

Classification d'images par Deep learning

Vision appliquée pour la Robotique

Majeure ROBIA/ module IA Vision

LONCHAMBON Alexis - 5IRC

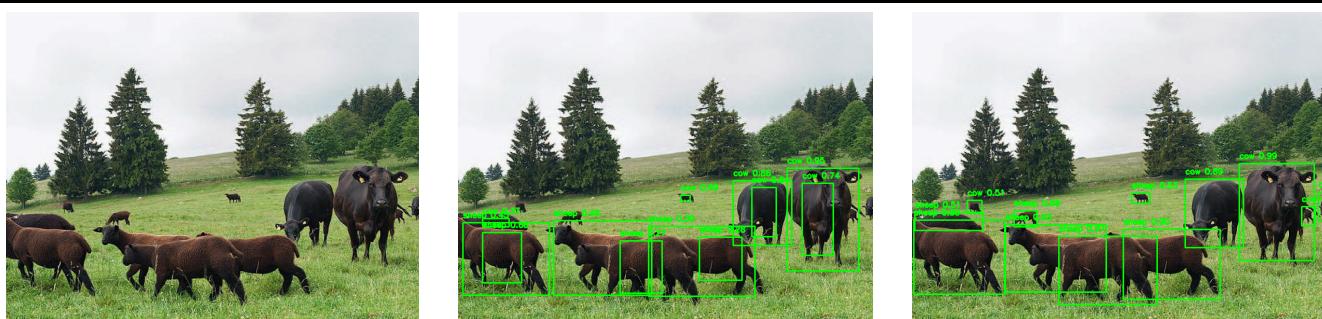
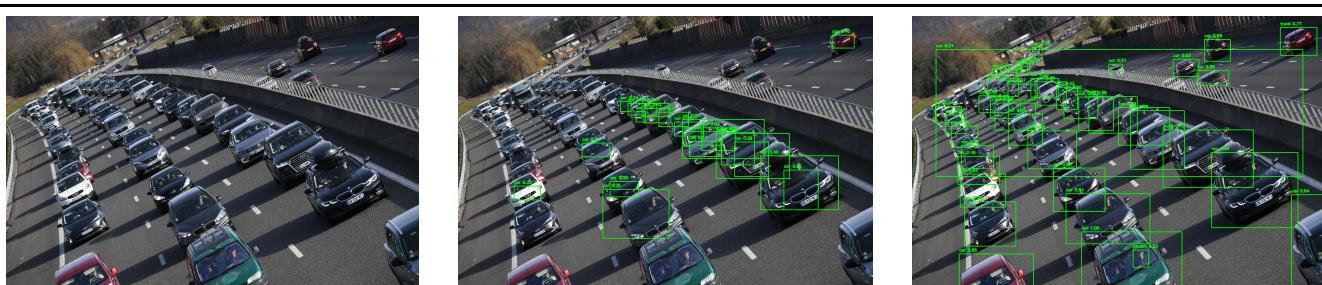
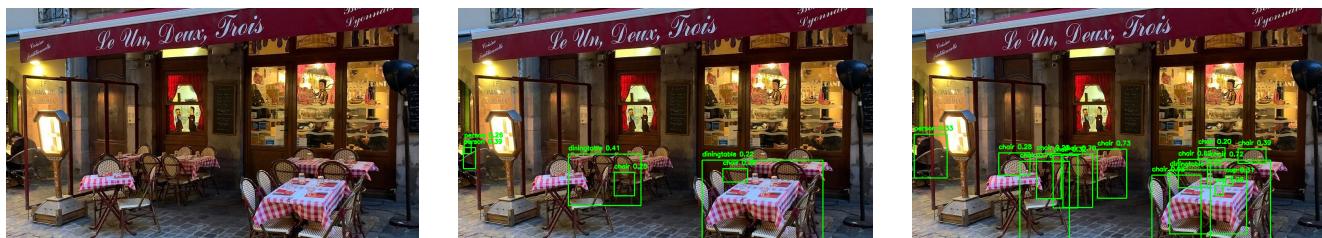
Mise en oeuvre de Yolo

