# Training : Computer Vision

## Hackathon Ecole des Ponts

06/09/2022

École des Ponts
ParisTech

eleven
strategy • data • digital

AGENDA

1. Computer vision approaches
2. An overview of the main Deep Learning algorithms
3. The “open” dataset : our gold mine
4. Measure the performance to improve your algorithm
5. Resources to help you start

# The amount of data available is a critical element to consider in an image-based application

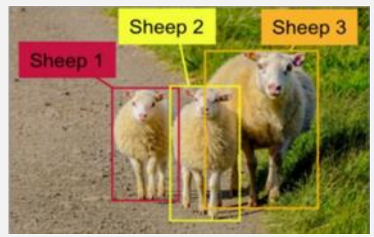
Little data

Lots of data

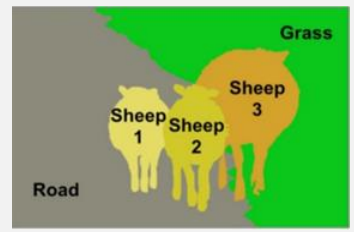
1. More hard engineering (“hacks”)
2. More ad-hoc algorithm
3. Transfer learning

Image recognition


Object detection




Object segmentation




1. More standard algorithms
2. Less hand engineering

**Sources of knowledge**

1. Labeled data (e.g. image → dog & cat)
2. Hand engineered features, network architecture, etc.

# Computer vision can be broken down into a multitude of blocks that can be combined as needed (1/2)

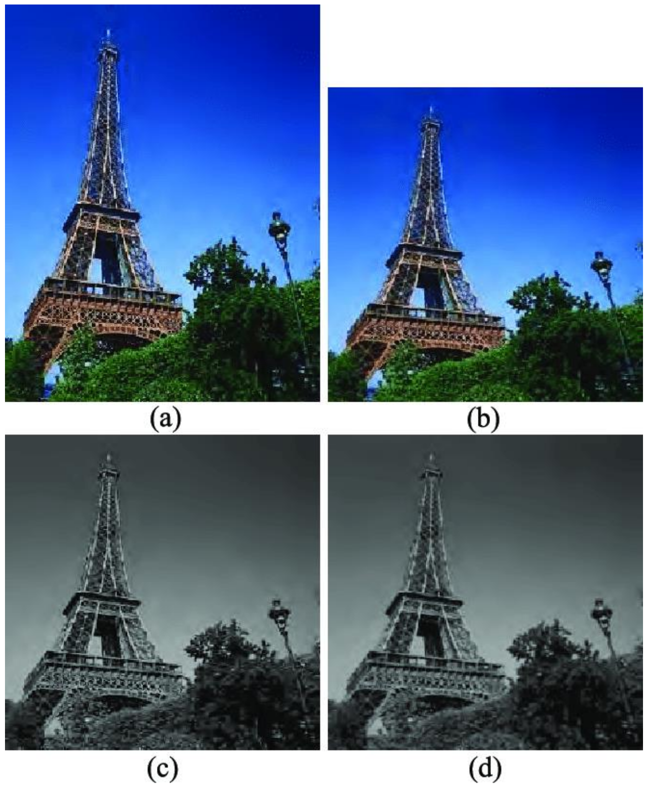
