Deep Learning for IoT
Lab 5 – Object and Pose Detection with YOLOv7

1. Introduction

Note: There are two parts to this lab. In this first part we look at how to do object detection and tracking. To access the second part of this lab, lease load the **Lab5B.ipynb** file in the yolo7 directory using Jupyter.

YOLOv7 is the state-of-the art object and pose detection algorithm in the YOLO (You Only Look Once) family of algorithms. It is architecturally similar to YOLOv3 which we talked about in the lecture, but has incorporated many improvements that has increased its speed and accuracy.

While the legacy versions of YOLO (YOLOv3 and below) were implemented in C, YOLOv7 is implemented in Python using the PyTorch library.

You can obtain the yolo7.zip file from:

https://www.dropbox.com/scl/fo/2887r62pmiwtpqtrmy7cy/AKO-S0nGn0BUTroI1qQhdoU?rlkey=wm74anevu5by1nt12imz5g8d7&dl=0

This is the standard YOLO7 distribution together with some special modifications that we have made:

a. It already has the yolov7.pt, yolov7-tiny.pt and yolov7-w6-pose.pt pretrained weights files.
b. It has the detect-cb.py file, which is a custom modified version of detect.py that lets you call "detect" as a library, and allows you to specify your own handlers for the objects that are detected.

There is no submission for this lab. However, you should do it anyway so that you are familiar with how to use the YOLOv7 detector.

There is also a yolo7train.zip file in the same Dropbox directory. This contains a sample directory tree for training the YOLO7 detector. You can unzip this file and take a look at how to set up a training directory.

2. Setting Up

a. Obtain the yolo7.zip file from:

https://www.dropbox.com/scl/fi/z4kyzrvfak4kpj0js3kx1/yolov7.zip?rlkey=7ydbtincmc2h5gyzl02lown1j&dl=0

b. Unzip yolo7.zip to create the yolo7 directory, which contains our special YOLO7 code. Do not use the yolo7 repository that is online as it does not have the special detect_cb.py file that we have created for you.
c. Change to the yolov7 directory.
d. Change to that directory, and install all the dependencies:

   pip install -r requirements.txt

e. Test your installation by detecting horses in an image:

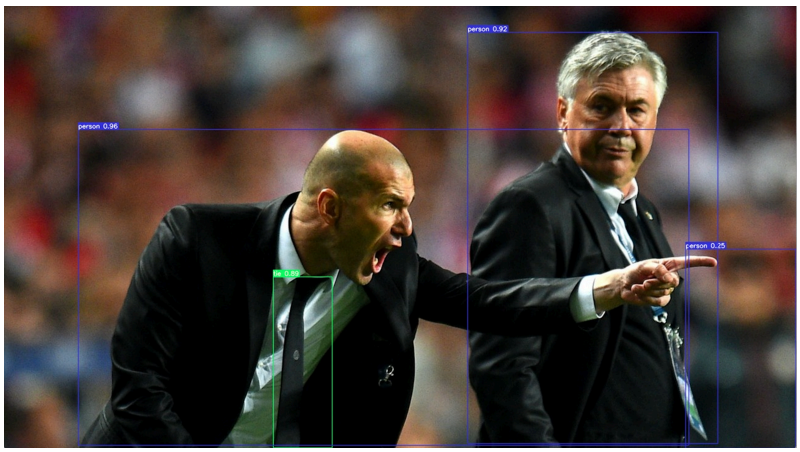
   python detect.py

   You will see an output like this:

```
RepConv.fuse_repvgg_block
Model Summary: 306 layers, 36905341 parameters, 6652669 gradients
 Convert model to Traced-model...
 traced_script_module saved!
 model is traced!

/Users/ctank/Library/Python/3.9/lib/python/site-packages/torch/functional.py:504
erWarning: torch.meshgrid: in an upcoming release, it will be required to pass t
ndexing argument. (Triggered internally at /Users/runner/work/pytorch/pytorch/py
h/aten/src/ATen/native/TensorShape.cpp:3484.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
4 persons, 1 bus, 1 tie, Done. (398.0ms) Inference, (0.9ms) NMS
 The image with the result is saved in: runs/detect/exp4/bus.jpg
5 horses, Done. (320.1ms) Inference, (0.5ms) NMS
 The image with the result is saved in: runs/detect/exp4/horses.jpg
2 persons, 1 tie, 1 cake, Done. (372.6ms) Inference, (0.6ms) NMS
 The image with the result is saved in: runs/detect/exp4/image1.jpg
2 persons, 1 sports ball, Done. (312.6ms) Inference, (0.5ms) NMS
 The image with the result is saved in: runs/detect/exp4/image2.jpg
1 dog, 1 horse, Done. (311.3ms) Inference, (0.4ms) NMS
 The image with the result is saved in: runs/detect/exp4/image3.jpg
3 persons, 1 tie, Done. (277.1ms) Inference, (0.4ms) NMS
 The image with the result is saved in: runs/detect/exp4/zidane.jpg
Done. (2.045s)
```

   If you go to runs/detect/exp4 you will be able to see the results. Here is an example:



   You can also do live detection on your webcam stream by doing:

   python detect.py --source 0

3. Fine-Tuning your YOLO Detector

The YOLOv7 detector has already been trained to detect a very large range of objects, but perhaps you'd like to train it to detect particular objects that you are interested in. One

example of why you might like to do this is that you have gathered a large number of images of the ***actual*** environment that you will be working with, and you want to increase the accuracy of your detector.

a. Downloading Yolo Label

There are several tools to help you label and identify objects in images, but YOLO Label is one of the easiest to use. Download and install YOLO Label by following the instructions at: https://github.com/developer0hye/Yolo_Label

Notes:

i. You can use other labelers if you are not comfortable with compiling YOLO Label. However, YOLO Label's workflow is very fast and convenient.
ii. You will need to install qt before compiling YOLO Label. Please see https://doc.qt.io/qt-6/get-and-install-qt.html for how to install qt.

b. Labeling Images

Assuming that you've installed YOLO Label installed, in this section we will practice labeling a few images just to give you a feel of how to do it.

i. In the git repo that you have pulled, there is a sample directory called Samples, and below that there is a directory called Images with some test images.
ii. In the Samples directory, create a file called obj_names.txt, and key in the following:

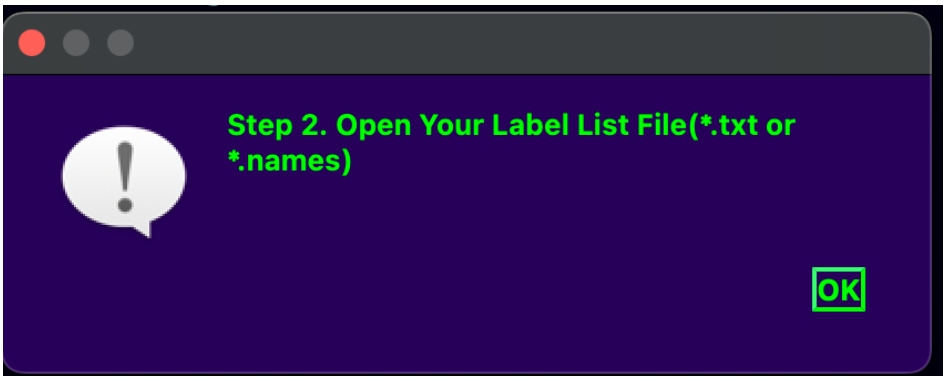
raccoon
kangaroo

Save the file.