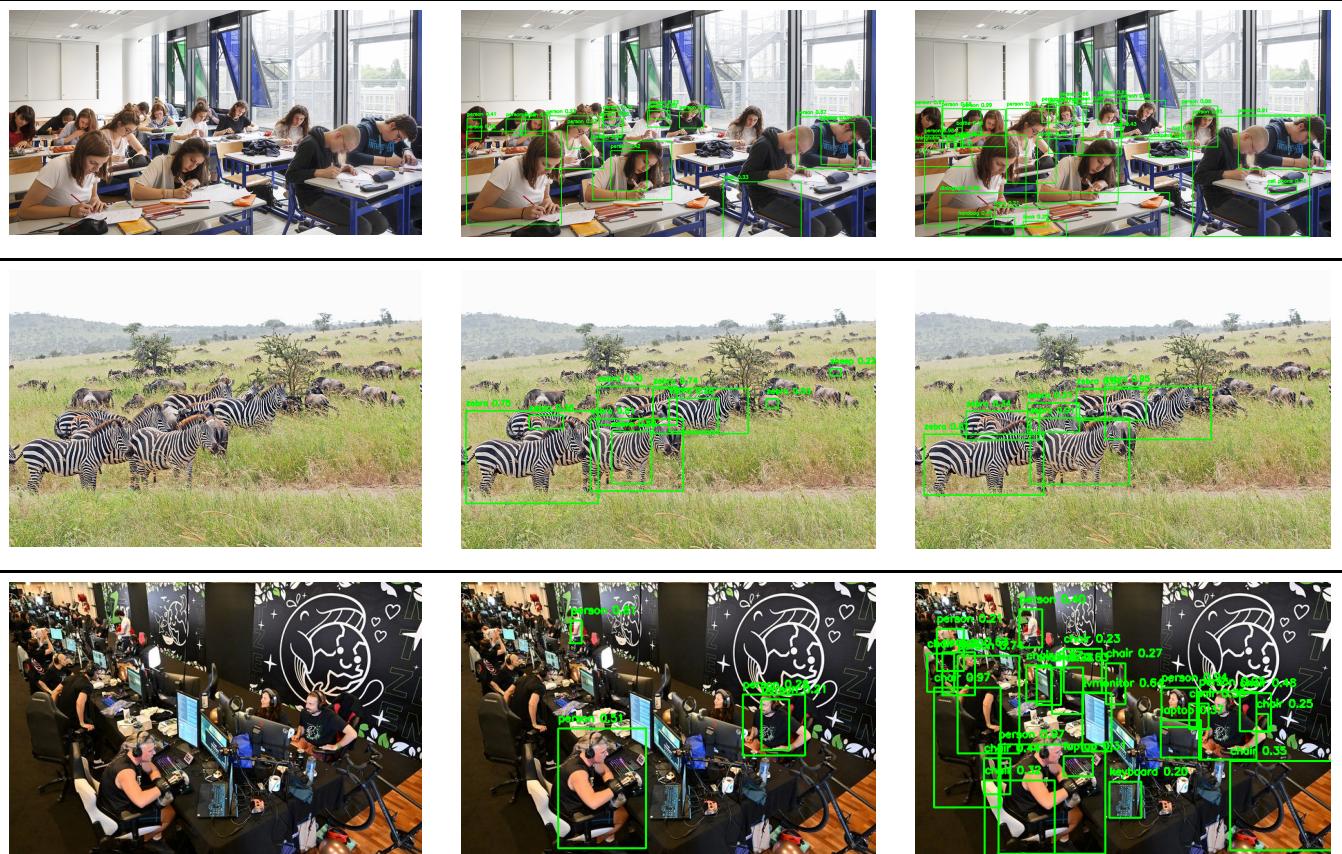
Partie 1

Question 1

1.1 Quelles sont les classes reconnues par le réseau ? tester sur au moins 20 images par classes (conserver ces images et les déposer sur le ecampus). Tester les 2 réseaux yolo et tiny-yolo

Le modèle est entraîné sur les classes de COCO 2017





1.2 A quoi sert le ou les thresholds ?

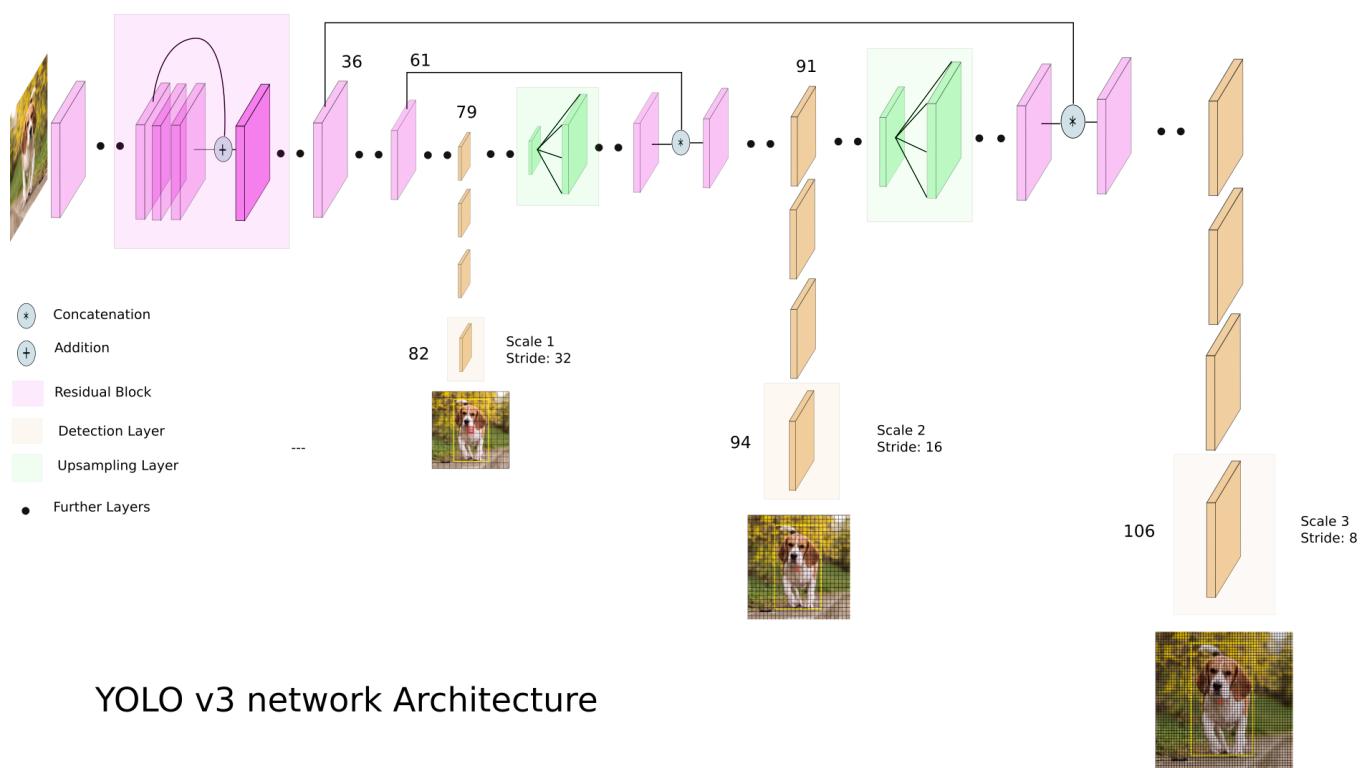
Le threshold sert à définir à partir de quelle certitude on détermine que la classe est reconnue par le modèle.