**Image Pre-processing**

Image scaling

Contrast enhancement

Cropping and padding

Noise reduction




(a)Original image

(b)Resized

(c)Grayed

(d)Blured

# Computer vision can be broken down into a multitude of blocks that can be combined as needed (1/2)

**Image Pre-processing**

Image scaling

Contrast enhancement

Cropping and padding

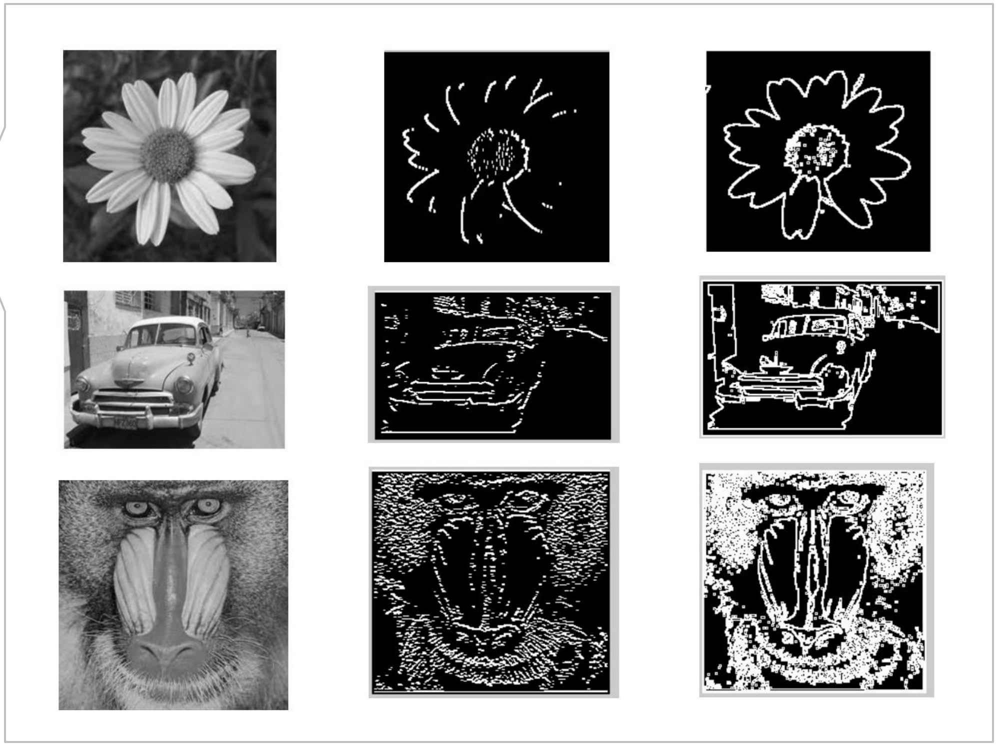
Noise reduction

**Feature extraction**

Lines/Edge detection

Corners/Blobs/Points

(SIFT/SURF, PCA, Watershed)



# Computer vision can be broken down into a multitude of blocks that can be combined as needed (1/2)

**Image Pre-processing**

Image scaling

Contrast enhancement

Cropping and padding

Noise reduction

**Feature extraction**

Lines/Edge detection

Corners/Blobs/Points

(SIFT/SURF, PCA, Watershed)

**Classification algorithm**

CNN

Cat

Computer vision can be broken down into a multitude of blocks that can be combined as needed (1/2)

