主讲人 ：正点原子团队
硬件平台：正点原子ATK-DLRV1126开发板

资料下载：www.openedv.com/docs/index.html
教学平台：www.yuanzige.com
天猫店铺：zhengdianyuanzi.tmall.com
技术论坛：www.openedv.com/forum.php
公众平台：正点原子

■ YOLOv5-v7.0预训练模型部署

1，说明

2，准备YOLOv5-v7.0工程

3， Windows下跑预训练模型

4，开发板下跑预训练模型

版本	说明	版本	说明
YOLOv1	核心思想： ● 将输入的图像划分为7x7个网格(grid cell)，每个网格负责预测中心点落在该网格内的目标； ● 每个网格会预测B个边界框(bounding box)和C个类别分数（其中B = 2）； ● 每个边界框预测4个位置信息（框中心点的x和y坐标，框的宽高w、h）和1个置信度(confidence)。 ● 输出B×5+C个预测值 缺点： ● 对小物体和重叠物体的检测效果较差； ● 定位精度和召回率相对较低，容易出现漏检。 论文：《You Only Look Once: Unified, Real-Time Object Detection》 工程代码：https://github.com/Tshzzz/pytorch_yolov1	YOLOv5	设计理念： ● 基于SOTA网络架构EfficientDet的思路，采用了新型的NAS搜索算法； ● 采用多尺度预测（可在3个不同尺度上进行预测）； 优点： ● 实现了更快、更准确的检测速度； ● 模型大小和性能之间提供了多种选择（YOLOv5s、YOLOv5m、YOLOv5l、YOLOv5x）； 论文：《You Only Look Once:Unified, Real-Time Object Detection》 工程代码：https://github.com/ultralytics/yolov5?tab=readme-ov-file
YOLOv2	对YOLOv1的改进： ● 引入了锚框(anchor boxes)用于检测不同尺寸和比例的目标； ● 使用Darknet-19作为骨干网络（Backbone）。 优点： ● 在保持高速度的同时，提高了检测的准确性； ● 可支持多达9000个类别的检测 论文：《YOLO9000: Better, Faster, Stronger》 工程代码：https://github.com/pjreddie/darknet	YOLOv6	由美团视觉智能部研发，致力于工业应用。 骨干网络（Backbone）、检测头（Head）和训练策略等多方面均有改进和优化，获得了更高的精度、更快的速度和更好的硬件友好性。 论文：《YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications》 工程代码：https://github.com/meituan/YOLOv6
YOLOv3	主要改进： ● 采用Darknet-53网络结构作为骨干网络； ● 采用尺度预测（可在3个不同尺度上进行预测）； 优点： ● 在保持速度的同时，进一步提升了检测的准确率，尤其对小目标的检测效果有所改善； ● 提高模型的鲁棒性。 论文：《YOLOv3: An Incremental Improvement》 工程代码：https://github.com/ultralytics/yolov3	YOLOv7	亮点： 使用更快的网络结构、更高效的卷积操作和数据并行技术，显著提高了检测速度； 增加了新的检测层和多项技术，提升了模型的检测精度； 网络结构更加紧凑，缩小了模型大小，更容易部署在资源受限的设备上； 多分辨率支持、数据增强探索、模型重参数化、高效层聚合网络（E-ELAN）等等。 论文：《YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors》 工程代码：https://github.com/WongKinYiu/yolov7
YOLOv4	亮点： ● 引入CSPDarknet53、SAM模块、YOLOv3-SPP、YOLOv3-PAN等技术，进一步提高了检测精度和速度。 优点： ● 在保持高速度的同时，大幅提高了检测准确率和效率。 论文：《YOLOv4: Optimal Speed and Accuracy of Object Detection》 工程代码：https://github.com/AlexeyAB/darknet	YOLOv8	亮点： 骨干网络（Backbone）：Darknet-53或Darknet-85，可提取更丰富的特征； 检测头：引入了新的检测头，可以在不同尺度上检测目标； 数据增强：如随机缩放、旋转、平移、对比度增强等，提高模型的泛化能力； 高效的训练策略、优化的算法、工程化、简洁易用性等。 工程代码：https://github.com/ultralytics/ultralytics
YOLOX	YoloX由旷视科技开源，以YOLOv3 (Darknet53作为backbone)作为基线进行开发。 论文：《YOLOX: Exceeding YOLO Series in 2021》 工程代码：https://github.com/Megvii-BaseDetection/YOLOX	其它	YOLO还有各种衍生版本或者变体，如TinyYOLO、MobileNet-YOLO、YOLOv4-Tiny、YOLOv5的各种变体（如YOLOv5-P6、YOLOv5-s、YOLOv5-m、YOLOv5-l、YOLOv5-x等）、Scaled YOLOv4、YOLOv3-SPP1、YOLOv3-SPP3、YOLOv3-tiny、YOLOF等

