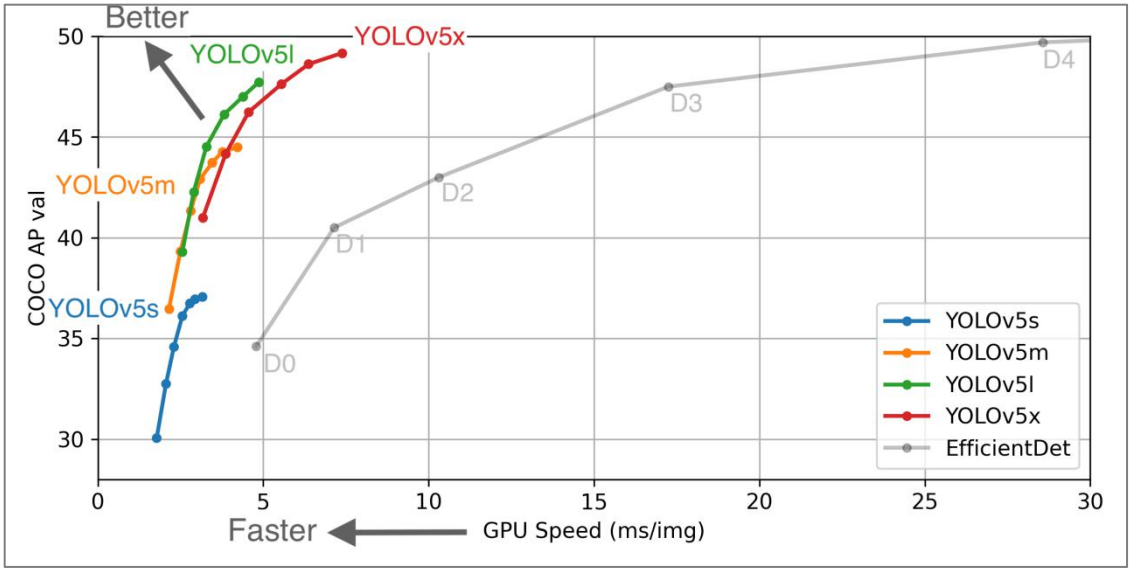
Path:
dofbot_ws/src/dofbot_color_identify/scripts/Garbage_sorting.ipynb
dofbot_ws/src/dofbot_color_identify/scripts/Garbage_sorting_single.ipynb

1. Yolov5 algorithm performance test chart.



2. Environmental requirements

Cython
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
pillow
PyYAML>=5.3
scipy>=1.4.1
tensorboard>=2.2
torch>=1.6.0
torchvision>=0.7.0
tqdm>=4.41.0
imgaug

3. Model training

3.1 Create your own data set in the following format:

```
garbage/train/images      be used to store data Images of data sets
garbage/train/labels    be used to store data labels of data sets
```

Tips:

The name of the picture and the label file should be one-to-one correspondence. The form of the label file is that category center point x, center point y, border width w, border height h, and the label format is as follows:

```
label    x y w h
14 0.5192307692307692 0.3425480769230769 0.4423076923076923 0.37259615384615385
```

3.2 Create your own garbage.yaml file

```
# train and val data
train: ./garbage/train/images/
val: ./garbage/train/images/
# number of classes
nc: 16
# class names
names: ["Zip_top_can", "Old_school_bag", "Newspaper", "Book", "Toilet_paper",
        "Peach_pit", "Cigarette_butts", "Disposable_chopsticks", "Egg_shell",
        "Apple_core", "Watermelon_rind", "Fish_bone", "Expired_tablets",
        "Expired_cosmetics", "Used_batteries", "Syringe"]
```

3.3 Modify models/ yolov5s.yaml file

```
# parameters
nc: 16    # number of classes
```

Find nc: The number of categories is consistent with the custom yaml file

3.4 Related parameter configuration for model training.

```
if __name__ == '__main__':
    check_git_status()
    parser = argparse.ArgumentParser()
    parser.add_argument('--epochs', type=int, default=5)
    parser.add_argument('--batch-size', type=int, default=4)
    parser.add_argument('--cfg', type=str, default='models/yolov5s.yaml', help='*.cfg path')
    parser.add_argument('--data', type=str, default='data/coco128.yaml', help='*.data path')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640], help='train,test
sizes')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', action='store_true', help='resume training from last.pt')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
```

```
    parser.add_argument('--evolve', action='store_true', help='evolve hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache images for faster
training')
    parser.add_argument('--weights', type=str, default='', help='initial weights path')
    parser.add_argument('--name', default='', help='renames results.txt to results_name.txt if
supplied')
    parser.add_argument('--device', default='0', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--adam', action='store_true', help='use adam optimizer')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%')
    parser.add_argument('--single-cls', action='store_true', help='train as single-class dataset')
    opt = parser.parse_args()
```