## Image Pre-processing

Image scaling

Contrast enhancement

Cropping and padding

Noise reduction

## Feature extraction

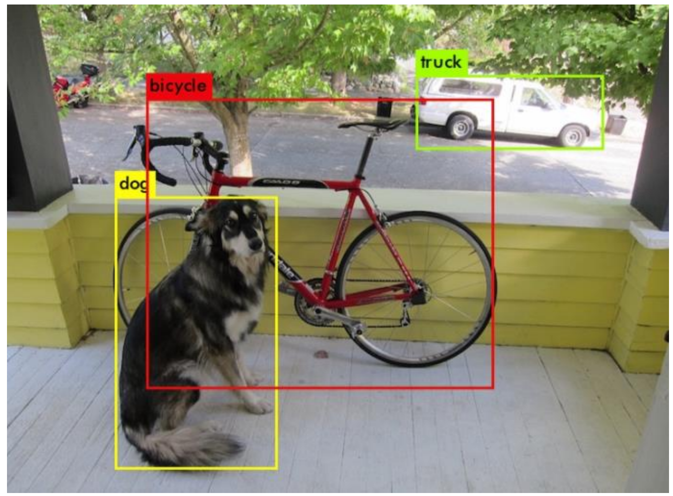
Lines/Edge detection

Corners/Bl

(SIFT/SU

Water

## Classification algorithm




## Object detection and/or Identification

YOLO

# Computer vision can be broken down into a multitude of blocks that can be combined as needed (1/2)

**Image Pre-processing**

Image scaling

Contrast enhancement

Cropping and padding

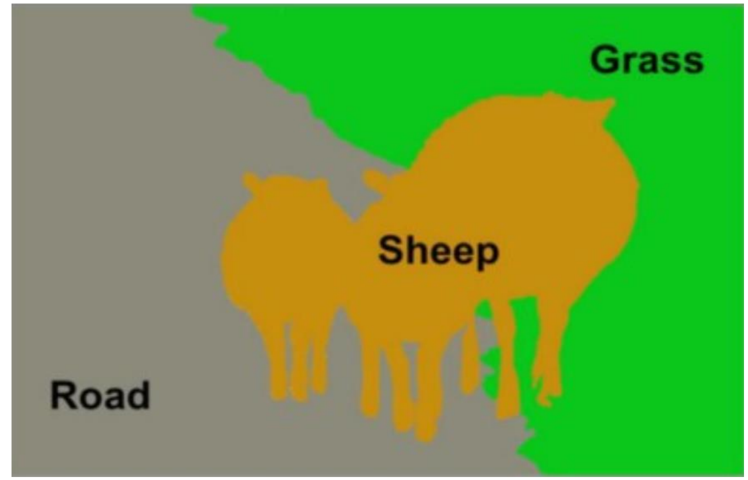
Noise reduction

**Feature extraction**

Lines/Edge detection

Corners/Blobs/Points

(SIFT/SURF, PCA, Watershed)




**Segmentation**

Faster R-CNN

# Computer vision can be broken down into a multitude of blocks that can be combined as needed (2/2)

**Image Pre-processing**

Image scaling

Contrast enhancement

Cropping and padding

Noise reduction

**Feature extraction**

Lines/Edge detection

Corners/Blobs/Points

(SIFT/SURF, PCA, Watershed)

**Classification algorithm**

CNN

**Object detection and/or Identification**

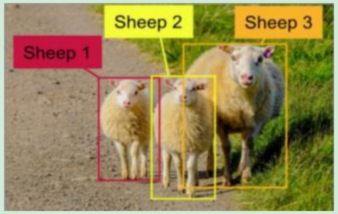
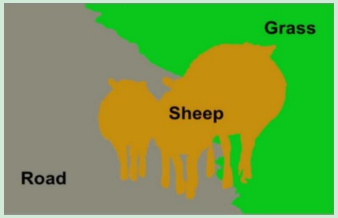
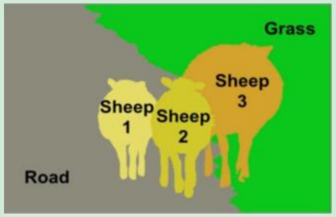
YOLO

- We can start our process by applying image pre-processing (for instance if we have few data)
- Later we can use detection algorithm in order to find the object of interest
- Finally we can leverage feature extraction to post process the results to make them more precise of follow some field rules

AGENDA

1. Computer vision approaches
2. **An overview of the main Deep Learning algorithms**
3. The “open” dataset : our gold mine
4. Measure the performance to improve your algorithm
5. Resources to help you start

# The 3 main approaches to machine learning vision are Classification, Detection and Segmentation

| Approach | Objective | Algorithms | Examples |
|---|---|---|---|
| **Classification** | Associate a class, defining its content, to each image | ▪ CNN |   |
| **Detection** | Spot objects in an image according to predefined categories | ▪ YOLO<br>▪ SSD<br>▪ Faster R-CNN<br>▪ RFCN |   |
| **Segmentation** | **Semantic**: Link a category to each pixel in an image. Recognize a set of pixels that form distinct categories | ▪ Unet<br>▪ PSPNet<br>▪ ICNet<br>▪ DeepLab |   |
| | **Per instance**: Associate each pixel with the instance to which it belongs. | ▪ Mask R-CNN<br>▪ ResNet |   |