1、说明

① 本实验以YOLOv5-v7.0版本为例子讲解：

- YOLOv5精度高，而且速度快，在相同的GPU硬件上，YOLOv5的检测速度比YOLOv3快几倍；
- YOLOv5比YOLOv3的工程代码实现更加模块化、自动化和可配置化；
- YOLOv5支持将模型导出为多种格式，如ONNX、TensorRT、paddle、openvino、torchscript等等，在部署方面，相较于YOLOv3，YOLOv5更加面向于工业应用场景。

1、说明

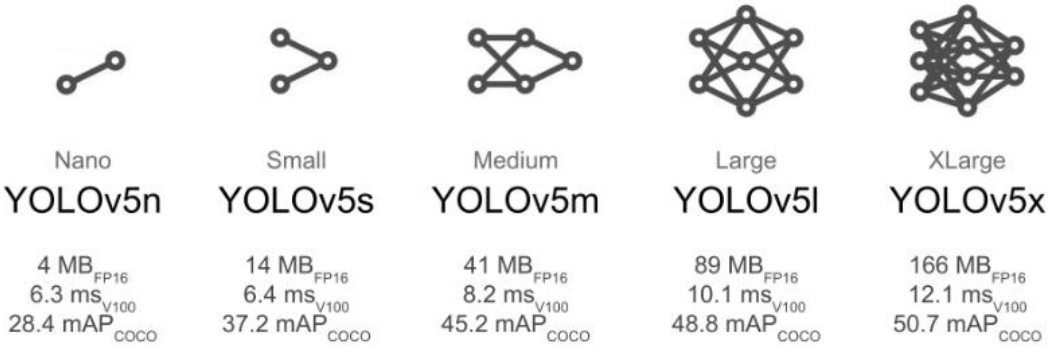
② 预训练模型权重使用yolov5s.pt，YOLOv5不同版本：

- 模型大小、网络深度和网络宽度、计算复杂度、检测准确性： YOLOv5n < YOLOv5s < YOLOv5m < YOLOv5l < YOLOv5x
- 检测速度： YOLOv5n >YOLOv5s > YOLOv5m > YOLOv5l > YOLOv5x
- 应用场景：

由于YOLOv5n和YOLOv5s计算复杂度低，更适合在计算资源受限的设备上使用，如移动设备或边缘设备；

YOLOv5x适用于需要高精度但计算资源充足的场景；

YOLOv5m和YOLOv5l可以根据具体需求进行选择，以达到速度与准确性的平衡；



1、说明

yolov5多个版本网络模型：

Model	size (pixels)	mAPval 50-95	mAPval 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

32倍下采样
3个输出

64倍下采样
4个输出

检测更大的
目标

1、说明

③ 本人电脑环境

操作系统：Windows 10

硬件配置：

CPU：Intel i7-11800H

GPU：NVIDIA GeForce RTX 3080 Ti（显存12GB，英伟达官网上显示算力8.6）

内存：32G

深度学习框架：PyTorch-GPU 1.13.0

VMWare下安装的Ubuntu不支持CUDA，没办法用到GPU，所以本人训练模型是在Windows下完成的。

若在PC安装的是Ubuntu实体机，则Ubuntu可以调用到CUDA，可以使用GPU。

2、准备YOLOv5-v7.0工程

① 手动在线下载： https://github.com/ultralytics/yolov5/releases

② git 克隆（推荐）

1. 安装git： https://git-scm.com/download/win
2. 新建一个文件夹，并进入到新建的文件夹下
3. 克隆YOLOv5-v7.0工程

```
git clone https://github.com/ultralytics/yolov5.git
```

