Project 1: COCOTEXT

Alireza Amiri

Part 2 - Object Detection

In this part of the project, you will use your skills in object detection to train a network on the COCO–Text dataset. All the design parameters including type of the object detection network, backbone (feature extractor), input resolution, batch size, number of steps for training, etc. are arbitrarily and should be chosen based on your knowledge, so be careful about choosing each parameter. The minimum requirement of this part is training one object detection network and testing and visualizing the network on the test images. Don't forget that the discussion on choosing parameters and the output result is important!

Data preparation

Downloading the dataset and unzipping it

[ ] ↳ *9 cells hidden*

Organizing the files and folders to match the requirements of the project.

In the following sections, we wish to organize files in such a way that is understandable for YOLO algorithm. This structure requires copying files inside ultralytics directory with two folders in which inside each of them, images and text files are seperated into different folders. This way, YOLO algorithm can be run on the dataset

```
source_path='/content/cocotext'

destination_path='/content/ultralytics/ultralytics/datasets'
if not os.path.exists(destination_path):
  os.makedirs(destination_path)
```

```
import os

destination_path = '/content/ultralytics/ultralytics/datasets'

# Check if directory exists
if os.path.exists(destination_path):
  print(f"Directory exists: {destination_path}")
else:
  print(f"Directory does not exist: {destination_path}")
```

```
Directory exists: /content/ultralytics/ultralytics/datasets
```

The process in the following code is as follows:

1. the currently existing folders are addresses.
2. we create new folders of images and labels inside both test and train folders
3. Next, data is copied to destination folders based on their types. in order to seperate images from text files, the ending format of files are assessed and if they were equall to ".jpeg", they will be transferred to images folder, else, they will be stored in label folder

```
import os
import shutil

source_train = "/content/cocotext/train"
source_test = "/content/cocotext/test"
dest_dir = "/content/ultralytics/ultralytics/datasets"
os.makedirs(os.path.join(dest_dir, "images", "train"), exist_ok=True)
os.makedirs(os.path.join(dest_dir, "images", "test"), exist_ok=True)
os.makedirs(os.path.join(dest_dir, "labels", "train"), exist_ok=True)
os.makedirs(os.path.join(dest_dir, "labels", "test"), exist_ok=True)
def move_data(source_folder, dest_images, dest_labels):
  for filename in os.listdir(source_folder):
    if filename.endswith(".jpg"):
      image_path = os.path.join(source_folder, filename)
      label_path = os.path.join(source_folder, os.path.splitext(filename)[0] + ".txt")  # Assuming consistent naming
      shutil.move(image_path, os.path.join(dest_images, filename))  # Use move to maintain structure
```

```
        shutil.move(label_path, os.path.join(dest_labels, os.path.splitext(filename)[0] + ".txt"))

move_data(source_train, os.path.join(dest_dir, "images", "train"), os.path.join(dest_dir, "labels", "train"))
move_data(source_test, os.path.join(dest_dir, "images", "test"), os.path.join(dest_dir, "labels", "test"))

print("Data moved to destination folders!")
```

```
Data moved to destination folders!
```

Deifne architecture of the dataset in .yaml file.

Besides redesigning the architecture of data, we must also give the algorithm a .yaml file to define the configurations of dataset. In this project, the .yaml file is as follows:

path: /content/ultralytics/ultralytics/datasets

train: images/train

val: images/test

nc: 1

names: 0: 'text'

```
shutil.copy('/content/AI.yaml' ,'/content/ultralytics/ultralytics/datasets')
```

```
'/content/ultralytics/ultralytics/datasets/AI.yaml'
```

```
import glob as glob
import matplotlib.pyplot as plt
import cv2
import requests
import random
import numpy as np
from google.colab.patches import cv2_imshow
```

