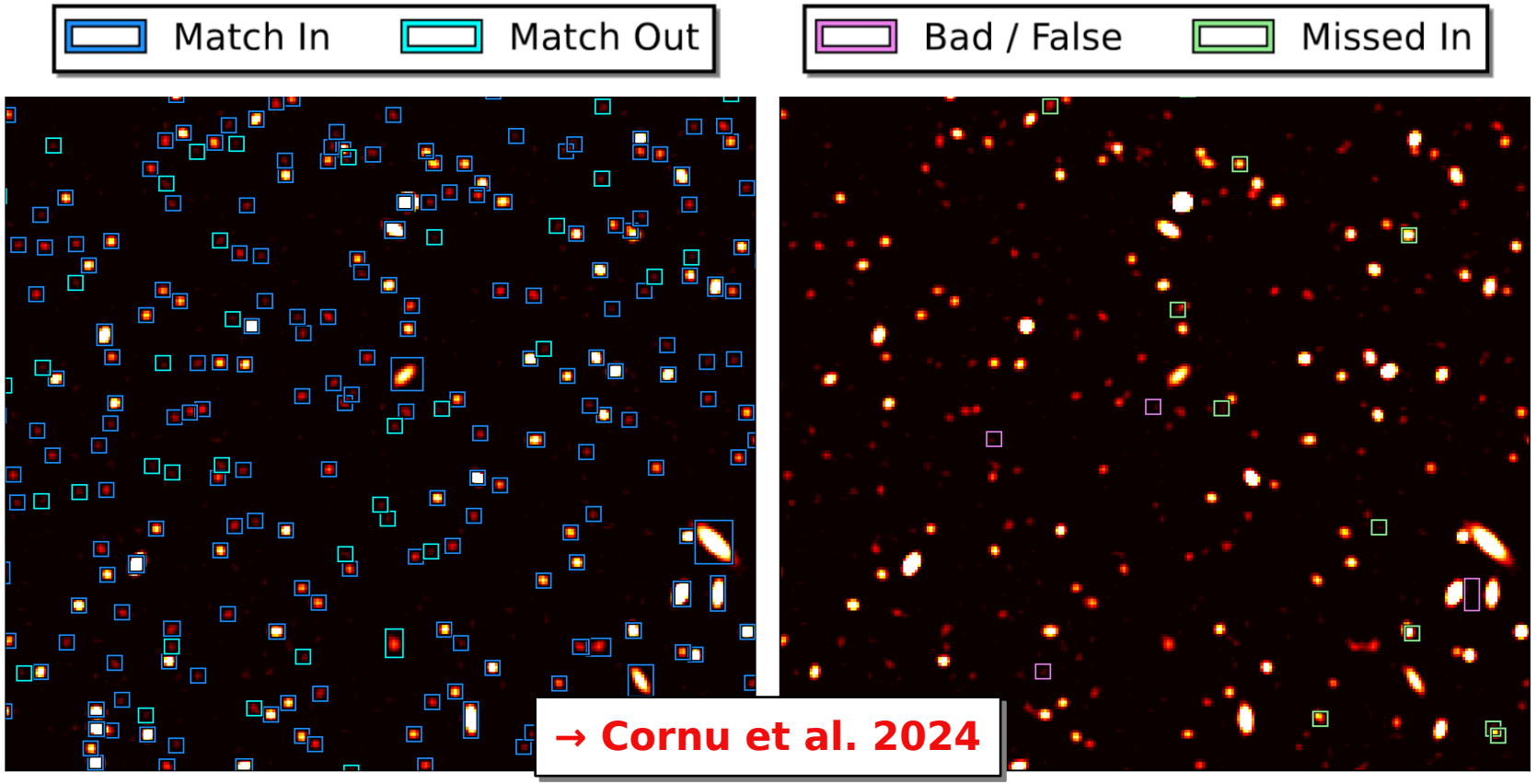
Result using a YOLO detector with a darknet-19 “like” backbone. 40.1 mAP over the 1000 classes, at a 416p resolution. The network run at 690 ips on an RTX 4090.