4. 若要基于指定的版本来操作，则需要做如下操作

查看提交的版本记录：

```
git log --oneline
```

切换到对应的commit id

```
git checkout -b 7.0 915bbf2
```

2、准备YOLOv5-v7.0工程

通过requirements.txt安装工程所需的依赖库

先修改requirements.txt文件，选择安装onnx后再执行如下命令安装库：

```
pip install -r requirements.txt
```

若个别库安装不上，可尝试换源、单独安装，例如，安装onnx：

```
pip install onnx==1.12.0 -i https://pypi.tuna.tsinghua.edu.cn/simple
```

若pip安装不上，可以使用conda 来安装：

```
conda install onnx
```

conda命令通过-c选项指定源：

```
conda install -c https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main onnx
```

关于conda和pip命令的基本使用，可看前面的课程

3、Windows下跑预训练模型

参考官方教程：

https://github.com/ultralytics/yolov5/blob/master/tutorial.ipynb

```
采用预训练模型yolov5s.pt，喂入模型中的图片被resize为640*640大小，置信度阈值为0.25，图片位于data/images下
python detect.py --weights yolov5s.pt --img 640 --conf 0.25 --source data/images

采用预训练模型yolov5s.pt，喂入模型中的图片被resize为640*640大小，置信度阈值为0.5，图片位于data/images下
python detect.py --weights yolov5s.pt --img 640 --conf 0.5 --source data/images
```

3、Windows下跑预训练模型

python detect.py命令后的参数说明

命令参数	说明
--help	查看帮助
--weights	模型权重文件的路径，默认配置为YOLOv5工程根目录下的yolov5s.pt文件
--source	网络的输入数据，即要检测的数据，可以是图片/视频路径，也可以是'0'(电脑自带摄像头),也可以是rtsp等视频流。 字符串类型，默认值是YOLOv5工程根目录下的data/images中的文件
--data	数据集配置文件的路径，默认为YOLOv5工程根目录下的data/coco128.yaml配置文件
--imgsz	输入图片的大小，int类型，默认值是640，大小表示为640 × 640个像素。模型在检测图片之前，先把图片resize成640 × 640大小，然后再喂进网络中进行前向传播，因此，我们用于预测的图片可以是各种尺寸。
--conf_thres	置信度阈值，float类型，默认值是0.25。设置置信度阈值，用于在进行NMS操作之前清理检测框，把低于阈值的检测框给去掉。该值设置的越小，检测到的目标就越多，在图片上画出的框就越多，检测的就越不准确，该值设置的越大，检测的越准确。
--iou_thres	进行NMS时的IoU阈值，float类型，默认值是0.45。
--max_det	最大检测数量，默认最大检测1000个目标。
--device	推理时的设备，是采用CPU还是GPU来进行推理，若不指定，系统会自动检测。 若用CPU推理，参数可指定为cpu，若有多个GPU，使用哪一个GPU推理时，可通过0、1、2等参数来指定。
--view_img	是否把检测结果给显示出来，默认不显示，若要显示，命令使用示例如下： python detect.py --view-img

3、 Windows下跑预训练模型

python detect.py命令后的参数说明

命令参数	说明
--save_txt	是否将检测结果（物体的类别索引和边框的位置信息）保存在一个.txt文件中，默认不保存，若要保存，命令使用示例如下： python detect.py --save-txt
--save_conf	是否保存目标的置信度，默认不保存，若要保存，命令使用示例如下（注意：要同时指定--save-txt）： python detect.py --save-txt --save-conf
--save-crop	是否把模型检测到的物体给裁剪下来，若指定该参数，若指定该参数，则在crops文件夹下看到以类别命名的子文件夹，里面保存裁剪下来的检测到的物体图片。
--nosave	开启该参数，表示不保存检测结果。
--agnostic-nms	在标准的NMS中，通常会基于每个类别分别进行NMS。若开启该参数，表示在不区分类别的情况下进行NMS，这称为类别不可知的NMS，或者跨类别的NMS，开启后，NMS将不再区分不同的类别，而是对所有检测到的边界框进行全局的抑制。
--augment	是否采用数据增强进行推理，若采用数据增强，则指定该参数，系统会在进行目标检测之前对输入的图像进行一系列随机变换，如旋转、缩放、裁剪、翻转等，目的是增加数据集的多样性和数量，从而提高模型的泛化能力和性能。
--visualize	是否把特征图可视化出来，若指定该参数，在exp文件夹下会多出一些以物体标签命名的子文件夹，子文件夹下会存放特征图。
--update	若指定该参数，则对所有模型进行strip_optimizer操作，即从保存的模型权重中移除优化器的状态，只保留模型的参数（即权重和偏置）。这样做可以确保你在加载模型时不会受到优化器配置的影响。