Data Visualization

In order to examine data and labels and have an understanding of the goal of this project, we must first visualise a few samples of data. This task, requires previewing an image, finding its corresponding label file and using data inside text files to plot rectangles at the correct position. To do so, we defined a function which executes the above operations. This function inputs the image and label directories and outputs the resulting stacked image

```
def visualize_bounding_boxes(image_path, annotation_path):
    # Load the image
    image_BGR = cv2.imread(image_path)

    with open(annotation_path) as f:
        lst = [line.rstrip() for line in f]

    # Going through all bounding boxes
    for i in range(len(lst)):
        # Getting current bounding box coordinates, width, and height
        bb_current = lst[i].split()
        x_center, y_center = int(float(bb_current[1]) * image_BGR.shape[1]), int(float(bb_current[2]) * image_BGR.shape[0])
        box_width, box_height = int(float(bb_current[3]) * image_BGR.shape[1]), int(float(bb_current[4]) * image_BGR.shape[0])

        # Calculate top-left corner coordinates (x_min, y_min)
        x_min = int(x_center - (box_width / 2))
        y_min = int(y_center - (box_height / 2))

        # Draw bounding box on the original image
        cv2.rectangle(image_BGR, (x_min, y_min), (x_min + box_width, y_min + box_height), [172, 10, 127], 2)

        # Prepare text with label and confidence for the current bounding box
        class_current = 'Class: {}'.format(bb_current[0])

        # Put text with label and confidence on the original image
        cv2.putText(image_BGR, class_current, (x_min, y_min - 5), cv2.FONT_HERSHEY_COMPLEX, 0.7, [172, 10, 127], 2)

    # Display the image with bounding boxes
    cv2_imshow(image_BGR)
```

```
image_path = '/content/ultralytics/ultralytics/datasets/images/test/100.jpg'
annotation_path = '/content/ultralytics/ultralytics/datasets/labels/test/100.txt'
visualize_bounding_boxes(image_path, annotation_path)
```

We can see that the class name is also written on the image besides each bounding box, but since there are no other classes than "text" that would be unnecessary

Model Initialization and Training

Having Ultralytics package installed, we may use YOLO v8 model.

```
from ultralytics import YOLO

model = YOLO("yolov8n.pt")
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov8n.pt to 'yolov8n.pt'...
100%|██████████| 6.23M/6.23M [00:00<00:00, 132MB/s]
```

In this part, hyperparameters of model is defined. number of epoches can be chosen as larger numbers, but due to the computation cost, it is considered to be relatively low in the first run to assess the performance of the model.

```
model.train(data='/content/AI.yaml', epochs=10, imgsz=640)
```

Show hidden output

```
model.save('/content/yolov8.h5')
```

Prediction

At the end of the project, we try to predict a few examples using the code below. in a few cells below, we can see the examples and results of this model.

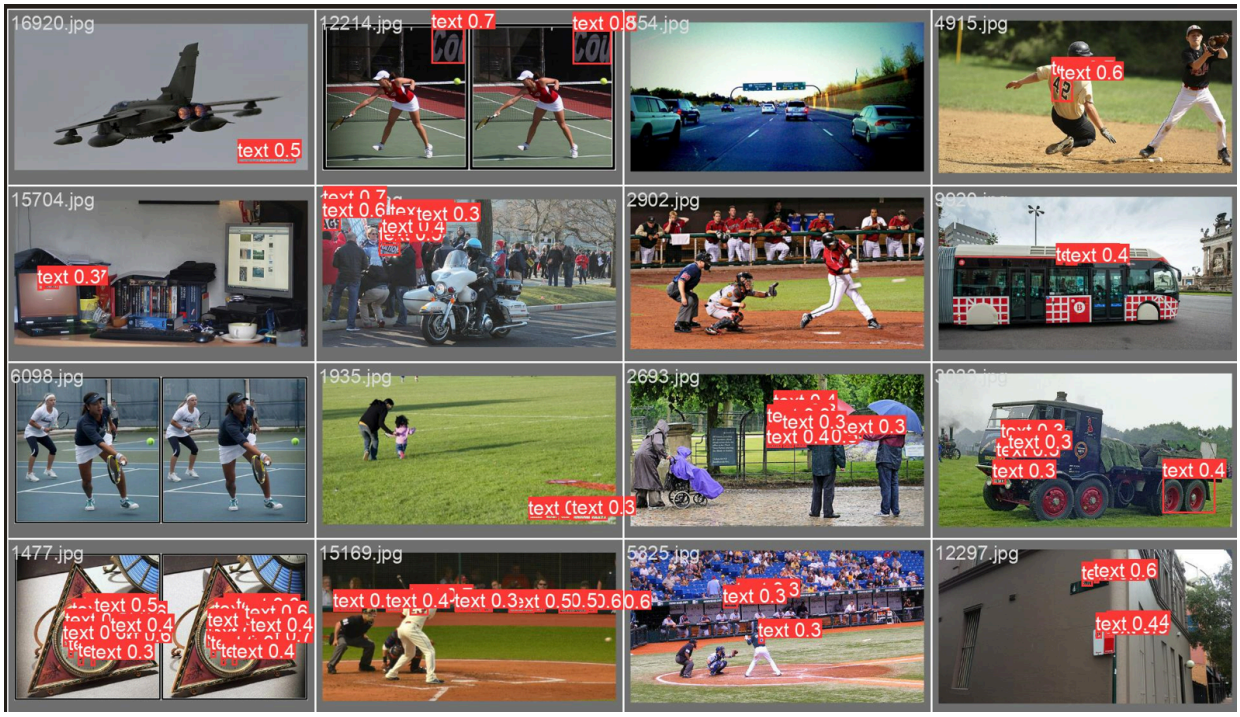
```
!yolo task=detect mode=predict model=/content/ultralytics/runs/detect/train/weights/best.pt conf=0.25 source=/content/ultralytics/ultralytics/datasets/image
```

