Tutorial of Oil Tank Detection with YOLO-v3

1 Purpose

In a set of **remote sensing images**(n. 遥感图片), we need to detect the oil tanks in these images as much as possible and for each oil tank detected by our model, we need to put a rectangular box which indicates the position of the oil tank object.

The source of data set is in **Kaggle**
If you don't want to read the detailed information on this website, just download the data set by this **link**

2 Introduction of the data set

This data set has 2 folders, 2 `.json` files and 1 `.csv` file.

2.1 Folders

The folders are image_patches and large_images, the images_pathches come from large_images.

2.1.1 large_images

This folder contains 100 raw 4800 * 4800 images saved from Google Earth, with `id_large.jpg` formate.

2.1.2 image_patches

The image_patches directory contains 512 * 512 patches generated from the large image. Each large image is split into 100 512 * 512 patches with an overlap of 37 pixels between patches on both axes. Image patches are named following an `id_row_column.jpg` format.

2.2 .json files

2.2.1 labels.json

labels.json contains labels for all images. Labels are stored as a list of dictionaries, one for each image. Images that do not contain any floating head tanks are given a label of 'skip'. Bounding box labels are in the format of (x, y) coordinate pairs of the four corners of the bounding box.

2.2.2 labels_coco.json

labels_coco.json contain the same labels as the previous file, converted into COCO label format. Here bounding boxes are formatted as [x_min, y_min, width, height]

2.3 large_image_data.csv

large_image_data.csv contains metadata about the large image files, including coordinates of the center of each image and the altitude.

3 Fundamental Steps & Tools

3.1 Steps

> Load the pre-training model. -> Partitioning data set. -> Train & Validate -> Quality Assessment -> Test

3.2 Tools Applied

YOLO Algorithm! You Only Look Once!
The introduction of YOLO v3 🚀
Train custom data with YOLO v3 🚀

Google Colab 🚀 **RECOMMEND!** FREE GPU for 12 HOURS per MONTH

4 Experimental Procedure

Some source code is from **YOLO v3 Google Colab**, I'll attribute them.

4.1 Install YOLO v3 & Detect the Environment

This block is from **YOLO v3 Google Colab**

When you run this block, it will **clone YOLO v3 from github into your current folder automatically**. Then, it will detect the environment such as version of python and the GPU name, version of CUDA.

```
!git clone https://github.com/ultralytics/yolov3  # clone repo
%cd yolov3
%pip install -qr requirements.txt  # install dependencies

import torch
from IPython.display import Image, clear_output  # to display images

clear_output()
print(f"Setup complete. Using torch {torch.__version__}
({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else
'CPU'})")
```

4.2 Install some Practical Tools(Optional)

This block is from **YOLO v3 Google Colab**

4.2.1 Tensorboard

```
# Tensorboard (optional)
%load_ext tensorboard
%tensorboard --logdir runs/train
```

4.2.2 Weights & Biases

```
# Weights & Biases  (optional)
%pip install -q wandb
import wandb
wandb.login()
# find your api key for wandb here: https://wandb.ai/authorize
```