1.3 Quelles sont les fichiers utilisés liés au réseau de neurones , que contiennent ils précisément ?

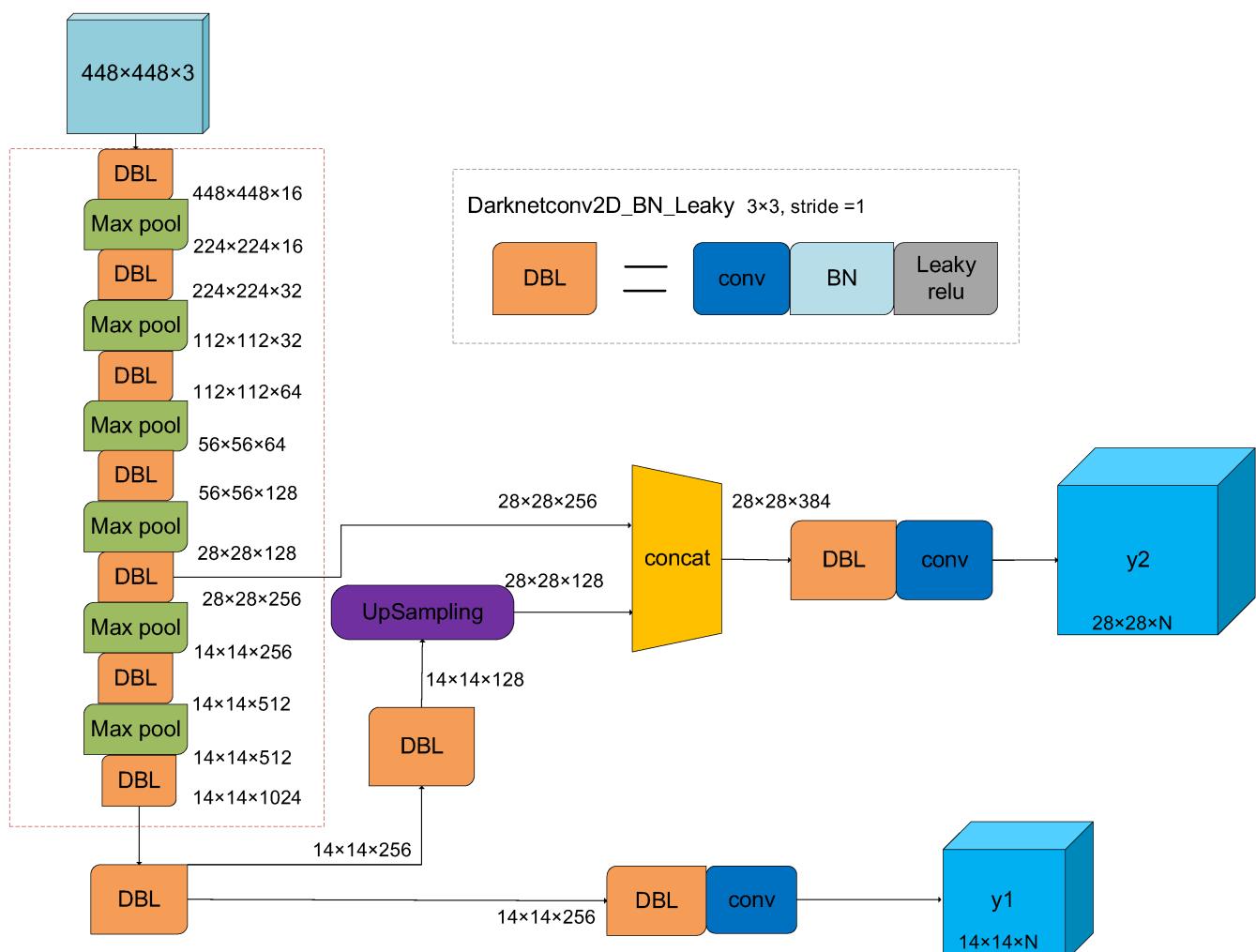
Les fichiers sont :

- yolov3.weights : les poids du modèle sous format binaire
- yolov3.cfg : toute la configuration de chaque layer du modèle sous format cfg

1.4 Quelle est l'architecture du réseau yolov3 et yolov3 tiny , expliquer les différentes couches du réseaux . Lesquelles dépendent du nombre de classes ?