Epochs: Refers to how many times the entire data set will be iterated during the training process. If the graphics card does not work, you can reduce it.
batch-size: How many images are read at a time before the weight is updated. For mini-batch with gradient descent, if the graphics card is not good, you can adjust it to a smaller size.
cfg: the configuration file that stores the model structure
data: files for storing training and test data
img-size: Enter the width and height of the image, if the graphics card fails, adjust it to a smaller size.
rect: Rectangle training
resume: resume the most recently saved model and start training
nosave: save only the final checkpoint
notest: only test the last epoch
evolve: evolutionary hyperparameters
bucket: gsutil bucket
cache-images: cache images to speed up training
weights: weight file path
name: Rename results.txt to results_name.txt
device: cuda device, i.e. 0 or 0,1,2,3 or cpu
adam: use adam optimization
multi-scale: Multi-scale training, img-size +/- 50%
single-cls: single-category training set

3.5 Open the terminal, execute the command to start training

```
python train.py --img 416 --batch 16 --epochs 300 --data ./data/garbage.yaml
--cfg ./models/yolov5s.yaml --weights yolov5s.pt
```

3.6 Model test

```
python detect.py --source 0    # webcam
                               file.jpg    # image
```

```
file.mp4    # video
path/    # directory
path/*.jpg    # glob
```

4. Single Garbage sorting

4.1 Import head file

```
#!/usr/bin/env python
# coding: utf-8
import Arm_Lib
import cv2 as cv
import threading
from time import sleep
import ipywidgets as widgets
from IPython.display import display
from single_garbage_identify import single_garbage_identify
```

4.2 Create an instance, initialize parameters

```
single_garbage = single_garbage_identify()
model = "General"
```

4.3 Initialize the robot position

```
import Arm_Lib
arm = Arm_Lib.Arm_Device()
joints_0 = [90, 135, 0, 0, 90, 0]
arm.Arm_serial_servo_write6_array(joints_0, 1000)
```

4.4 Create control widgets

```
button_layout          = widgets.Layout(width='320px', height='60px', align_self='center')
output = widgets.Output()

exit_button = widgets.Button(description='Exit', button_style='danger', layout=button_layout)
imgbox = widgets.Image(format='jpg', height=480, width=640,
layout=widgets.Layout(align_self='center'))
controls_box = widgets.VBox([imgbox, exit_button], layout=widgets.Layout(align_self='center'))
```

4.5 Exit program button callback

```
def exit_button_Callback(value):
    global model
    model = 'Exit'
    with output: print(model)
exit_button.on_click(exit_button_Callback)
```

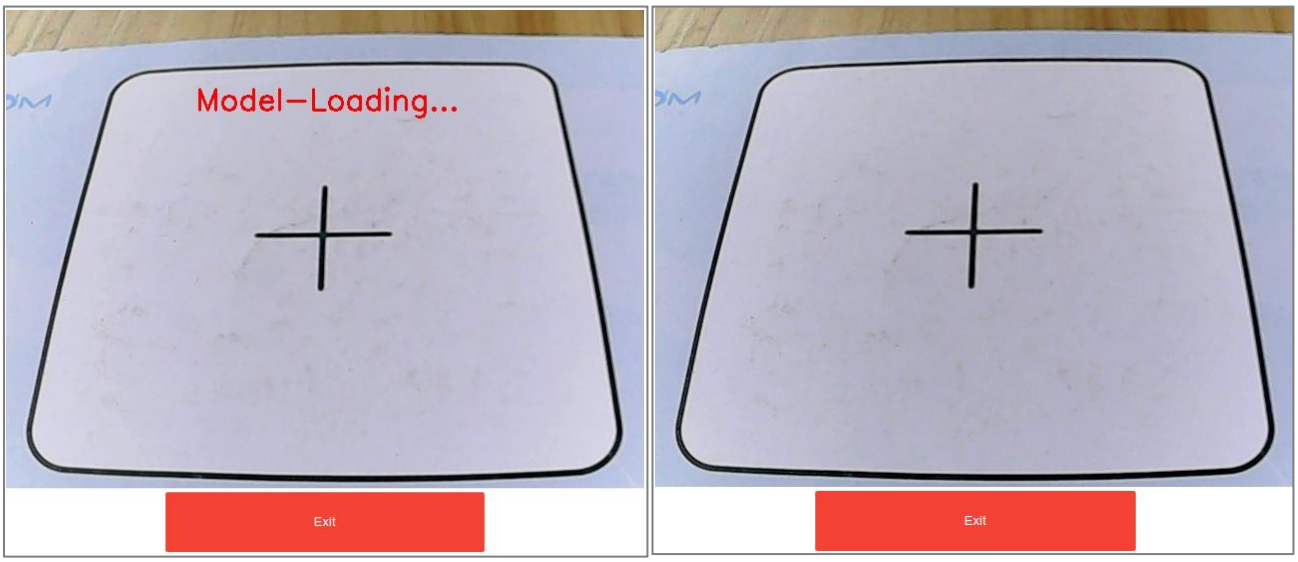
4.6 Main process

```
def camera():
    # Open camera
    capture = cv.VideoCapture(0)
    # When the camera is normally opened, the following code is executed in loop
        while capture.isOpened():
        try:
            _, img = capture.read()
            img = cv.resize(img, (640, 480))
            img = single_garbage.single_garbage_run(img)
            if model == 'Exit':
                cv.destroyAllWindows()
                capture.release()
                break
            imgbox.value = cv.imencode('.jpg', img)[1].tobytes()
        except KeyboardInterrupt:capture.release()
```

