实验：基于 YOLOv5s 的目标检测

5.4.1 实验内容

本实验使用 YOLO 官网的 YOLOv5s 做为预训练模型，使用 COCO128 数据集进一步训练，实现 80 个类别的目标检测网络模型，并进行该网络模型的测试，然后将模型部署到 SE5 上。本实验的训练代码源于 YOLOv5 官网（https://github.com/ultralytics/yolov5）。

5.4.2 数据集

MS COCO 包含大量的室内外日常场景图片，可用于对目标检测、分割、人体关键点检测和图像描述等任务进行分析和评测。数据集包括 2014、2015 和 2017 三个版本。COCO2017 包含 118287 张图像的训练集，5000 张图像的验证集和 40670 张图像的测试集。训练集图片和验证集图片都有标注，总共 80 个目标类别。

表 COCO 数据集类别设置

0: 人	1: 自行车	2: 汽车	3: 摩托车	4: 飞机
5: 公共汽车	6: 火车	7: 卡车	8: 船	9: 信号灯
10: 消防栓	11: 停车标志	12: 停车计费器	13: 长凳	14: 鸟
15: 猫	16: 狗	17: 马	18: 羊	19: 牛
20: 大象	21: 熊	22: 斑马	23: 长颈鹿	24: 背包
25: 雨伞	26: 手提包	27: 领带	28: 手提箱	29: 飞盘
30: 双板滑雪板	31: 单板滑雪板	32: 运动球	33: 风筝	34: 棒球棒
35: 棒球手套	36: 滑板	37: 冲浪板	38: 网球拍	39: 瓶子
40: 高脚杯	41: 茶杯	42: 叉子	43: 刀子	44: 勺子
45: 碗	46: 香蕉	47: 苹果	48: 三明治	49: 橙子
50: 西兰花	51: 胡萝卜	52: 热狗	53: 披萨	54: 甜甜圈
55: 蛋糕	56: 椅子	57: 沙发	58: 盆栽植物	59: 床
60: 餐桌	61: 卫生间	62: 电视机	63: 笔记本电脑	64: 鼠标
65: 遥控器	66: 键盘	67: 电话	68: 微波炉	69: 烤箱
70: 烤面包机	71: 水槽	72: 冰箱	73: 书	74: 闹钟
75: 花瓶	76: 剪刀	77: 泰迪熊	78: 吹风机	79:牙刷

5.4.3 YOLOv5s 在 PyTorch 框架下程序实现

本实验完整源代码的工程文件夹“YOLOv5s”包括以下文件：

```
root@ubuntu:~/bmnnsdk2/bmnnsdk2-bm1684_v2.7.0/YOLOv5s$ tree -L 2
├── data
└── street1.png
└── street2.png
└── street3.png
├── model
└──YOLOv5s.pt
└──best.torchscript
├── YOLOv5-master
```

```
├── SE5
└── bmodel
└── street1.png
└── street2.png
└── street3.png
└── test_SE5.py
├── test.py
├──bmodel.py
```

1、在 PC 端执行的模型测试文件

./data	测试用图片文件夹
./model	训练好的网络模型文件夹
./YOLOv5-master	YOLOv5 官方源代码
test.py	PC 端 YOLOv5s 模型测试源代码
bomdel.py	转换成在 SE5 端执行的 bmodel 模型源代码

2、在 SE5 端执行的模型测试文件

./ SE5　　在 SE5 上执行的 bmodel 模型、测试程序及测试用图片，部署时将此文件夹下载到 SE5 端。./ SE5 文件夹下包括：

/bmodel	SE5 端的模型文件
test_SE5.py	SE5 端的测试程序

1. 网络模型训练代码

YOLOv5s 网络模型的训练使用./YOLOv5-master 文件夹下 YOLOv5 官方源代码。

YOLOv5-master 主要文件夹和文件的作用如下：

./data	数据集的参数以及训练时需要的超参数
./datasets	COCO128 数据集
./models	网络结构源代码以及使用 YOLOv5s 所需的网络参数信息
./runs	训练和测试之后的输出结果，没有该文件夹时会自动生成
train.py	YOLOv5s 的训练源代码
detect.py	YOLOv5s 的测试源代码
val.py	YOLOv5s 的验证源代码，使用验证集评估模型性能
export.py	将 Pytorch 模型转换为其他框架模型的源代码
yolov5s.pt	YOLOv5 官方提供的预训练模型

