Object Detection & Instance Segmentation

YoloV5, Faster-RCNN, EfficientDet d7, Mask-RCNN, and Detectron2

Hichem Felouat

hichemfel@gmail.com

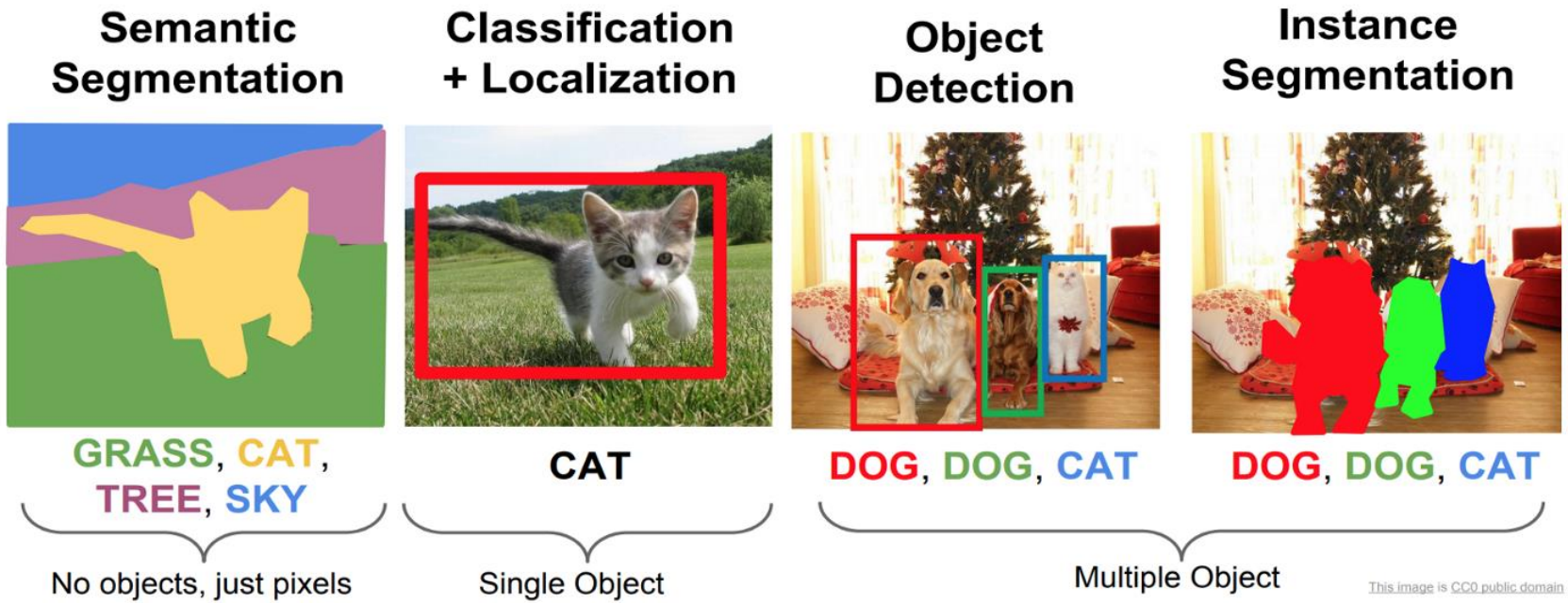
https://www.researchgate.net/profile/Hichem_Felouat

https://www.linkedin.com/in/hichemfelouat/

Object Detection

- We all know about the image **classification** problem. Given an image, the Net finds out the class to which the image belongs.

- Localizing an object in a picture means predicting a **bounding box** around the object and can be expressed as a **regression** task.



Object Detection

Problem: the dataset does not have bounding boxes around the objects, how can we train our model?

- We need to **add them ourselves**. This is often one of the **hardest and most costly parts** of a Machine Learning project: **getting the labels**.

- It is a good idea to spend time looking for the **right tools**.

Image Labeling Tool

An image labeling or annotation tool is used to label the images for bounding box object detection and segmentation.

Open-source image labeling tool like :

- VGG Image
- Annotator
- LabelImg
- OpenLabeler
- ImgLab

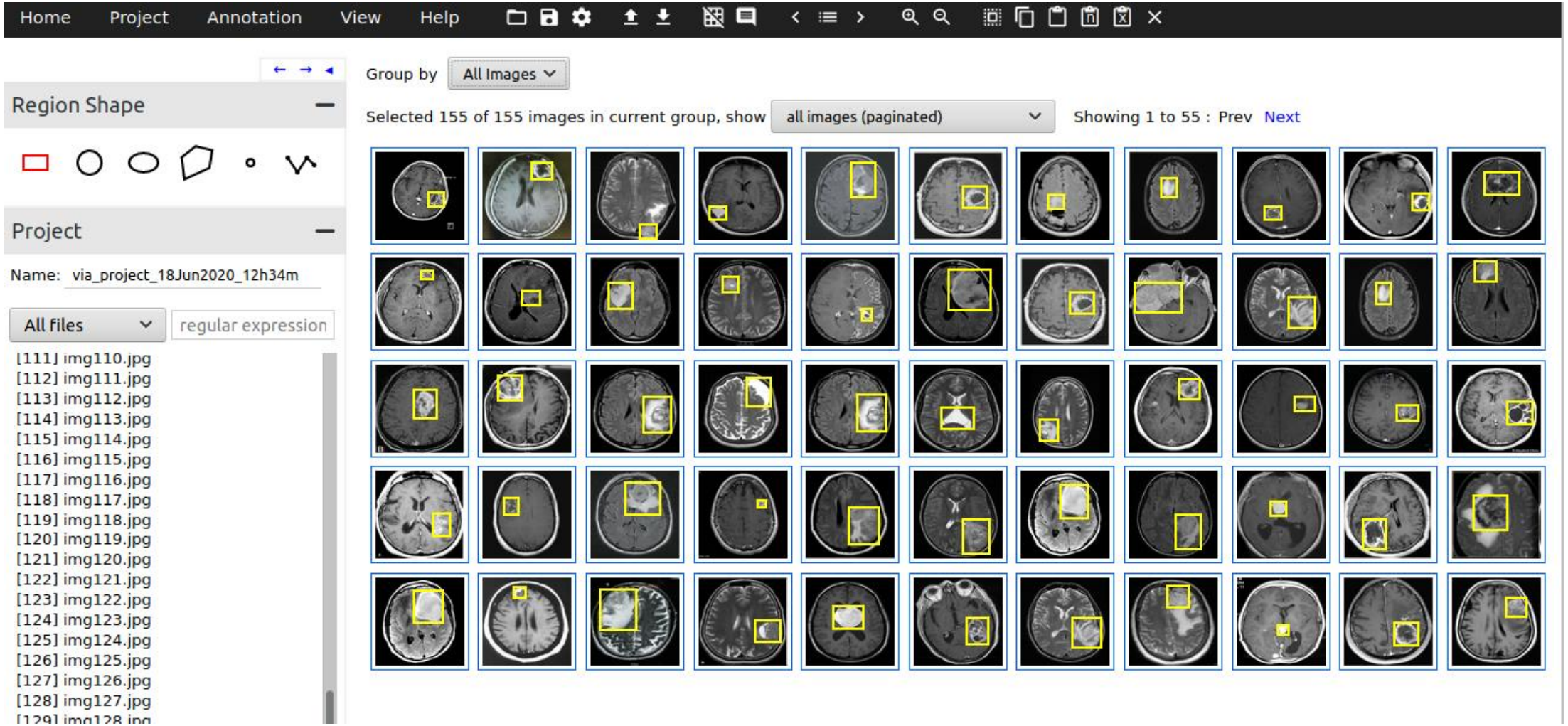
Commercial tool like :

- LabelBox
- Supervisely

Crowdsourcing Platform like :

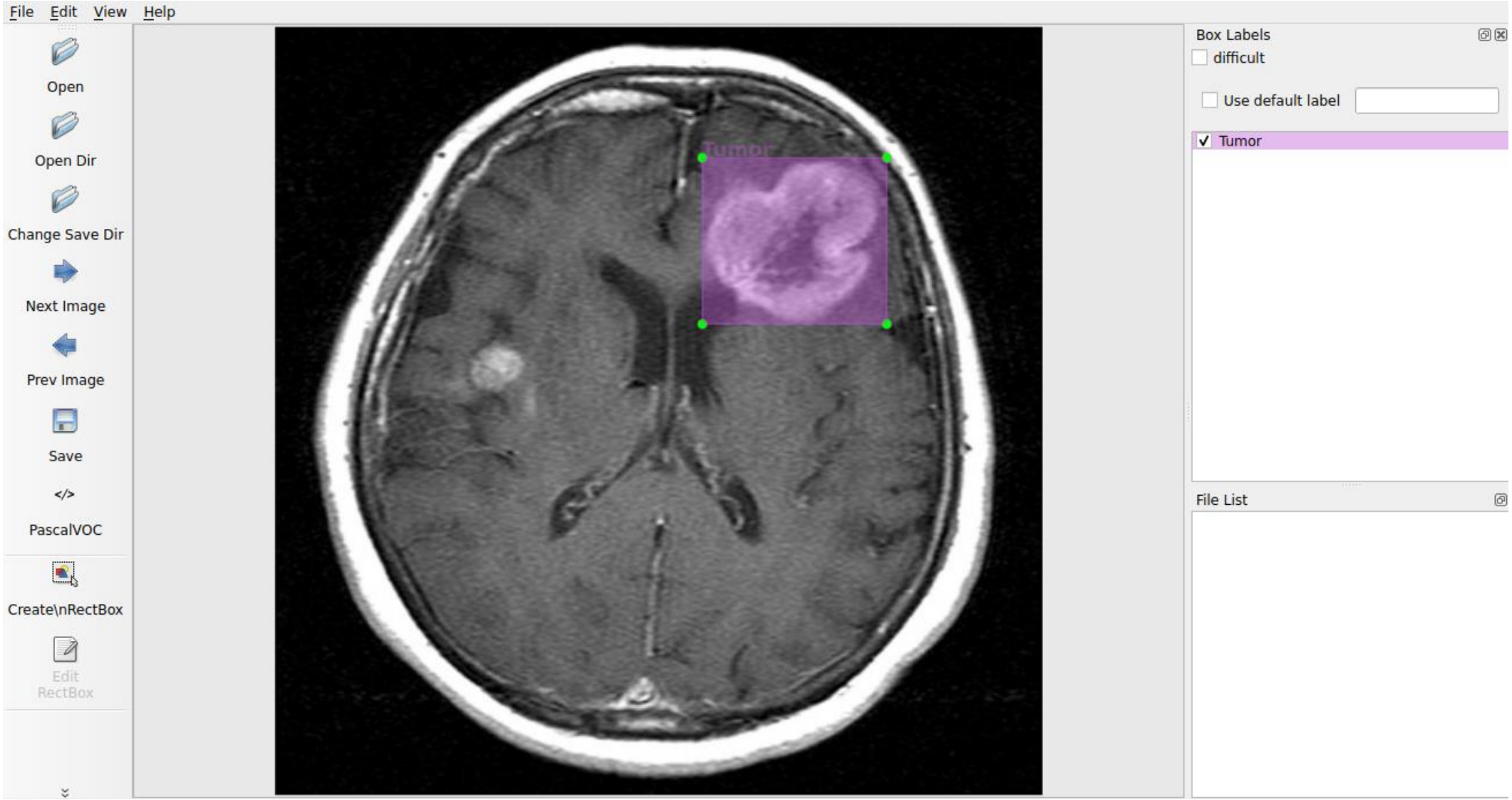
- Amazon Mechanical Turk

Image Labeling Tool - VGG Image



VGG Image: *http://www.robots.ox.ac.uk/~vgg/software/via/*

Image Labeling Tool - labelImg



labelimg

Object Detection - Notes

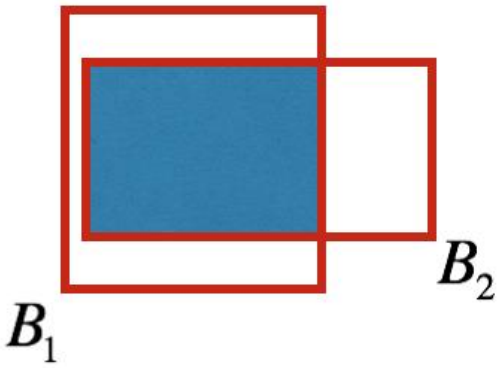
- The bounding boxes should be **normalized** so that the **horizontal** and **vertical** coordinates, as well as the **height** and **width**, all range from **0 to 1**.

- It is common to predict **the square root of the height and width** rather than the height and width directly: this way, a **10-pixel error** for a large bounding box will not be penalized as much as a 10-pixel error for a small bounding box.

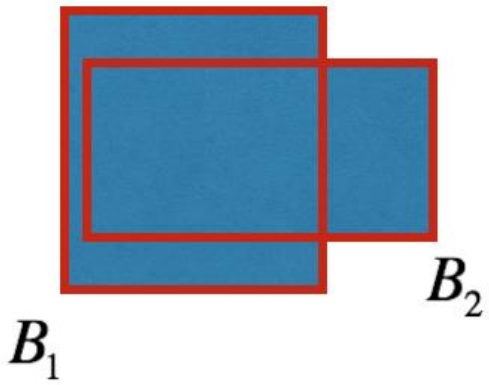
How to Evaluate Object Detection Model?

- The **MSE** often works fairly well as a **cost function** to train the model, but it is not a great metric to evaluate how well the model can predict bounding boxes.
- The most common metric for this is **the Intersection over Union (IoU)**.
- **tf.keras.metrics.MeanIoU**

Intersection

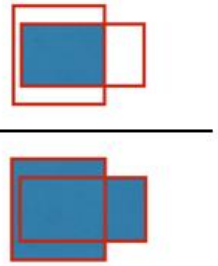


Union



Intersection over Union

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} =$$



Evaluate Object Detection Model : mAP

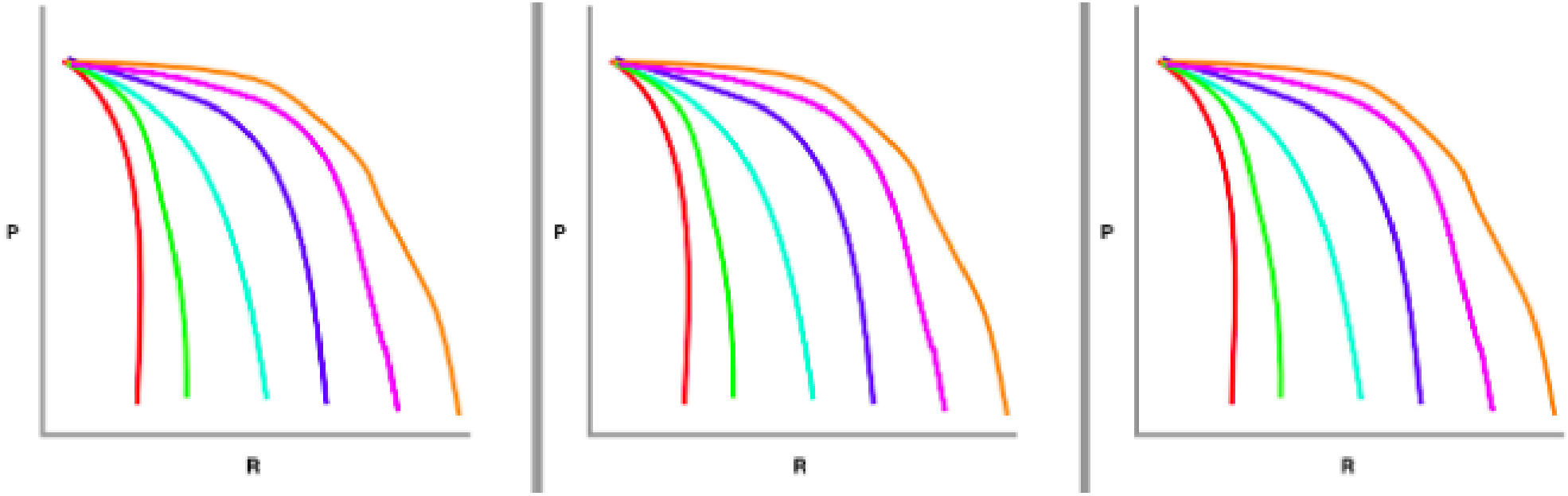
mean Average Precision (mAP)

In order to calculate **mAP**, we draw a series of **precision-recall curves** with the **IoU threshold** set at varying levels of difficulty. In COCO evaluation, the IoU threshold ranges from 0.5 to 0.95 with a step size of 0.05 represented as AP@[.5:.05:.95]

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\#\text{ ground truths}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\#\text{ predictions}}$$

Evaluate Object Detection Model : mAP



We draw these precision-recall curves for the dataset split out by class type (for example **3 classes** and **6 threshold ranges**).

Evaluate Object Detection Model : mAP

These precision and recall values are then plotted to get a PR (precision-recall) curve. **The area under the PR curve** is called **Average Precision (AP)**.

- For each class, calculate AP at different IoU thresholds and take their average to get the AP of that class.

$$\text{AP}[\text{class}] = \frac{1}{\#\text{thresolds}} \sum_{\text{iou} \in \text{thresholds}} AP[class, iou]$$

Exp: in **PASCAL VOC challenge 2007**, it is defined as the mean of precision values at a set of **11** equally spaced recall levels [0,0.1,…,1] (0 to 1 at step size of 0.1).

$$AP = \frac{1}{11} \sum_{r \in (0, 0.1, \ldots, 1)} p_{interp(r)}$$

Object detection metrics : *https://github.com/rafaelpadilla/Object-Detection-Metrics*

Evaluate Object Detection Model : mAP

- Calculate the final AP by averaging the AP over different classes.