YOLO v3 network Architecture



1.5 Est-ce que vous trouvez d'autres modèles pré entraînés proposés aux téléchargements ? Si oui tester les

Oui mais flemme

Partie 2 Interfaçage Python

2.1 On désire une option pour n'afficher qu'une liste de classe configurée en ligne de commande et mettre en place des boîtes englobantes avec un effet de transparence ou équivalent pour mettre les objets détectés en surbrillance ou équivalents. Adapter le code pour travailler sur un flux webcam ou des images (En changeant la ligne de commande) . Mettre le code commenté ici.

```
import argparse

import cv2.dnn
import numpy as np

def main(input_image):
    # Load the YOLOv3 model with OpenCV
    net = cv2.dnn.readNet("yolov3-tiny.weights", "yolov3-tiny.cfg")
    # net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

    # Load the COCO class labels
    with open("coco.names", "r") as f:
        classes = f.read().strip().split("\n")

    # Initialize webcam capture
    if(input_image == None):
        cap = cv2.VideoCapture(0) # 0 corresponds to the default webcam
        (change as needed)
    else:
        cap = cv2.imread(input_image)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 800);
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 600);
    # cap.set(cv2.CAP_PROP_FPS,120)
    # example resolution logi hd1080 : 2304x1536 2304x1296 1920x1080
    1600x896 1280x720 960x720 1024x576 800x600 864x480 800x448 640x480 640x360
    432x240 352x288 320x240 320x180 176x144 160x120 160x90
    # example resolution logi 720 : 1280x960 1280x720 1184x656 960x720
    1024x576 800x600 864x480 800x448 752x416 640x480 640x360 432x240 352x288
    320x240 320x176 176x144 160x120 160x90
    while True:
        if(input_image == None):
            ret, frame = cap.read() # Read a frame from the webcam

            if not ret:
                break
        else:
            frame = cap

        # Get the height and width of the frame
        height, width = frame.shape[:2]

        # Create a blob from the frame (preprocess)
        blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416),
        swapRB=True, crop=False)
```

```
# Set the input to the neural network
net.setInput(blob)

# Get the output layer names
layer_names = net.getUnconnectedOutLayersNames()

# Run forward pass through the network
outs = net.forward(layer_names)

# Initialize lists to store detected objects' class IDs,
confidences, and bounding boxes
class_ids = []
confidences = []
boxes = []

# Minimum confidence threshold for object detection
conf_threshold = 0.5

# Iterate through each output layer
for out in outs:
    # Iterate through each detection in the output
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]

        if confidence > conf_threshold:
            # Scale the bounding box coordinates to the original
            frame size
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Calculate the top-left corner of the bounding box
            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])

# Non-maximum suppression to remove redundant boxes
nms_threshold = 0.4
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold,
                           nms_threshold)

# Draw bounding boxes and labels on the frame
for i in indices:
    x, y, w, h = boxes[i]
    label = str(classes[class_ids[i]])
    confidence = confidences[i]
    color = (0, 255, 0) # Green color for the bounding box
```

```
        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
        cv2.putText(frame, f"{label} {confidence:.2f}", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    # Display the frame with detected objects
    cv2.imshow("Object Detection", frame)

    if(input_image != None):
        cv2.waitKey(0)
        break

    # Exit the loop if 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    # Release the webcam and close all OpenCV windows
    if(input_image == None):
        cap.release()

    cv2.destroyAllWindows()
    return

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--img', help='Path to input image.')

    args = parser.parse_args()
    main(args.img)
```

Partie 3 Préparation à l'entraînement votre propre classifier

L'idée est maintenant de comprendre comment créer votre propre réseau . L'objectif est de créer votre classifier sur 3 ou 4 classes (à vous de choisir parmi les objets proposés ou autres). Il faut prévoir idéalement dans les 200 images labélisés par classe (ne pas hésiter à mixer les fonds et les éclairages) mais on peut avoir des performances corrects à partir de 40.

3.1

On utilisera le processus suivant : S'inscrire sur roboflow

3.1.1 Créer le dataset et le labéliser sur roboflow

Generating New Version

Prepare your images and data for training by compiling them into a version.
Experiment with different configurations to achieve better training results.



