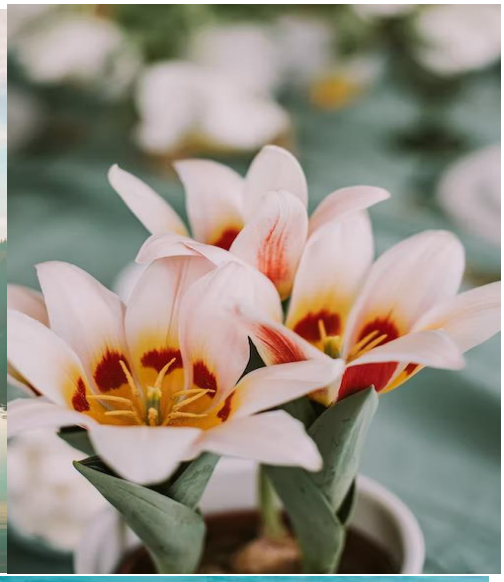
RECENT ADAVANCES IN MACHINE LEARNIG

During this assignment we have mostly seen how a Convolutional network for different task la segmentation or detection. The goal for this project is to firstly use a pretrained model with new data in order to see if this model can really adapt on new images. For this task we took different style of images, hard one from the cifar10 dataset and other with high quality but with several information to detect in it.

The pretrained model YOLOV8 is trained on the COCO dataset which contains several images like bike bear etc. It has a pretty large spectrum of images. We choosed 10 images.



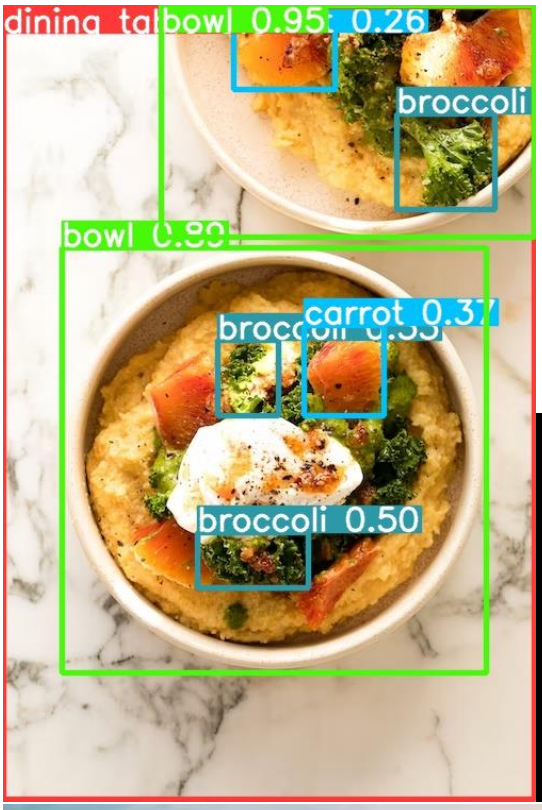
激安
ホッピー
乾杯屋
処方せん




We first passed this image to the prétrained model to see how they will predict this sample

```
img=['bateau','comp','lune','nourriture','personnes','stonehedge_1','tigre_1','lac','fleurs']
for n in img:
  model.predict('/content/{}.jpg'.format(n),save=True,save_txt=True)
```