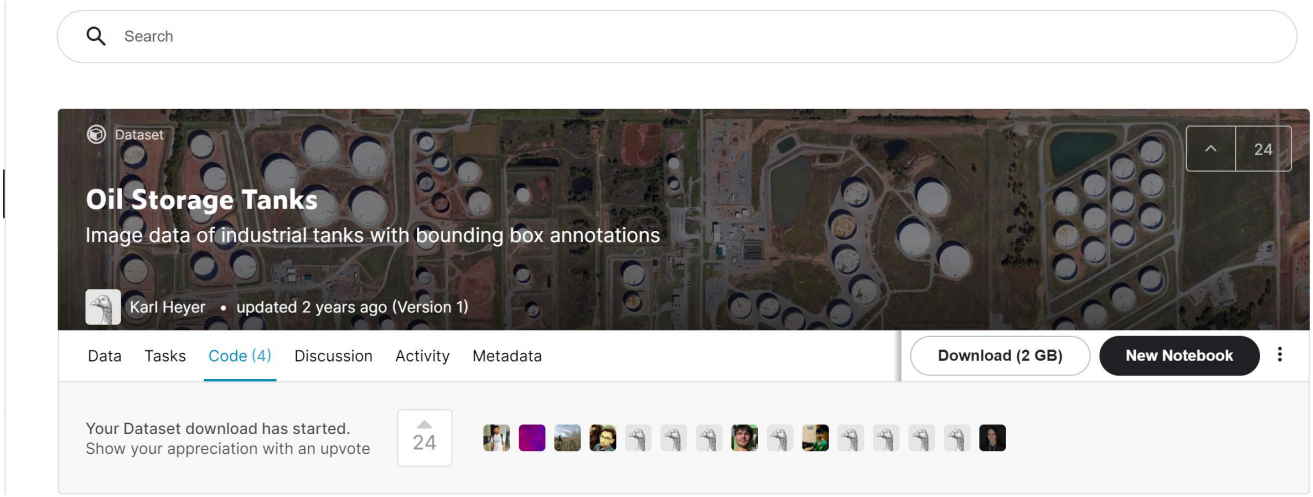
4.3 Download Oil Tanks Data Set

```
# Download OilTanks
torch.hub.download_url_to_file('https://storage.googleapis.com/kaggle-data-sets/217778/471818/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20210529%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20210529T015211Z&X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=7841f273484f6db0a67b132b1ffd092b68d370dadde25e54c2ebfecaf88b58bfe1d2c90e6d75eb945e6135a74abcaee91b29e75c4a731375148a0e5873556e311f04991c33bd7b6908837eabbb05376d1b391654fd3a06397553798b5f1462b516cf07168c1c10c4a5fa8cd1cb802d13b91f436dec896fc893858497755f4616d18e8ec20570733ab23fc47e0a3d068b0c090b70e53122802937a434610b0b5dcb6fd1d2f7c2d68acdefa24c579cbab6c6460261ef77fa4f70c2c539519922bb5ac6288251a1adeb6eb0b7a79e483d137c81d5389f1f666e9e7d856cc16f980852f8de25992c01f88593ad588ae2d4de34e083fa19bdf8e3effeb0746eb9cfeb', 'tmp.zip')
!unzip -q tmp.zip -d ../ && rm tmp.zip
```

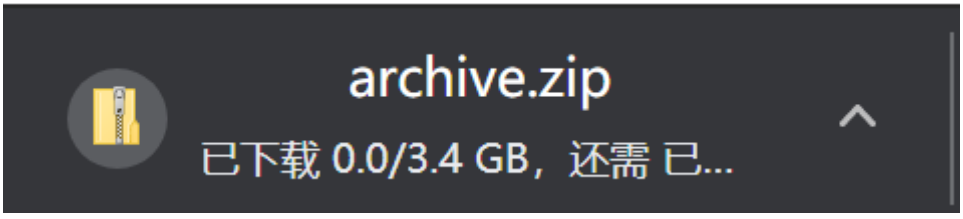
When you run it, the oil tank data set will load into your current folder.

Oops, seems the download link is changed. Never mind, try to use this method:

1. click download button.



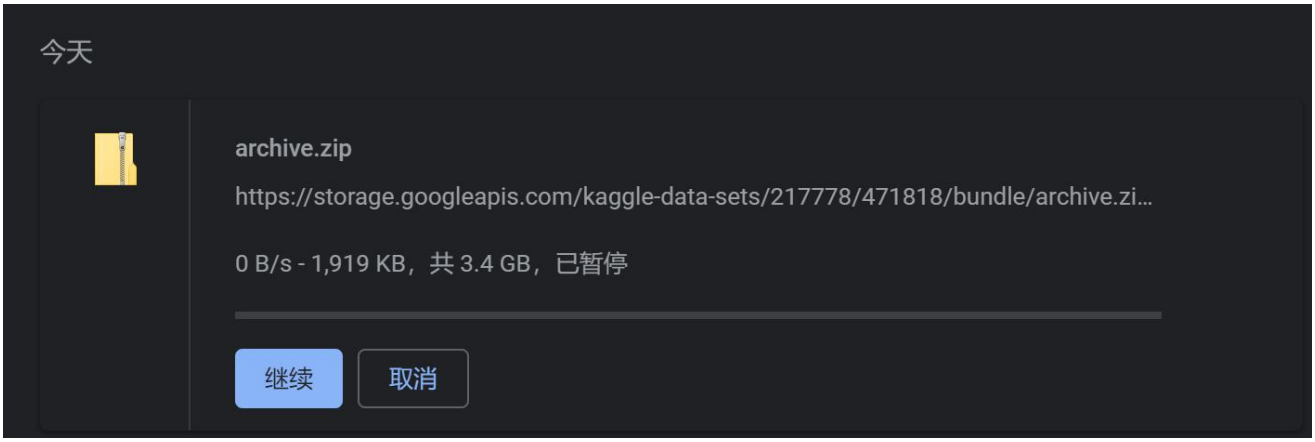
2. suspend the downloading



3. click "全部显示"(Um, I haven't use chrome in English version so I don't know how to say it in English)

4. copy the download link

Look at the line that "**https://storage**...", this is the download link, copy it.