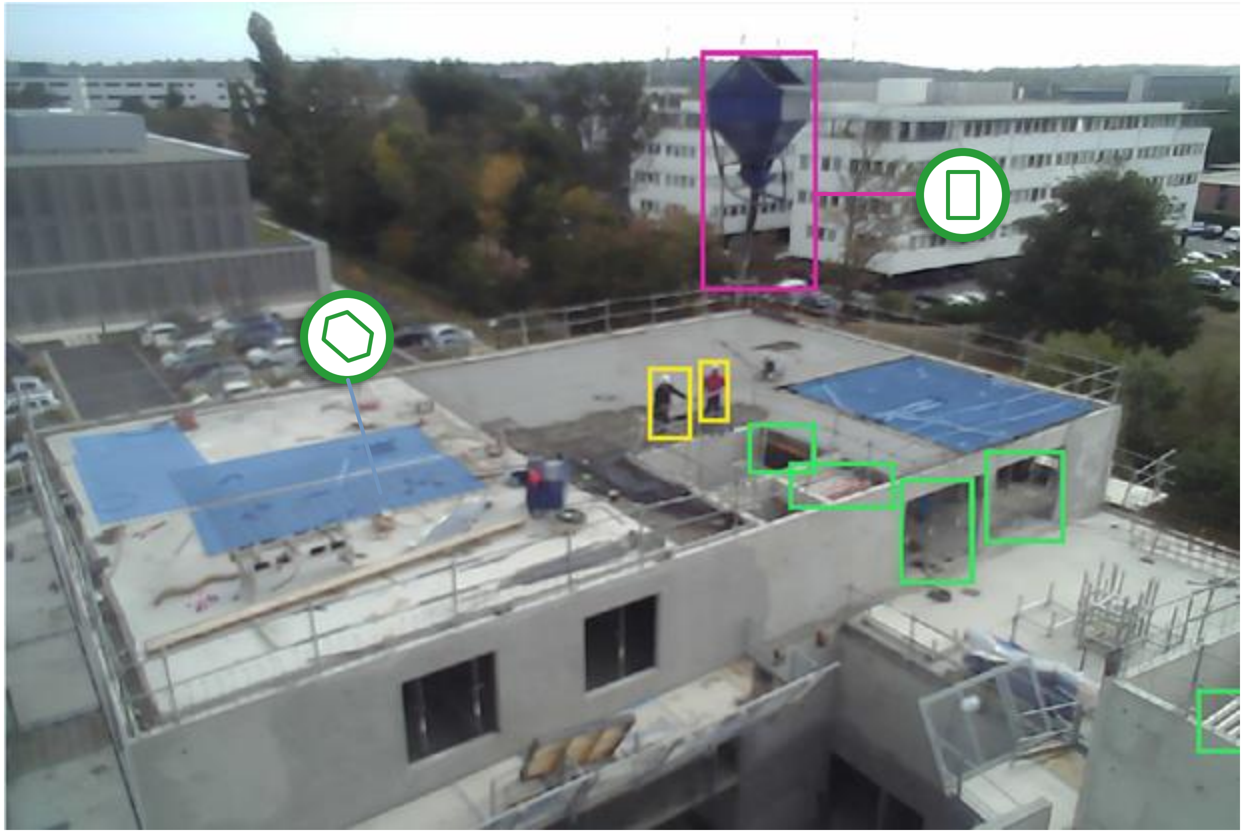
# These different approaches are based on two different types of annotation

**Polygon**

- Polygon annotations select all pixels belonging to an instance. It is used in segmentation

**Bounding box**

- Bounding box annotations give position, size and type of object contained in a rectangle parallel to the axes: used in detection



AGENDA

1. Computer vision approaches
2. An overview of the main Deep Learning algorithms
3. **The "open" dataset : our gold mine**
4. Measure the performance to improve your algorithm
5. Resources to help you start

## To train these models several datasets are available online

| Name | Type | Annotation |
|---|---|---|
| COCO | Natural landscapes | Instance segmentation |
| ADE20K | Natural landscapes | Instance segmentation |
| SUN | Natural landscapes | Instance segmentation |
| IMAGE NET | Natural landscapes | Bounding boxes detection |
| PASCAL VOC2012 | Natural landscapes | Bounding boxes detection |
| CITYSCAPES | City landscapes | Instance segmentation |

- Is the context of the dataset close to context of my images?
- Does the format of the dataset compatible with my algorithm?
- What classes are labeled in the dataset?
- Is the dataset balanced between classes?