3、 Windows下跑预训练模型

python detect.py命令后的参数说明

命令参数	说明
--project	检测结果的保存目录，检测结果默认保存在YOLOv5工程根目录的runs/detect下，可通过该参数指定保存的目录
--name	默认值为exp，表示runs/detect下的exp子文件夹，表示预测结果保存的文件夹名字
--exist-ok	该参数表示每次模型检测的结果是否保存在原来的文件夹下，每次检测时，默认在runs/detect下新建一个exp子文件夹，并把图片保存在新建的子文件夹下，若指定该参数，则检测后不会新建exp子文件夹，检测见过会保存在上一次的文件夹下
--line-thickness	通过该选项参数调节检测框的线条的粗细，默认值是3
--hide-labels	是否在检测结果的图像上隐藏标签labels，默认为False，表示显示标签
--hide-conf	是否在检测结果的图像上隐藏标签置信度得分，默认为False，表示显示置信度得分
--half	是否使用FP16半精度推理（默认采用高精度FP32来推理，在推理阶段，对精度要求没那么严格，可以选择低精度来推理）
--dnn	是否使用 OpenCV DNN 进行 ONNX 推理，若使用，指定该参数开启即可
--vid-stride	该参数表示在使用视频进行推理时，每隔多少帧进行一次检测或处理，默认值是1，表示对每一帧都进行推理

4、开发板下跑预训练模型

① 需要修改export.py文件：

② 将yolov5s.pt通过export.py文件导出为yolov5s.onnx文件或者yolov5s.torchscript文件

③ 通过rknn-toolkit工具将yolov5s.onnx文件或者yolov5s.torchscript文件转化为.rknn文件

④ 将.rknn进行预编译

⑤ 部署到RV1126开发板上

```
// 转换为TorchScript格式，得到yolov5s.torchscript文件
python export.py --weights yolov5s.pt --img 640 --batch 1 --include torchscript
// 转换为ONNX格式，得到yolov5s.onnx文件
python export.py --weights yolov5s.pt --img 640 --batch 1 --include onnx
// 若需要指定opset，可根据安装的onnx库的版本来调整，如安装的onnx库的版本是1.12.0，后面加上--opset 12
python export.py --weights yolov5s.pt --img 640 --batch 1 --include onnx --opset 12
```