5. replace the 1st parameter in `download_url_to_file()` with the copied link.

4.4 Partitioning Data Set

I just partitioning as train : validate = 8 : 2, in the training folder, there are 8000 images while in validation folder, there are 2000 images.

```
# making train and val
import os
import natsort
import shutil
from pathlib import Path

def make_fold_if_not_exist(dest_path):
  if not Path(dest_path).exists():
    os.mkdir(dest_path)

def move_to(source,to_train,to_val,to_test,files):
  files = natsort.natsorted(files)
  for i in range(int(0.7*len(files))):
    full_path = os.path.join(source,files[i])
    des_path = os.path.join(to_train,files[i])
    shutil.move(full_path,des_path)
  for i in range(int(0.2*len(files))):
    full_path = os.path.join(source,files[i+int(0.7*len(files))])
    des_path = os.path.join(to_val,files[i+int(0.7*len(files))])
    shutil.move(full_path,des_path)
  for i in range(int(0.1*len(files))):
    full_path = os.path.join(source,files[i+int(0.9*len(files))])
    des_path = os.path.join(to_test,files[i+int(0.9*len(files))])
    shutil.move(full_path,des_path)

path_origin = "../Oil Tanks/image_patches"
path_train = "../Oil Tanks/images/train"
path_val = "../Oil Tanks/images/val"
path_test = "../Oil Tanks/images/test"

# make dir for train ,val and test
```

```
make_fold_if_not_exist(path_origin)
make_fold_if_not_exist(path_train)
make_fold_if_not_exist(path_val)
make_fold_if_not_exist(path_test)

files = os.listdir(path_origin)
move_to(path_origin,path_train,path_val,path_test,files)

files_train = os.listdir(path_train)
nums_train = len(files_train)
print(nums_train)
files_train=natsort.natsorted(files_train)
print(files_train)

files_val = os.listdir(path_val)
nums_val = len(files_val)
print(nums_val)
files_val=natsort.natsorted(files_val)
print(files_val)

files_test = os.listdir(path_test)
nums_test = len(files_test)
print(nums_test)
files_test=natsort.natsorted(files_test)
print(files_test)
```

4.5 Modify `.yaml` file

4.5.1 `coco128.yaml`

This is the original coco128.yaml, to find it, follow this path:

> `yolov3` (folder)-> `data` (folder)-> `coco128.yaml`

```
# COCO 2017 dataset http://cocodataset.org
# Train command: python train.py --data coco.yaml
# Default dataset location is next to YOLOv3:
#   /parent_folder
#     /coco
#     /yolov3


# download command/URL (optional)
download: bash data/scripts/get_coco.sh

# train and val data as 1) directory: path/images/, 2) file:
path/images.txt, or 3) list: [path1/images/, path2/images/]
train: ../coco128/train2017/  # 118287 images
```

```
14 val: ../coco128/val2017/  # 5000 images
15
16 # number of classes
17 nc: 80
18
19 # class names
20 names: [ 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
   'train', 'truck', 'boat', 'traffic light',
21          'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird',
   'cat', 'dog', 'horse', 'sheep', 'cow',
22          'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella',
   'handbag', 'tie', 'suitcase', 'frisbee',
23          'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat',
   'baseball glove', 'skateboard', 'surfboard',
24          'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife',
   'spoon', 'bowl', 'banana', 'apple',
25          'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
   'donut', 'cake', 'chair', 'couch',
26          'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop',
   'mouse', 'remote', 'keyboard', 'cell phone',
27          'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book',
   'clock', 'vase', 'scissors', 'teddy bear',
28          'hair drier', 'toothbrush' ]
```

Where I need to modify is the `download`, this step is optional and in this experiment we don't need it. So just add `#` at the head of line 10.

Then, I need to modify the train/val/test path. These path are from

Finally, in this lab, although we have 3 kinds of classes(This is from data set, in `label.json`), but I just map these 3 kinds of objects all into 1 kind: `tanks`, in order to test how to use YOLO v3. Therefore, I also need to modify the `nc` in line 18 and the `names`, an array start from line 21.