本实验需要使用 train.py 程序进行模型训练，yolov5s.pt 作为预训练模型，./datasets/coco128 作为训练集，export.py 将训练后的.pt 模型转为 torchscript 模型。

2. 网络模型测试代码

主目录下 test.py 为网络模型的测试源代码。

本实验采用主目录下 test.py 作为 YOLOv5s 网络模型的测试程序，相比 YOLOv5 官网的测试程序 detect.py，更加简单易懂。

（1）网络参数设置：图片尺寸 640×640、模型路径“best.torchscript”、检测图片路径“./data/”、文件名“street1.png”。

```
    def parse_opt():
        parser = argparse.ArgumentParser(prog=__file__)
        parser.add_argument('--img_size', type=int, default=640, help='infer
ence size (pixels)')
        parser.add_argument('--model_path', default='best.torchscript')
        parser.add_argument('--img_path', type=str, default="./data/")
```

```
        parser.add_argument('--img_name', type=str, default="street1.png")
        parser.add_argument('--tpu', type=int, default=0, help='tpu id')
        opt = parser.parse_args()
        return opt
```

（2）目标检测器类定义，参数初始化

```
    class Detector(object):
        def __init__(self, img_size):
            print("using model {}".format(opt.model_path))
            # 1、加载 YOLOv5s 模型
            self.net = torch.jit.load(opt.model_path)
            # 2、生成锚框
            self.nl = 3
            anchors = [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [1
16, 90, 156, 198, 373, 326]]
            self.anchor_grid = np.asarray(anchors, dtype=np.float32).reshape
(self.nl, 1, -1, 1, 1, 2)
            self.grid = [np.zeros(1)] * self.nl
            self.stride = np.array([8., 16., 32.])
            # 3、设定初筛和 NMS 阈值
            self.confThreshold = 0.5   #分类置信度
            self.nmsThreshold = 0.5    #nms 阈值
            self.objThreshold = 0.5    #有/无目标置信度
            self.img_size = img_size
            # 4、定义检测类别
            self.classes = ['person', 'bicycle', 'car', 'motorbike', 'aeropla
ne', 'bus', 'train', 'truck', 'boat','traffic light', 'fire hydrant', 'stop
sign', 'parking meter', 'bench', 'bird', 'cat',
                            'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
 'zebra', 'giraffe', 'backpack','umbrella', 'handbag', 'tie', 'suitcase', '
frisbee', 'skis', 'snowboard', 'sports ball',
                            'kite', 'baseball bat', 'baseball glove', 'skateboa
rd', 'surfboard', 'tennis racket','bottle', 'wine glass', 'cup', 'fork', 'kn
ife', 'spoon', 'bowl', 'banana', 'apple',
                            'sandwich', 'orange', 'broccoli', 'carrot', 'hot do
g', 'pizza', 'donut', 'cake', 'chair','sofa', 'pottedplant', 'bed', 'dining
table', 'toilet', 'tvmonitor', 'laptop', 'mouse',
                            'remote', 'keyboard', 'cell phone', 'microwave', 'o
ven', 'toaster', 'sink', 'refrigerator','book', 'clock', 'vase', 'scissors
', 'teddy bear', 'hair drier', 'toothbrush']
```

（3）图像预处理函数，图片填充成正方形并缩放到 640×640 并归一化

```
        def preprocess(self, img):
            target_size = self.img_size
            h, w, c = img.shape
            # 1、计算缩放参数
            # 1.1 计算缩放比例
            r_w = target_size / w
            r_h = target_size / h
            if r_h > r_w:
            # 1.2 计算缩放后的尺寸
                tw = target_size
                th = int(r_w * h)
                tx1 = tx2 = 0
            # 1.3 计算黑边填充数
```

```
            ty1 = int((target_size - th) / 2)
            ty2 = target_size - th - ty1
        else:
            tw = int(r_h * w)
            th = target_size
            tx1 = int((target_size - tw) / 2)
            tx2 = target_size - tw - tx1
            ty1 = ty2 = 0
        # 2、按照缩放参数改变图像大小
        img = cv2.resize(img, (tw, th), interpolation=cv2.INTER_LINEAR)
        # 3、填充
        padded_img = cv2.copyMakeBorder(
            img, ty1, ty2, tx1, tx2, cv2.BORDER_CONSTANT, (114, 114, 114)
        )
        # 4、BGR => RGB
        resized_img = cv2.cvtColor(padded_img, cv2.COLOR_BGR2RGB)
        # 5、图像归一化
        image = resized_img.astype(np.float32)
        image /= 255.0
        # 6、HWC 格式转为 CHW 格式
        image = np.transpose(image, [2, 0, 1])
        # 7、CHW 格式转为 NCHW 格式
        image = np.expand_dims(image, axis=0)
        # 8、将图像转换为连续内存，便于推理
        image = np.ascontiguousarray(image)
        return image, padded_img, (max(r_w, r_h), tx1, ty1)
```

