Downloading Object Detection dataset for custom classes from COCO dataset and converting it into YOLOv7 format

Install `fiftyone` library to download the custom dataset from COCO

- In this notebook I will be downloading datapoints from `TRAIN` split of `COCO` dataset.
- The custom classes are `Person` and `Car`.
- For knowing class names in the `COCO` dataset: Click Here!
- To download the dataset I will use `fiftyone` library.
- To know more about `fiftyone` : More about fiftyone!

In [1]:
```
%%capture
!pip install fiftyone tqdm
```

- I will download 5k instances containing either of the classes (`Person` or `Car`)

In [ ]:
```
import fiftyone.zoo as fzoo
# To download the COCO dataset for only the 'PERSON', and 'CAR' classes

train_dataset = fzoo.load_zoo_dataset(
    "coco-2017",
    splits=["train"],
    label_types=["detections"], # by default only detections are loaded
    # label_field=["detections"],
    classes=["person","car"],
    max_samples=5000,
    dataset_dir="./dataset/", # Save it to the dataset folder
    only_matching=True
)
```

Converting COCO format to YOLO format

In [8]:
```
import json
from pathlib import Path
```

In [137…
```
# setting input_path and output_path
```

```
input_path = Path("./dataset/train/")
output_path = Path("./car-person-detection/")
```

Reading JSON file

In [ ]:
```
with open(input_path/"labels.json","rb") as f:
    data = json.load(f)

# data
```

Preprocessing Images

Reading Images from the Input directory and saving them to the Output directory with updated name in TRAIN, VAL, and TEST folder. Also saving filenames of images for further use.

You have to make two changes:

1. Use shutile.move() or shutile.copy() instead of cv2.imwrite()
2. You have to change the line where Category_id will be written....(category_id - 1) or plus 1 according to your use case.

In [140…
```
train_filenames = None
val_filenames = None
test_filenames = None
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import shutil

def load_images_from_folder(folder, train_size=0.7, val_size=0.2, test_size=0.1):
      """""""
      folder = Path(folder)
      filenames = [filename.name for filename in folder.iterdir()]
      train, validation = train_test_split(filenames, train_size=train_size,random_state=42)
      test_size = (test_size)/(val_size+test_size)
      validation, test = train_test_split(validation, test_size=test_size, random_state=42)

      ### Moving TRAIN
      train_filenames = train
      count=0
      train_images_path = output_path/"images/train/"
      if not train_images_path.exists():
```

```
            train_images_path.mkdir(parents=True)
      for filename in tqdm(train_filenames):
            #   img = cv2.imread(os.path.join(folder,filename))
            #   cv2.imwrite(f"{output_path}images/train/img{count}.jpg", img)
            image_name = f"img{count}.jpg"
            shutil.copyfile(src = folder/filename,dst=train_images_path/image_name)
            count+=1

      ### Moving VAL
      val_filenames = validation
      count=0
      val_images_path = output_path/"images/val/"
      if not val_images_path.exists():
            val_images_path.mkdir(parents=True)
      for filename in tqdm(val_filenames):
            image_name = f"img{count}.jpg"
            shutil.copyfile(src = folder/filename,dst=val_images_path/image_name)
            count+=1

      ### Moving TEST
      test_filenames = test
      count=0
      test_images_path = output_path/"images/test/"
      if not test_images_path.exists():
            test_images_path.mkdir(parents=True)
      for filename in tqdm(test_filenames):
            image_name = f"img{count}.jpg"
            shutil.copyfile(src = folder/filename,dst=test_images_path/image_name)
            count+=1

      return (train_filenames, val_filenames, test_filenames)

train_filenames, val_filenames, test_filenames = load_images_from_folder('./dataset/train/data/')
```

```
100%|██████████| 3500/3500 [00:09<00:00, 379.68it/s]
100%|██████████| 1000/1000 [00:02<00:00, 404.33it/s]
100%|██████████| 500/500 [00:01<00:00, 377.21it/s]
```

Helper functions

In [142…

```
def get_img_ann(image_id):
    """
    this method return annotation of the image from the data(json)
    this method is dependent on global variable(data)
```

```
    """
    img_ann = []
    isFound = False
    for ann in data['annotations']:
        if ann['image_id'] == image_id:
            img_ann.append(ann)
            isFound = True
    if isFound:
        return img_ann
    else:
        return None
```

In [143…

```
def get_img(filename):
  """this method returns img file"""
  for img in data['images']:
    if img['file_name'] == filename:
      return img
```