AGENDA

1. Computer vision approaches
2. An overview of the main Deep Learning algorithms
3. The “open” dataset : our gold mine
4. **Measure the performance to improve your algorithm**
5. Resources to help you start

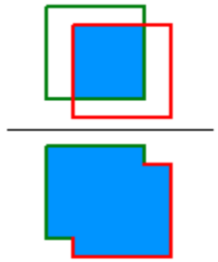
# Several metrics can be used in computer vision depending on the objectives to reach

## Local Metrics

- **Intersect over Union (IoU)**

Measuring the common cover on an object can be formalized as:

$$\boldsymbol{IoU} = \frac{\mathbf{aire}(\mathbf{Box}_{\mathbf{True}} \cap \mathbf{Box}_{\mathbf{Predicted}})}{\mathbf{aire}(\mathbf{Box}_{\mathbf{True}} \cup \mathbf{Box}_{\mathbf{Predicted}})}$$



We set a threshold $\varepsilon \in [0,1]$

Prediction are considered true if $\boldsymbol{IoU} \geq \boldsymbol{\varepsilon}$.

## Model Metrics

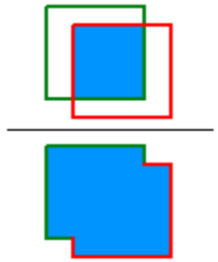
# Several metrics can be used in computer vision depending on the objectives to reach

## Local Metrics

- **Intersect over Union (IoU)**

Measuring the common cover on an object can be formalized as:

$$IoU = \frac{\mathbf{aire}(\mathbf{Box}_{\mathbf{True}} \cap \mathbf{Box}_{\mathbf{Predicted}})}{\mathbf{aire}(\mathbf{Box}_{\mathbf{True}} \cup \mathbf{Box}_{\mathbf{Predicted}})}$$



We set a threshold $\varepsilon \in [0,1]$

Prediction are considered true if $\boldsymbol{IoU} \geq \boldsymbol{\varepsilon}$.

## Model Metrics

- **Mean Intersect over Union (mIoU)**

For a **fixed minimum confidence** level, the IoU is calculated for each annotation and then averaged over a class or globally.

- No indication of the type of error: over/under-prediction?
- Depends on a confidence threshold set

- Very "visual" metric, easy to interpret

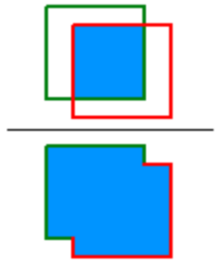
# Several metrics can be used in computer vision depending on the objectives to reach

## Local Metrics

- **Intersect over Union (IoU)**

Measuring the common cover on an object can be formalized as:

$$IoU = \frac{\text{aire}(\mathbf{Box}_{\mathbf{True}} \cap \mathbf{Box}_{\mathbf{Predicted}})}{\text{aire}(\mathbf{Box}_{\mathbf{True}} \cup \mathbf{Box}_{\mathbf{Predicted}})}$$



We set a threshold $\varepsilon \in [0,1]$

Prediction are considered true if $IoU \geq \varepsilon$.

- **Precision/Recall :**

$$\text{precision} = \frac{TP}{TP + FP}$$

Precision answers the question: **What share of the detected objects were the right ones?**

$$\text{recall} = \frac{TP}{TP + FN}$$

Recall answers the question: **What is the share of the objects that have been detected?**

## Model Metrics

- **Mean Intersect over Union (mIoU)**

For a **fixed minimum confidence** level, the IoU is calculated for each annotation and then averaged over a class or globally.

- No indication of the type of error: over/under-prediction?
- Depends on a confidence threshold set

- Very "visual" metric, easy to interpret

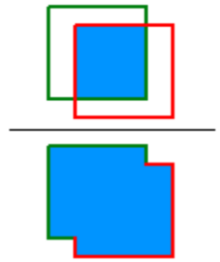
# Several metrics can be used in computer vision depending on the objectives to reach

## Local Metrics

- **Intersect over Union (IoU)**

Measuring the common cover on an object can be formalized as:

$$IoU = \frac{\text{aire}(\text{Box}_{\text{True}} \cap \text{Box}_{\text{Predicted}})}{\text{aire}(\text{Box}_{\text{True}} \cup \text{Box}_{\text{Predicted}})}$$



We set a threshold $\varepsilon \in [0,1]$

Prediction are considered true if $IoU \geq \varepsilon$.