（4）定义预测函数，输入：图像数据，输出：目标检测结果

```
    def predict(self, data_in):
        self.net.eval()
        output = self.net(torch.FloatTensor(data_in))
        return np.array(output)
```

（5）目标检测后处理：扫描网络输出的所有边界框，只保留置信度高的边界框，采用非极大抑制法消除冗余的重叠框，将得分最高的框的标签指定为框的类别。

```
    def postprocess(self, frame, outs):
        classIds = []
        confidences = []
        boxes = []
        for out in outs:
            for detection in out:
                scores = detection[5:]
                classId = np.argmax(scores)
                confidence = scores[classId]
                # 1.初步筛选：保留有目标置信度>0.5 且类别概率得分>0.5 的目标边界框
                if confidence > self.confThreshold and detection[4] > sel
f.objThreshold:
                # 2.计算目标左上角坐标和宽高
                    center_x = int(detection[0])
                    center_y = int(detection[1])
                    width = int(detection[2])
                    height = int(detection[3])

                    left = int(center_x - width / 2)
```

```
                    top = int(center_y - height / 2)
                    classIds.append(classId)
                # 3.计算边界框得分：有目标置信度*目标类别概率
                    confidences.append(float(confidence) * detection[4])
                    boxes.append([left, top, width, height])
        # 4.执行非极大抑制算法消除冗余框
        indices = cv2.dnn.NMSBoxes(boxes, confidences, self.confThreshold, self.nmsThreshold)
        # 5.在图像上绘制目标类别、边界框和边界框得分
        for i in indices:
            box = boxes[i]
            left = box[0]
            top = box[1]
            width = box[2]
            height = box[3]
            frame = self.drawPred(frame, classIds[i], confidences[i], left, top, left + width, top + height)
        return frame
```

（6）程序主函数

```
def main(opt):
    src_img = cv2.imread(opt.img_path + opt.img_name)
    if src_img is None:
        print("Error: reading image '{}'".format(opt.img_name))
        return -1
    # 1.加载目标检测模器
    YOLOv5 = Detector(opt.img_size)
    start_time = time.time()
    # 2、图片预处理
    img, padded_img, (ratio, tx1, ty1) = YOLOv5.preprocess(src_img)
    print("img.shape: {}".format(img.shape))
    # 3、目标检测
    dets = YOLOv5.predict(img)
    # 4、目标检测后处理
    plot_img = YOLOv5.postprocess(padded_img, dets)
    cv2.imwrite("YOLOv5s_out_{}".format(opt.img_name), plot_img)
    print("-" * 66)
    print("saved as YOLOv5s_out_{}".format(opt.img_name))
    print("cost time: {:.5f} s".format(time.time() - start_time))
    print("-" * 66)
```


5.4.4 YOLOv5s 网络模型训练和测试过程

本实验选择使用 YOLOv5s 的官方训练源代码 train.py 进行模型训练实践，先加载自官方网站下载的预训练模型 yolov5s.pt，再使用 COCO128 数据集进行进一步的训练。训练输出的模型文件用于接下来的测试和 SE5 部署。由于 YOLO 官方的测试程序 detect.py 过于复杂，涉及到诸多的设置，不利于学习实践，因此本实验选择了一个较为简单的程序 test.py 进行 YOLOv5s 的推理测试，使用训练过程输出的模型文件“best.torchscript”作为 YOLOv5s 测试模型。

在训练和测试过程中，模型、程序的结构关系如图 2 所示。

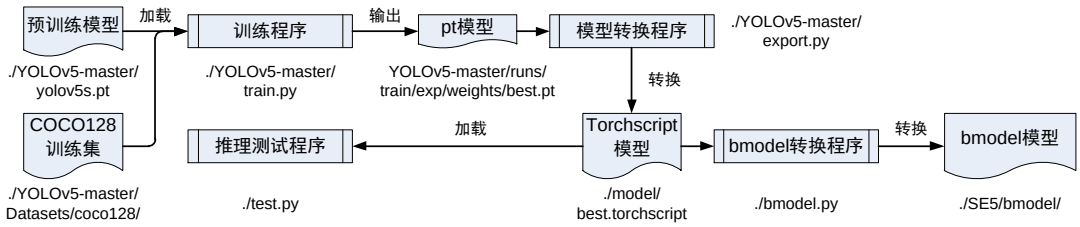

图 2 模型、程序结构关系

训练和测试的步骤如下：

（1）进入工程文件夹“YOLOv5s”