giraffe 0.76
refrige
person 0.88
handbag 0.42
handbag 0.35


broccoli
carrot
broccoli 0.50

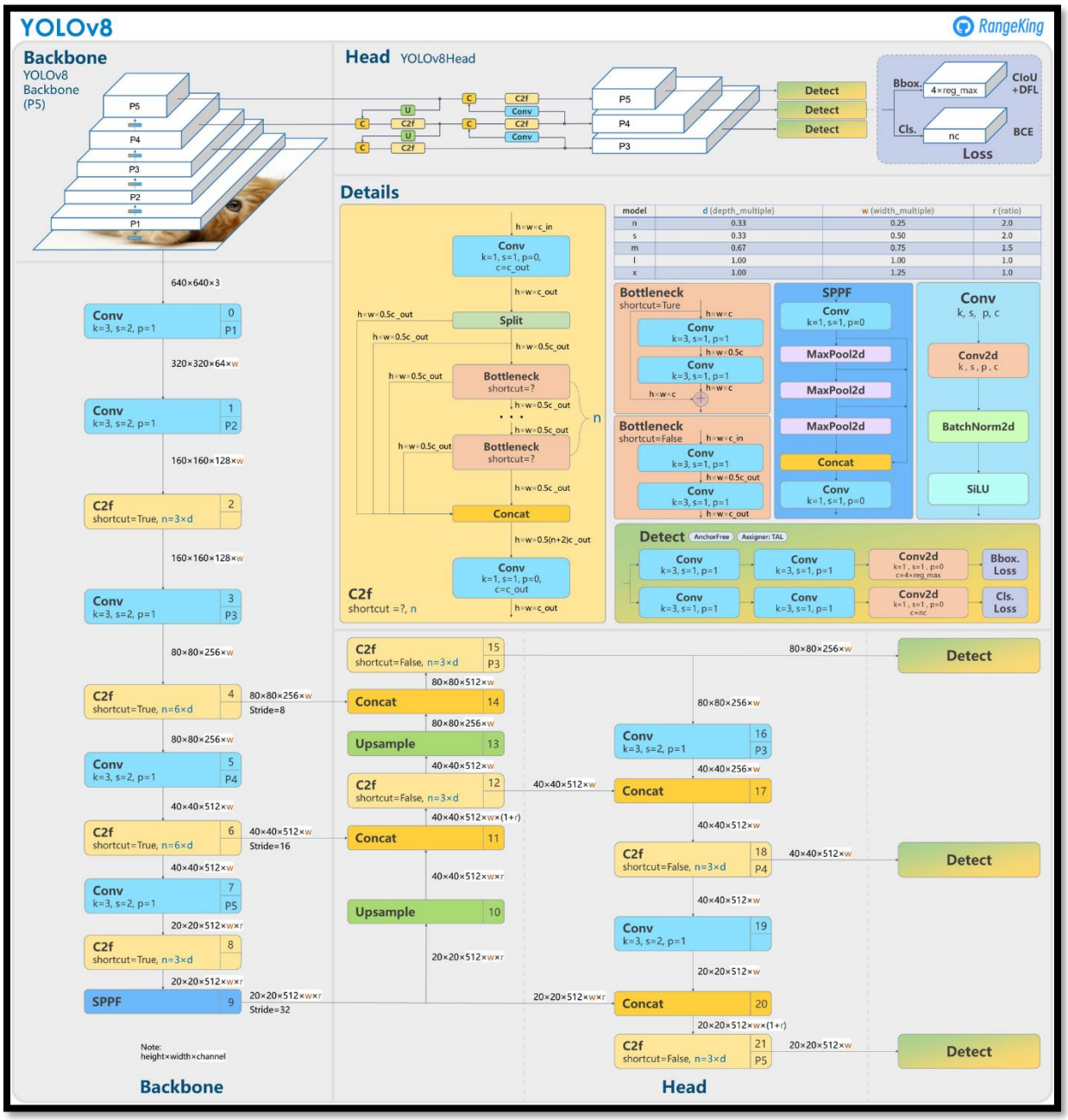

wine glass


potted plant 0.51
vase 0.50
vase 0.76


boat 0.79
boat 0.66

As we can see the images are not well interpreted by the model, and it is normal because some of them are not in the COCO database that it used for the training of YOLOv8.

YOLOv8 Architechture



It is a deep learning algorithm that uses convolutional neural networks (CNNs) to detect and classify objects in images or video frames in real-time.

The Yolov8 architecture consists of a backbone network (Darknet-53), neck (SPP, PAN) and head (YOLOv3). The backbone network is a deep CNN that extracts features from the input image. The neck layers are used to aggregate features from different scales to provide a more comprehensive representation of the image. The head consists of a set of fully connected layers that perform object detection and classification.