Source Images

Images: 238

Classes: 5

Unannotated: 0



Train/Test Split

Training Set: 166 images

Validation Set: 48 images

Testing Set: 24 images

3.1.2 Vous exporterez à la fin le dataset au format yolo et au format tensorflow. Que contienne ses archives ? (vous renderez ces fichiers sur le e-campus)

Les archives contiennent les images, les detections, les classes et des fichiers textes d'instruction

Exemple TensorFlow CSV

```
filename,width,height,class,xmin,ymin,xmax,ymax
picture_2023-11-07_09-30-
01.jpg.rf.04f51206cbf77b3f2cdc9c9404fb26a1.jpg,640,480,moose,83,313,202,433
picture_2023-11-07_09-30-
01.jpg.rf.04f51206cbf77b3f2cdc9c9404fb26a1.jpg,640,480,moose,534,305,640,46
1
picture_2023-11-07_09-30-
01.jpg.rf.04f51206cbf77b3f2cdc9c9404fb26a1.jpg,640,480,moose,306,308,406,44
5
```

Exemple Yolo Darknet

```
1 0.0828125 0.834375 0.165625 0.2270833333333333
1 0.35045312500000003 0.8296041666666666 0.23884375000000008
0.21279166666666663
1 0.664625 0.8065416666666666 0.1462031249999999 0.20237499999999997
1 0.8415156250000001 0.8415208333333334 0.16295312499999995
0.2247083333333337
1 0.890625 0.7886875 0.1160781249999995 0.22916666666666666
```

3.1.3 Vous pourrez ensuite lancer l'apprentissage sur roboflow (vous avez 3 credits associés à la création de votre compte). Tester le résultat ? Etes vous satisfait du résultat , expliquer pourquoi et comment vous tester.

v1 2023-11-07 11:51am

Generated on Nov 7, 2023

toy-animals-get4r/1

Model Type: Roboflow 3.0 Object Detection (Fast)
Checkpoint: COCOn

mAP 98.2% | Precision 98.8% | Recall 96.4%

[View Detailed Model Evaluation](#)

Deploy Your Model



TRY THIS MODEL
Drop an image or [browse your device](#)



Try on mobile

On peut tester directement sur le site en déployant le modèle. Et les résultats sont impressionnantes !

3.1.4 Comment peut-on récupérer le réseau généré ? Comment l'utiliser ? Pourquoi ce choix ?

Le modèle est utilisable en ligne et il faut l'intégration pour l'utiliser (avec une Hosted API) ou en utilisant un inference server en local

Switch Model:

v1 toy-animals-get4r/1

Trained On: toy-animals-get4r 570 Images [View Version →](#)

Model Type: Roboflow 3.0 Object Detection (Fast)

Checkpoint: COCOn

mAP 98.2%

Precision 98.8%

Recall 96.4%

[View Model Graphs →](#)

Samples from Test Set



[View Test Set →](#)

Upload Image or a Video File

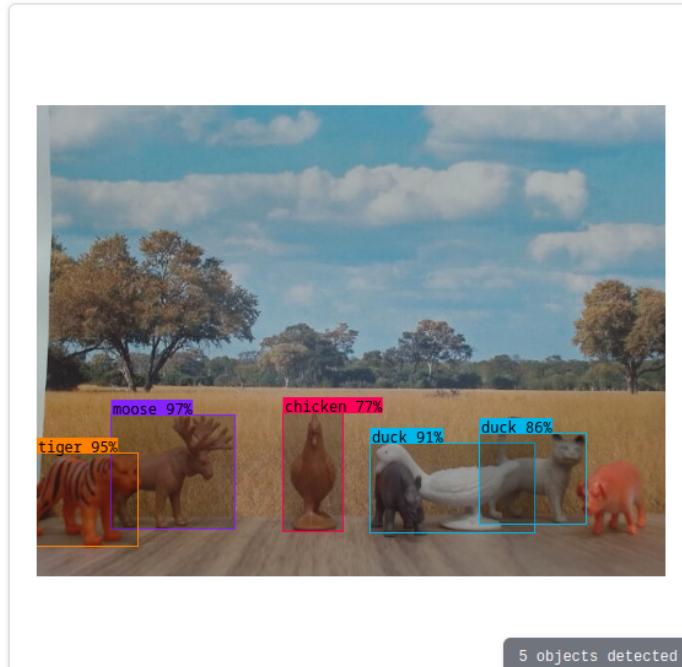
Drop files here or [Select File](#)

Paste YouTube or Image URL

[Paste a link...](#)

[Try With Webcam](#)

[Try On My Machine](#)