（2）在 Terminal 模式或 PyCharm 环境中运行“./YOLOv5-master/train.py”代码

运行 train.py 后，计算机开始加载数据进行 YOLOv5s 的模型训练，在运行窗口输出

```
root@ubuntu:/workspace/YOLOv5s/YOLOv5-master# python3 train.py --data coco1
28.yaml --epochs 100 --weights ''yolov5s.pt' ' --cfg models/yolov5s.yaml  --
batch-size 16

Epoch    GPU_membox_lossobj_losscls_lossInstances     Size
99/99      3.59G    0.03069     0.0394    0.004793      188          640: 100%|
          Class     Images  Instances      P       R     mAP50  mAP50-95: 100%|
           all       128        929      0.92   0.894    0.95     0.762

100 epochs completed in 0.057 hours.
Optimizer stripped from runs/train/exp2/weights/last.pt, 14.9MB
Optimizer stripped from runs/train/exp2/weights/best.pt, 14.9MB

Validating runs/train/exp2/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
    Class     Images  Instances      P         R      mAP50   mAP50-95: 100%|
     all       128        929     0.921      0.893       0.95      0.765
 person    128        254      0.986      0.848      0.962     0.758
 bicycle     128          6      0.986         1      0.995    0.63
     car       128         46          1      0.615      0.798     0.415
    ...        ...        ...        ...        ...         ...        ...
   scissors    128          1          1          0         0.497     0.348
  teddy bear   128         21       0.983         1         0.995     0.841
  toothbrush   128          5       0.926         1         0.995     0.833
    Results saved to runs/train/exp
```

训练结束后，模型文件“best.pt”存储在./YOLOv5-master/runs/train/exp/weights 文件夹下。

（4）转换模型

运行 export.py，将模型文件“best.pt”转换成模型文件“best.torchscript”

```
root@ubuntu:/workspace/YOLOv5s/YOLOv5-master# python3 export.py
```

转换结束后，模型文件“best.torchscript”存储在./model 文件夹下。

（5）在 Docker 环境或 PyCharm 环境中运行“test.py”的程序。

进入虚拟机 Docker 环境方式如下：

a) 进入解压好的目录，运行 Docker Image

```
$ cd bmnnsdk2-bm1684_v2.7.0/
$ sudo ./docker_run_bmnnsdk.sh
```

执行上述命令成功进入工作空间（workspace），如下所示，这就是 Docker 环境。

```
ngit@ubuntu:~$ cd bmnnsdk2-bm1684_v2.7.0/
ngit@ubuntu:~/bmnnsdk2-bm1684_v2.7.0$ sudo ./docker_run_bmnnsdk.sh
/home/ngit/bmnnsdk2-bm1684_v2.7.0
/home/ngit/bmnnsdk2-bm1684_v2.7.0
bmnnsdk2-bm1684/dev:ubuntu16.04
docker run --network=host --workdir=/workspace --privileged=true -v /home/n
git/bmnnsdk2-bm1684_v2.7.0:/workspace -v /dev/shm --tmpfs /dev/shm:exec -v
/etc/localtime:/etc/localtime -e LOCAL_USER_ID=0 -it bmnnsdk2-bm1684/dev:ub
untu16.04 bash
root@ubuntu:/workspace#
```

b) 配置 DockerContainer 的环境变量：

```
$ cd /workspace/scripts/
$ ./install_lib.sh nntc
$ source envsetup_cmodel.sh
```