Processing labels

Applying Conversion

In [144…

```
# sample annotation
data['annotations'][0]
```

Out[144]:

```
{'segmentation': [[214.59,
   205.04,
   218.39,
   203.27,
   218.39,
   198.97,
   221.18,
   195.42,
   225.73,
   193.9,
   228.77,
   192.39,
   241.17,
   193.4,
   243.45,
   212.13,
   252.57,
   213.65,
   252.06,
   199.98,
   256.87,
   201.25,
   260.92,
   204.03,
   263.45,
   206.56,
   267.75,
   223.27,
   259.91,
   230.86,
   249.78,
   256.68,
   253.58,
   261.24,
   243.39,
   262.67,
   241.78,
   258.9,
   236.94,
   258.1,
   237.21,
   252.45,
   239.9,
   252.45,
   240.17,
```

```
  236.05,
  237.48,
  224.49,
  233.17,
  219.92,
  225.11,
  219.11,
  219.73,
  216.42,
  214.62,
  210.77,
  213.81,
  206.47,
  215.43,
  205.13],
 [247.96, 237.39, 246.89, 254.87, 248.77, 238.2, 248.77, 238.2]],
'area': 1698.440800000001,
'iscrowd': 0,
'image_id': 116061,
'bbox': [213.81, 192.39, 53.94, 70.28],
'category_id': 18,
'id': 1728}
```

In [145…

```python
# Processing labels for each split TRAIN, VAL, and TEST

def process_label(filenames, output_path, class_ids):
    count = 0
    for filename in tqdm(filenames):
        # Extracting Image
        img = get_img(filename)
        img_id = img['id']
        img_w = img['width']
        img_h = img['height']

        # Get annotations for this image
        img_ann = get_img_ann(img_id)

        # If img_ann is not None
        if img_ann:
            # Opening file for current image
            file_object = open(f"{output_path}/img{count}.txt", "a")

            for ann in img_ann:
                current_category = ann['category_id']
                if current_category in class_ids:
```

```
                    current_bbox = ann['bbox']

                    x = current_bbox[0]
                    y = current_bbox[1]
                    w = current_bbox[2]
                    h = current_bbox[3]

                    # Finding midpoints
                    x_centre = (x + (x+w))/2
                    y_centre = (y + (y+h))/2

                    # Normalization
                    x_centre = x_centre/img_w
                    y_centre = y_centre/img_h
                    w = w/img_w
                    h = h/img_h

                    # Limiting unpto fix number of decimal places
                    x_centre = format(x_centre, '.6f')
                    y_centre = format(y_centre, '.6f')
                    w = format(w, '.6f')
                    h = format(h, '.6f')

                    # Writing current object
                    # category_name = category_map[category_name] # category map from
                    if current_category==1:
                        current_category=0
                    else:
                        current_category=1
                    file_object.write(f"{current_category} {x_centre} {y_centre} {w} {h}\n")

            file_object.close()
            count+=1
    print(f"total_count: {count}")
```

In [146…

```
# this class_ids will be used to filter out other classes
# because in annotation we are having all classes details
class_ids = [1,3]# 1-person, 3-car
```

In [149…

```
# Label Conversion for TRAIN
train_labels_dir = Path("./car-person-detection/labels/train/")
if not train_labels_dir.exists():
    train_labels_dir.mkdir(parents=True)
process_label(filenames=train_filenames, output_path="./car-person-detection/labels/train/",class_ids=class_ids)
```

```
100%|██████████| 3500/3500 [00:57<00:00, 60.42it/s]
total_count: 3500
```

In [150…
```
# Label Conversion for VAL
val_labels_dir = Path("./car-person-detection/labels/val/")
if not val_labels_dir.exists():
    val_labels_dir.mkdir(parents=True)
process_label(filenames=val_filenames, output_path="./car-person-detection/labels/val/",class_ids=class_ids)
```

```
100%|██████████| 1000/1000 [00:19<00:00, 50.30it/s]
total_count: 1000
```

In [151…
```
# Label Conversion for TRAIN
test_labels_dir = Path("./car-person-detection/labels/test/")
if not test_labels_dir.exists():
    test_labels_dir.mkdir(parents=True)
process_label(filenames=test_filenames, output_path="./car-person-detection/labels/test/",class_ids=class_ids)
```

```
100%|██████████| 500/500 [00:08<00:00, 56.52it/s]
total_count: 500
```

Uploading dataset to kaggle

Install kaggle

In [105…
```
%%capture

!pip install kaggle --user
```

run the below command

In [ ]:
```
!mkdir ~/.kaggle/
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

initialize the dataset

- I have created zip file of the dataset
- It is better to zip the dataset and then upload to Kaggle.

- Kaggle will extract the dataset by default

In [ ]:
```
# init the dataset metadata
# car-persob-detection is the dataset name, change it by your name,
# otherwise will get an error of data already exist.
!kaggle datasets init -p car-person-detection
```

In [ ]:
```
import os
os.environ['KAGGLE_USERNAME'] = 'mdiqbalbajmi'
os.environ['KAGGLE_KEY'] = '<kaggle_api_token>'
```

In [ ]:
```
# this will zip the folder, upload and kaggle will unzip it by default
!kaggle datasets create -p car-person-detection -r zip
```

- Now you are ready to go
- Make the dataset public
- Add it to you kaggle notebook
- And Voila! Train your own Yolov7 models for your custom dataset