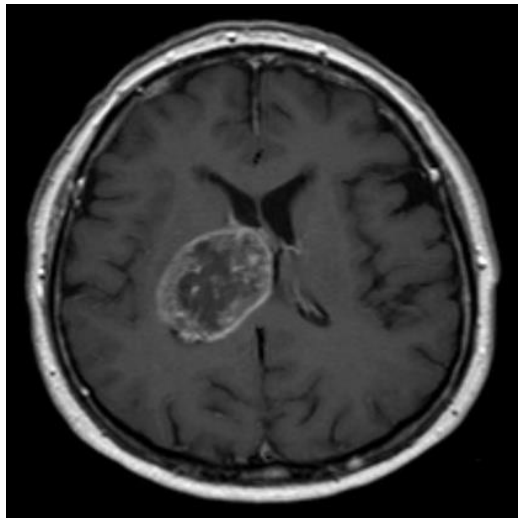
$$\text{AP} = \frac{1}{\#\text{classes}} \sum_{\text{class} \in \text{classes}} AP[class]$$

= **mAP-IoU_thresholds**

mAP - example COCO dataset

$$mAP_{\text{COCO}} = \frac{mAP_{0.50} + mAP_{0.55} + \ldots + mAP_{0.95}}{10}$$

Object Detection - Example



raw image
416*416

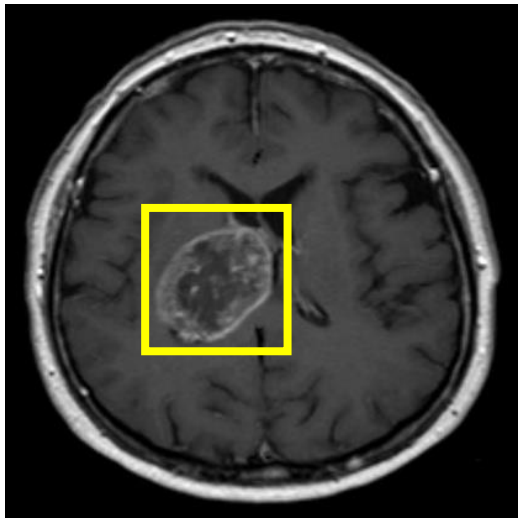
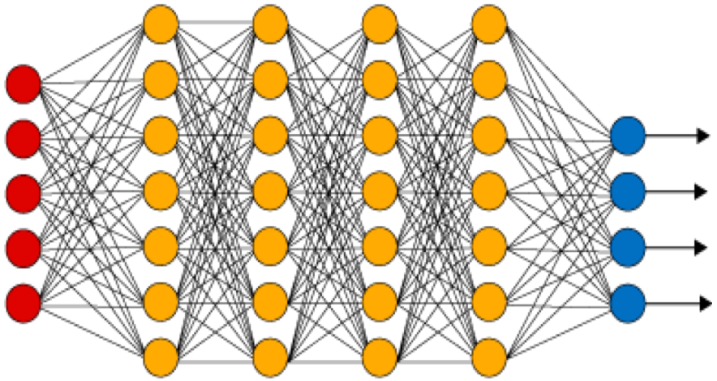


image labeling
(C, X,Y, W, H)



model

(C, X,Y, W, H)

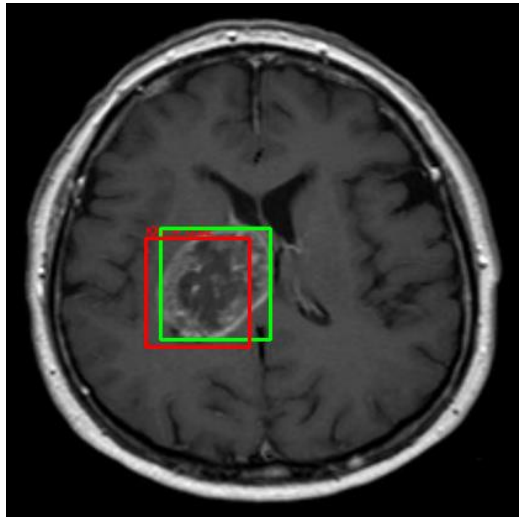


image result

- Each item should be a tuple of the form :

(images,
(class_labels, bounding_boxes))

Object Detection - Example

```
81  model = keras.models.Sequential()
82  model.add(keras.layers.Conv2D(filters=64, kernel_size=5, strides=1, padding="same",
83                                activation="relu", input_shape= (416,416,3)))
84  model.add(keras.layers.MaxPool2D(pool_size=2))
85  model.add(keras.layers.Conv2D(filters=128, kernel_size=3, strides=1, padding="same",
86                                activation="relu"))
87  model.add(keras.layers.MaxPool2D(pool_size=2))
88  model.add(keras.layers.Conv2D(filters=128, kernel_size=3, strides=1, padding="same",
89                                activation="relu"))
90  model.add(keras.layers.MaxPool2D(pool_size=2))
91
92  model.add(keras.layers.Flatten())
93  model.add(keras.layers.Dense(64, activation="relu"))
94  model.add(keras.layers.Dense(32, activation="relu"))
95  model.add(keras.layers.Dense(4))
96
97  opt = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
98  model.compile(loss="mean_squared_error", optimizer=opt, metrics=["mse"])
99  history = model.fit(X_train, y_train, epochs=10, batch_size=8)
```

Full code: *https://www.kaggle.com/hichemfelouat/braintumorlocalization*

Object Detection - Exp TL

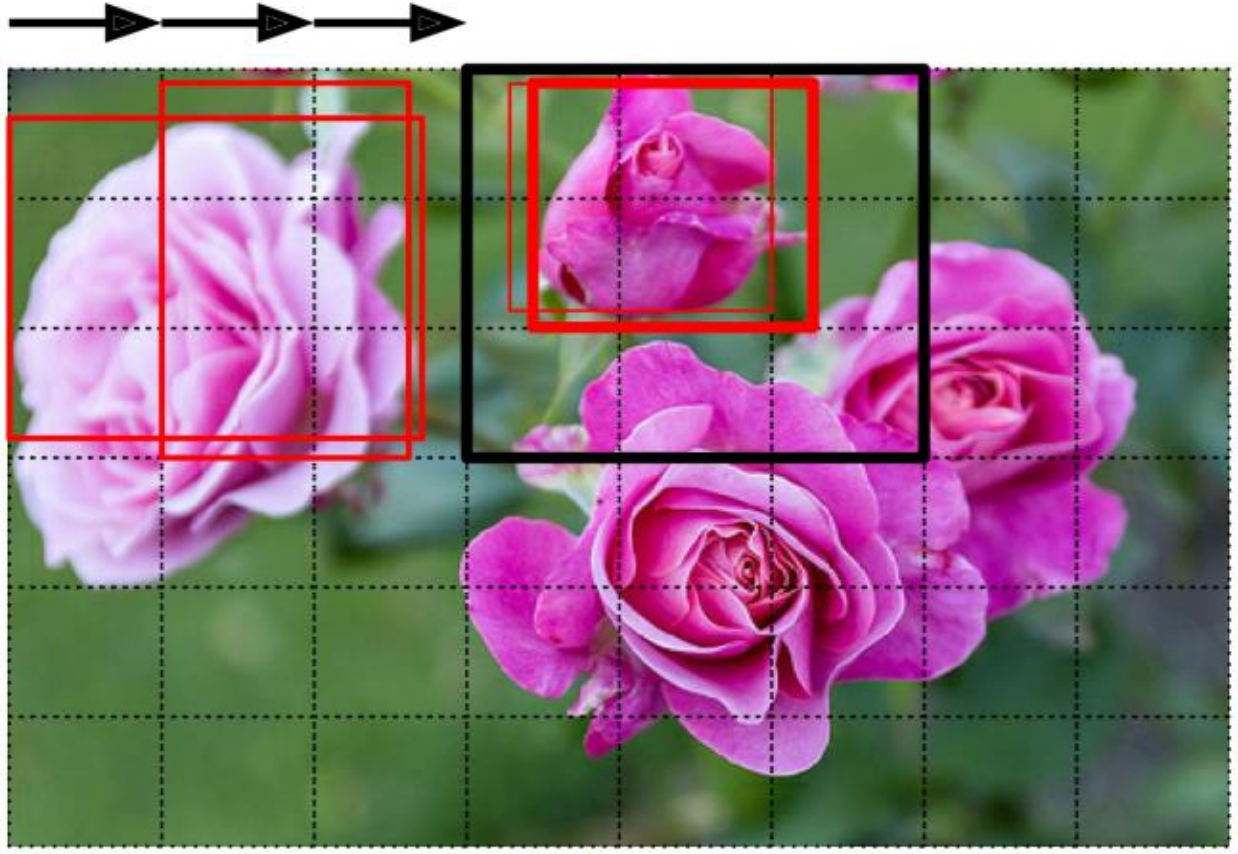
```
82  base_model = keras.applications.inception_resnet_v2.InceptionResNetV2(weights=
83                                                      "imagenet",include_top=False)
84  avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
85  layer_h1 = keras.layers.Dense(64, activation="relu")(avg)
86  layer_h2 = keras.layers.Dense(32, activation="relu")(layer_h1)
87  output = keras.layers.Dense(4)(layer_h2)
88  model = keras.Model(inputs=base_model.input, outputs=output)
```

Full code: *https://www.kaggle.com/hichemfelouat/braintumorlocalization*

Multiple Objects Detection

The task of **classifying and localizing multiple objects** in an image is called **object detection**.

Detecting multiple objects by **sliding a CNN across the image**.



Multiple Objects Detection

1. You need to add an **extra objectness output** to your CNN, to estimate the probability that an object is indeed present in the image.

2. Find **the bounding box with the highest objectness score**, and get rid of all the other bounding boxes that overlap a lot with it (**IoU**).

3. **Repeat step two** until there are no more bounding boxes to get rid of.

Multiple Objects Detection

In general, object detectors have three (3) main components:

1) The **backbone** that **extracts features** from the given image.

2) The **feature network** that takes multiple levels of features from the backbone as input and **outputs a list of fused features** that represent salient characteristics of the image.

3) The final **class/box network** that uses the fused features to **predict the class and location** of each object.

What is Yolo?

- **You Only Look Once (YOLO)** is an algorithm that uses convolutional neural networks for object detection.

- It is one of **the faster** object detection algorithms out there.

- It is a very good choice when we need **real-time detection**, without loss of too much accuracy.

You Only Look Once: Unified, Real-Time Object Detection *https://arxiv.org/abs/1506.02640*
YOLOV3: *https://arxiv.org/abs/1804.02767*
YOLOV4: *https://arxiv.org/abs/2004.10934*

What is Yolo?

Backbone
refers to the **feature-extraction** architecture. **Tiny YOLO** has only 9 convolutional layers, so it's less accurate but faster and better suited for mobile and embedded projects. **Darknet53** (The backbone used in YOLOV3) has 53 convolutional layers, so it's more accurate but slower.

In **YOLOv4**, backbones can be VGG, ResNet, SpineNet, EfficientNet, ResNeXt, or Darknet53.

What is Yolo?

Neck

additional layers between the backbone and the head (dense prediction block), its purpose is to **add extra information** in the layers. (Feature Pyramid Networks, Path Aggregation Network ...).

Head

The head block is the part used to: **locate bounding boxes** and **classify** what's inside each box.

Interpreting The YOLO Output

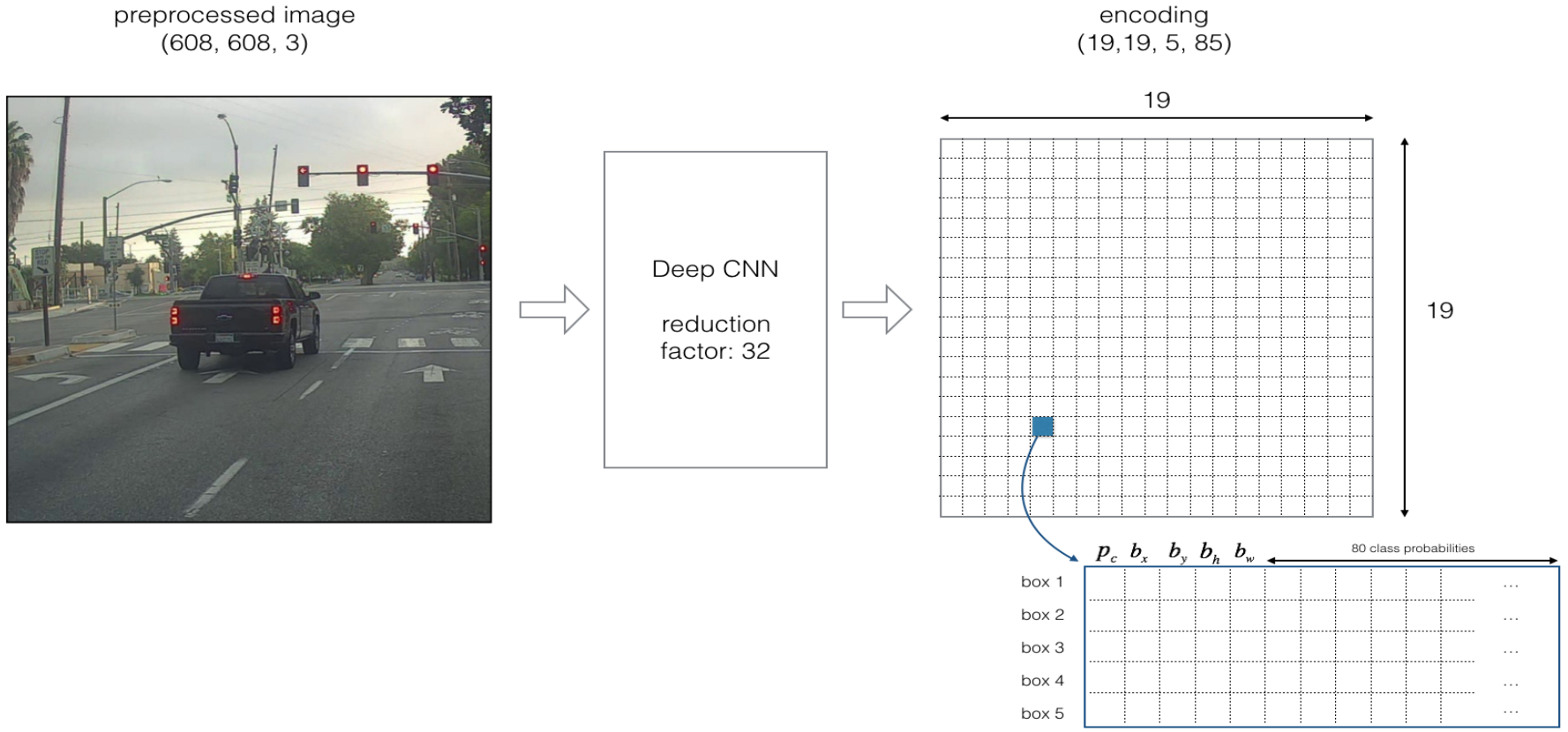
- **The input** is a batch of images of shape **(m, 608, 608, 3)**.

- **The output** is a list of bounding boxes along with the recognized classes. Each bounding box is represented by **6 numbers (pc, bx, by, bh, bw, c)**. If we expand **c** into an 80-dimensional vector (The number of classes), each bounding box is then represented by **85 numbers**.

- To do that, **we divide the input image into a grid of dimensions equal** to that of the final feature map. For example, the input image is **608 x 608**, and the dimensions of the feature map are **19 x 19**. So the stride of the network will be **32** (608/19=32).

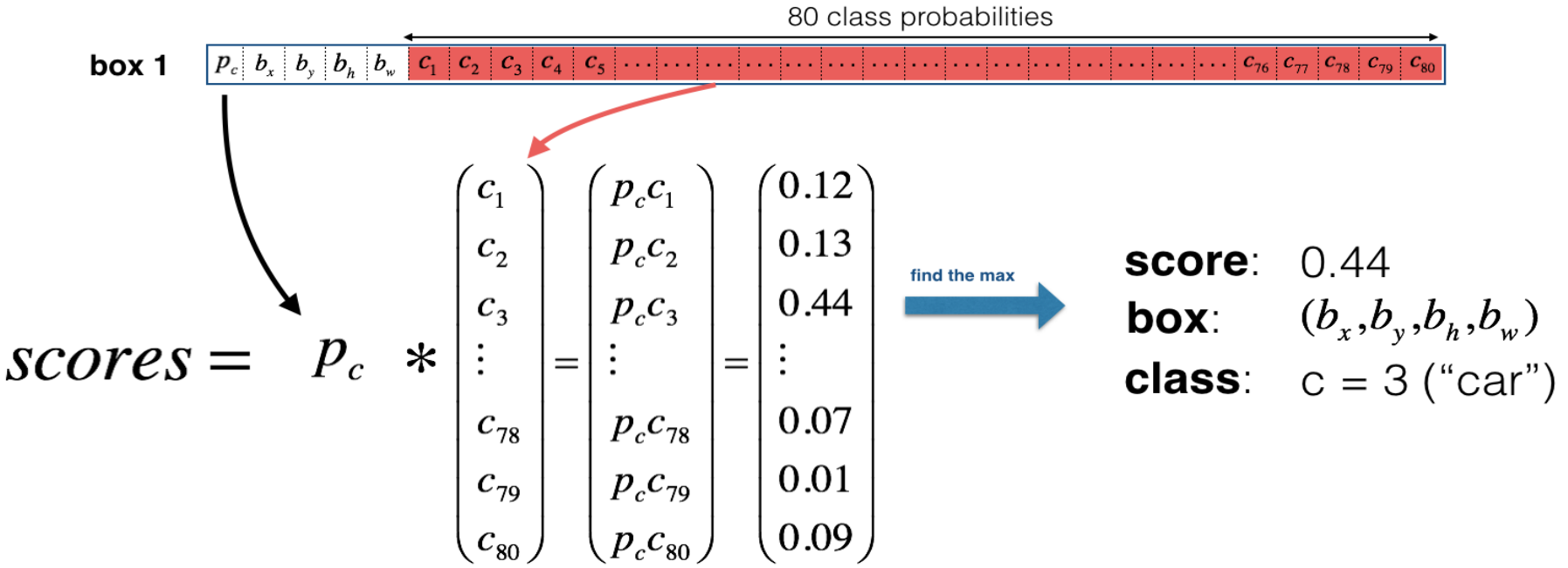
Interpreting The YOLO Output

- IMAGE (m, 608, 608, 3) -> DEEP CNN -> ENCODING (m, 19, 19, 5, 85).
- We are using **5 anchor boxes**, each of the 19x19 cells thus encodes information about **5 boxes** (YOLOV3).