执行上述命令后环境配置成功的结果如下所示，进入了仿真 TPU 运行环境的 cmodel 开发模式。

```
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/site-pack
ages (from pexpect>4.3->ipython>=5.0.0->ipykernel==5.3.4->ufw==1.0.0) (0.7.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/site-pack
ages (from plotly>=4.0->ufw==1.0.0) (8.0.1)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/site-packages (fr
om prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython>=5.0.0->ipykernel==5.3.4->u
fw==1.0.0) (0.2.5)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.7/site-packages
 (from Flask>=1.0.4->dash->ufw==1.0.0) (3.0.3)
Requirement already satisfied: typing-extensions>=3.6.4 in /usr/local/lib/python3.7/
site-packages (from importlib-metadata->jsonschema==3.2.0->ufw==1.0.0) (4.1.1)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/site-packages
(from importlib-metadata->jsonschema==3.2.0->ufw==1.0.0) (3.7.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.7/site-pack
ages (from Jinja2>=3.0->Flask>=1.0.4->dash->ufw==1.0.0) (2.1.1)
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.7/site-
packages (from jupyter-client->ipykernel==5.3.4->ufw==1.0.0) (4.9.2)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.7/site-packages
(from jupyter-client->ipykernel==5.3.4->ufw==1.0.0) (22.3.0)
Requirement already satisfied: entrypoints in /usr/local/lib/python3.7/site-packages
 (from jupyter-client->ipykernel==5.3.4->ufw==1.0.0) (0.4)
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.7/site-pa
ckages (from jupyter-client->ipykernel==5.3.4->ufw==1.0.0) (1.5.4)
Successfully installed Flask-2.2.2 Werkzeug-2.2.2 brotli-1.0.9 click-8.1.3 dash-2.6.
2 dash-bootstrap-components-1.2.1 dash-core-components-2.0.0 dash-cytoscape-0.3.0 da
sh-draggable-0.1.2 dash-html-components-2.0.0 dash-split-pane-1.0.0 dash-table-5.0.0
```

```
 flask-compress-1.13 ipykernel-5.3.4 itsdangerous-2.1.2 jsonschema-3.2.0 ufw-1.0.0 u
fwio-0.9.0
/workspace/scripts
/workspace/scripts
root@ubuntu:/workspace/scripts#
```

在 Docker 环境或 PyCharm 环境中运行“test.py”的程序。

运行 test.py 后，程序读入图片“../data/street1.jpg”开始目标检测，在运行状态窗口输出

```
root@ubuntu:/workspace/YOLOv5s# python3 test.py
using model ./model/YOLOv5s.pt
img.shape: (1, 3, 640, 640)
------------------------------------------------------------------------
saved as YOLOv5s_out_street1.png
cost time: 0.36653 s
    ------------------------------------------------------------------------
```

运行结果如图 3 所示。

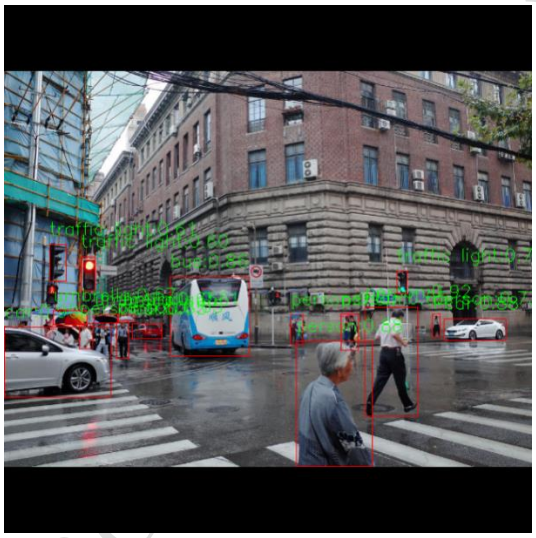

图 4YOLOv5s 网络 PC 端测试结果

5.4.5 YOLOv5s 网络模型在 SE5 上的部署

1. 模型编译

（2）BMNETp 编译器

输入以下命令调用编译器工具 BMNETp 的编译函数 compile 将 YOLOv5s 网络模型文件“best.torchscript”编译为 SE5 支持的 bmodel 模型文件。

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/workspace/lib/bmcompiler/
$ python3 -m bmnetp \
--model="./model/best.torchscript",\
--outdir="./SE5/bmodel",\
--shapes=[[1, 3, 640, 640]],\
--net_name="yolov5s", \
--target="BM1684", \
--opt=2,
```

为了便于执行，将以上命令写入 bmodel.py，运行 bmodel.py 即可调用 BMNETp 编译器将 pt

模型编译为 bmodel 模型。其中 BMNETp 命令的参数见 ResNet 实验。

```
#进入 bmodel.py 文件的路径下
root@ubuntu:/workspace/YOLOv5s# python3 bmodel.py
```

编译成功提示信息如下。

```
==========================================================
*** Store bmodel of BMCompiler...
==========================================================
BMLIB Send Quit Message
Compiling succeeded.
```

出现提示信息“Compiling succeeded”，表明模型编译完成。在 SE5 文件夹下可以看到新生成文件夹“bomdel”。其中，compilation.bmodel 是可在 SE5 端进行测试所需的 bmodel 文件。

（3）模型验证

在 PC 端的 BMNNSDK 环境下进行模型验证。