4、开发板下跑预训练模型

python export.py命令后的参数说明

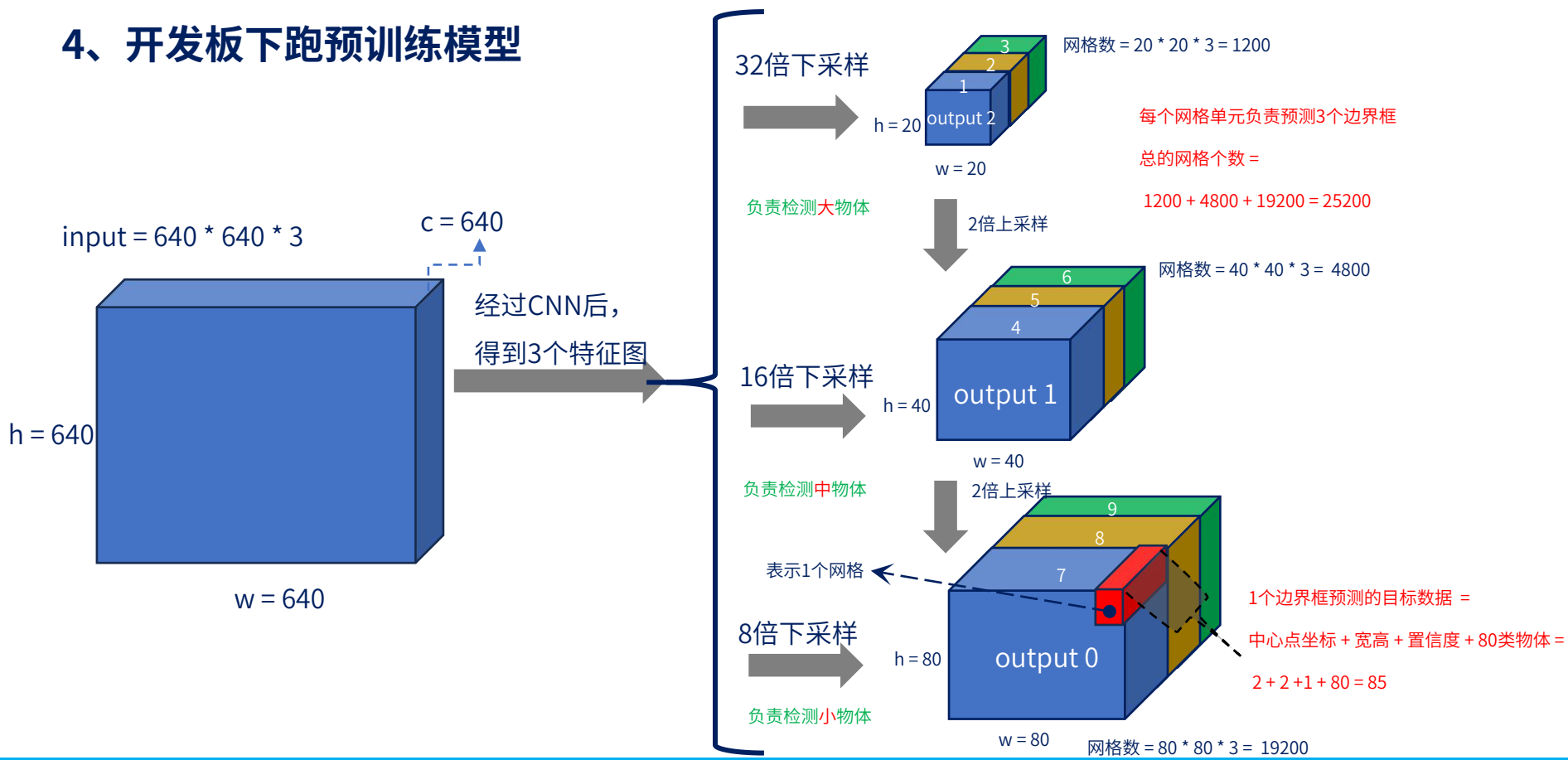
命令参数	说明
--help	查看帮助
--data	数据集配置文件的路径，默认为YOLOv5工程根目录下的data/coco128.yaml配置文件
--weights	模型权重文件的路径，默认配置为YOLOv5工程根目录下的yolov5s.pt文件
--imgsz	输入图片的大小，int类型，默认值是[640, 640]，大小表示为640 × 640个像素
--batch-size	批处理大小，即一次能够处理的样本数量，默认值为1，表示每次推理一张图片。
--device	默认使用cpu来导出模型，如果有多个GPU，可以通过0、1、2等指定使用的是哪一块GPU
--half	是否将导出的模型采用半精度FP16来导出，默认是FP32精度导出的，若指定，则采用FP16导出模型
--inplace	是否设置YOLOv5 Detect() inplace=True
--keras	是否将模型导出为Keras格式，如需要导出为Keras格式，则指定该参数
--optimize	TorchScript格式转换参数，导出的TorchScript格式的模型，是否要优化，以便于模型部署到资源受限的设备上，如手机或嵌入式系统，注意，优化可能会导致模型精度的轻微下降。
--int8	导出CoreML或者TFLIte格式时，是否采用8位整数（INT8）量化，即将模型的权重和激活值从通常使用的32位浮点数（FP32）转换为8位整数，量化后，显著减少模型的大小，可实现更快的推理速度。注意，量化会导致模型精度的轻微下降。
--dynamic	针对ONNX/TF/TensorRT模型，通过该参数可以使得导出的模型保持其动态形状的特性。一般来说，模型通常可以接受不同大小的输入，并据此调整其内部计算图的大小，然而，在某些导出格式（如ONNX）中，模型可能会被要求具有固定的输入大小，通过--dynamic 参数可以确保模型保持其接受不同输入大小的能力。