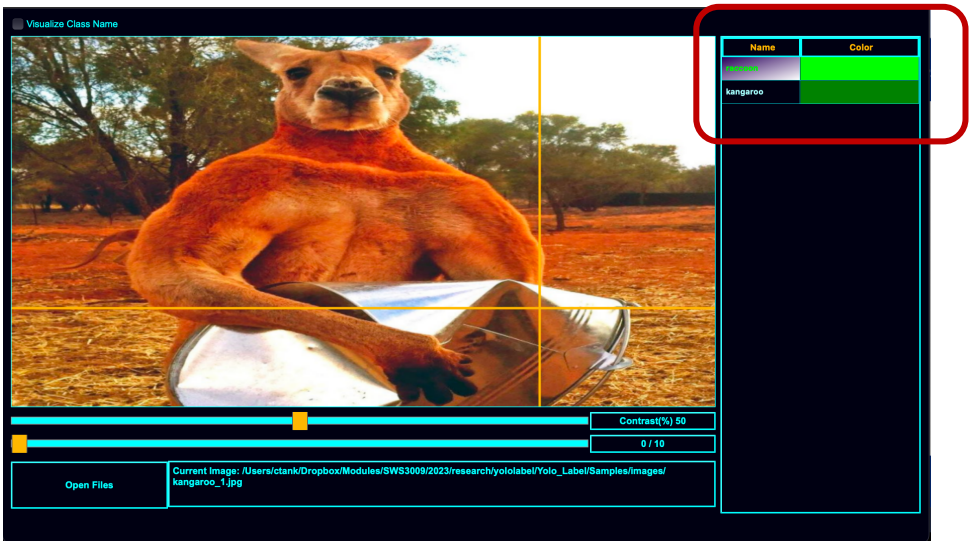
iii. Now launch Yolo Label, then click on Open Files at the bottom left of the screen:

iv. This will open a dialog that says “Step 1. Open Your Data Set Directory”. Click OK, then navigate to the Samples/Images directory with all the images.

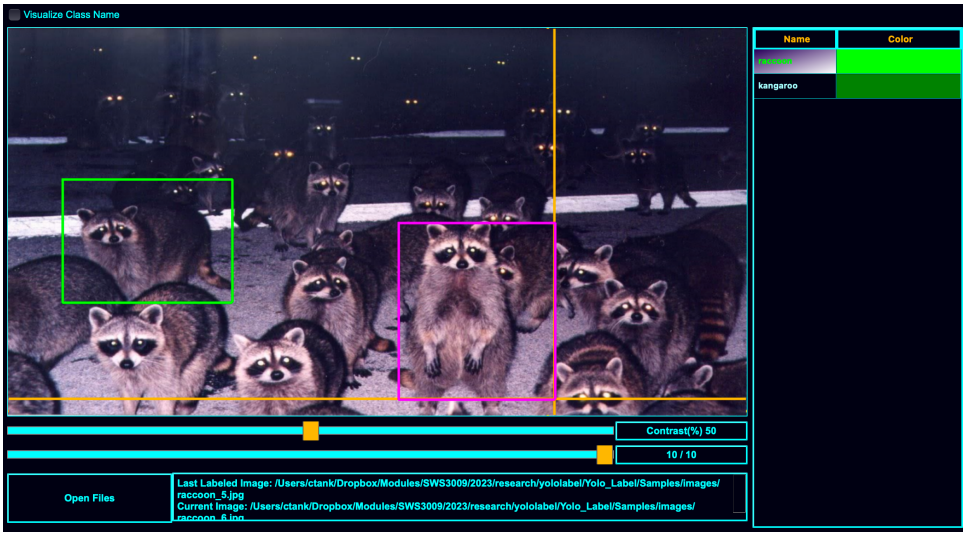
v. Once you have selected your data set directory, YOLO Label will now ask you to select your Label List File:



vi. Click OK, then select the obj_names.txt file you created earlier. Once you have done this, YOLO Label will open the first image:



vii. Select the object name from the right box (circled in red above), then click the top left corner of the object. A purple box will appear. Move to the bottom right of the object and click again. A box will now surround the object you marked:



viii. Use "D" to move to the next image, and "A" to move to the previous image. As you move between images your markings will automatically be saved.

ix. Run through the images and label them using the steps above. At the end you will see that every image in the directory will now have a .txt file that has the label, top-left and bottom-right coordinates in it for every object in the image.