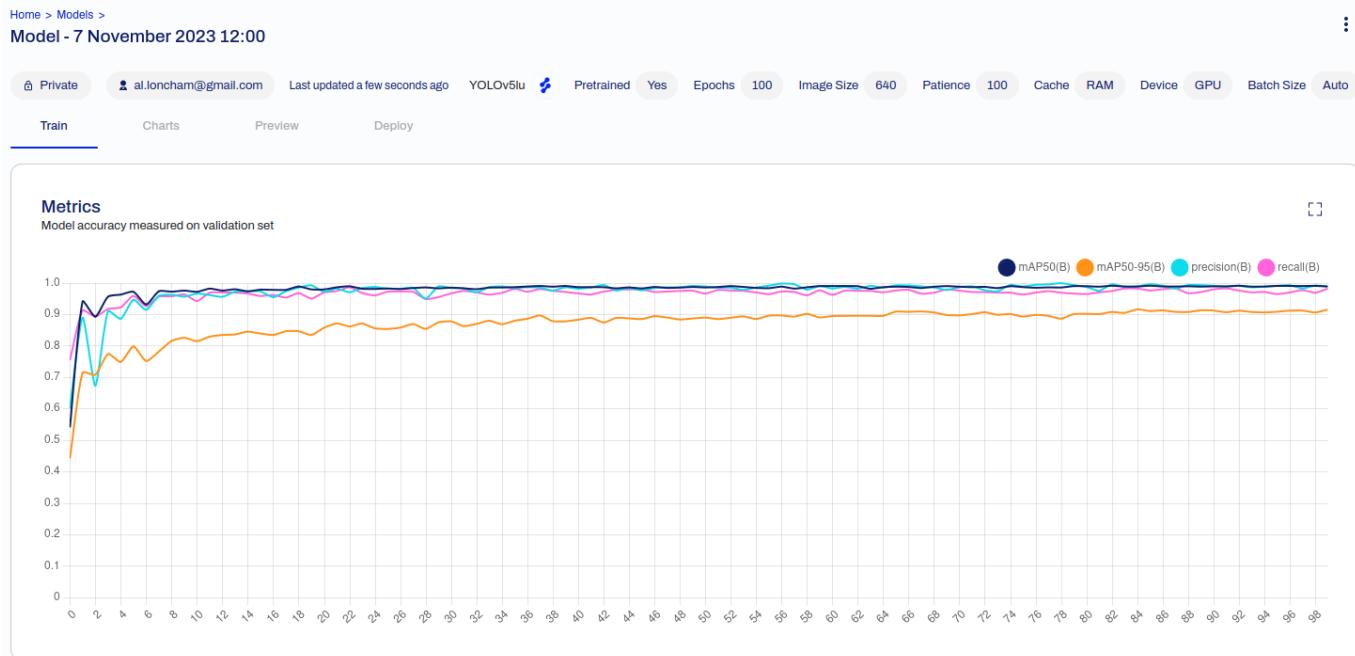
Confidence Threshold: 50%
Overlap Threshold: 50%

Label Display Mode:
[Draw Labels](#)

```
{
  "predictions": [
    {
      "x": 111.5,
      "y": 299.5,
      "width": 101,
      "height": 93,
      "confidence": 0.974,
      "class": "moose",
      "class_id": 2
    },
    {
      "x": 41.5,
      "y": 322,
      "width": 83,
      "height": 76,
      "confidence": 0.953,
      "class": "tiger",
      "class_id": 3
    }
  ]
}
```

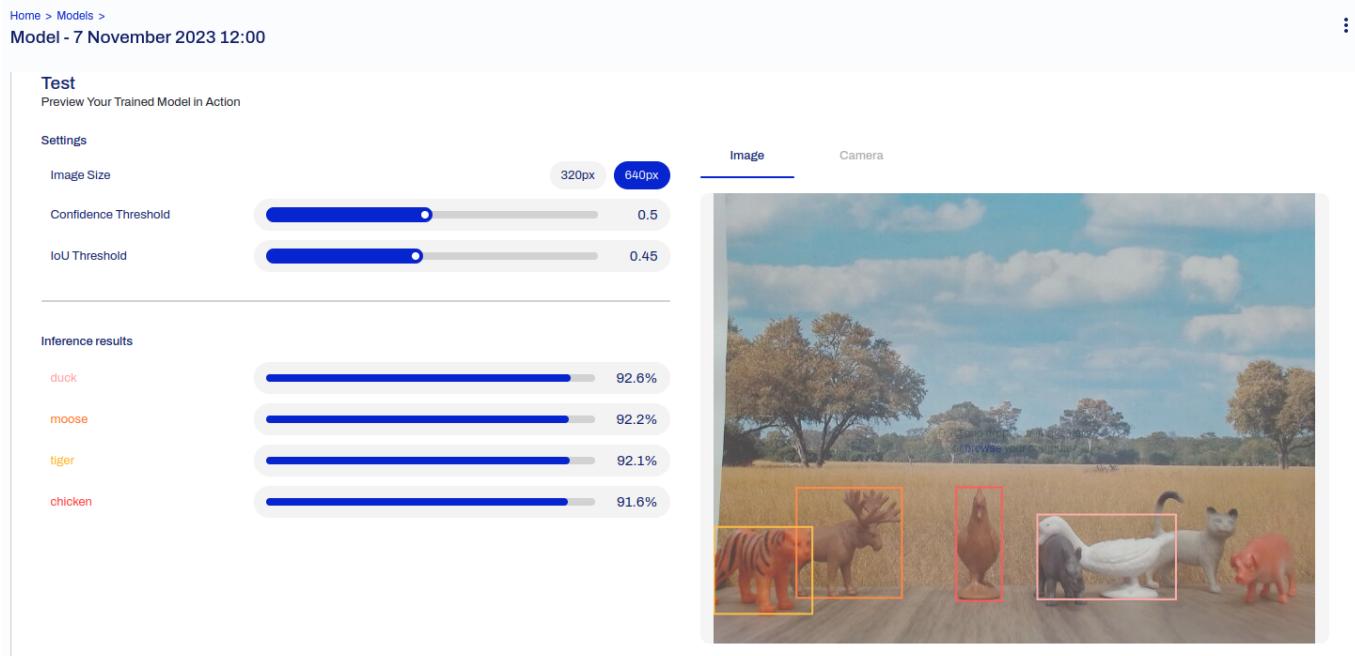
[Copy](#)