4、开发板下跑预训练模型

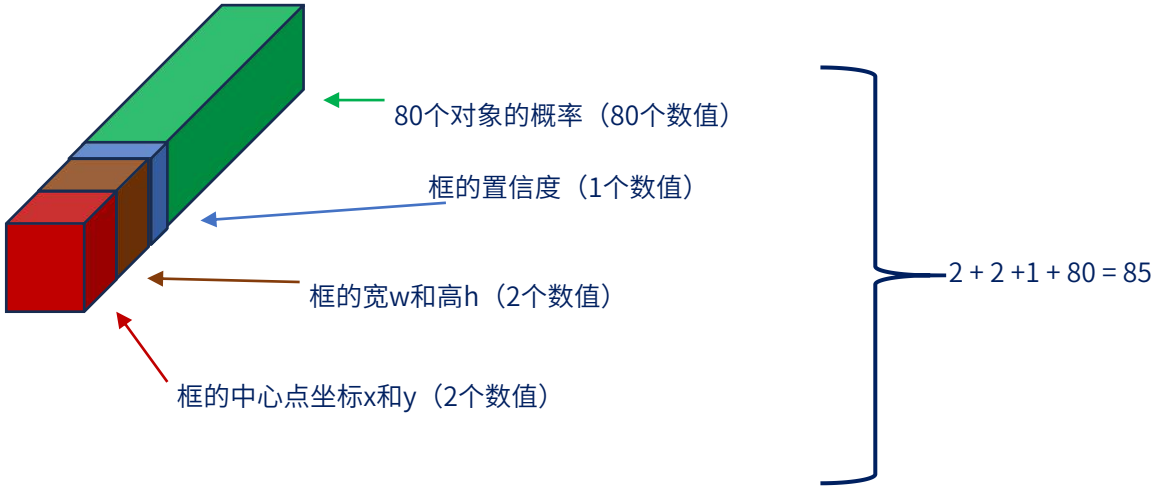
python export.py命令后的参数说明

命令参数	说明
--simplify	ONNX转换参数，导出ONNX的过程是否对模型进行简化，简化的目的是减小模型复杂度、提高推理速度或减少资源消耗，同时尽量保持模型的性能
--opset	ONNX转换参数，指定导出模型时所使用的ONNX操作集（Operator Set）的版本，默认为12版本
--verbose	TensorRT转换，是否开启详细输出模式以便输出更多的信息
--workspace	TensorRT转换参数，导出TensorRT时，指定一个工作空间目录，该目录用于存放与模型导出相关的临时文件、日志文件或中间结果
--nms	TF转换参数，导出的模型是否包含NMS操作，默认为False，表示不包含
--agnostic-nms	TF转换参数，导出的模型是否包含agnostic NMS操作，默认为False
--topk-per-class	TF.js 转换参数，是否要保留每一类别
--topk-all	TF.js 转换参数，是否所有类别都要保留
--iou-thres	TF.js 转换参数，IoU阈值，默认为0.45
--conf-thres	TF.js 转换参数，置信度阈值，默认为0.25