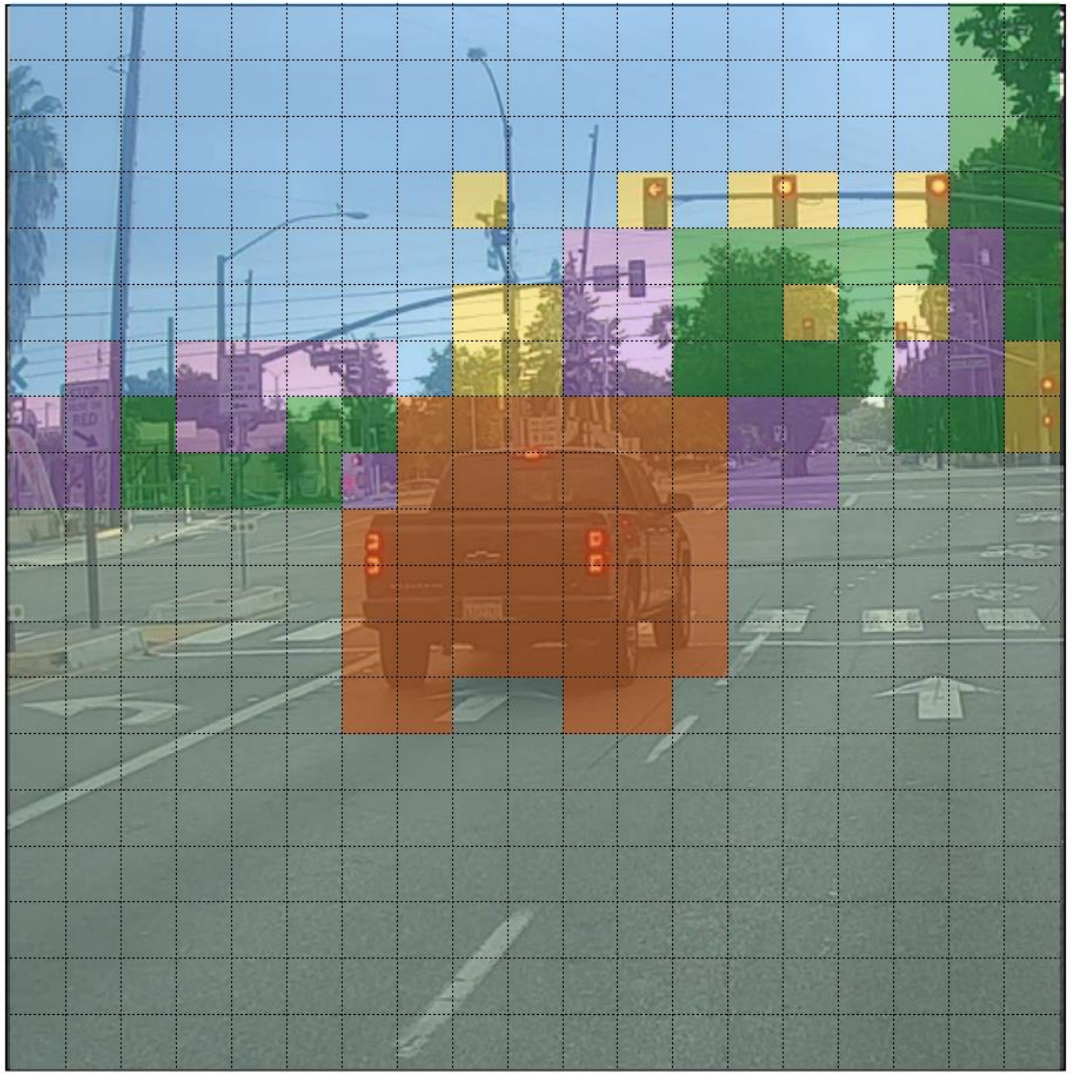
Interpreting The YOLO Output

- Now, for **each box** (of each cell) we will compute the following elementwise product and extract a **probability** that the box contains a certain **class**.



the box (b_x, b_y, b_h, b_w) has detected c = 3 ("car") with probability score: 0.44

What does YOLO Predict on an Image



Output Processing

We will carry out these steps:

- Get rid of boxes with a **low score** (meaning, the box is not very confident about detecting a class (**exp: pc < 0.5**)).
- Select only one box when several boxes overlap with each other and detect the same object (**Non-max suppression**).

YOLOV5

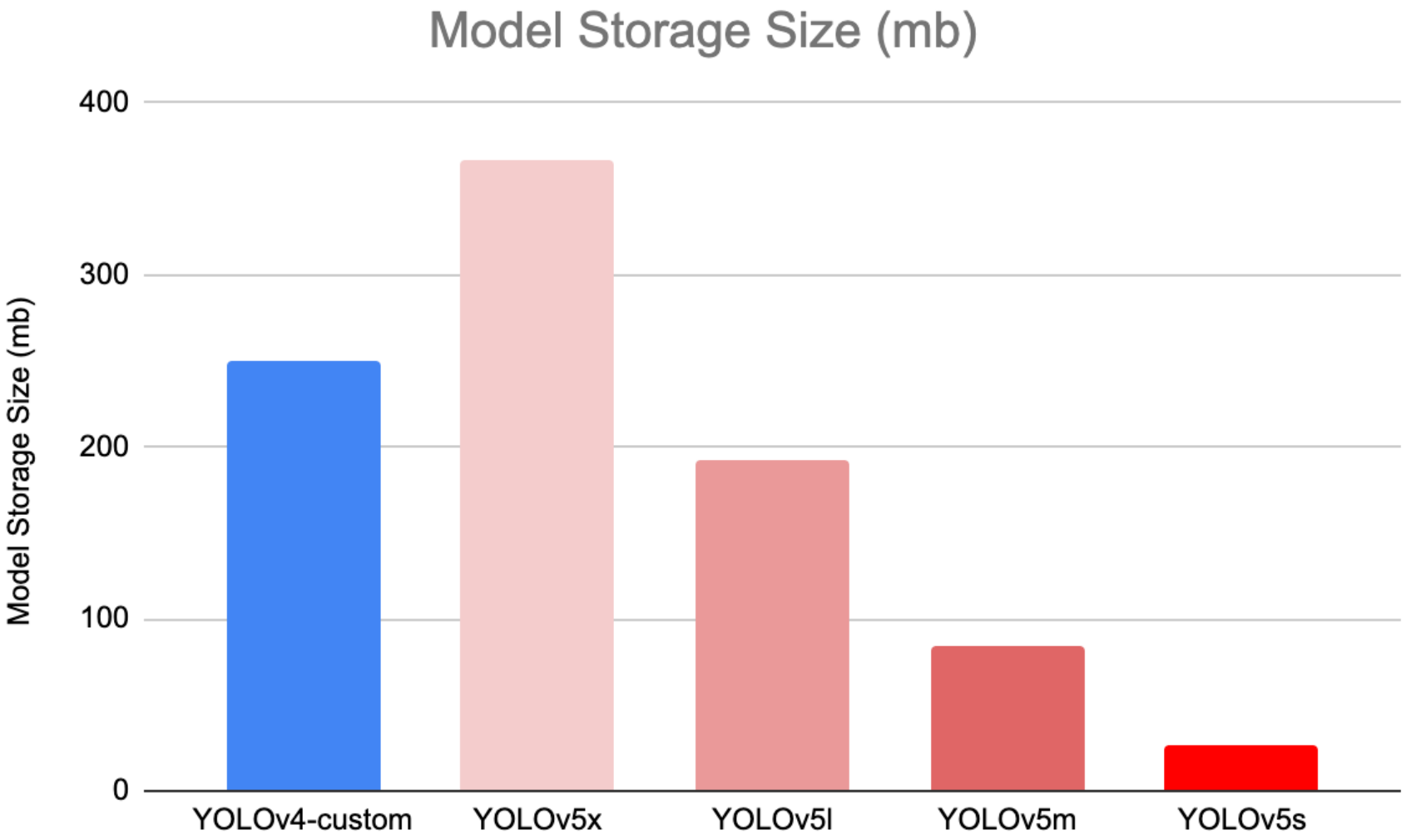
- **YOLOv5** was released by Glenn Jocher on **June 9, 2020**. It follows the recent releases of YOLOv4 (April 23, 2020).

- YOLOv5 is **smaller** and generally **easier** to use in production. Given it is natively implemented in **PyTorch** (rather than Darknet), modifying the architecture and exporting to many deploy environments is straightforward.

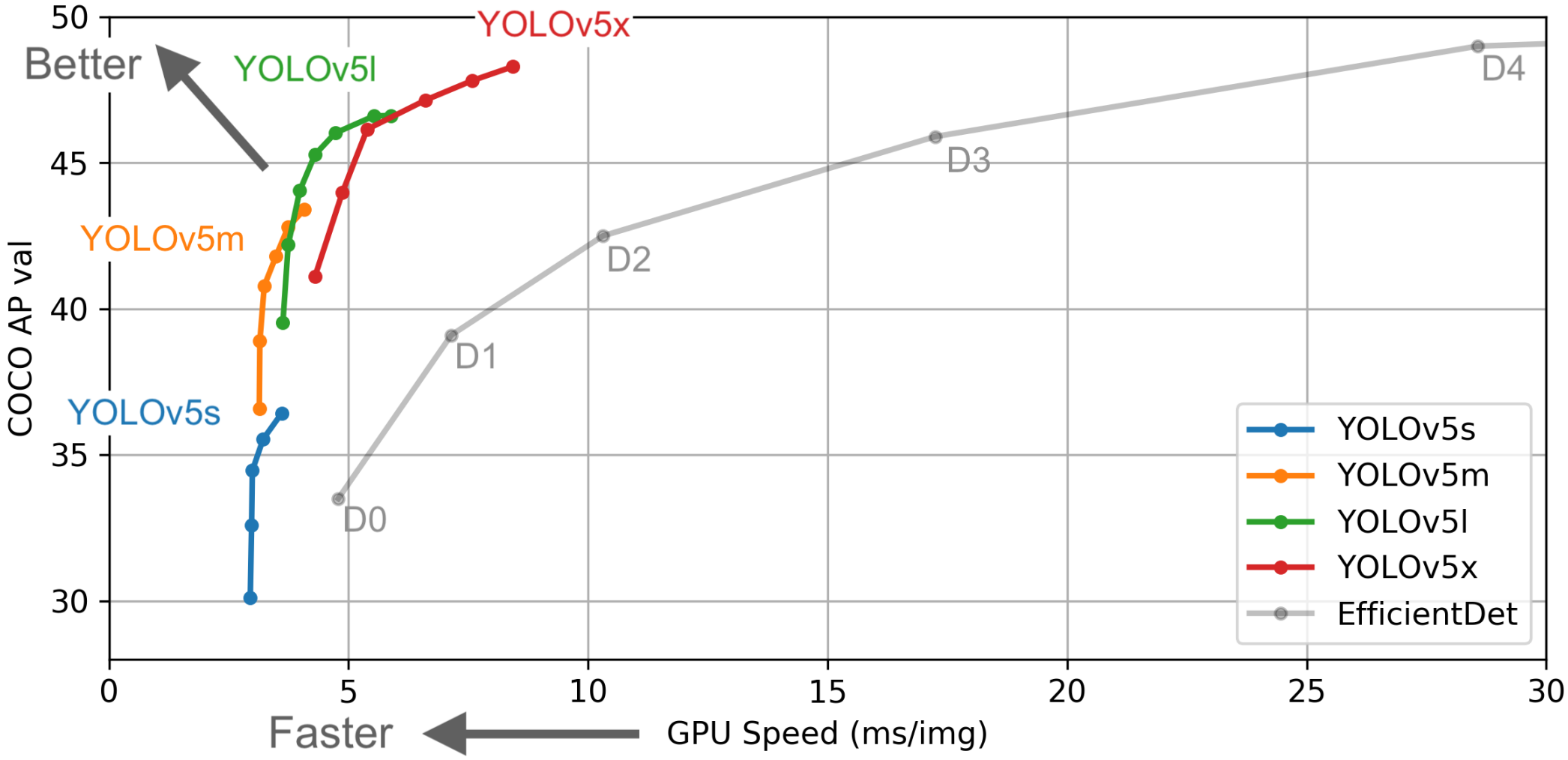
- **YOLOv5s** is about **88% smaller** than big-YOLOv4 (27 MB vs 244 MB).

ultralytics /yolov5 :
https://github.com/ultralytics/yolov5

YOLOV5

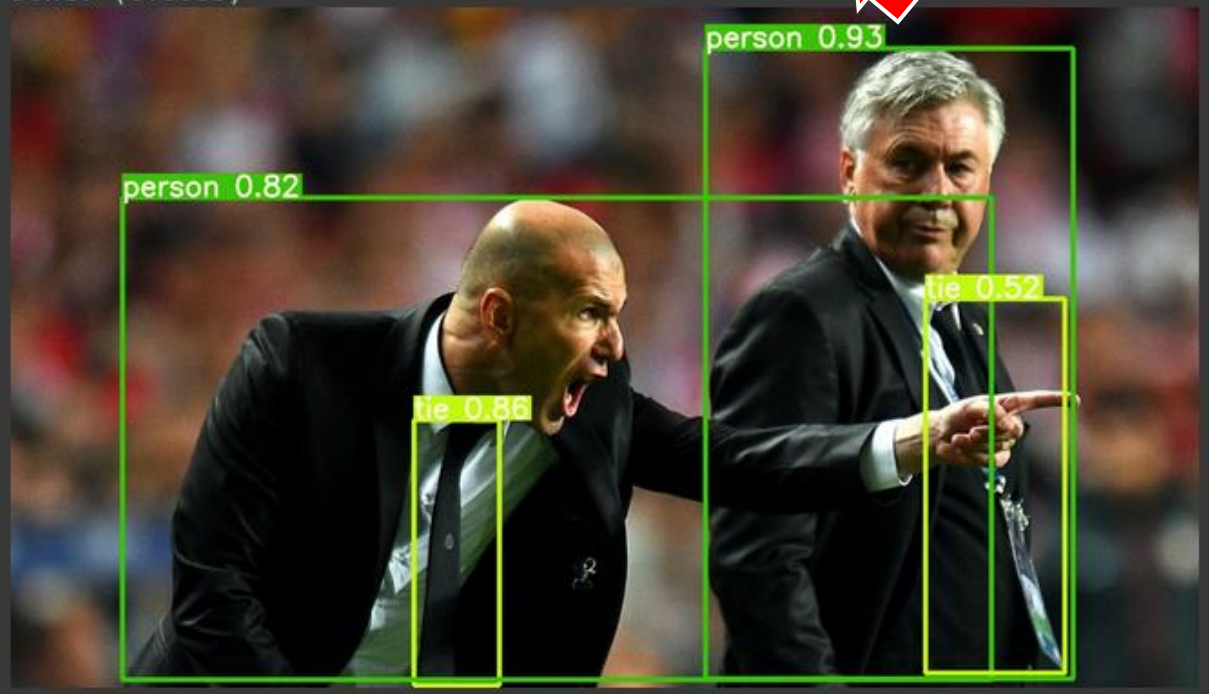

Model Storage Size (mb)
Model Storage Size (mb)
400
300
200
100
0
YOLOv4-custom
YOLOv5x
YOLOv5l
YOLOv5m
YOLOv5s

YOLOV5

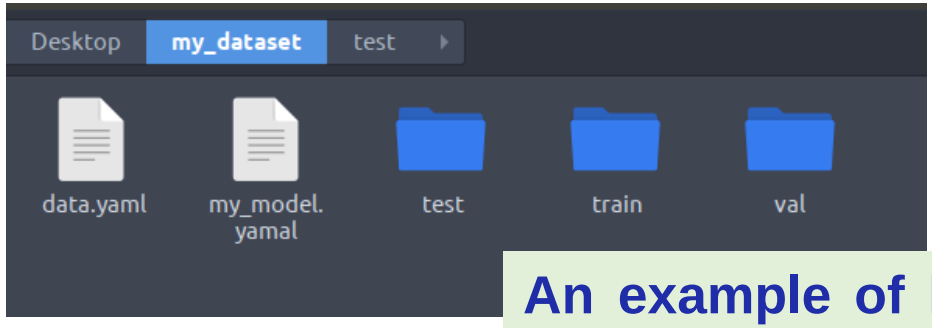

50
45
40
35
30
Better
YOLOv5x
YOLOv5l
YOLOv5m
YOLOv5s
D0
D1
D2
D3
D4
COCO AP val
0
5
10
15
20
25
30
Faster
GPU Speed (ms/img)
YOLOv5s
YOLOv5m
YOLOv5l
YOLOv5x
EfficientDet