- **Precision/Recall :**

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

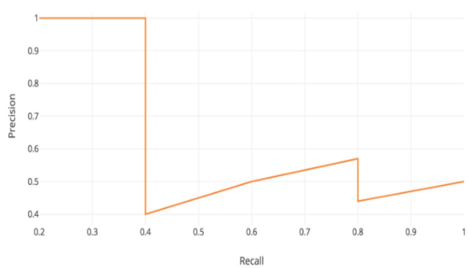
Precision answers the question: **What share of the detected objects were the right ones?**

Recall answers the question: **What is the share of the objects that have been detected?**

- **Precision-recall curve :**

Precision and recall are calculated for different confidence levels (proba of the class model) in order to obtain the curve:

$$precision = f(recall)$$




Precision-recall curve

## Model Metrics

- **Mean Intersect over Union (mIoU)**

For a **fixed minimum confidence** level, the IoU is calculated for each annotation and then averaged over a class or globally.

- No indication of the type of error: over/under-prediction?
- Depends on a confidence threshold set

- Very "visual" metric, easy to interpret

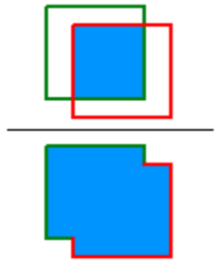
# Several **metrics** can be used in computer vision depending on the objectives to reach

## Local Metrics

### Intersect over Union (IoU)

Measuring the common cover on an object can be formalized as:

$$IoU = \frac{\text{aire}(\text{Box}_{\text{True}} \cap \text{Box}_{\text{Predicted}})}{\text{aire}(\text{Box}_{\text{True}} \cup \text{Box}_{\text{Predicted}})}$$



We set a threshold $\varepsilon \in [0,1]$

Prediction are considered true if $\boldsymbol{IoU \geq \varepsilon}$.

### Precision/Recall :

$$\text{precision} = \frac{TP}{TP + FP}$$

Precision answers the question: **What share of the detected objects were the right ones?**

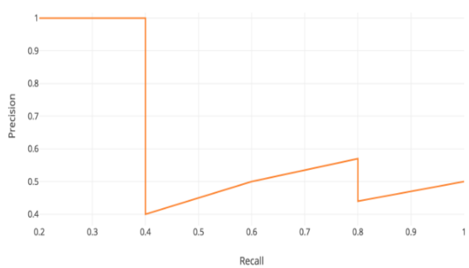
$$\text{recall} = \frac{TP}{TP + FN}$$

Recall answers the question: **What is the share of the objects that have been detected?**

### Precision-recall curve :

Precision and recall are calculated for different confidence levels (proba of the class model) in order to obtain the curve:

$$precision = f(recall)$$



Precision-recall curve

## Model Metrics

### Mean Intersect over Union (mIoU)

For a **fixed minimum confidence** level, the IoU is calculated for each annotation and then averaged over a class or globally.

- No indication of the type of error: over/under-prediction?
- Depends on a confidence threshold set

- Very "visual" metric, easy to interpret

### Average Precision (AP)

Once the IoU threshold is set, the AP of a class can be calculated as a measure of the area below the precision-recall curve.

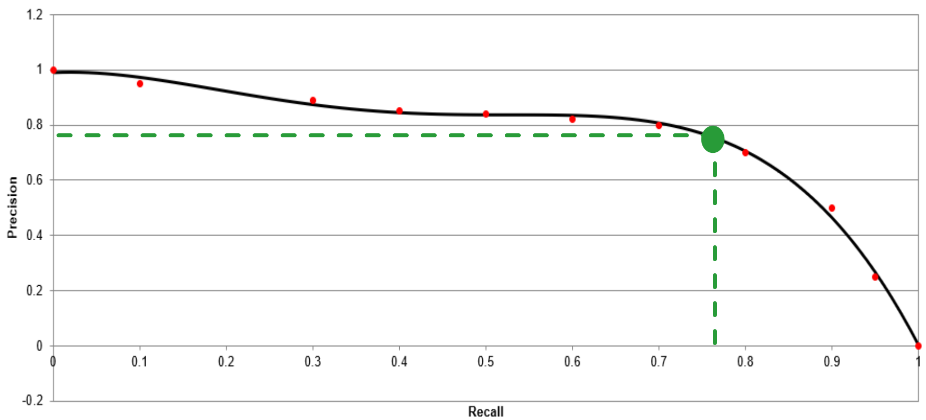
- Difficult to interpret
- Depends on a fixed IoU threshold