YOLO-CIANNA for galaxy detection - SKAO SDC1

Simulated continuum observation at 560 MHz for 1000h integration time with SKA

Objective: detect (Ra, Dec) and characterize (Flux, Bmaj, Bmin, PA, ...) the sources

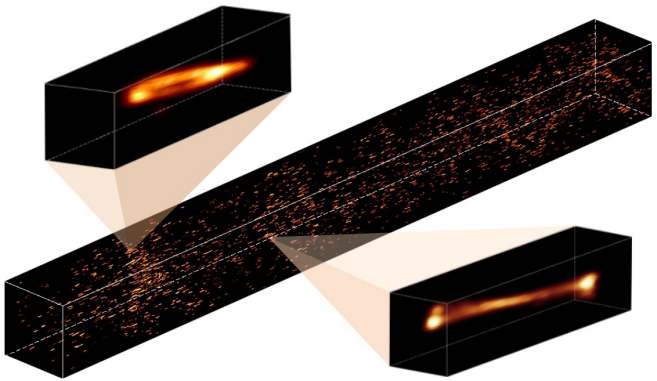


Fast (~130 Mpix/s - RTX 4090) and light network (17 layers, 13M param)

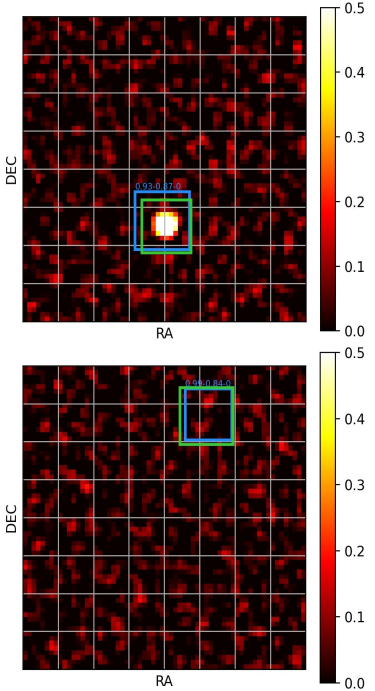
YOLO-CIANNA for galaxy detection - SKAO SDC2

Simulated HI cube for 2000h integration time with SKA

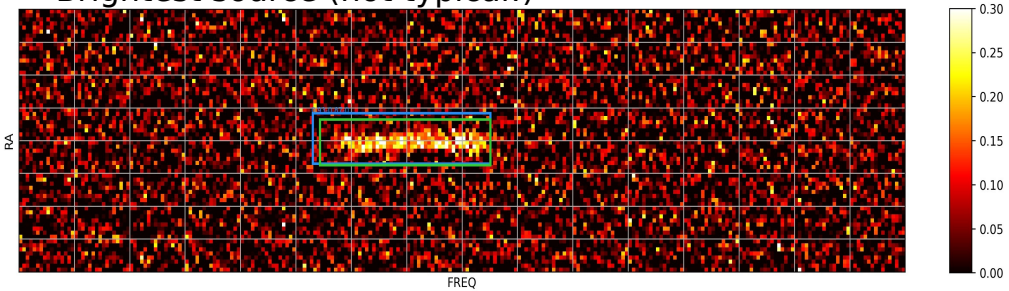
Objective: detect (Ra, Dec, Freq) and characterize (Line flux, HI size, PA, I) the sources



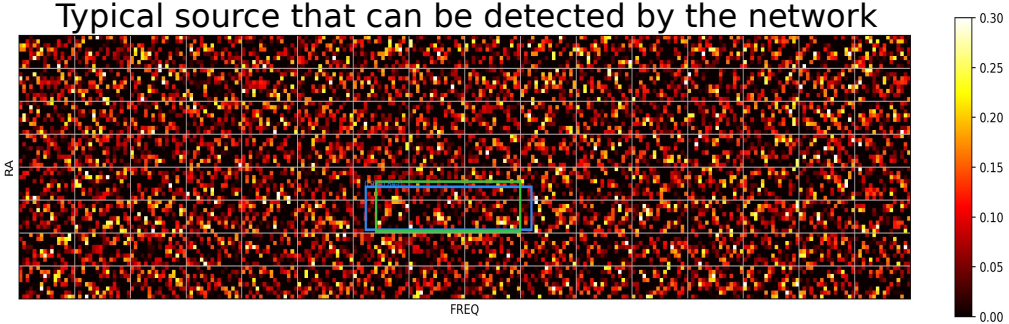
True boxes vs Predicted boxes



Brightest source (not typical!)



Typical source that can be detected by the network



*averaged over 20 channels in FREQ and 20 pixels in DEC respectively

→ Harley et al. 2023
→ Cornu et al. In prep

Fully 3D, fast (~120 Mpix/s - RTX 4090) and light network (23 layers, 4M param)