Inference - Example

```
10

Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', img_size=640, iou_thres=0.5, output='infere
Using CUDA device0 _CudaDeviceProperties(name='Tesla K80', total_memory=11441MB)

Fusing layers... Model Summary: 284 layers, 8.89222e+07 parameters, 8.45317e+07 gradients
image 1/2 /content/yolov5/inference/images/bus.jpg: 640x512 4 persons, 1 buss, Done. (0.196s)
image 2/2 /content/yolov5/inference/images/zidane.jpg: 384x640 2 persons, 2 ties, Done. (0.138s)
Results saved to /content/yolov5/inference/output
Done. (0.688s)
```

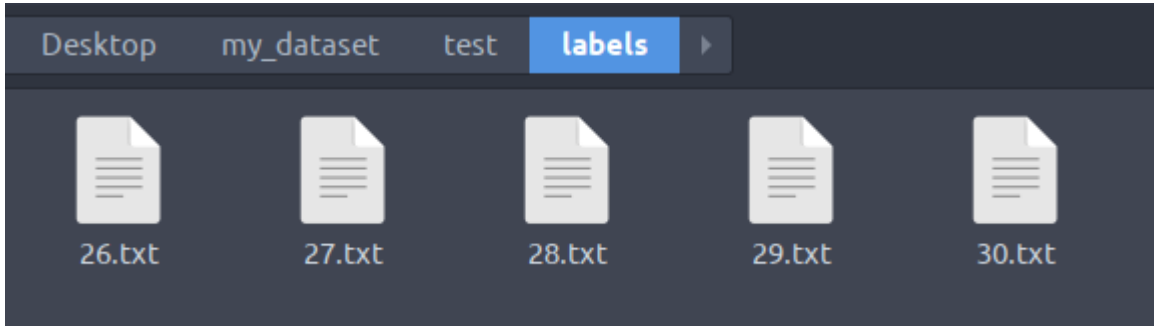


Train YOLOV 5On a Custom Dataset

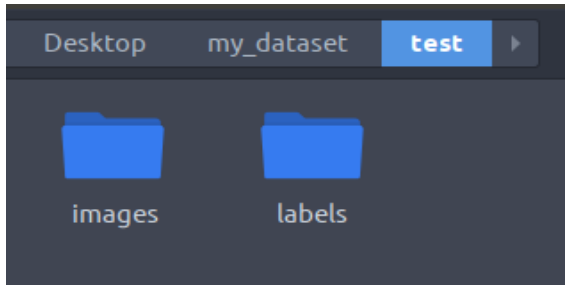


My dataset

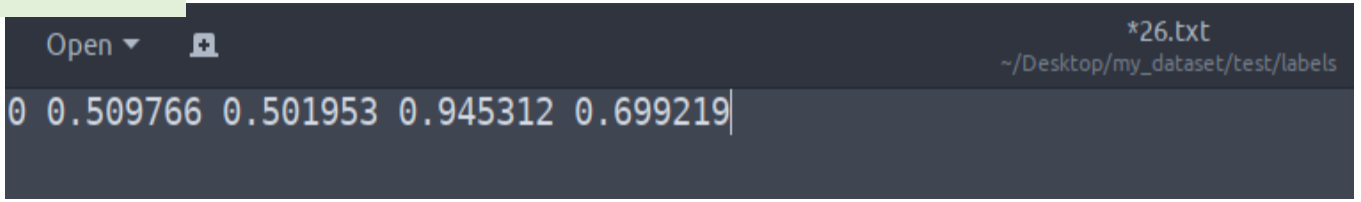
An example of how to organize a dataset



Test labels

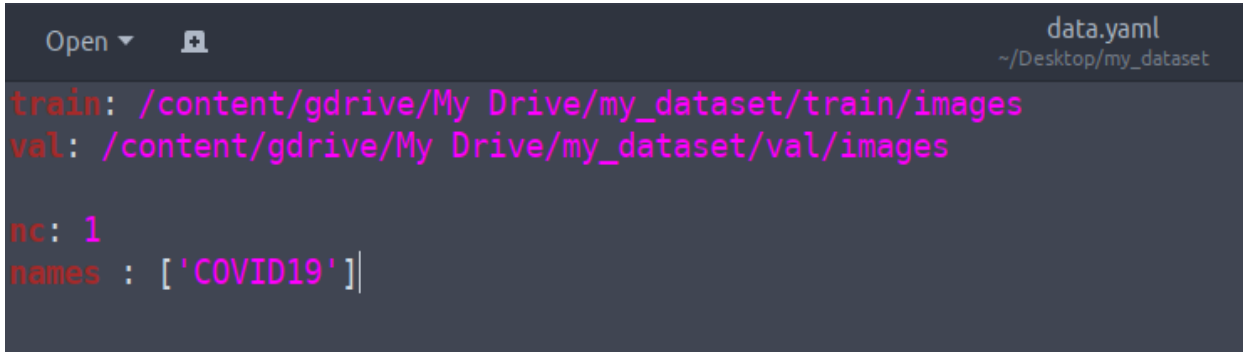


Test set



label

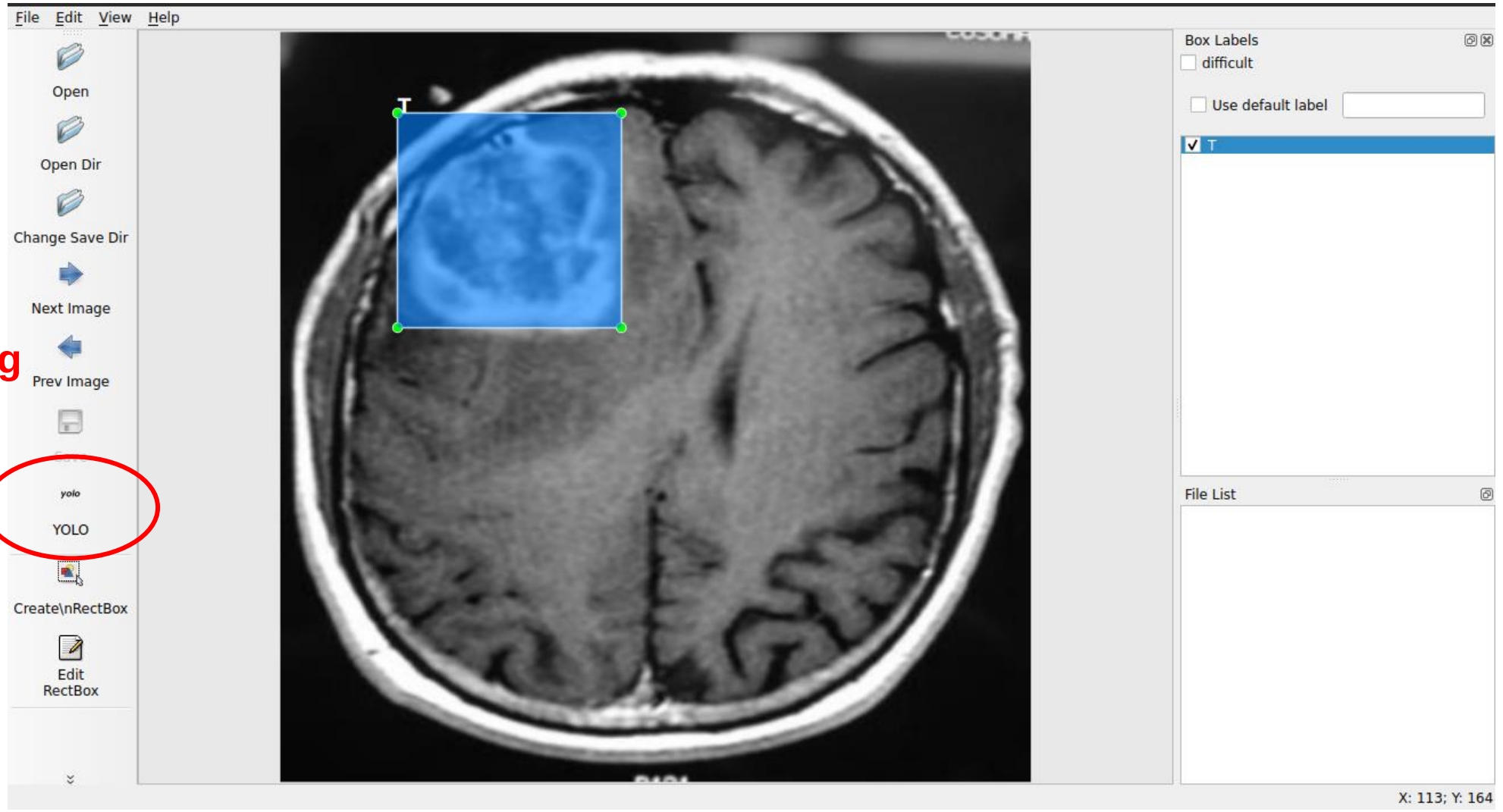
Test images



data.yaml

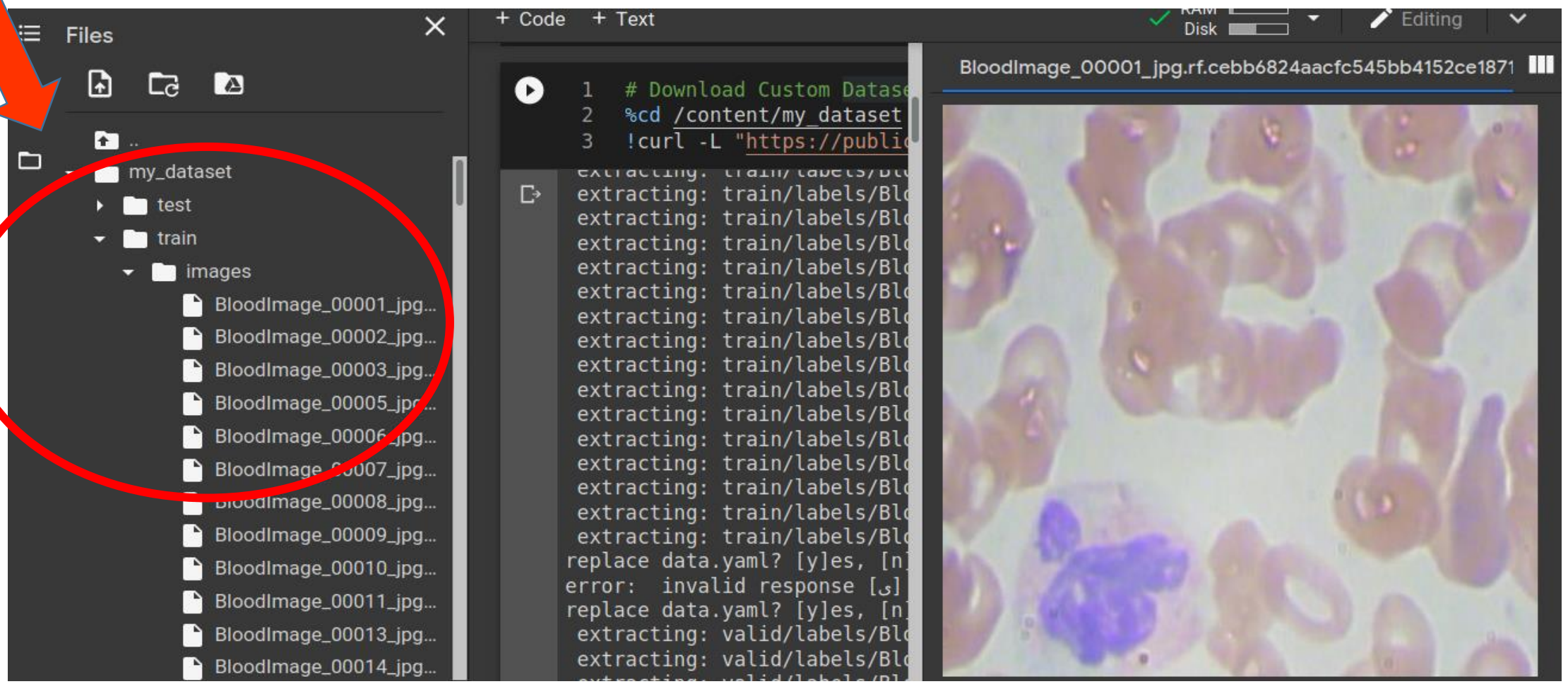
Train YOLOV 5On a Custom Dataset

labelImg
YOLO



Train YOLOV 5On a Custom Dataset

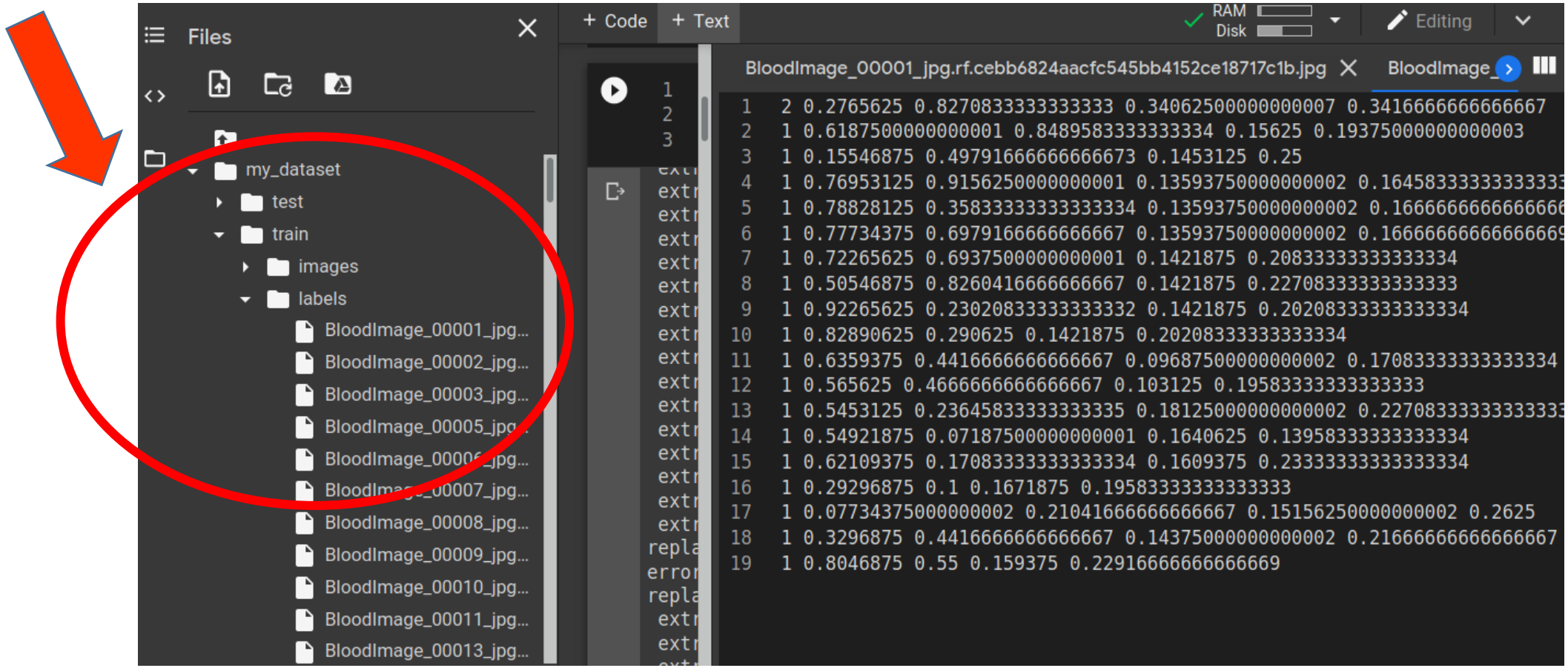
Custom object detection data in YOLOV5 format - image



public blood cell detection dataset

Train YOLOV 5On a Custom Dataset

Custom object detection data in YOLOv5 format - label



Label: class bx by bh bw.txt

Train YOLOV 5On a Custom Dataset

Download custom object detection data in YOLOV5 format from Roboflow.

```
# Download Custom Dataset
%cd /content/my_dataset
!curl -L "YOUR LINK HERE" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