- Independent of confidence level
- Gives an indication of the precision/recall trade-off

➤ **The mean Average Precision (mAP)** is the **AP** averaged over all classes

# The AP reflects the ability to combine precision and recall for a model

## Example n°1



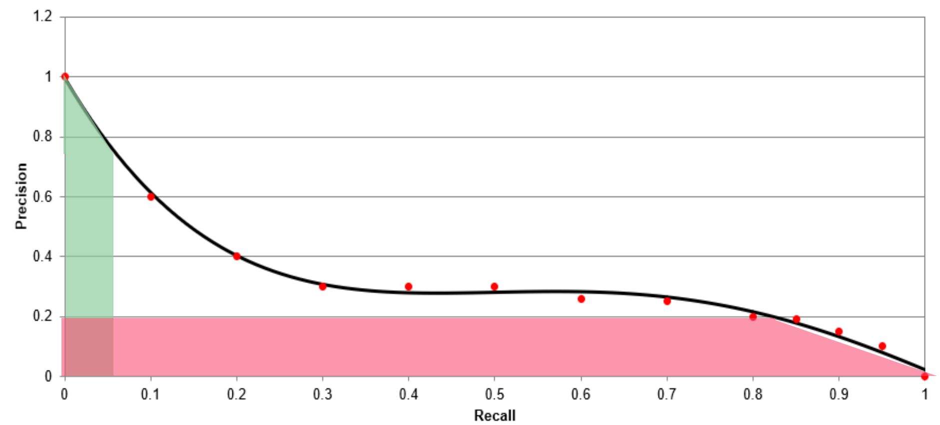

$$AP \approx 0.74$$

**Interpretation :**

- We see that we can maintain a good trade-off between the level of precision and the level of recall
- The green dot guarantees an accuracy of about 80% for a 75% recall: most objects are spotted without too many errors

➢ An important AP ensures a certain efficiency of the model that will combine precision and recall

## Example n°2




$$AP \approx 0.28$$

**Interpretation :**

We see here that we cannot guarantee precision and recall: we must choose to favour one or the other

- The green zone guarantees a precision $> 75\%$ accuracy but also a recall $< 20\%$ : we don't make much mistake but we spot very few objects
- The pink zone guarantees a recall > 80% but a precision < 5% : We spot almost all the object but most of the objects identified are not the one we want

➢ A low AP means it is difficult to reconcile accuracy and recall: one should be favored

AGENDA

## Resources: all classical Machine Learning might come handy and a good understanding of Computer Vision libraries will be helpful

### Computer Vision libraries

**PyTorch**

*Standard library for deep learning*

**OpenCV**

*Standard library classical computer vision*

**TensorFlow**

*An alternative to Pytorch*

### ML + Viz libraries

**scikit learn**

*To develop a wide range of ML models*

**pandas**

*To exploite model's output and aggregate them*

**Streamlit**

*To efficiently develop a dashboard / front-end*

To use a pretrained detection model in Pytorch, several steps need to be done such as the download of the model, the analysis of the image by the model, and the printing of the prediction on the image

## Illustration of the different steps

- Import libraries
- Download pretrained model
- Define the classes of the pretrained model
- Process the input image to the right format
- Predict the objects present in the image
- Plot the results of the prediction model

To use a pretrained detection model in Pytorch, several steps need to be done such as the download of the model, the analysis of the image by the model, and the printing of the prediction on the image

## Illustration of the different steps

- **Import libraries**
- Download pretrained model
- Define the classes of the pretrained model
- Process the input image to the right format
- Predict the objects present in the image
- Plot the results of the prediction model

```python
from PIL import Image
from torchvision import models
from torchvision import transforms as T
import matplotlib.pyplot as plt
import cv2
```