E.g. kangaroo_5.txt has the box coordinates for each kangaroo in kangaroo_5.jpg:

```
1 0.193980 0.441737 0.196210 0.565678
1 0.319398 0.513771 0.596433 0.849576
1 0.706243 0.284958 0.460424 0.302966
```

After labeling all your data, you will need to:

i. Decide which images are used for training and which are used for validation.
ii. Create a "dataset" directory.
iii. Create "train" and "valid" directories under your "dataset" directory.
iv. Within the "train" directory, create an "images" directory and copy all your training images there.
v. Within the "train" directory, create a "labels" directory and copy all the label (*.txt) files for the images there.
vi. Repeat steps iv and v for the "valid" directory, but this time copy over the validation images and validation label files.
vii. Create a "data.yaml" file in the "dataset" directory, which contains the path to the training and validation images, the number of categories (nc) and the names for each category (names):

```
train: ../train/images
val: ../valid/images

nc: 7
names: ['fish', 'jellyfish', 'penguin', 'puffin', 'shark', 'starfish', 'stingray']
```

Note that the train and valid directories are relative to the YOLOv7 directory.

c. Fine-tuning YOLOv7

Since it will take too long for you to label all the images yourself, we will instead use a set of pre-labeled images.

We have created a dataset directory in the yolov7 directory, and it contains test and training images together with label files and a data.yaml file. This file comes from Roboflow and you can find other datasets there.

Unzip this file in the SAME directory where you unzipped the yolov7.zip file. Launch the training using:

```
python train.py --weights yolov7.pt --data dataset/data.yaml --workers 4 --batch-size 4 --img 416 --cfg cfg/training/yolov7.yaml --name yolov7 --hyp data/hyp.scratch.p5.yaml --epochs 25
```

This will start the training, for 25 epochs. This will take a **very** long time.

4. **Running your YOLO Detector**

Unfortunately, the YOLO detector is too complex for you to build on your own, and unlike the transformers there are no Hugging Face models that you can just download and use. So, we will use a modified version of the YOLO detector that is provided.

In our modified version, YOLO will call a callback function for each image it receives either from a file or from the video camera.

Create a file called “mydetect.py” in the **same** directory where you have your YOLOv7 files, and enter the following code. First we have the imports, and a series of comments explaining how the callback is created:

```
import matplotlib.pyplot as plt
from detect_cb import detect
from utils.plots import plot_one_box
import cv2

# This is an example of a callback. The callback is called
# every time detect finishes detecting objects and drawing
# the bounding boxes.
# The result_list parameter is a list of results, one for
# each object detected.
# Index 0 xyxy: Bounding box in xyxy format (actual coordinates)
# Index 1 xywh: Bounding box in xywh normalized format
# Index 2 cls: Class,
# Index 3 name: Object name,
# Index 4 color: Object color,
# im0: The original image without padding and normalization
# waitvalue: Needed only if you call cv2.imshow.
#   Call cv2.waitKey with this value. Use 0 for images and
#       15 for videos
```

Now we build the callback. The callback receives the xyxy bounding boxes (top-left and lower right corners), xywh bounding boxes (top-left, width and height), the object class (cls), object name (name) and the color of the bounding box (color). We will use this information and the plot_one_box helper to draw boxes around the objects detected:

```
def mycallback(result_list, im0, waitvalue=0):

    for result in result_list:
        xyxy = result[0]
        label = result[3]
        color = result[4]
        coords = [t.tolist() for t in result[0]]

        print("Object detected: ", label, " Coordinates: ", coords)

        plot_one_box(xyxy, im0, label=label, color=color, line_thickness=1)

    cv2.imshow("test", im0)
    cv2.waitKey(waitvalue)
```

Finally we call “detect” with the source (a filename for a JPG file or “0” for the video camera), a pointer to the callback, and a waitValue for waiting for keystrokes. Use 0 for image files and 15 for the video camera:

```
# Call detect with image3 in inference/images
#detect("inference/images/image3.jpg", callback = mycallback)

# For detecting on webcam
detect("0", callback = mycallback, waitvalue=15)
```

Run your mydetect.py program using:

```
python mydetect.py
```

Your program will continuously read the video camera until you press a key to exit. You may need to press a key many times before the program exits.

5. **Creating a Pose Estimator**

One of the new additions to the classic YOLOv3 architecture is a pose estimation head that detects the positions of a person's head, arms and legs. It will draw a "stick figure" like this:

Please launch Jupyter Notebook and load up the Lab5B.ipynb notebook to continue this lab.

6. **Summary**

In this lab we saw how to use the YOLOv7 object detector to perform object detection and tracking, as well as pose