4.7 Start

```
display(controls_box,output)
threading.Thread(target=camera, ).start()
```

After running the above code, it will take some time to load the model, please be patient.
Place map as shown below.
After the robotic arm continues to accurately identify 10 times, it will automatically sort the blocks.



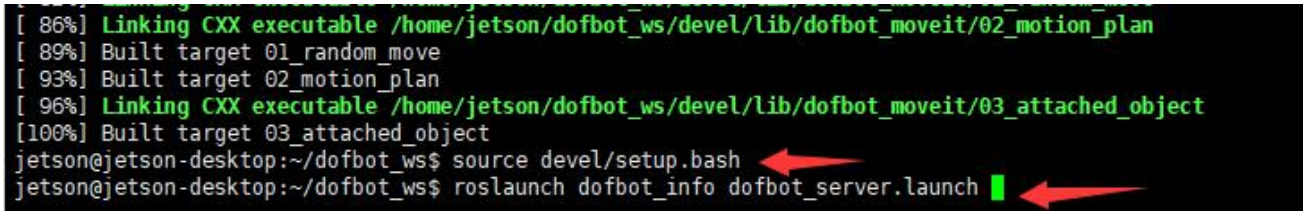
5. Garbage sorting

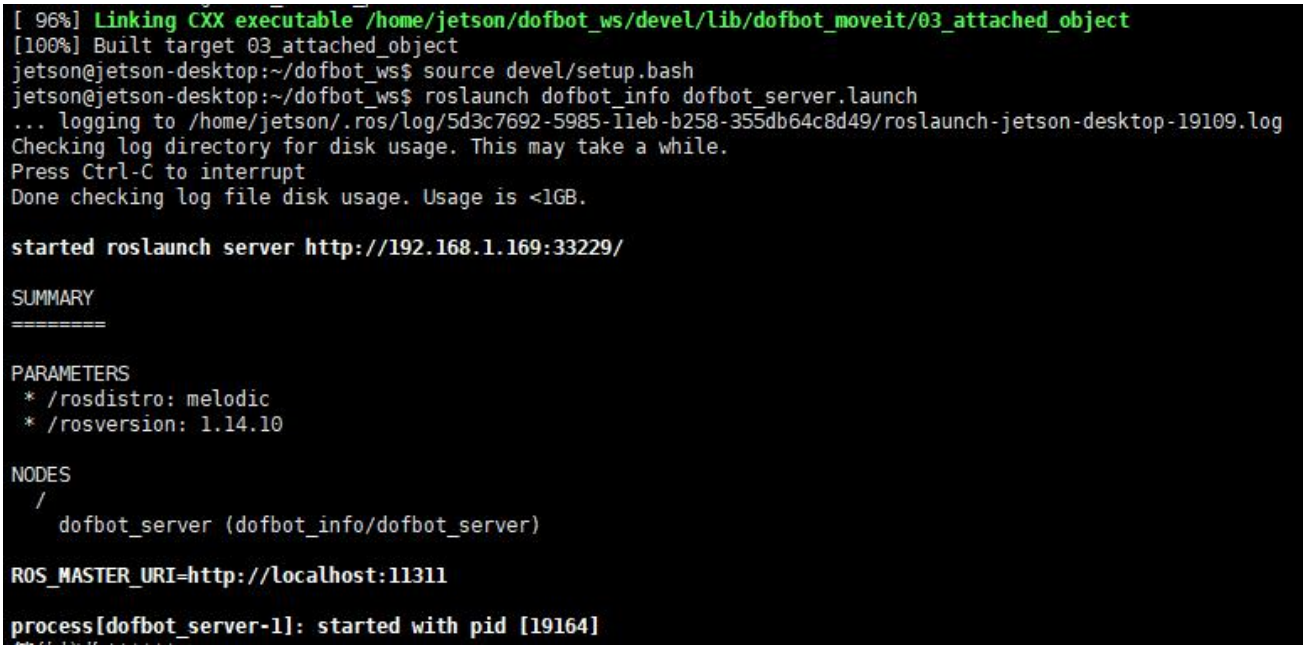
5.1 Before starting color sorting and stacking function, you must start the reverse solution server and keep running.