The Yolov8 algorithm works by dividing the input image into a grid of cells, and each cell predicts bounding boxes and class probabilities for objects that are present in that cell. Each bounding box is described by four coordinates (x, y, width, height) that define the position and size of the object. Class probabilities are estimated for each bounding box to determine the class of the object (e.g., car, pedestrian, bicycle, etc.). Non-maximum suppression is then used to remove overlapping bounding boxes and retain only the most likely ones.

Yolov8 is a state-of-the-art object detection algorithm that is fast, accurate, and suitable for real-time applications. It has been widely used in various domains, including self-driving cars, robotics, security, and surveillance.

YOLOv8 detection uses a series of building blocks to perform object detection. Here are some of the key building blocks and their characteristics:

1. Convolutional Layers: YOLOv8 uses a series of convolutional layers with different filter sizes and strides to extract features from the input image. The number of convolutional layers and their parameters depend on the specific YOLOv8 model architecture.
2. Activation Function: YOLOv8 uses the Mish activation function in its convolutional layers, which is known to perform well in object detection tasks.
3. Loss Function: YOLOv8 uses a combination of several loss functions to train the model, including the binary cross-entropy loss for object presence prediction, the smooth L1 loss for bounding box regression, and the focal loss for class prediction. The specific combination and weighting of these loss functions depend on the YOLOv8 model architecture.
4. Spatial Pyramid Pooling (SPP): YOLOv8 uses SPP to capture features at multiple scales. SPP allows the model to perform object detection on objects of different sizes and scales.
5. Feature Fusion: YOLOv8 uses feature fusion to combine features from different scales and layers. This allows the model to detect objects with greater accuracy and robustness.
6. Object Detection Head: YOLOv8 uses an object detection head that includes several convolutional layers and fully connected layers to predict object presence, bounding box coordinates, and class probabilities.

225 layers, 3157200 parameters, 3157184 gradients, 8.9 GFLOPs

Training on a custom Dataset

In order to train our own model we'll use different set of images, we will try to predict wether or not there is a tiger or lion in the picture. To achieve thaht the usi,ng of roboflow is vividly recommended.

What is roboflow ?

Roboflow assists programmers and data scientists in building and maintaining computer vision datasets for machine learning models. It offers a simple user interface for annotating photos and videos, converting datasets into other formats, training and analyzing models, and putting models into use. Many computer vision tasks, including object identification, image segmentation, and classification, are supported by Roboflow. A number of well-known deep learning frameworks, including TensorFlow, PyTorch, and Keras, are also integrated. Developers may focus on creating high-quality machine learning models rather than data administration by saving time with Roboflow.

So basically we give robofolow a various dataset of images that contains leopard or tiger and then we annotate it manually. After you can decide how to split your dataset, we use the default config which was 70% training,20%validation,10%test.

Import and training

After we annoted all our images roboflow give you a code in order to run their API to install and train the model.