```

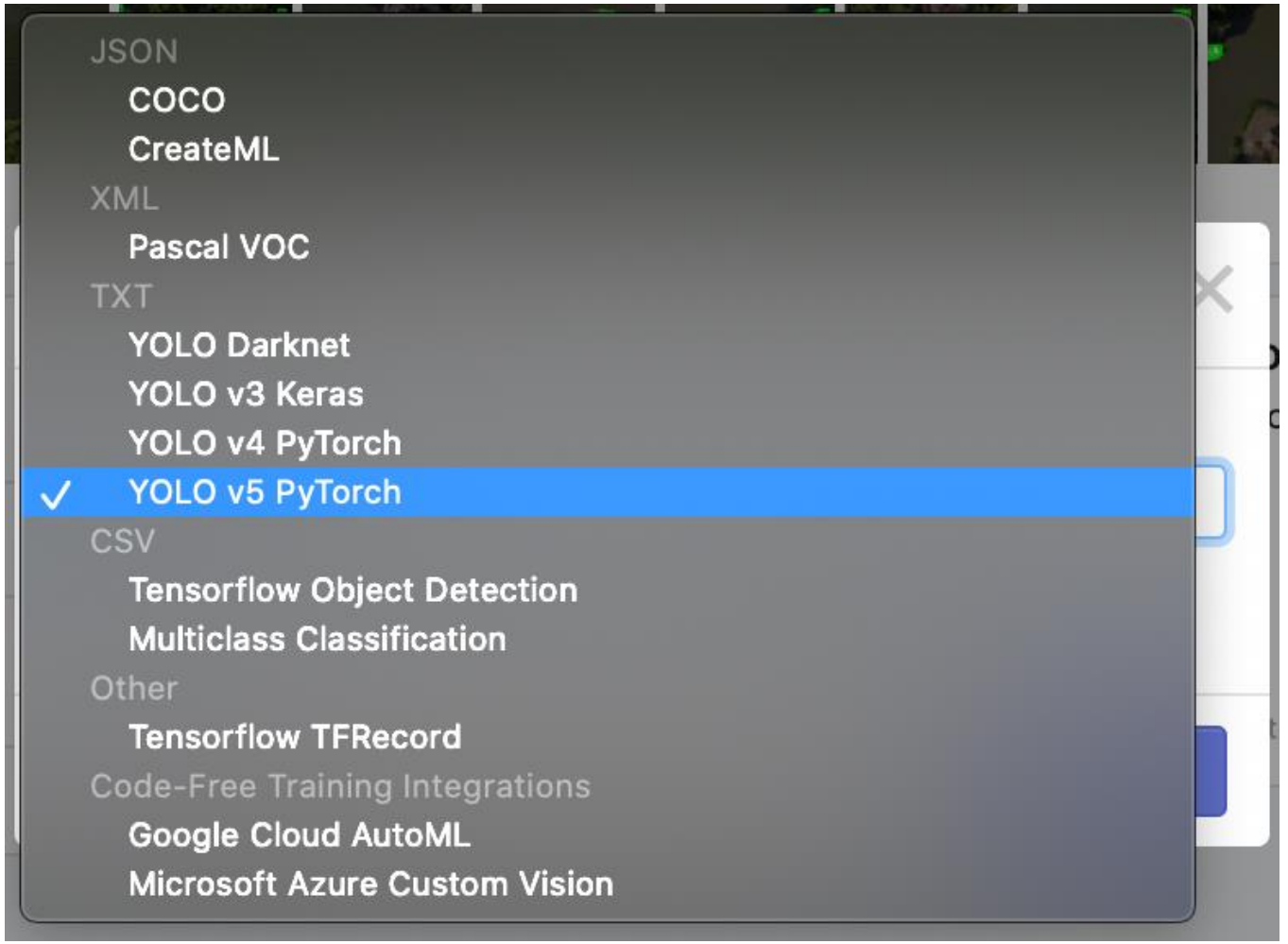
 # Download Custom Dataset
%mkdir /content/my_dataset/
%cd /content/my_dataset
!curl -L "YOUR LINK HERE" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip

Roboflow simplifies your computer workflow from data organization, annotation verification, preprocessing, augmentations to exporting to your required model format. *https://app.roboflow.ai/login*

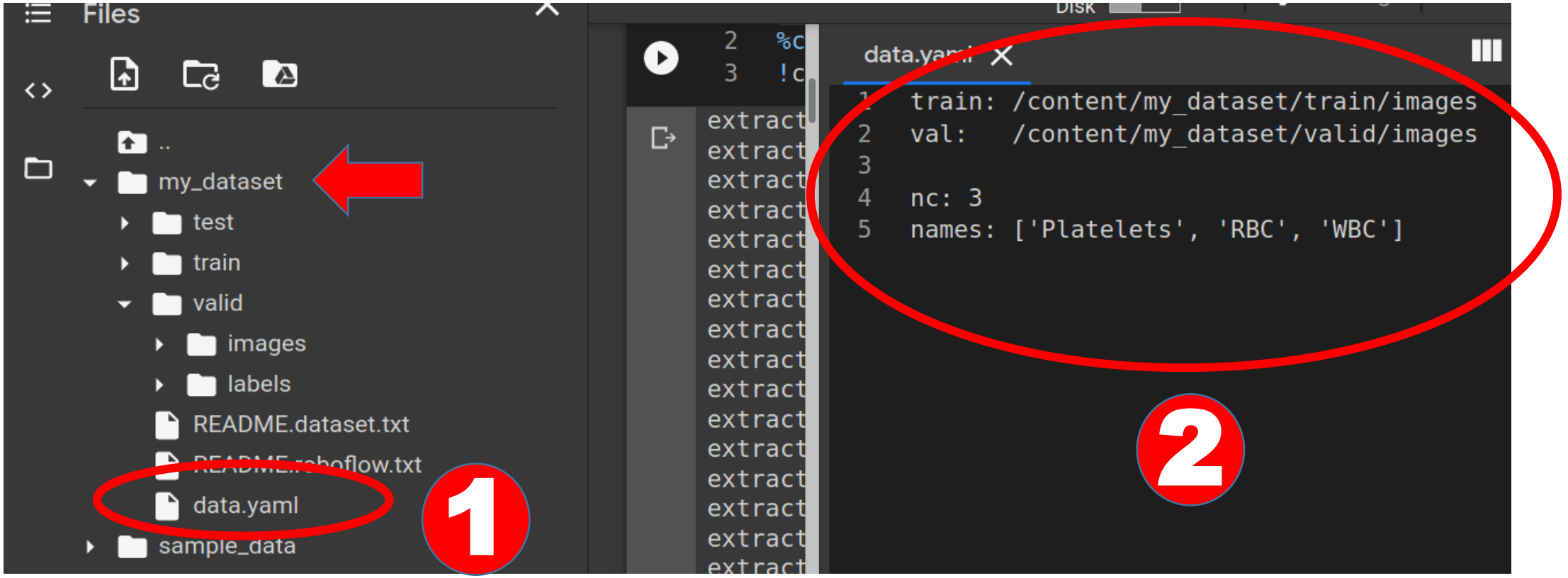
Getting Started with Roboflow : *https://blog.roboflow.com/getting-started-with-roboflow/*

Train YOLOV 5On a Custom Dataset

Use the "**YOLOv5 PyTorch**" export format. Note that the Ultralytics implementation calls for a **YAML** file defining where your training and test data is. The Roboflow export also writes this format for us.



Define Model Configuration and Architecture



data.yaml file is for specifying the location of a YOLOv5 images folder, a YOLOv5 labels folder, and information on our custom classes.

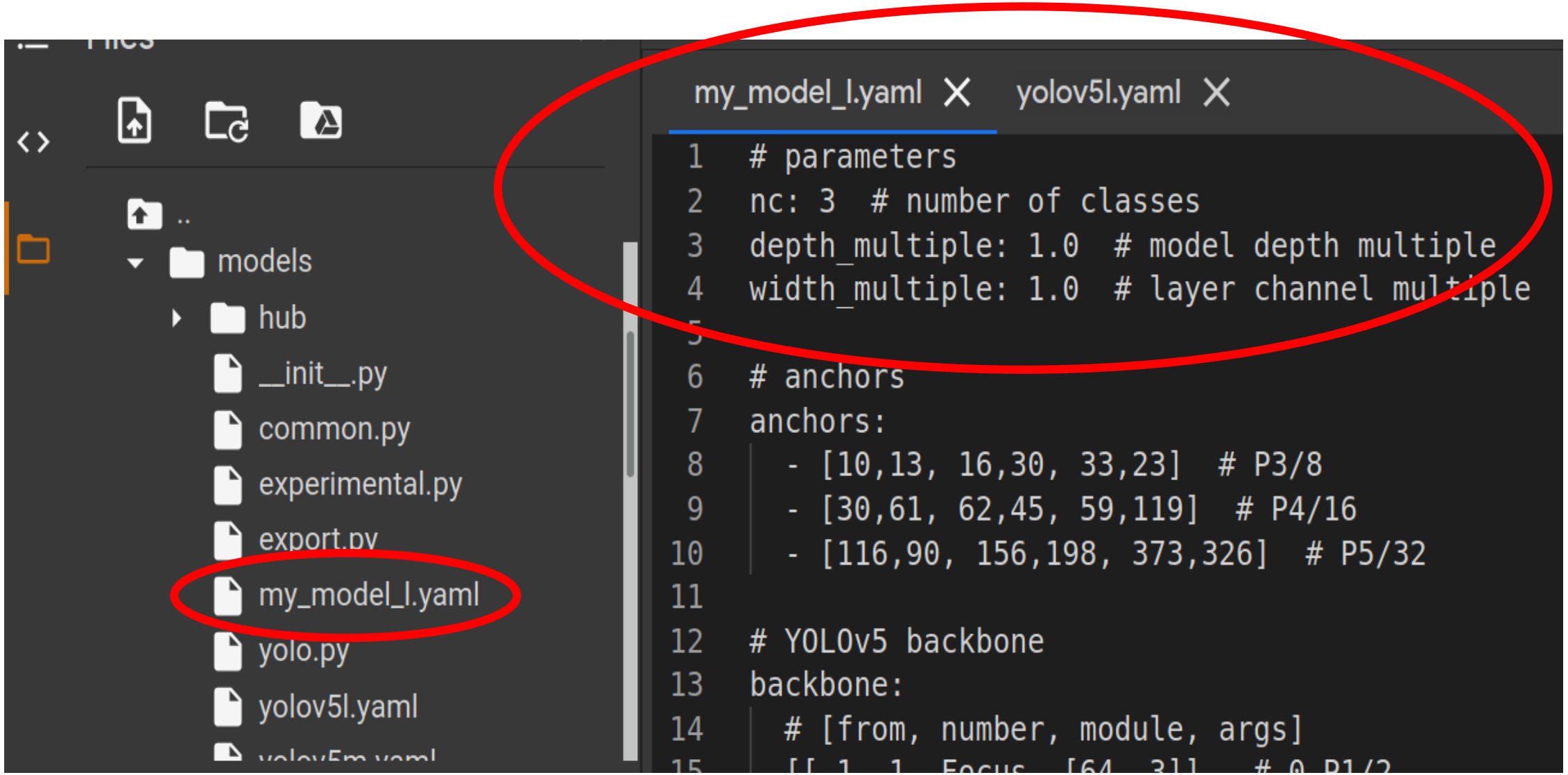
Define Model Configuration and Architecture

1) We will write a new yaml script file (exp: **my_model_l.ymal**) that defines the parameters for our model like **the number of classes**, **anchors**, and **each layer**.

2) Copy then paste the script from one of the files (**yolov5s, m, l, x**) into your ymal script file (exp: **my_model_l.ymal**).

3) Update the parameters.

Define Model Configuration and Architecture



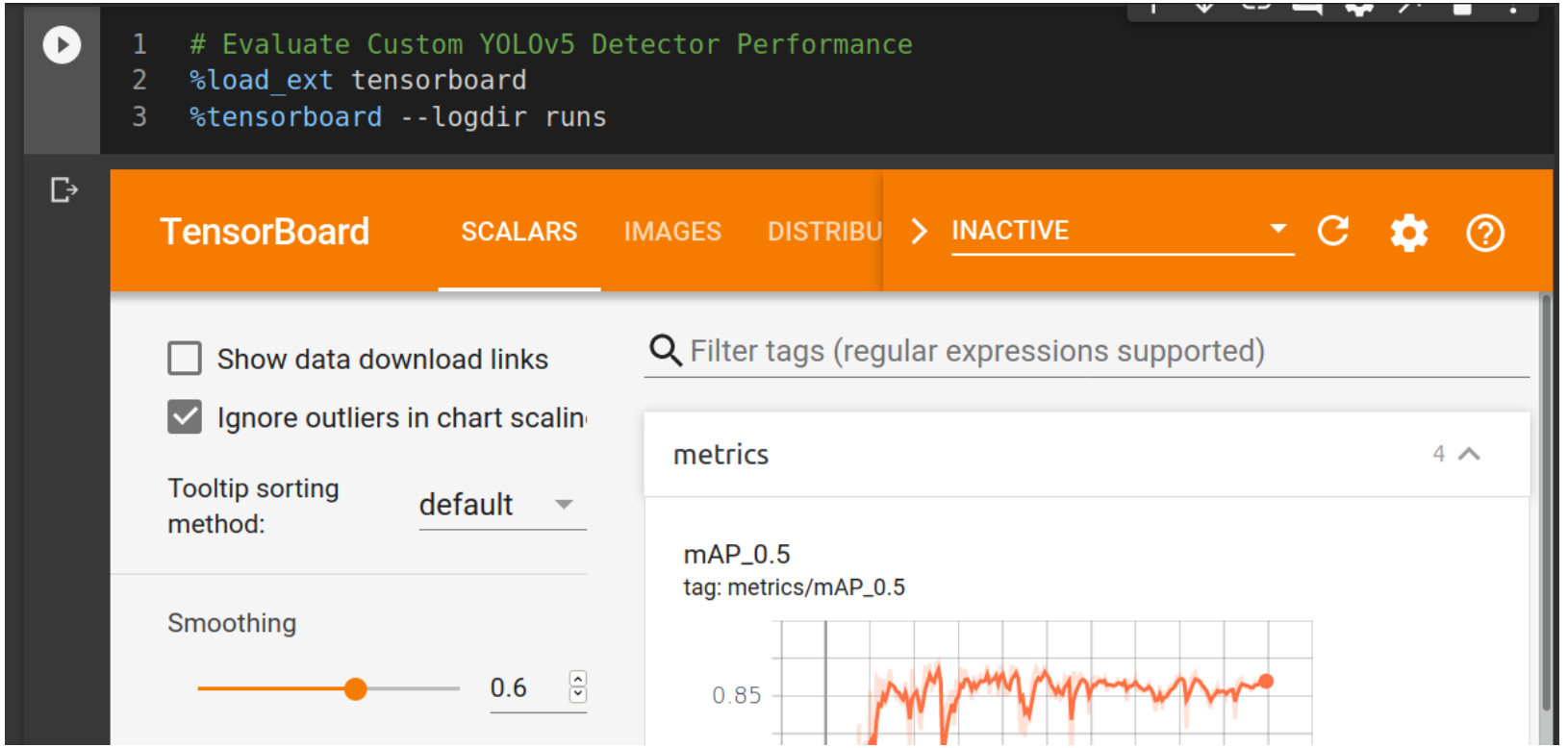
Train YoloV5 on Custom Dataset

```
# Train Yolov5 on Custom Dataset
# weights :
# https://drive.google.com/drive/folders/1Drs_Aiu7xx6S-ix95f9kNsA6ueKRpN2J

%%time
%cd /content/yolov5/
!python train.py --img 416 --batch 16 --epochs 200 --data
      /content/my_dataset/data.yaml --cfg models/my_model_l.yaml
                --weights yolov5l.pt --name my_model_l_results  --cache
```

Train YoloV5 on Custom Dataset

- **img:** define input image size
- **batch:** determine batch size
- **epochs:** define the number of training epochs. (Note: often, 3000+ are common here!)
- **data:** set the path to our yaml file
- **cfg:** specify our model configuration
- **weights:** specify a custom path to weights. (Note: you can download weights from the Ultralytics Google Drive folder)
- **name:** result names
- **nosave:** only save the final checkpoint
- **cache:** cache images for faster training

Evaluate Custom YOLOV5 Detector Performance




```
# Evaluate Custom YOLOV5 Detector Performance
%load_ext tensorboard
%tensorboard --logdir runs

# we can also output some older school graphs if the tensor board isn't working for whatever reason...
Image(filename='/content/yolov5/runs/exp0_my_model_l_results/results.png', width=1000)  # view results.png
```

Run Inference With Trained Weights

```
# Inference 1 image :
%cd /content/yolov5/
!python detect.py --weights /content/yolov5/runs/exp0_my_model_l_results/weights/best.pt --
img 416 --conf 0.5 --source
/content/my_dataset/test/images/BloodImage_00038_jpg.rf.63da20f3f5538d0d2be8c4633c703
4a1.jpg
```

```
# display 1 image :
import torch
from IPython.display import Image, clear_output  # to display images
Image(filename='/content/yolov5/inference/output/BloodImage_00038_jpg.rf.63da20f3f5538d0d
2be8c4633c7034a1.jpg', width=600)
```

Run Inference With Trained Weights

```
# Inference all test images :
%cd /content/yolov5/
!python detect.py --weights
/content/yolov5/runs/exp0_my_model_l_results/weights/best.pt --img 416 --conf
0.5 --source /content/my_dataset/test/images

#display inference on ALL test images
import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/inference/output/*.jpg'):
    display(Image(filename=imageName))
    print("\n")
```

Export Saved YOLOV5 Weights for Future Inference

```
# Export Saved YOLOv5 Weights for Future Inference

from google.colab import drive
drive.mount('/content/gdrive')

%cp /content/yolov5/runs/exp2_my_model_results/weights/best_my_model_results.pt
                                                    /content/gdrive/My\ Drive
```

We recommend following the full code in this YOLOv5 Colab Notebook:
https://colab.research.google.com/drive/1gDZ2xcTOgR39tGGs-EZ6i3RTs16wmzZQ

How to Train YOLOv5 On a Custom Dataset:
https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/

Faster R-CNN

- **Faster R-CNN** (Faster Region-based Convolutional Neural Network) is now a canonical model for deep learning-based object detection. It helped inspire many detection and segmentation models that came after it.

- Faster R-CNN was originally published in NIPS **2015**.

- We can not understand Faster R-CNN without understanding its own predecessors, **R-CNN** and **Fast R-CNN**.