My coco128.yaml is look like this:

```
1  # COCO 2017 dataset http://cocodataset.org
2  # Train command: python train.py --data coco.yaml
3  # Default dataset location is next to YOLOv3:
4  #   /parent_folder
5  #     /coco
6  #     /yolov3
7
8
9  # download command/URL (optional)
10 # download: bash data/scripts/get_coco.sh
11
12 # train and val data as 1) directory: path/images/, 2) file:
   path/images.txt, or 3) list: [path1/images/, path2/images/]
13 train: ../Oil Tank/images/train
14 val: ../Oil Tank/images/val
```

```
15
16 # number of classes
17 nc: 1
18
19 # class names
20 names: [ 'Tank']
```

4.5.2 `coco.yaml`

The principle of modifying `coco.yaml` is same as modifying `coco128.yml`, the difference is that in `coco.yaml`, there is a test fold you need to point out. Then I modify my `coco.yaml` like this:

```
1  # COCO 2017 dataset http://cocodataset.org
2  # Train command: python train.py --data coco.yaml
3  # Default dataset location is next to YOLOv3:
4  #   /parent_folder
5  #     /coco
6  #     /yolov3
7
8
9  # download command/URL (optional)
10 # download: bash data/scripts/get_coco.sh
11
12 # train and val data as 1) directory: path/images/, 2) file:
   path/images.txt, or 3) list: [path1/images/, path2/images/]
13 train: ../Oil Tanks/images/train  # 7000 images
14 val: ../Oil Tanks/images/val  # 2000 images
15 test: ../Oil Tanks/images/test  # 1000 images
16
17 # number of classes
18 nc: 1
19
20 # class names
21 names: [ 'Tank']
22
23 # Print classes
24 # with open('data/coco.yaml') as f:
25 #   d = yaml.safe_load(f)  # dict
26 #   for i, x in enumerate(d['names']):
27 #     print(i, x)
```

4.6 make labels

The fundamental idea in this step is that for every pictures in train/val/test, generating the specific file to store labels automatically(If an oil tank is in this picture, else we could skip generating).

For instance, if there is a picture called `01_5_9.jpg` with 6 oil tanks in `labels.json`:

```
...
{"id": 59, "file_name": "01_5_9.jpg", "label": {"Tank": [{"geometry": [{"x": 373, "y": 47}, {"x": 373, "y": 2}, {"x": 321, "y": 2}, {"x": 321, "y": 47}]}, {"geometry": [{"x": 500, "y": 46}, {"x": 500, "y": 2}, {"x": 450, "y": 2}, {"x": 450, "y": 46}]}], "Floating Head Tank": [{"geometry": [{"x": 47, "y": 251}, {"x": 47, "y": 124}, {"x": -1, "y": 124}, {"x": -1, "y": 251}]}, {"geometry": [{"x": 190, "y": 171}, {"x": 190, "y": 234}, {"x": 251, "y": 234}, {"x": 251, "y": 171}]}, {"geometry": [{"x": 193, "y": 62}, {"x": 193, "y": 123}, {"x": 255, "y": 123}, {"x": 255, "y": 62}]}, {"geometry": [{"x": 316, "y": 94}, {"x": 316, "y": 162}, {"x": 381, "y": 162}, {"x": 381, "y": 94}]}]}}
...
```

> Why could there be -1?
>
> Because every picture is 512 * 512, while they are come from splitting original 4800 * 4800 picture into 100 smaller pictures, therefore, the majority of the images have **borrow the board** from former and latter, upper and lower images. However, some position is not a part of the image its' content, then the position turns into some negative numbers like "-1".

- In this file, I have ignored the labels with negative numbers, otherwise the calculation will be more complex than now.
- Another point is that there may be some tanks that are labeled as "**Floating Head Tank**", I just labeled them as "**Tank**", but in the **next edition** of this experiment, I will **not** cover them as "**Tank**".
- Some tanks in this data set are not labeled. This is the "**missing annotation**".