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="RGBVm9HBvYWRBrPG4RYK")
project = rf.workspace("raml").project("raml")
dataset = project.version(4).download("yolov8")
```

We then run this command to start training the model, note that we modify the epochs because it was to long and doesn't improve the model accuracy.

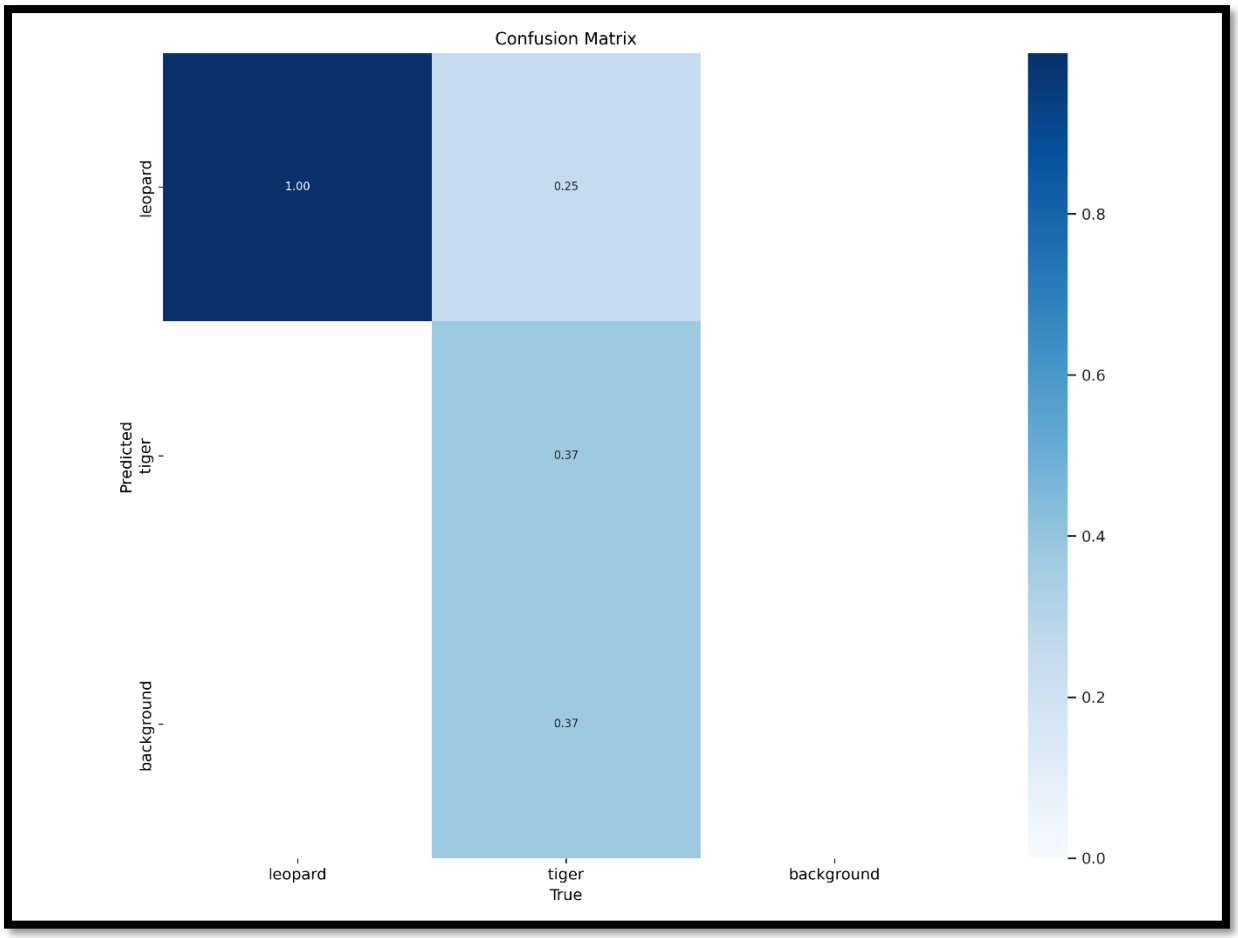
```
!yolo task=detect \mode=train \model=yolov8s.pt \data=/content/RAML-4/data.yaml \epochs=100 \imgsz=640
```

Once the model is trained we use the validation command with the best weight the training of our model have found.

```
!yolo task=detect mode=val model=/content/runs/detect/train9/weights/best.pt data=/content/RAML-3/data.yaml
```

And we finish with the test dataset.