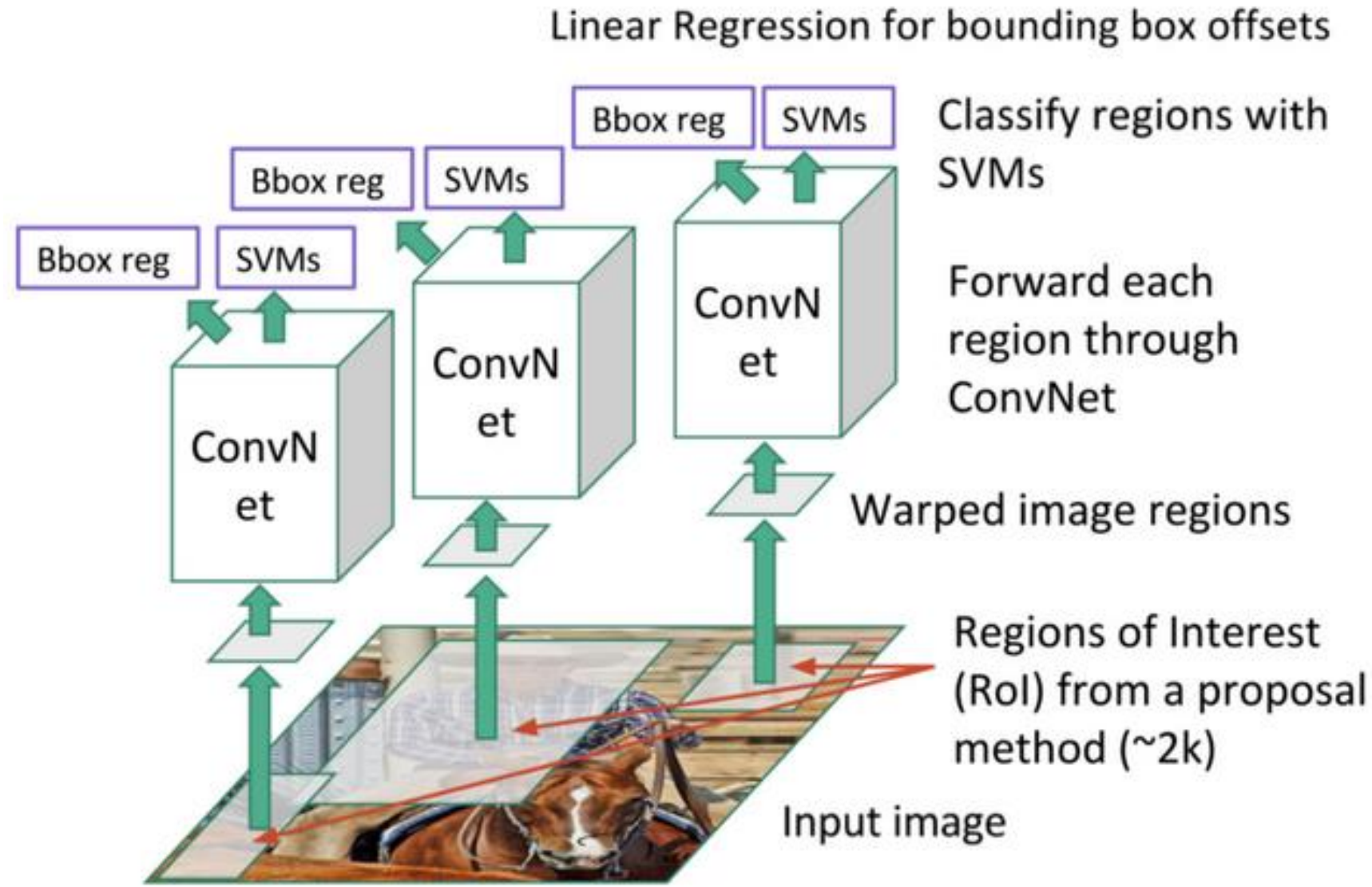
R-CNN

R-CNN (Region-based Convolutional Neural Network) (2014):

- Scan the input image for possible objects using an algorithm called **Selective Search**, generating **~2000 region** proposals.
- Run a convolutional neural net (CNN) on top of each of these region proposals.
- Take the output of each CNN and feed it into **a)** an **SVM** to classify the region and **b)** a **linear regressor** to tighten the bounding box of the object if such an object exists.

Selective Search for Object Recognition: *https://link.springer.com/article/10.1007%252Fs11263-013-0620-5*
R-CNN: *https://arxiv.org/abs/1311.2524*

R-CNN


Linear Regression for bounding box offsets
Bbox reg
SVMs
Classify regions with SVMs
Bbox reg
SVMs
Bbox reg
SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

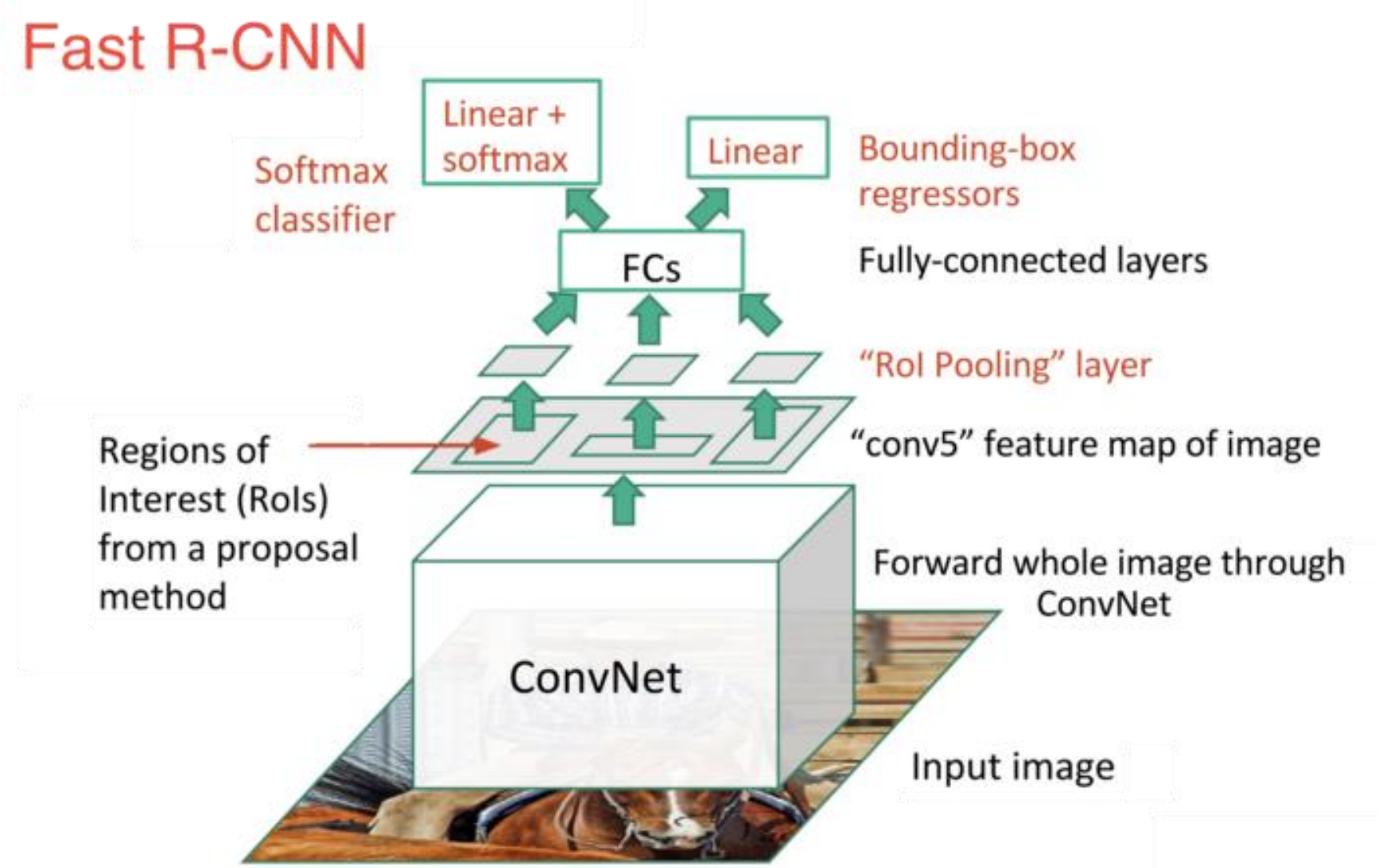
Fast R-CNN

The purpose of the Fast Region-based Convolutional Network (**Fast R-CNN**) is to **reduce the time consumption** related to the high number of models necessary to analyze all region proposals.

- Fast R-CNN developed by Ross Girshick in **2015**.
- Performing feature extraction over the image before proposing regions, thus only **running one CNN over the entire image** instead of 2000 CNN's over 2000 overlapping regions.
- **Replacing the SVM with a softmax layer**, thus extending the neural network for predictions instead of creating a new model.

Fast R-CNN: *https://arxiv.org/abs/1504.08083*

Fast R-CNN


Fast R-CNN
Linear + softmax
Linear
Softmax classifier
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image

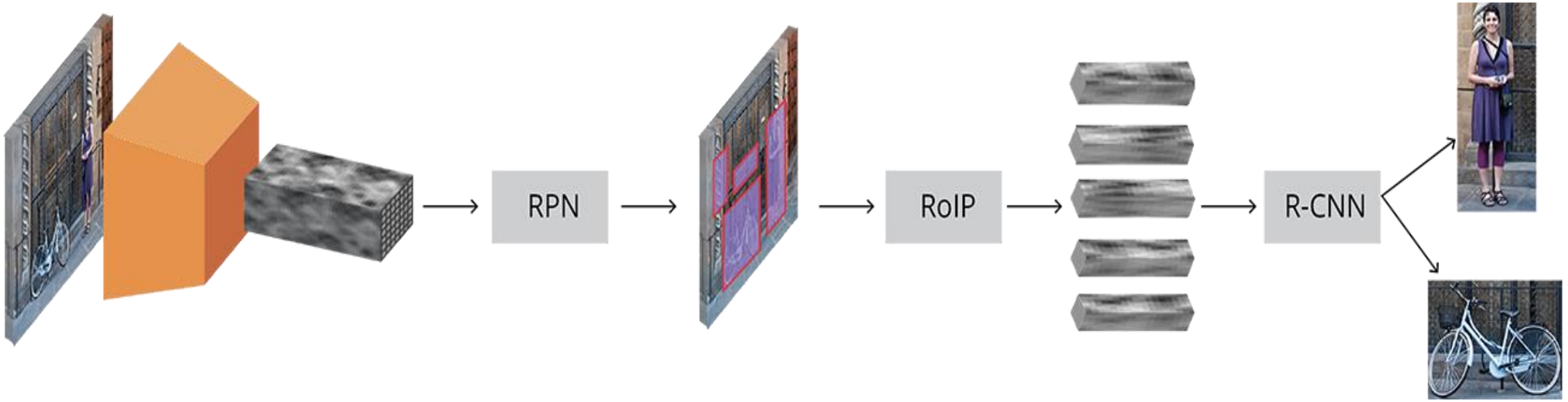
Faster R-CNN

- The input image is passed through a **pre-trained CNN** (The original Faster R-CNN used VGG), then we use **the convolutional feature map** we get for the next part.

- Next, **the Region Proposal Network (RPN)** uses the features that the CNN computed to find up to a **predefined number of regions** (bounding boxes), which may contain objects. It generates multiple possible regions based on **k fixed-ratio anchor boxes** (default bounding boxes). **RPN accelerates the training and testing processes**.

- Finally, comes the **R-CNN** module, which uses that information to: **Classify the content** in the bounding box (or discard it, using “background” as a label). **Adjust the bounding box coordinates** (so it better fits the object).

Faster R-CNN



Faster R-CNN: Down the rabbit hole of modern object detection
https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/
Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
https://arxiv.org/abs/1506.01497

EfficientDet-D7

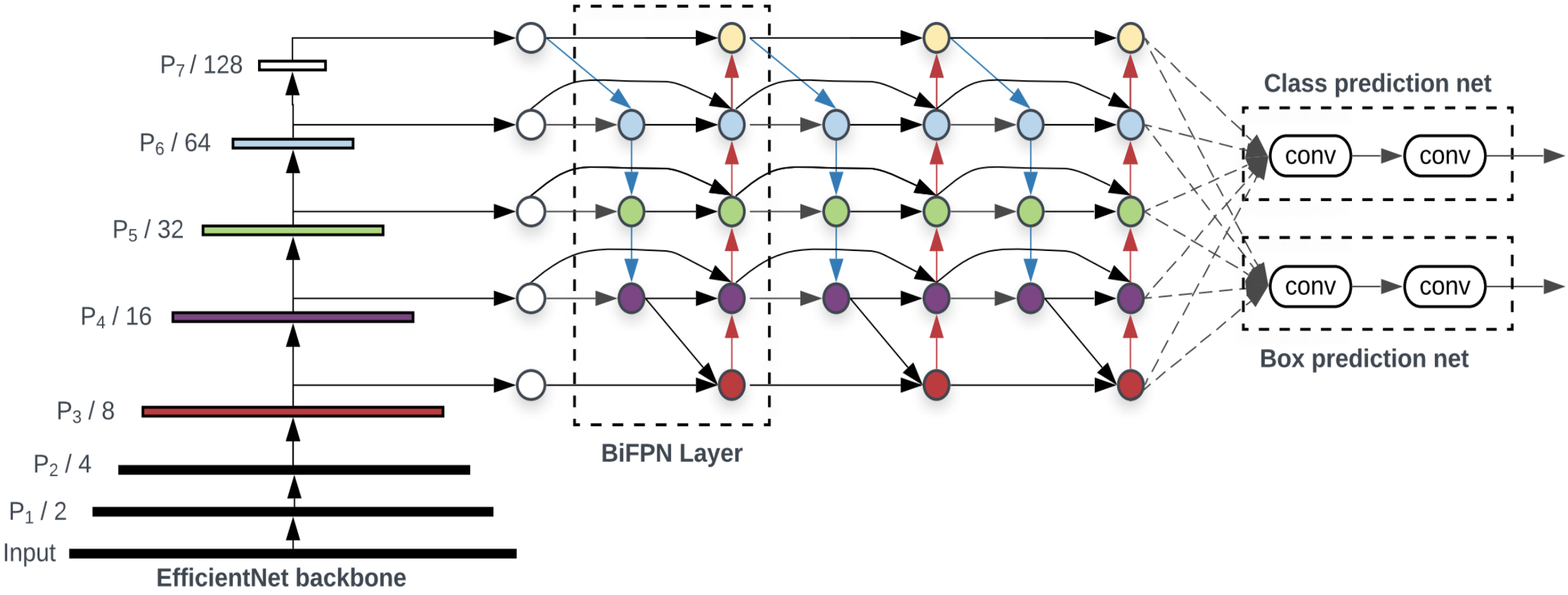
- EfficientDets are a **family of object detector models** that are based on EfficientNet and is reportedly much efficient than other states of the art models.

- In the world of object detection balancing the trade-off between accuracy and performance efficiency of the detectors is a major challenge; EfficientDet attempts to minimize the trade-off and **give the best detector both in terms of accuracy and performance**.

- EfficientDet-D7 achieves state-of-the-art **51.0 mAP** on COCO dataset with **52M parameters** and **326B FLOPS1** , being **4x smaller** and using **9.3x fewer FLOPS**.

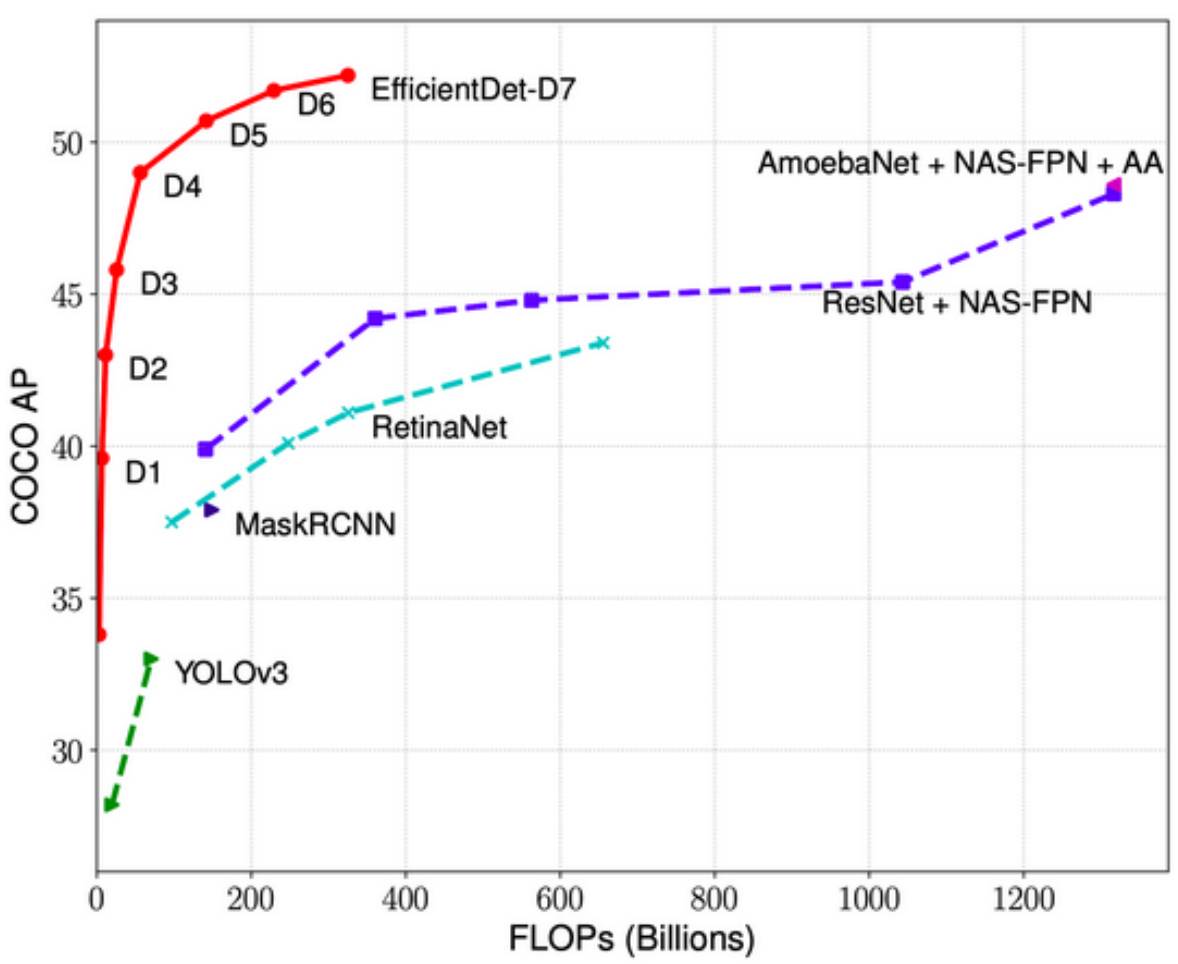
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks: *https://arxiv.org/abs/1905.11946*
EfficientDet: Scalable and Efficient Object Detection : *https://arxiv.org/abs/1911.09070*
Github: *https://github.com/google/automl/tree/master/efficientdet*