Input following command:

```
cd ~/dofbot_ws/    # Entering Workspace
catkin_make      # compile
source devel/setup.bash    # Update the system environment
roslaunch dofbot_info dofbot_server.launch    # Start the server terminal node
```

As shown below.





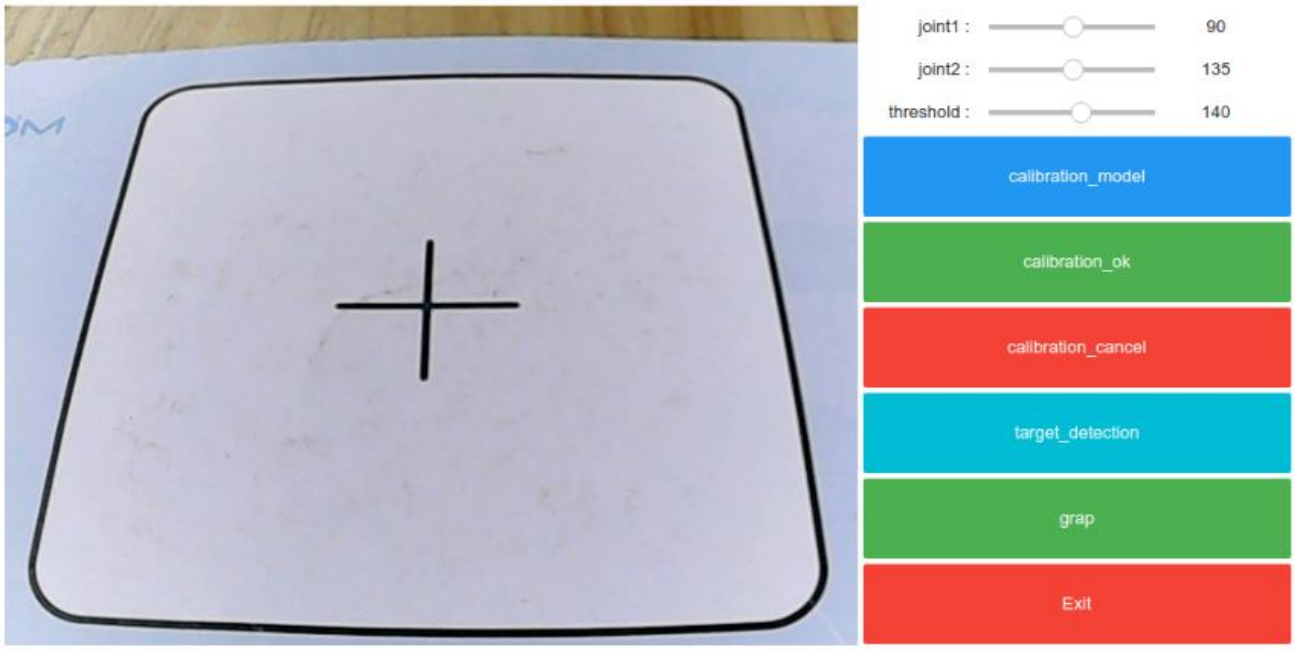
5.2 Box calibration

After the color calibration is completed, box calibration is required to obtain the position information of the block in the box.

More Details, please check [Visual positioning] course.

5.3 Running garbage sorting program

After running the program, you will see the interface as shown below.



5.4 After the [box calibration] is completed, click the [target_detection] button to load the model for identification.

5.5 When the block is recognized on map, click the [grap] button to start DOFBOT.

5.6 When the game is over, please click the [Exit] button to exit the program.