4、开发板下跑预训练模型

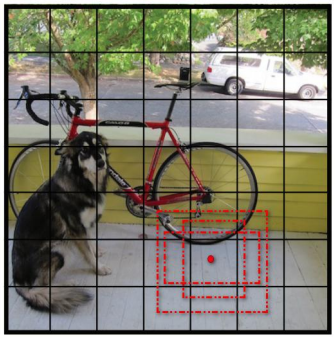


4、开发板下跑预训练模型

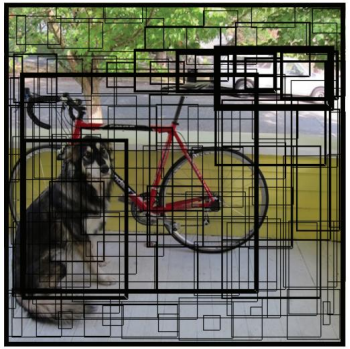


4、开发板下跑预训练模型

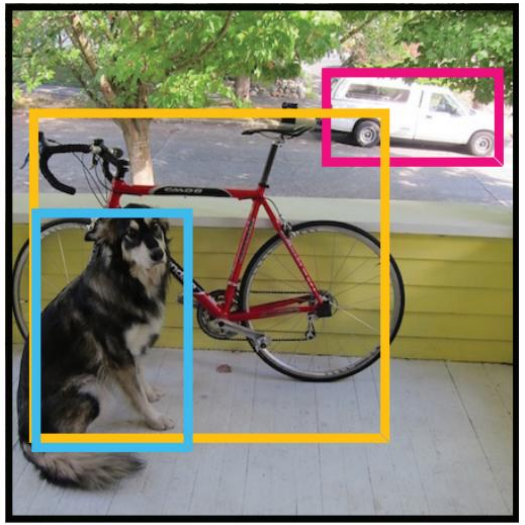
将特征图划分为s * s个网格单元



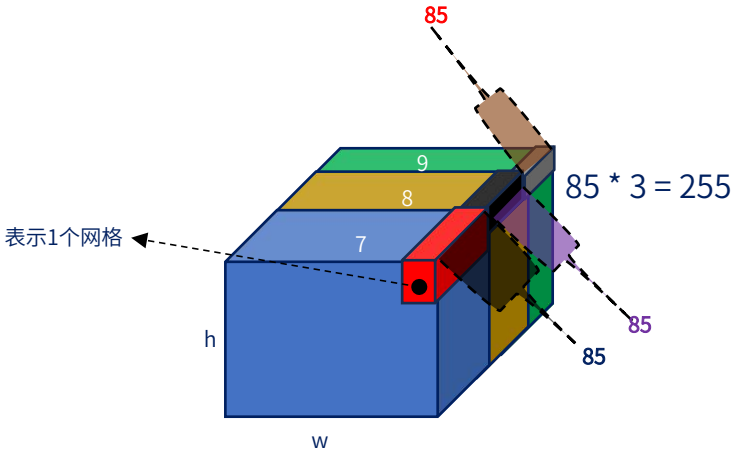
每个网格单元预测3个bounding box

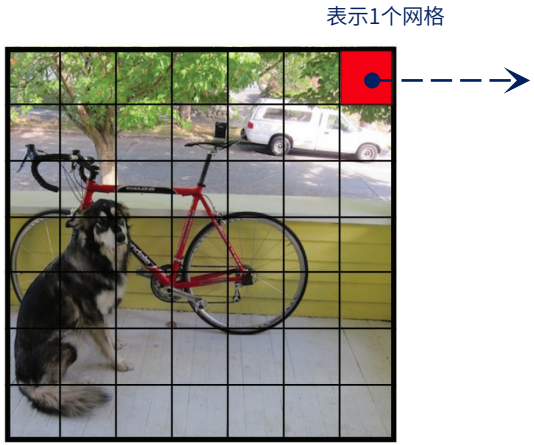


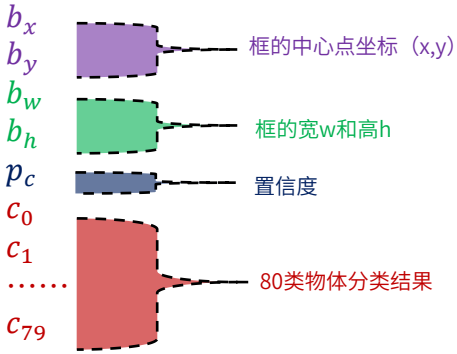
设置置信度阈值 + NMS



4、开发板下跑预训练模型







1个边界框预测的目标数据 =

中心点坐标 + 宽高 + 置信度 + 80类物体 =

2 + 2 +1 + 80 = 85

4、开发板下跑预训练模型

在将.pt文件导出为.onnx或者. torchscript格式文件之前，先修改models/yolo.py文件的后处理部分的代码，原来代码：

```
def forward(self, x):
        z = []  # inference output
        for i in range(self.nl):
            x[i] = self.m[i](x[i])  # conv
        #return x[0],x[1],x[2]

            bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
            x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

            if not self.training:  # inference
                if self.dynamic or self.grid[i].shape[2:4] != x[i].shape[2:4]:
                    self.grid[i], self.anchor_grid[i] = self._make_grid(nx, ny, i)

                if isinstance(self, Segment):  # (boxes + masks)
                    xy, wh, conf, mask = x[i].split((2, 2, self.nc + 1, self.no - self.nc - 5), 4)
                    xy = (xy.sigmoid() * 2 + self.grid[i]) * self.stride[i]  # xy
                    wh = (wh.sigmoid() * 2) ** 2 * self.anchor_grid[i]  # wh
                    y = torch.cat((xy, wh, conf.sigmoid(), mask), 4)
                else:  # Detect (boxes only)
                    xy, wh, conf = x[i].sigmoid().split((2, 2, self.nc + 1), 4)
                    xy = (xy * 2 + self.grid[i]) * self.stride[i]  # xy
                    wh = (wh * 2) ** 2 * self.anchor_grid[i]  # wh
                    y = torch.cat((xy, wh, conf), 4)
                z.append(y.view(bs, self.na * nx * ny, self.no))

        return x if self.training else (torch.cat(z, 1),) if self.export else (torch.cat(z, 1), x)
```

4、开发板下跑预训练模型

将def forward(self, x)修改为如下（此处修改只是用于转换模型，若训练模型的话，请把修改的地方还原回原来的代码形式）。此处修改，针对的是YOLOV5的V7.0版本！其它版本不一定是这样子改！此处修改，就是让模型有3个输出（原来的代码是只有一个输出）：

```
def forward(self, x):
        z = []  # inference output
        for i in range(self.nl):
            x[i] = self.m[i](x[i])  # conv
        return x[0],x[1],x[2]