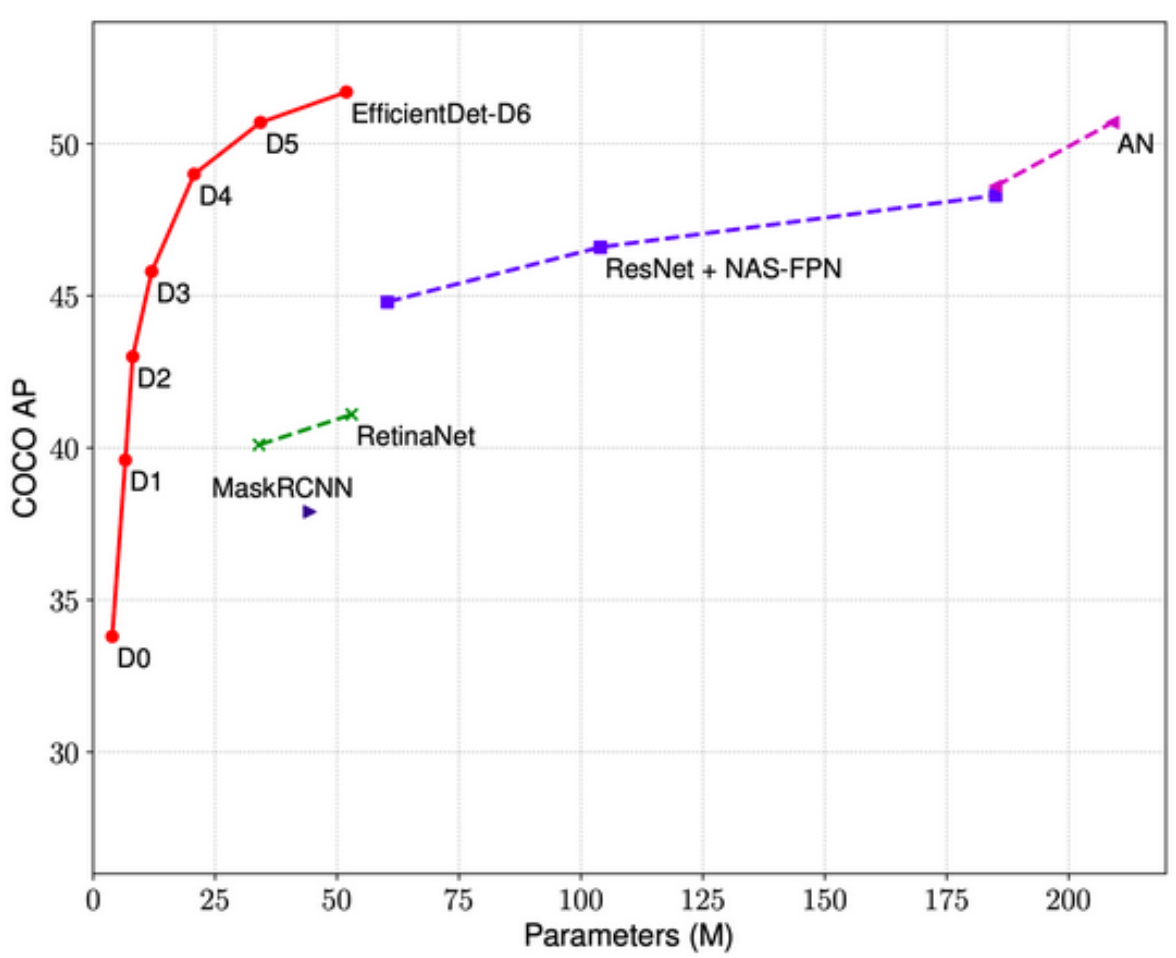
EfficientDet-D7



EfficientDet architecture. EfficientDet uses EfficientNet as the backbone network and a newly proposed bi-directional feature pyramid network (BiFPN) feature network.

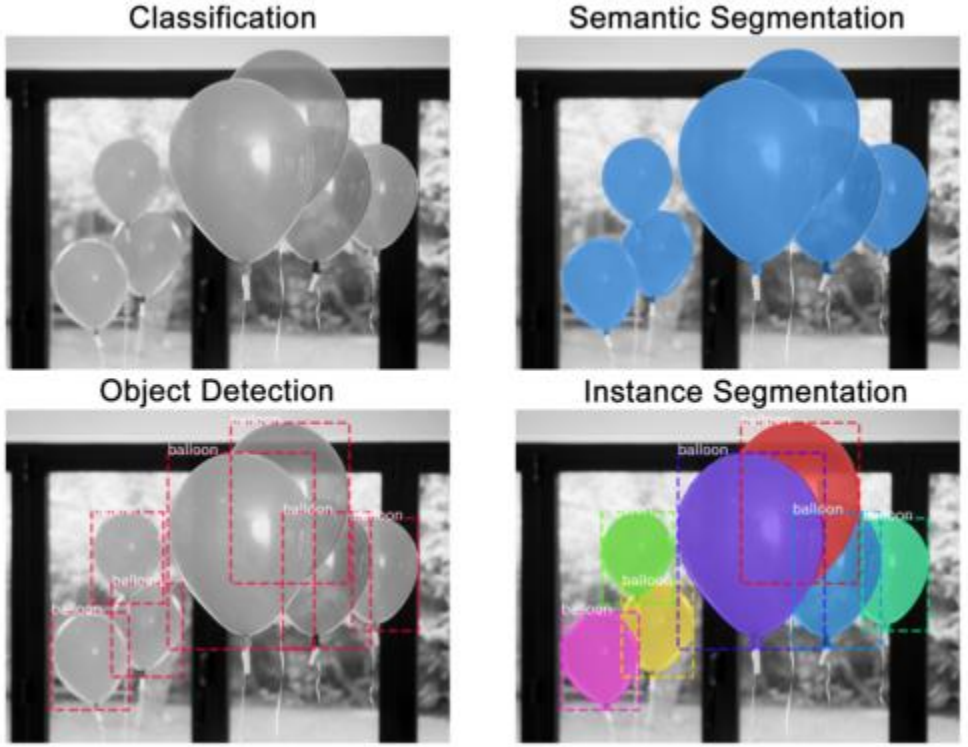
EfficientDet-D7





Instance Segmentation

Instance Segmentation aims to predicting the **object class-label** and the **pixel-specific object** instance-mask, it localizes different classes of object instances present in various images. Instance segmentation aims to help largely robotics, autonomous driving, surveillance, etc.



- **Classification:** There is a balloon in this image.
- **Semantic Segmentation:** These are all the balloon pixels.
- **Object Detection:** There are 7 balloons in this image at these locations. We're starting to account for objects that overlap.
- **Instance Segmentation:** There are 7 balloons at these locations, and these are the pixels that belong to each one.

A Survey on Instance Segmentation: State of the art
https://arxiv.org/abs/2007.00047

Mask R-CNN

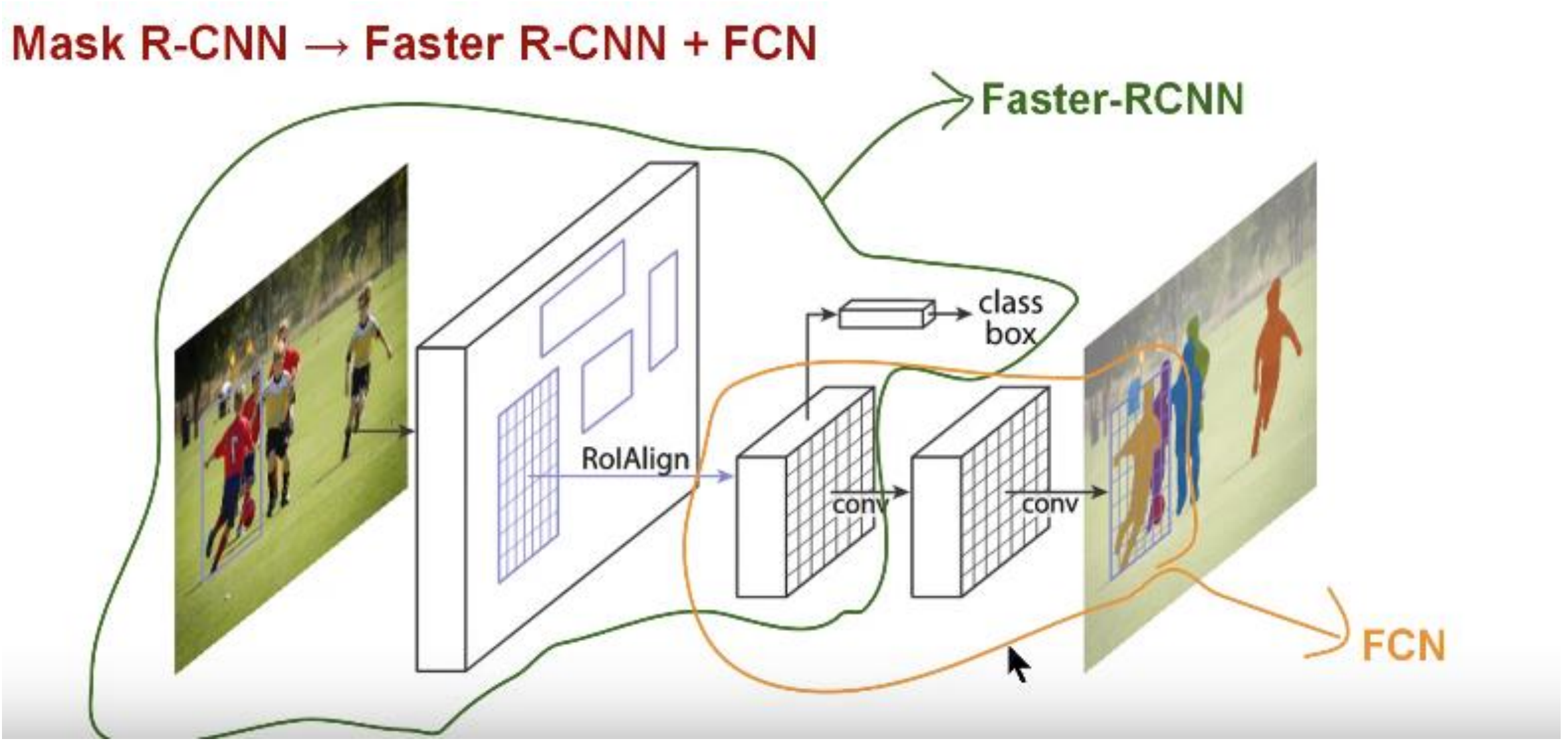
- **Mask R-CNN** (Mask Regional Convolutional Neural Network) is a state of the art model for **instance segmentation**, developed on top of **Faster R-CNN**.

- For a given image, Mask R-CNN, in addition to the class label and bounding box coordinates for each object, will also **return the object mask**.

- The Mask R-CNN model introduced in the **2018** paper titled **"Mask R-CNN"**.

Mask R-CNN: *https://arxiv.org/abs/1703.06870*

Mask R-CNN


Mask R-CNN → Faster R-CNN + FCN
Faster-RCNN
class
box
RoIAlign
conv
conv
FCN

Detectron2

- Detectron2 was built by **Facebook AI** Research (FAIR) to support rapid implementation and evaluation of **novel computer vision research**.
- Detectron2 is now implemented in **PyTorch**.
- Detectron2 is **flexible** and **extensible**, and able to provide fast training on **single or multiple GPU** servers.
- Detectron2 **can be used as a library** to support different projects on top of it.

Detectron2: A PyTorch-based modular object detection library
https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/
Detectron2 :
https://github.com/facebookresearch/detectron2?fbclid=IwAR2CdXQoTU9i-ebKPZIc7BQw8R6NKgp0B-yUkGr1BF3w1VKWzNhxFHi6Zbw
detectron2's documentation: *https://detectron2.readthedocs.io/*

Detectron 2Model Zoo

COCO Object Detection Baselines

Faster R-CNN:

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model | metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model | metrics
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model | metrics
R50-C4	3x	0.543	0.104	4.8	38.4	137849393	model | metrics
R50-DC5	3x	0.378	0.070	5.0	39.0	137849425	model | metrics
R50-FPN	3x	0.209	0.038	3.0	40.2	137849458	model | metrics
R101-C4	3x	0.619	0.139	5.9	41.1	138204752	model | metrics
R101-DC5	3x	0.452	0.086	6.1	40.6	138204841	model | metrics
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model | metrics
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model | metrics

Detectron 2Model Zoo

COCO Instance Segmentation Baselines with Mask R-CNN

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	mask AP	model id	download
R50-C4	1x	0.584	0.110	5.2	36.8	32.2	137259246	model | metrics
R50-DC5	1x	0.471	0.076	6.5	38.3	34.2	137260150	model | metrics
R50-FPN	1x	0.261	0.043	3.4	38.6	35.2	137260431	model | metrics
R50-C4	3x	0.575	0.111	5.2	39.8	34.4	137849525	model | metrics
R50-DC5	3x	0.470	0.076	6.5	40.0	35.9	137849551	model | metrics
R50-FPN	3x	0.261	0.043	3.4	41.0	37.2	137849600	model | metrics
R101-C4	3x	0.652	0.145	6.3	42.6	36.7	138363239	model | metrics
R101-DC5	3x	0.545	0.092	7.6	41.9	37.3	138363294	model | metrics
R101-FPN	3x	0.340	0.056	4.6	42.9	38.6	138205316	model | metrics
X101-FPN	3x	0.690	0.103	7.2	44.3	39.5	139653917	model | metrics

Detectron2 Model Zoo : *https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md*

Detectron2 - Inference

Train Detectron2 on a custom dataset

Convert your data-set to COCO-format :

The COCO dataset is formatted in **JSON** and has five annotation types: object detection, keypoint detection, stuff segmentation, panoptic segmentation, and image captioning. It is a collection of “info”, “licenses”, “images”, “annotations”, “categories” (in most cases), and “segment info” (in one case).

- If your dataset is not in coco format, you have to convert it.

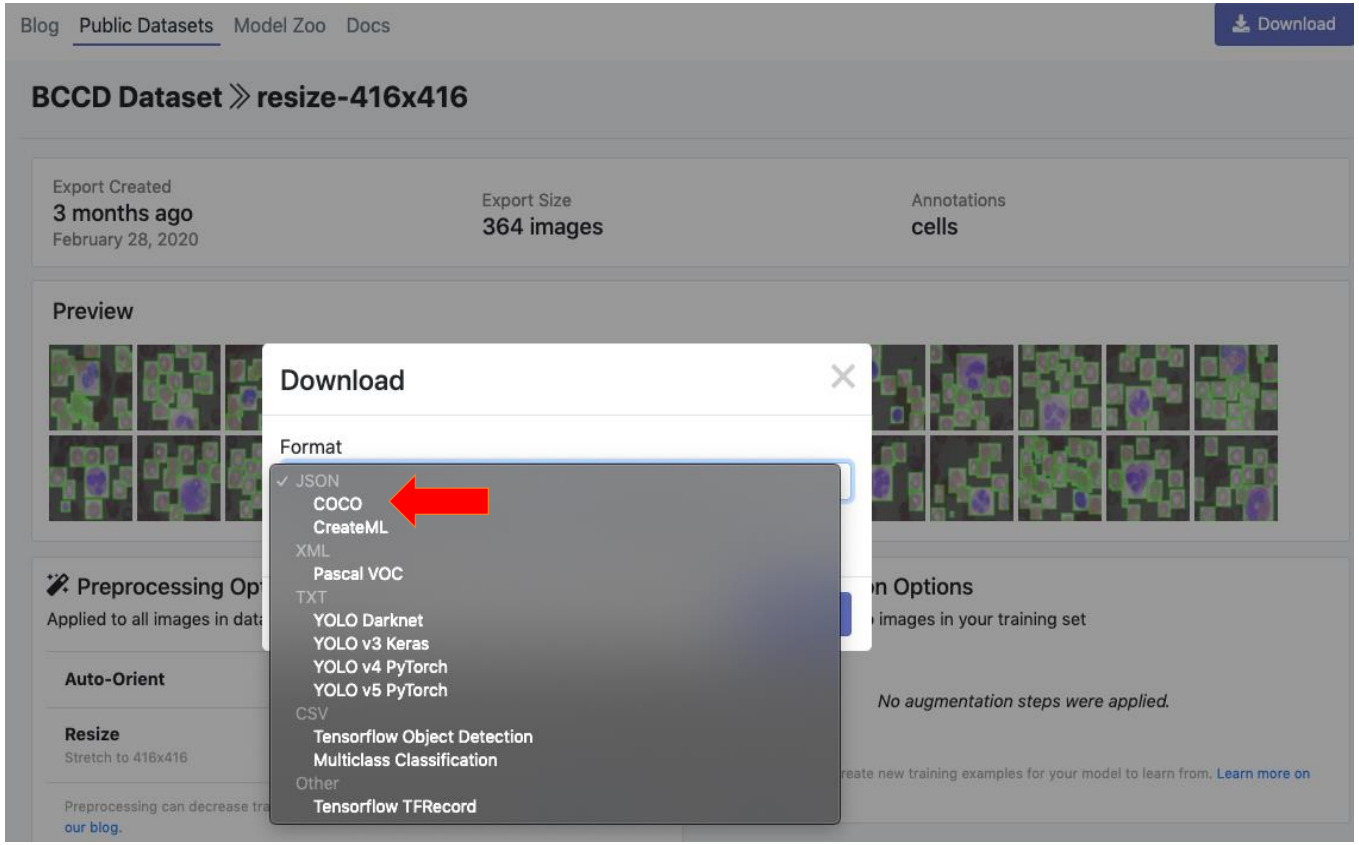
Train Detectron2 on a custom dataset


```
{
    "info": {
        "year": "2020",
        "version": "1",
        "description": "Exported from roboflow.ai",
        "contributor": "Roboflow",
        "url": "https://public.roboflow.ai/object-detection/bccd",
        "date_created": "2020-03-16T01:31:26+00:00"
    },
    "licenses": [
        {
            "id": 1,
            "url": "https://creativecommons.org/publicdomain/zero/1.0/",
            "name": "Public Domain"
        }
    ],
    "categories": [
        {
            "id": 0,
            "name": "cells",
            "supercategory": "none"
        },
        {
            "id": 1,
            "name": "Platelets",
            "supercategory": "cells"
        },
        {
            "id": 2,
            "name": "RBC",
            "supercategory": "cells"
        },
        {
            "id": 3,
            "name": "WBC",
            "supercategory": "cells"
        }
    ],
    "images": [
        {
            "id": 0,
            "license": 1,
            "file_name": "BloodImage_00240_jpg.rf.02e03d6cdaa6a0c4446fbede5b024dd2.jpg",
            "height": 416,
            "width": 416,
            "date_captured": "2020-03-16T01:31:26+00:00"
        },

    "annotations": [
        {
            "id": 0,
            "image_id": 0,
            "category_id": 2,
            "bbox": [
                242,
                240,
                67.60000000000001,
                93.60000000000001
            ],
            "area": 6327.3600000000015,
            "segmentation": [],
            "iscrowd": 0
        }

    ]
}
```