Example of code to import libraries

To use a pretrained detection model in Pytorch, several steps need to be done such as the download of the model, the analysis of the image by the model, and the printing of the prediction on the image

## Illustration of the different steps

- Import libraries
- **Download pretrained model**
- Define the classes of the pretrained model
- Process the input image to the right format
- Predict the objects present in the image
- Plot the results of the prediction model

```
# Loading the model and the dataset
# Loads pretrained VGG model and sets it to eval mode

model = models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
model = model.eval()
```

Example of code to download a detection model

To use a pretrained detection model in Pytorch, several steps need to be done such as the download of the model, the analysis of the image by the model, and the printing of the prediction on the image

## Illustration of the different steps

- Import libraries
- Download pretrained model
- **Define the classes of the pretrained model**
- Process the input image to the right format
- Predict the objects present in the image
- Plot the results of the prediction model

```
COCO_INSTANCE_CATEGORY_NAMES = [
    "__background__",
    "person",
    "bicycle",
    "car",
    "motorcycle",
    "airplane",
    "bus",
    "train",
    "truck",
    "boat",
```

Example of code to define the classes of the detection model

To use a pretrained detection model in Pytorch, several steps need to be done such as the download of the model, the analysis of the image by the model, and the printing of the prediction on the image

## Illustration of the different steps

- Import libraries
- Download pretrained model
- Define the classes of the pretrained model
- **Process the input image to the right format**
- Predict the objects present in the image
- Plot the results of the prediction model

```
img = Image.open(img_path)  # Load the image
transform = T.Compose([T.ToTensor()])  # Defing PyTorch Transform
img = transform(img)  # Apply the transform to the image
```

Example of code to process the input image

To use a pretrained detection model in Pytorch, several steps need to be done such as the download of the model, the analysis of the image by the model, and the printing of the prediction on the image

## Illustration of the different steps

- Import libraries
- Download pretrained model
- Define the classes of the pretrained model
- Process the input image to the right format
- **Predict the objects present in the image**
- Plot the results of the prediction model

```
pred = model([img])  # Pass the image to the model
pred_class = [
    COCO_INSTANCE_CATEGORY_NAMES[i]
    for i in list(pred[0]["labels"].numpy())
]  # Get the Prediction Score
pred_boxes = [
    [(i[0], i[1]), (i[2], i[3])]
    for i in list(pred[0]["boxes"].detach().numpy())
]  # Bounding boxes
pred_score = list(pred[0]["scores"].detach().numpy())
pred_t = [pred_score.index(x) for x in pred_score if x > threshold][
    -1
]  # Get list of index with score greater than threshold.
pred_boxes = pred_boxes[: pred_t + 1]
pred_class = pred_class[: pred_t + 1]
```

Example of code to predict the objects in the image

To use a pretrained detection model in Pytorch, several steps need to be done such as the download of the model, the analysis of the image by the model, and the printing of the prediction on the image

## Illustration of the different steps

- Import libraries
- Download pretrained model
- Define the classes of the pretrained model
- Process the input image to the right format
- Predict the objects present in the image
- **Plot the results of the prediction model**

```
img = cv2.imread(img_path)  # Read image with cv2
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert to RGB
for i in range(len(boxes)):
    cv2.rectangle(
        img, boxes[i][0], boxes[i][1], color=(0, 255, 0), thickness=rect_th
    )  # Draw Rectangle with the coordinates
    cv2.putText(
        img,
        pred_cls[i],
        boxes[i][0],
        cv2.FONT_HERSHEY_SIMPLEX,
        text_size,
        (0, 255, 0),
        thickness=text_th,
    )  # Write the prediction class
```

Example of code to plot the detected objects on the image

To use a pretrained detection model in Pytorch, several steps need to be done such as the download of the model, the analysis of the image by the model, and the printing of the prediction on the image

## Illustration of the different steps

- Import libraries
- Download pretrained model
- Define the classes of the pretrained model
- Process the input image to the right format
- Predict the objects present in the image
- Plot the results of the prediction model

This tutorial is in the python file:
*Tutorial_Computer_Vision_Detection.py*

*It can be used as a baseline for your project*