Here is the code:

```
# making labels
import os
import cv2
import json

def valid(L):
  if L[0]['x']<0 or L[0]['y']<0 or L[1]['x']<0 or L[1]['y']<0 or L[2]['x']<0 or L[2]['y']<0 or L[3]['x']<0 or L[3]['y']<0:
    return False
  else:
    return True
```

```
11
12  def get_point_w_h(L):
13    h = L[2]['y']-L[0]['y']
14    w = L[2]['x']-L[0]['x']
15    point_x = L[0]['x']+w/2
16    point_y = L[0]['y']+h/2
17    return point_x,point_y,abs(w),abs(h)
18
19  def get_double(x,y,w,h):
20    s = 512*1.0
21    x = 1.0*x/s
22    y = 1.0*y/s
23    w = 1.0*w/s
24    h = 1.0*h/s
25    return x,y,w,h
26
27  path = ""
28  path_origin = "../Oil Tanks/labels"
29  path_train = "../Oil Tanks/labels/train/"
30  path_val = "../Oil Tanks/labels/val/"
31  path_test = "../Oil Tanks/labels/test/"
32
33  # mkdir for label
34  make_fold_if_not_exist(path_origin) # make_fold_if_not_exists() is
    implemented in section 4.4
35  make_fold_if_not_exist(path_train)
36  make_fold_if_not_exist(path_val)
37  make_fold_if_not_exist(path_test)
38
39  with open("../Oil Tanks/labels.json",'r') as load_f:
40    load_dict = json.load(load_f)
41    print(load_dict)
42    print(len(load_dict))
43    for dic in load_dict:
44      if dic['label']!="Skip":
45        # print(dic['file_name'][0:-4])
46        if dic['label'].keys():
47          # make the path of train or val
48          img_name = dic['file_name'][0:-4]
49          if (img_name[0:1] == '9' and img_name[0:2] != '90') or
    img_name[0:3] == '100':
50            path = path_test
51          elif (img_name[0:1] == '7' and img_name[0:2] != '70') or
    img_name[0:2] == '90' or img_name[0:1] == '8':
52            path = path_val
53          else:
54            path = path_train
55          file_name = path+img_name+".txt"
56          print(file_name)
57          # create the txt file
58          f = open(file_name,'w')
59          for i in range(len(dic['label'].keys())):
```

```
            for item in dic['label'][list(dic['label'].keys())[i]]:
                point_x,point_y,w,h = get_point_w_h(item['geometry'])
                point_x,point_y,w,h = get_double(point_x,point_y,w,h)
                print(item['geometry'])
                print("%.6f %.6f %.6f %.6f"%(point_x,point_y,w,h))
                formate_str = "%d %.6f %.6f %.6f %.6f\r\n"%
    (0,point_x,point_y,w,h)
                f.write(formate_str)
```

4.7 Train

```
# Train YOLOv3 on COCO for 4 epochs
!python train.py --img 512 --batch 16 --epochs 4 --data coco.yaml --weights
yolov3.pt --nosave --cache
```

If you want to use `coco128.yaml`, just replace the `coco.yaml` with it.

This step is very time consuming

When it has been done, we will got a file names `best.pt` in `yolov3/runs/train/exp/weights/`. This file is very **significant**. Because this file is the **target of training**!

4.8 Visualization

4.8.1 Local Logging

```
Image(filename='runs/train/exp/train_batch0.jpg', width=800)  # train batch
0 mosaics and labels
Image(filename='runs/train/exp/test_batch0_labels.jpg', width=800)  # test
batch 0 labels
Image(filename='runs/train/exp/test_batch0_pred.jpg', width=800)  # test
batch 0 predictions

Image(filename='runs/train/exp/train_batch1.jpg', width=800)  # train batch
1 mosaics and labels
Image(filename='runs/train/exp/test_batch1_labels.jpg', width=800)  # test
batch 1 labels
Image(filename='runs/train/exp/test_batch1_pred.jpg', width=800)  # test
batch 1 predictions

Image(filename='runs/train/exp/train_batch2.jpg', width=800)  # train batch
2 mosaics and labels
Image(filename='runs/train/exp/test_batch2_labels.jpg', width=800)  # test
batch 2 labels
Image(filename='runs/train/exp/test_batch2_pred.jpg', width=800)  # test
batch 2 predictions
```

```
from utils.plots import plot_results
plot_results(save_dir='runs/train/exp')  # plot all results*.txt as
results.png
Image(filename='runs/train/exp/results.png', width=800)
```

4.8.2 wandb

This step will be available if you have run section 4.2.2.

Open the **wandb** in your browser and you will find the record for your experiment.

4.9 Detect

If you have a director and there are many images needed detecting in that director, try as this:

```
!python detect.py --weights runs/train/exp/weights/best.pt --img 512 --conf
0.25 --source ../Oil\ Tanks/images/test/
```

The `weights` is the `.pt` file we got in train. The `conf` is set by yourself between 0-1. The `source` is the director holding images needed detecting. I just use the `test` director to detect.

4.10 Test

```
!python test.py --weights runs/train/exp/weights/best.pt --data coco.yaml --
img 512 --iou 0.5 # 暂时用不来 遇到瓶颈了
```

This is supposed to be section 4.9, however, I got something wrong in this command. Therefore, I put it in section 4.10, which means I need time to repair it.