```
$ cd /workspace/bin/x86
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/workspace/lib/bmnn/cmodel/
$ ./bmrt_test --context_dir=/workspace/bmodel 文件夹路径
```

或者，将待验证的 bmodel 文件夹复制到 SE5。

```
$ bmrt_test --context_dir=./bmodel 模型文件夹的路径
```

显示验证结果。

```
[BMRT][bmrt_test:1004] INFO:==>comparing #0 output ...
[BMRT][bmrt_test:1009] INFO:+++ The network[yolov5s] stage[0] cmp success +
++
[BMRT][bmrt_test:1029] INFO:load input time(s): 0.011449
[BMRT][bmrt_test:1030] INFO:calculate  time(s): 0.023695
[BMRT][bmrt_test:1031] INFO:get output time(s): 0.009436
[BMRT][bmrt_test:1032] INFO:compare    time(s): 0.044996
```

提示信息中出现“+++ The network [yolov5s] stage[0] cmp success +++”，表示模型验证成功。

2. 算法移植

算法移植主要包括导入数据、预处理、模型加载、测试和后处理等，完整程序见附录。

（1）加载 bmodel

SE5 提供了模块化接口 Sophon Python API：sophon.sail.Engine，用于加载 bmodel 并驱动 TPU 进行推理。

```
import sophon.sail as sail
…
#加载 bmodel
self.net = sail.Engine(model_path, tpu_id, sail.IOMode.SYSIO) #加载 bmodel
self.graph_name = self.net.get_graph_names()[0]  #获取网络名字
self.input_names = net.get_input_names(graph_name)  #获取网络输入名字
```

（2）预处理

通常情况下，在目标检测网络中的输入图片需要进行归一化。本实验中，对图片进行测试时需要将图片大小设置为 460×460：对图像按比例缩小后进行填充。

```
    def preprocess(self, img):
target_size = self.img_size
…以下同 PC 端程序。
        return image, padded_img, (max(r_w, r_h), tx1, ty1)
```

（3）模型测试

```
#运行预测网络
    def predict(self, tensor):
```

```
input_data = {self.input_name: np.array(tensor, dtype=np.float32)}
        output = self.net.process(self.graph_name, input_data)
        return list(output.values())[0]
```

（4）后处理

扫描网络输出的所有边界框，只保留置信度高的边界框，采用非极大抑制法消除冗余的重叠框，将得分最高的框的标签指定为框的类别。

```
      def postprocess(self, frame, outs):
classIds = []
…以下同 PC 端程序。
       return frame
```

（5）模型测试主函数

模型测试的主函数如下，包含预处理、模型加载、测试和结果显示等。

```
def main(opt):
src_img = cv2.imread(opt.img_path + opt.img_name)
    if src_img is None:
print("Error: reading image '{}'".format(opt.img_name))
       return -1
    YOLOv5 = Detector(opt.img_size, tpu_id=opt.tpu, model_format="fp32")
…以下同 PC 端程序。
```

3. SE5 端模型测试

（1）建立测试文件夹

将准备的测试图片、SE5 端程序“test_SE5.py”和生成的 bmodel 模型文件夹放入同一个文件夹并命名为 YOLOv5s_SE5，如图 1 所示。

图 1 测试文件夹

（2）SE5 云空间申请

SE5 云平台的使用以及云空间的申请，详情请见 SE5 云平台使用的说明。

（3）SE5 端运行程序

进入预测程序文件夹路径下，运行预测程序。

```
$ cd  /YOLOv5s_SE5/
$ python3  test_SE5.py
```

```
using model ./bmodel/compilation.bmodel
bmcpuinit: skip cpu_user_defined
open usercpu.so, inituser_cpu_init
[BMRT][load_bmodel:823] INFO:Loadingbmodel from [./bmodel/compilation.bmode
l]. Thanks for your patience...
[BMRT][load_bmodel:787] INFO:pre net num: 0, load net num: 1
img.shape: (1, 3, 640, 640)
------------------------------------------------------------------
saved as YOLOv5s_result_street1.png
cost time: 0.85246 s
------------------------------------------------------------------
```

程序运行完毕后，SE5 端会以图片形式保存测试结果“YOLOv5s_result_street1.png”。可

以按照云平台使用说明将云空间的文件下载到本地。