```
!yolo task=detect mode=predict model=/content/runs/detect/train9/weights/best.pt conf=0.25 source=/content/RAML-4/test/images
```



We can see that the leopard are have 100% of accuracy but tiger which is the class we want to predict have 25 % of wrong prediction.

Pytorch

Deep learning models are created using PyTorch, an open-source machine learning library. Based on the Torch library, it was created by Facebook's AI research team. The dynamic computational graph that PyTorch is known for enables more adaptable and logical deep learning models.

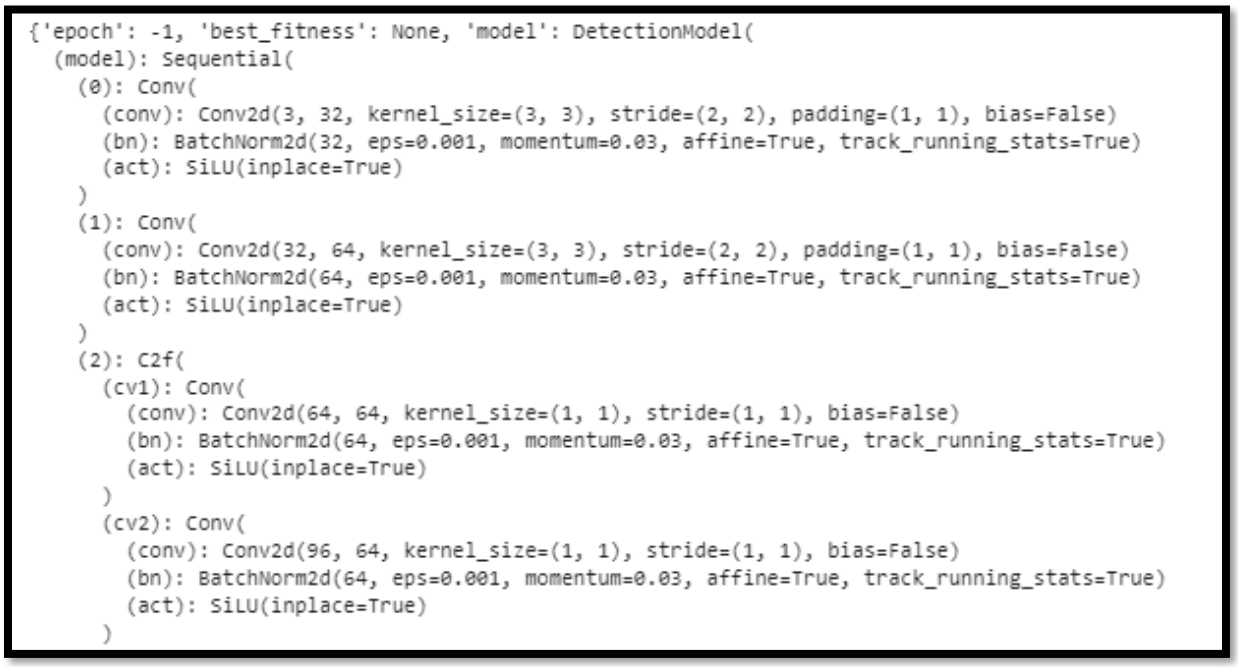
PyTorch's salient characteristics include:

A dynamic computational graph is one that is formed during runtime rather than being pre-defined, and PyTorch enables the development of such graphs. This gives more creative freedom when developing sophisticated deep learning models.

PyTorch provides GPU acceleration, making it possible to train deep learning models more quickly.

Autograd: Autograd, a PyTorch module for automatic differentiation, facilitates the automatic determination of gradients.

We used py torch to see the architecture of our model. Below you can see a part of this model.



```
{'epoch': -1, 'best_fitness': None, 'model': DetectionModel(
  (model): Sequential(
    (0): Conv(
      (conv): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(32, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
      (act): SiLU(inplace=True)
    )
    (1): Conv(
      (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn): BatchNorm2d(64, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
      (act): SiLU(inplace=True)
    )
    (2): C2f(
      (cv1): Conv(
        (conv): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
        (act): SiLU(inplace=True)
      )
      (cv2): Conv(
        (conv): Conv2d(96, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn): BatchNorm2d(64, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
        (act): SiLU(inplace=True)
      )
```

We havent succeed to modify the architecture of the Yolov8 model for detection, however we really searched it using all of this link :

https://github.com/ultralytics/ultralytics/blob/main/ultralytics/nn/modules.py

https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html

https://colab.research.google.com/github/omarsar/pytorch_notebooks/blob/master/pytorch_quick_start.ipynb#scrollTo=tmaCTw5tXowR

Conclusion

This project allowed us to really explore the different model architectures for segmentation or detection, we chose YOLO because there is already a lot of documentation on it. Going through roboflow to understand the importance of labeling and image attributes for prediction allowed us to understand that in addition to being essential this work is tedious and time-consuming. We are well aware that we have not been able to achieve the objectives set in terms of results for this project, however it has allowed us to improve our understanding of models such as CNN. The main difficulty of this project for us is that we were only novices in this field and we had to try to accumulate knowledge as quickly as possible.