3.1.5 On procédera ensuite à l'export du dataset sous ultralytics puis au training en utilisant le format YOLOv5lu.



3.1.6 On testera le résultat sur le site , est ce meilleur ou moins bon que le résultat de roboflow ?

C'est un peu meilleur



3.1.7 Dans les options de déploiement de ultralytics, on pourra ensuite générer et télécharger le format onnx .

Done.

3.1.8 Vous pourrez finalement le tester sous opencv directement. en utilisant le code d'exemple fourni sur le e-campus. Vous devrez en particulier trouver une solution pour récupérer le nom de vos classes, expliquer comment ?

On peut utiliser le dataset yolo qui a la liste des classes dans un fichier texte.

3.1.9 Cela est il fonctionnelles, comment procéder à tests ? Etes-vous satisfaits ? Joindre le code et les commandes d'appels, inclure des copies d'écran

```
#!/usr/bin/env python3

__author__      = "Alexis Lonchambon"
__license__     = "CC0"
__version__     = "0.1"

import cv2
import numpy as np

from ultralytics.utils import ASSETS, yaml_load
from ultralytics.utils.checks import check_yaml

CLASSES = yaml_load(check_yaml('./toynimals.yaml'))['names']
colors = np.random.uniform(0, 255, size=(len(CLASSES), 3))

def draw_bounding_box(img, class_id, confidence, x, y, x_plus_w, y_plus_h):
    """
    Draws bounding boxes on the input image based on the provided arguments.

    Args:
        img (numpy.ndarray): The input image to draw the bounding box on.
        class_id (int): Class ID of the detected object.
        confidence (float): Confidence score of the detected object.
        x (int): X-coordinate of the top-left corner of the bounding box.
        y (int): Y-coordinate of the top-left corner of the bounding box.
        x_plus_w (int): X-coordinate of the bottom-right corner of the bounding box.
        y_plus_h (int): Y-coordinate of the bottom-right corner of the bounding box.
    """
    label = f'{CLASSES[class_id]} ({confidence:.2f})'
    color = colors[class_id]
    cv2.rectangle(img, (x, y), (x_plus_w, y_plus_h), color, 2)
    cv2.putText(img, label, (x - 10, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

# Load YOLOv3 model with OpenCV
net = cv2.dnn.readNetFromONNX("./toynimalsv1.onnx")

# Initialize webcam capture
cap = cv2.VideoCapture(0) # 0 corresponds to the default webcam (change as needed)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 800);
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 600);
```

```
# cap.set(cv2.CAP_PROP_FPS,120)
# example resolution logi hd1080 : 2304x1536 2304x1296 1920x1080 1600x896
1280x720 960x720 1024x576 800x600 864x480 800x448 640x480 640x360 432x240
352x288 320x240 320x180 176x144 160x120 160x90
# example resolution logi 720 : 1280x960 1280x720 1184x656 960x720 1024x576
800x600 864x480 800x448 752x416 640x480 640x360 432x240 352x288 320x240
320x176 176x144 160x120 160x90
while True:
    ret, frame = cap.read() # Read a frame from the webcam

    if not ret:
        break

    # Get the height and width of the frame
    height, width = frame.shape[:2]

    # Prepare a square image for inference
    length = max((height, width))
    image = np.zeros((length, length, 3), np.uint8)
    image[0:height, 0:width] = frame

    # Calculate scale factor
    scale = length / 640

    # Create a blob from the frame (preprocess)
    blob = cv2.dnn.blobFromImage(image, scalefactor=1 / 255, size=(640,
640), swapRB=True)

    # Set the input to the neural network
    net.setInput(blob)

    # Perform inference
    outputs = net.forward()

    # Prepare output array
    outputs = np.array([cv2.transpose(outputs[0])])
    rows = outputs.shape[1]

    boxes = []
    scores = []
    class_ids = []

    # Iterate through output to collect bounding boxes, confidence scores,
    and class IDs
    for i in range(rows):
        classes_scores = outputs[0][i][4:]
        (minScore, maxScore, minClassLoc, (x, maxClassIndex)) =
cv2.minMaxLoc(classes_scores)
        if maxScore >= 0.25:
            box = [
                outputs[0][i][0] - (0.5 * outputs[0][i][2]), outputs[0][i]
[1] - (0.5 * outputs[0][i][3]),
                outputs[0][i][2], outputs[0][i][3]]
```

```
boxes.append(box)
scores.append(maxScore)
class_ids.append(maxClassIndex)

# Apply NMS (Non-maximum suppression)
result_boxes = cv2.dnn.NMSBoxes(boxes, scores, 0.25, 0.45, 0.5)

detections = []

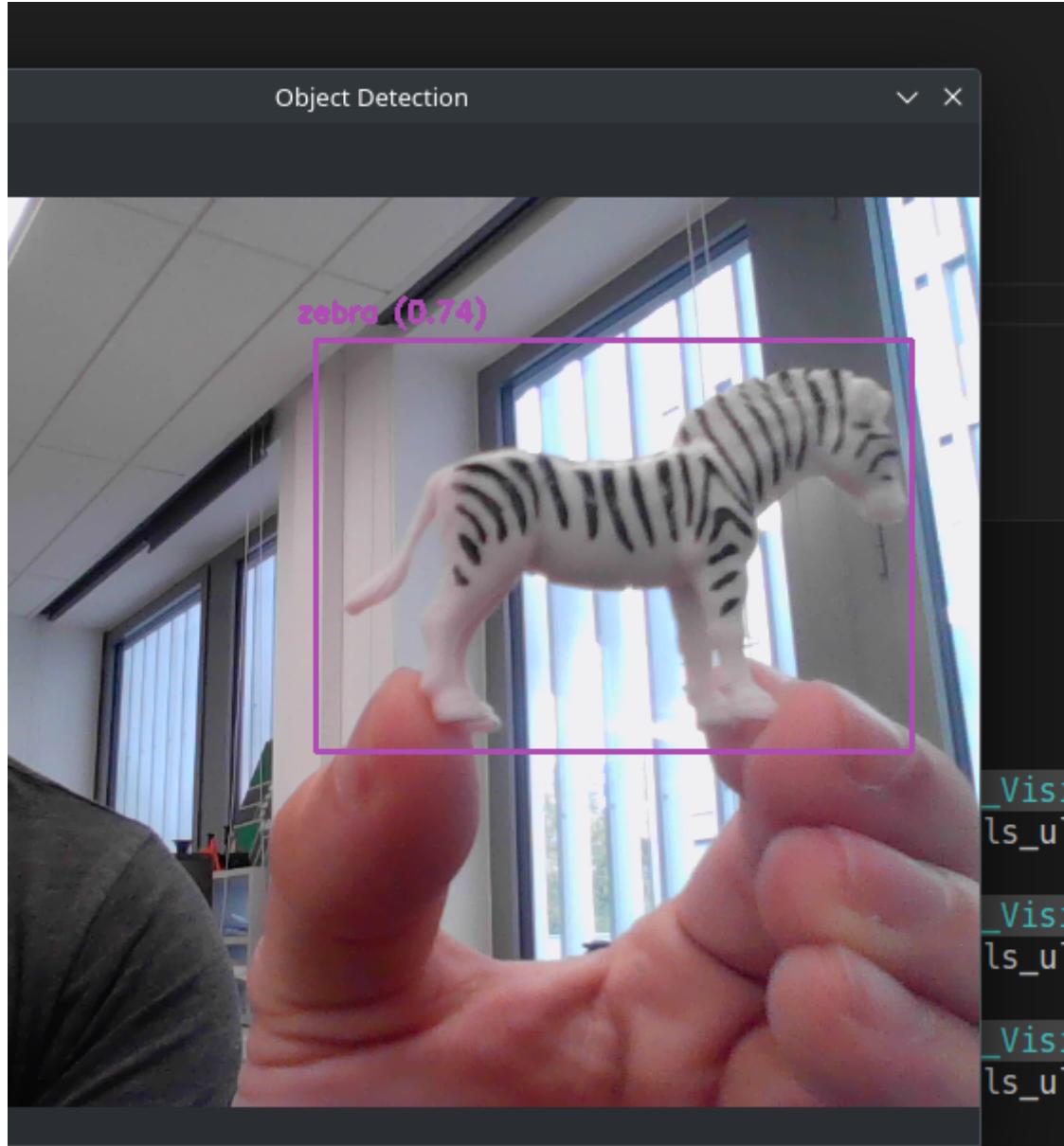
# Iterate through NMS results to draw bounding boxes and labels
for i in range(len(result_boxes)):
    index = result_boxes[i]
    box = boxes[index]
    detection = {
        'class_id': class_ids[index],
        'class_name': CLASSES[class_ids[index]],
        'confidence': scores[index],
        'box': box,
        'scale': scale}
    detections.append(detection)
    draw_bounding_box(frame, class_ids[index], scores[index],
round(box[0] * scale), round(box[1] * scale),
                    round((box[0] + box[2]) * scale), round((box[1] +
box[3]) * scale))

# Display the image with bounding boxes

# Display the frame with detected objects
cv2.imshow("Object Detection", frame)

# Exit the loop if 'q' key is pressed
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Release the webcam and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```



Question 3.1 (Bonus) Il est ensuite demandé de repartir de votre code de la question 2 et de l'adapter en changeant le minimum de chose pour utiliser votre réseau onnx. Expliquer les modifications qui ont dû être faites et inclure votre code Joindre le code et les commandes d'appels, inclure des copies d'écran

Pour info, suite à un bug bientôt corriger du module dnn , seul le yolov5lu semble être reconnu au format onnx sous opencv. Si il vous reste du temps vous pourrez tester d'autre format pour m'indiquer si ils sont fonctionnelles