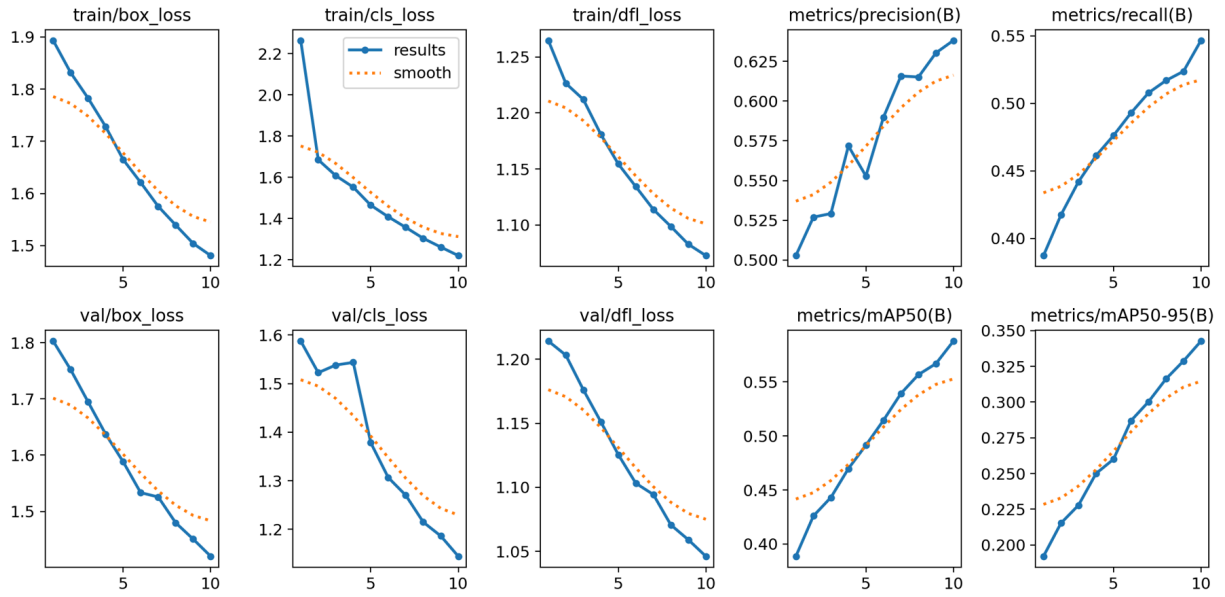
```
Ultralytics YOLOv8.2.29 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs

image 1/1 /content/ultralytics/ultralytics/datasets/images/test/10974.jpg: 640x640 12 texts, 19.6ms
```

```
Speed: 2.9ms preprocess, 19.6ms inference, 1006.0ms postprocess per image at shape (1, 3, 640, 640)
Results saved to /content/ultralytics/runs/detect/predict4
```
💡 Learn more at https://docs.ultralytics.com/modes/predict

Ground Truth1:

Prediction 1:






```
import locale
locale.getpreferredencoding = lambda: "UTF-8"
```

In order to download the runs file from colab, we musst first zip the folder so that it is downloadable from colab.

```
!zip -r /content/ultralytics.zip /content/ultralytics/
```

Show hidden output

Thank you