COCO-format

Train Detectron2 on a custom dataset

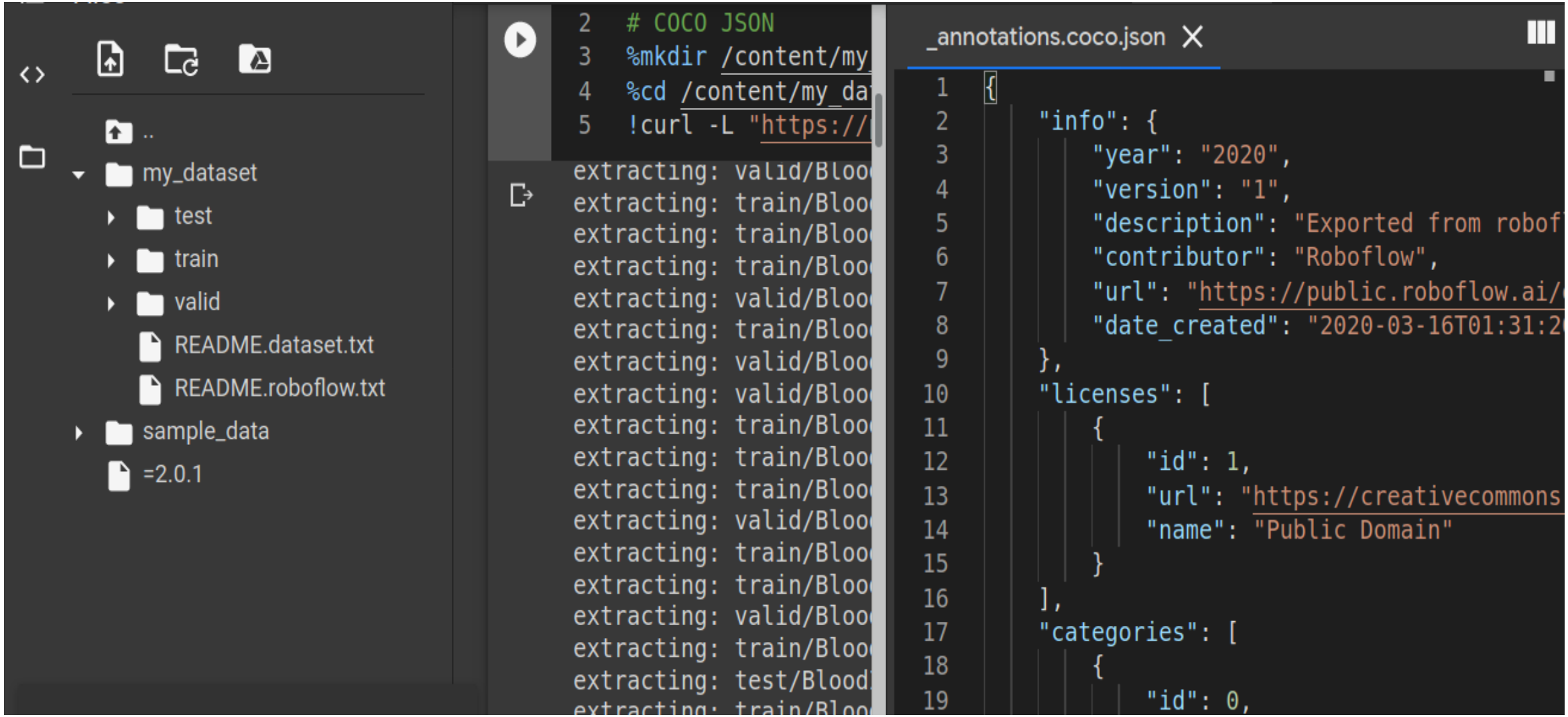
Import and Register Custom Detectron2 Data in COCO JSON format From Roboflow :




```
# Download Custom Dataset
# COCO JSON
%mkdir /content/my_dataset/
%cd /content/my_dataset
!curl -L "{YOUR LINK HERE}" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

Full code : *https://colab.research.google.com/drive/1-TNOcPm3Jr3fOJG8rnGT9gh60mHUsvaW#scrollTo=zVBjf0DE7HEW*

Train Detectron2 on a custom dataset



Public blood cell dataset

Train Detectron2 on a custom dataset

- Detectron2 keeps track of a list of available datasets in a registry, so we must register our custom data with Detectron2 so it can be invoked for training.

```
# Register Custom Detectron2 Data
from detectron2.data.datasets import register_coco_instances
register_coco_instances("my_dataset_train", {}, "/content/my_dataset/train/_annotations.coco.json",
"/content/my_dataset/train")
register_coco_instances("my_dataset_val", {},   "/content/my_dataset/valid/_annotations.coco.json",
"/content/my_dataset/valid")
register_coco_instances("my_dataset_test", {},  "/content/my_dataset/test/_annotations.coco.json",
"/content/my_dataset/test")
```

Train Detectron2 on a custom dataset

```
model_link = "COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file(model_link))
cfg.DATASETS.TRAIN = ("my_dataset_train",)
cfg.DATASETS.TEST  = ("my_dataset_val",)

cfg.DATALOADER.NUM_WORKERS = 4 # Number of data loading threads.
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(model_link)  # Let training initialize from model zoo.
cfg.SOLVER.IMS_PER_BATCH = 4 # Number of images per batch across all machines.
cfg.SOLVER.BASE_LR = 0.001

cfg.SOLVER.WARMUP_ITERS = 1000
cfg.SOLVER.MAX_ITER = 1500 # Adjust up if val mAP is still rising, adjust down if overfit.
cfg.SOLVER.STEPS = (1000, 1500)  # The iteration number to decrease learning rate by GAMMA.
cfg.SOLVER.GAMMA = 0.05

cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 64
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 4 # Your number of classes

cfg.TEST.EVAL_PERIOD = 100 # The period (in terms of steps) to evaluate the model during training.

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Detectron2.config package

https://detectron2.readthedocs.io/modules/config.html

Train Detectron2 on a custom dataset

```
# Test evaluation
from detectron2.data import DatasetCatalog, MetadataCatalog, build_detection_test_loader
from detectron2.evaluation import COCOEvaluator, inference_on_dataset

cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.85
predictor = DefaultPredictor(cfg)
evaluator = COCOEvaluator("my_dataset_test", cfg, False, output_dir="./output/")
val_loader = build_detection_test_loader(cfg, "my_dataset_test")
inference_on_dataset(trainer.model, val_loader, evaluator)
```

```
# Look at training curves in tensorboard

%load_ext tensorboard
%tensorboard --logdir output
```

Train Detectron2 on a custom dataset

```
# Inference & evaluation using the trained model
# cfg already contains everything we've set previously. Now we changed it a little bit for inference:
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.DATASETS.TEST = ("my_dataset_test", )
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7   # set the testing threshold for this model
predictor = DefaultPredictor(cfg)
test_metadata = MetadataCatalog.get("my_dataset_test")

from detectron2.utils.visualizer import ColorMode
import glob

for imageName in glob.glob("/content/my_dataset/test/*jpg")):
  img = cv2.imread(imageName)
  outputs = predictor(img)
  v = Visualizer(img[:, :, ::-1],
                 metadata=test_metadata,
                 scale=0.8
                  )
  out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
  cv2_imshow(out.get_image()[:, :, ::-1])
  print("\n")
```

Train Detectron2 - Save/Load Trained Model

```
# Save model_final.pth
%cp /content/my_dataset/output/model_final.pth  /content/drive/MyDrive/
```

```
my_cfg = get_cfg()
my_cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))
my_cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
my_cfg.MODEL.WEIGHTS = "/content/gdrive/My Drive/model_final.pth"
my_cfg.MODEL.ROI_HEADS.NUM_CLASSES = 4
my_cfg.DATASETS.TEST = ("my_dataset_test", )
my_model_detectron2 = DefaultPredictor(my_cfg)
test_metadata = MetadataCatalog.get("my_dataset_test")

from detectron2.utils.visualizer import ColorMode
import glob
for imageName in glob.glob("/content/my_dataset/test/*jpg"):
  im = cv2.imread(imageName)
  outputs = my_model_detectron2(im)
  print("Classes : \n", outputs["instances"].pred_classes)
  print("Scores : \n", outputs["instances"].scores)

  v = Visualizer(im[:, :, ::-1],
                 metadata=test_metadata,
                 scale=0.8
                  )
  out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
  cv2_imshow(out.get_image()[:, :, ::-1])
  print("\n")
```

Instance Segmentation on a custom dataset

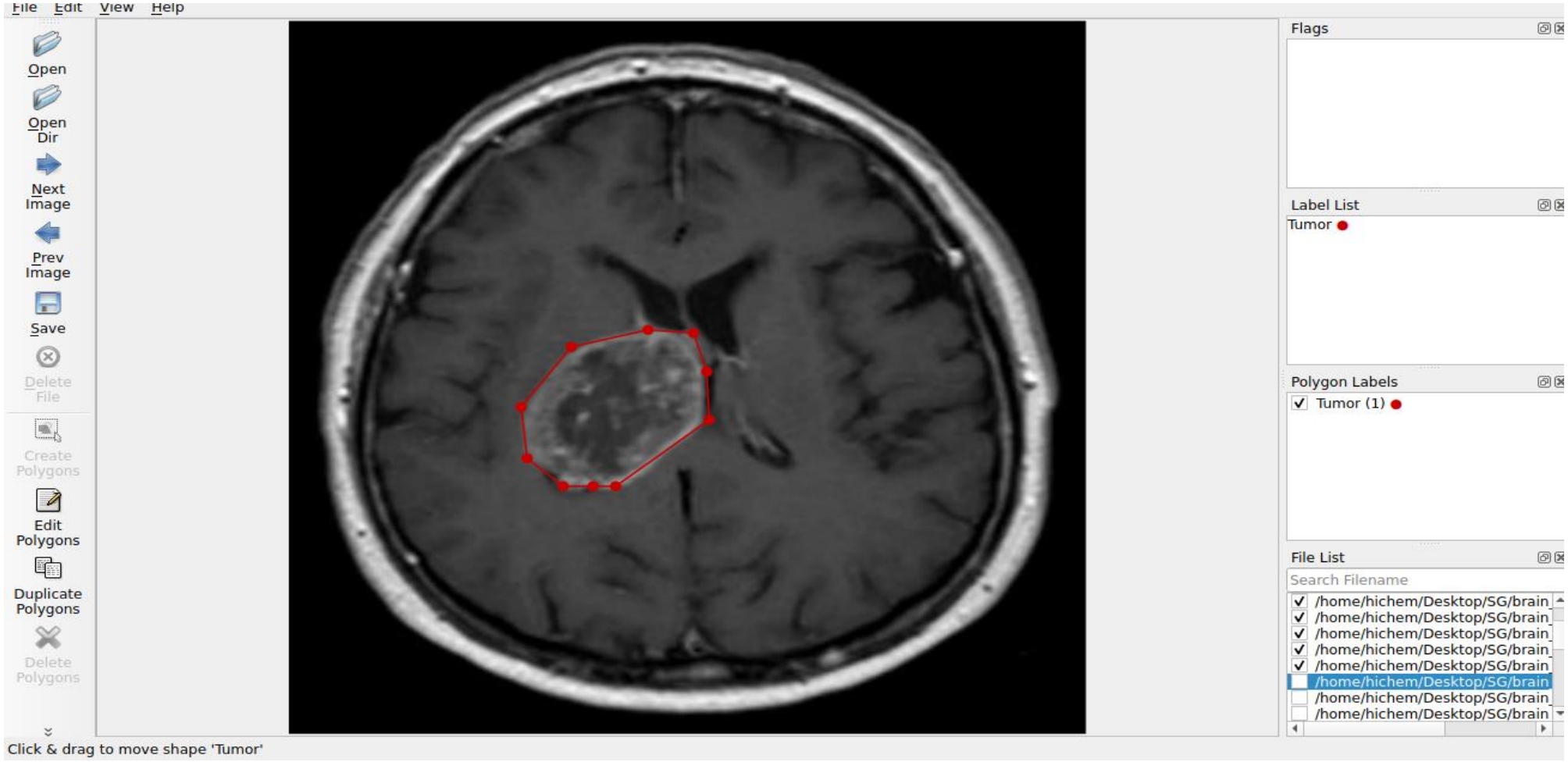
Dataset Gathering :

As always, at the beginning of an AI Project after the problem statement has been identified we move on to gathering the data or in this case images for training.

Data Labeling :

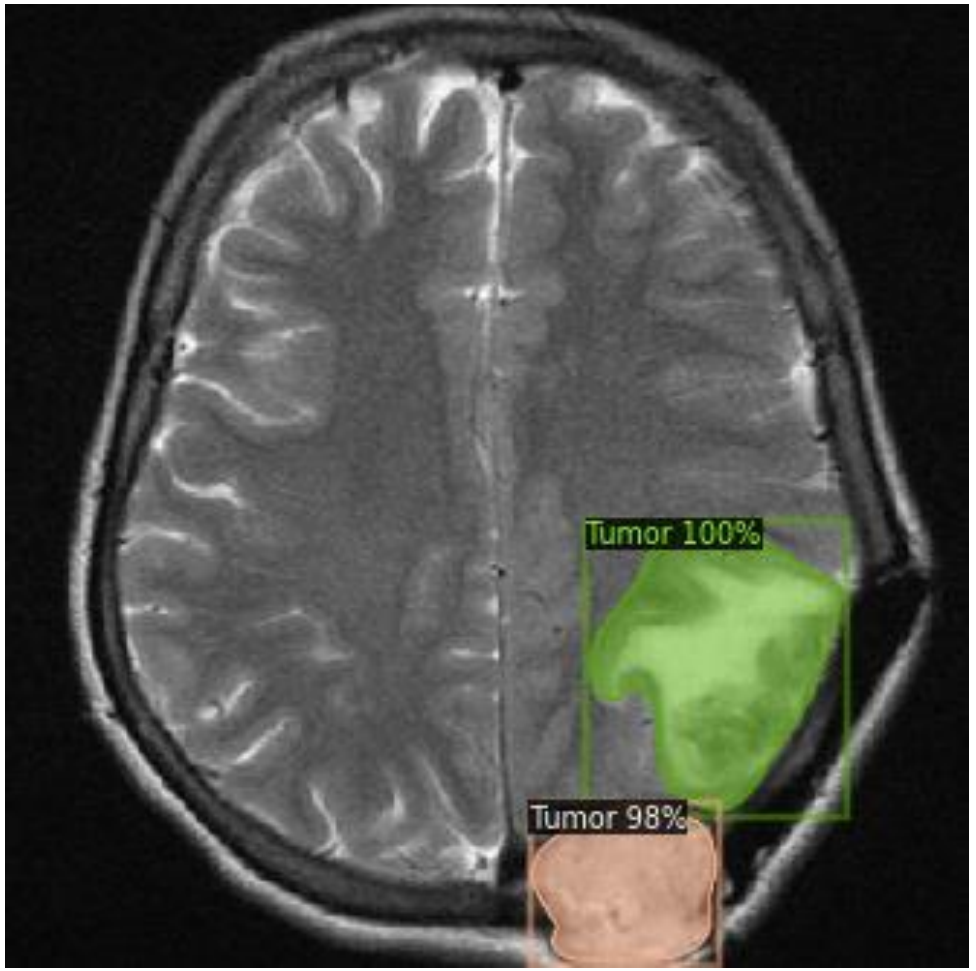
Now that we have gathered the dataset we need to label the images so that the model understands what is the interesting object on the picture. To label the images we need a **labeling software**, for example : **labelme** (**instance segmentation**)

Instance Segmentation on a custom dataset



labelme

Instance Segmentation on a custom dataset



Thanks For Your Attention

Hichem Felouat ...