        """
            bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
            x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

            if not self.training:  # inference
                if self.dynamic or self.grid[i].shape[2:4] != x[i].shape[2:4]:
                    self.grid[i], self.anchor_grid[i] = self._make_grid(nx, ny, i)

                if isinstance(self, Segment):  # (boxes + masks)
                    xy, wh, conf, mask = x[i].split((2, 2, self.nc + 1, self.no - self.nc - 5), 4)
                    xy = (xy.sigmoid() * 2 + self.grid[i]) * self.stride[i]  # xy
                    wh = (wh.sigmoid() * 2) ** 2 * self.anchor_grid[i]  # wh
                    y = torch.cat((xy, wh, conf.sigmoid(), mask), 4)
                else:  # Detect (boxes only)
                    xy, wh, conf = x[i].sigmoid().split((2, 2, self.nc + 1), 4)
                    xy = (xy * 2 + self.grid[i]) * self.stride[i]  # xy
                    wh = (wh * 2) ** 2 * self.anchor_grid[i]  # wh
                    y = torch.cat((xy, wh, conf), 4)
                z.append(y.view(bs, self.na * nx * ny, self.no))

        return x if self.training else (torch.cat(z, 1),) if self.export else (torch.cat(z, 1), x)
        """
```

4、开发板下跑预训练模型

修改好models/yolo.py文件中的def forward(self, x)函数以后，可执行如下命令，将.pt文件导出.onnx或者. torchscript格式文件：

```
// 转换为TorchScript格式，得到yolov5s.torchscript文件
python export.py --weights yolov5s.pt --img 640 --batch 1 --include torchscript
// 转换为ONNX格式，得到yolov5s.onnx文件
python export.py --weights yolov5s.pt --img 640 --batch 1 --include onnx
// 若需要指定opset，可根据安装的onnx库的版本来调整，如安装的onnx库的版本是1.12.0，后面加上--opset 12
python export.py --weights yolov5s.pt --img 640 --batch 1 --include onnx --opset 12
```

4、开发板下跑预训练模型

将前面导出的.onnx文件或者. torchscript文件转化为.rknn文件，请参考前面课程《 RKNN Toolkit转换模型》的代码：

convert-onnx-to-rknn-copy.py

convert-pytorch-to-rknn.py

本节课程的代码：

模型转换

convert-onnx-to-rknn.ipynb	将onnx转换为rknn
convert-pytorch-to-rknn.ipynb	将torchscript转化为rknn
precompile_rknn_model.ipynb	对rknn进行预编译操作

部署到RV1126

12_atk_yolov5_object_recognize下的代码

将编译得到的可执行文件atk_yolov5_object_recognize、标签文件coco_80_labels_list.txt 拷贝到开发板的/demo/bin目录下：

```
./atk_yolov5_object_recognize
```

天猫店铺：https://zhengdianyuanzi.tmall.com