2. Target detection

2. Target detection

The main problem solved in this section is how to use the dnn module in OpenCV to import a trained target detection network. But there are requirements for the version of opencv.

Currently, there are three main methods for using deep learning for target detection:

- Faster R-CNNs
- You Only Look Once(YOLO)
  -Single Shot Detectors(SSDs)

Faster R-CNNs is the most commonly heard neural network based on deep learning. However, this approach is technically difficult to understand (especially for deep learning newbies), difficult to implement, and difficult to train.

In addition, even if the "Faster" method is used to implement R-CNNs (where R represents the region proposal), the algorithm is still relatively slow, about 7FPS.

If we are pursuing speed, we can turn to YOLO because it is very fast and can reach 40-90 FPS on TianXGPU, and the fastest version may reach 155 FPS. But the problem with YOLO is that its accuracy needs to be improved.

SSDs were originally developed by Google and can be said to be a balance between the above two. Compared with Faster R-CNNs, its algorithm is more straightforward. Compared with YOLO, it is more accurate.

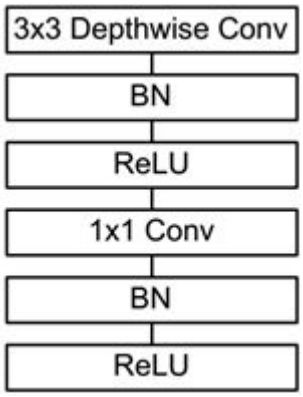
2.1. Model structure

The main work of MobileNet is to use depthwise sparable convolutions (depth-level separable convolutions) to replace the past standard convolutions (standard convolutions) to solve the problems of computational efficiency and parameter quantity of convolutional networks. The MobileNets model is based on depthwise sparable convolutions (depth-level separable convolutions), which can decompose standard convolutions into a depth convolution and a point convolution (1 × 1 convolution kernel). **Depthwise convolution applies each convolution kernel to each channel, while 1 × 1 convolution is used to combine the outputs of channel convolutions.**

Batch Normalization (BN) will be added to the basic components of MobileNet, that is, in each SGD (stochastic gradient descent), standardization processing will be performed so that the mean of the result (all dimensions of the output signal) is 0 and the variance is 1. Generally, when you encounter problems such as slow convergence or gradient explosion during neural network

training, you can try BN to solve the problem. In addition, in general use cases, BN can also be added to speed up training and improve model accuracy.

In addition, the model also uses the ReLU activation function, so the basic structure of depthwise separable convolution is as shown below:



The MobileNets network is composed of many depthwise separable convolutions shown in the figure above. Its specific network structure is shown in the figure below:

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s1	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

2.2. Code analysis

List of recognized objects

```
[person, bicycle, car, motorcycle, airplane, bus, train,
  truck, boat, traffic light, fire hydrant, street sign,
  stop sign, parking meter, bench, bird, cat, dog, horse,
  sheep, cow, elephant, bear, zebra, giraffe, hat, backpack,
  umbrella, shoe, eye glasses, handbag, tie, suitcase,
  frisbee, skis, snowboard, sports ball, kite, baseball bat,
  baseball glove, skateboard, surfboard, tennis racket,
  bottle, plate, wine glass, cup, fork, knife, spoon, bowl,
  banana, apple, sandwich, orange, broccoli, carrot, hot dog,
  pizza, donut, cake, chair, couch, potted plant, bed, mirror,
  dining table, window, desk, toilet, door, tv, laptop, mouse,
  remote, keyboard, cell phone, microwave, oven, toaster,
  sink, refrigerator, blender, book, clock, vase, scissors,
  teddy bear, hair drier, toothbrush]
```

Load the category [object_detection_coco.txt], import the model [frozen_inference_graph.pb], and specify the deep learning framework [TensorFlow]

```
#Load COCO class name
with open('object_detection_coco.txt', 'r') as f: class_names =
f.read().split('\n')
# Display different colors for different targets
COLORS = np.random.uniform(0, 255, size=(len(class_names), 3))
#Load DNN image model
model = cv.dnn.readNet(model='frozen_inference_graph.pb',
config='ssd_mobilenet_v2_coco.txt', framework='TensorFlow')
```

Import the image, extract the height and width, calculate a 300x300 pixel blob, and pass this blob into the neural network

```
def Target_Detection(image):
     image_height, image_width, _ = image.shape
     #Create blob from image
     blob = cv.dnn.blobFromImage(image=image, size=(300, 300), mean=(104, 117,
123), swapRB=True)
     model.setInput(blob)
     output = model.forward()
     # Iterate through each test
     for detection in output[0, 0, :, :]:
         # Extract the confidence of the detection
         confidence = detection[2]
         # Only draw bounding boxes when the detection confidence is higher than
a certain threshold, otherwise skip
         if confidence > .4:
             # Get the ID of the class
             class_id = detection[1]
             # Map class id to class
             class_name = class_names[int(class_id) - 1]
             color = COLORS[int(class_id)]
             # Get bounding box coordinates
```

```
                box_x = detection[3] * image_width
                box_y = detection[4] * image_height
                # Get the width and height of the bounding box
                box_width = detection[5] * image_width
                box_height = detection[6] * image_height
                # Draw a rectangle around each detected object
                cv.rectangle(image, (int(box_x), int(box_y)), (int(box_width),
int(box_height)), color, thickness=2)
                #Write the class name text on the detected object
                cv.putText(image, class_name, (int(box_x), int(box_y - 5)),
cv.FONT_HERSHEY_SIMPLEX, 1, color, 2)
     return image
```

2.3. Start

```
cd /home/pi/yahboomcar_ws/src/yahboomcar_visual/detection
python target_detection_CSI.py (CSI camera)
python target_detection_USB.py (USB camera)
```

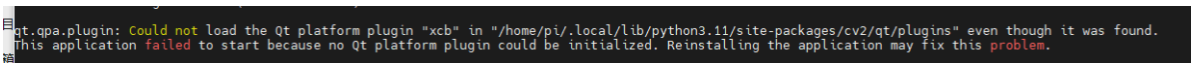
After clicking on the image box, use the [f] key on the keyboard to switch to human pose estimation.

```
if action == ord('f') or action == ord('F'):state = not state # Function
switching
```

2.4. Precautions

When running this tutorial, you need to use libqxcb.so, which will be provided in the source code and in the system.

If running the tutorial displays the following error:



You need to find the library and then add it according to the following command

```
cp libqxcb.so /home/pi/.local/lib/python3.11/site-
packages/cv2/qt/plugins/platforms
```