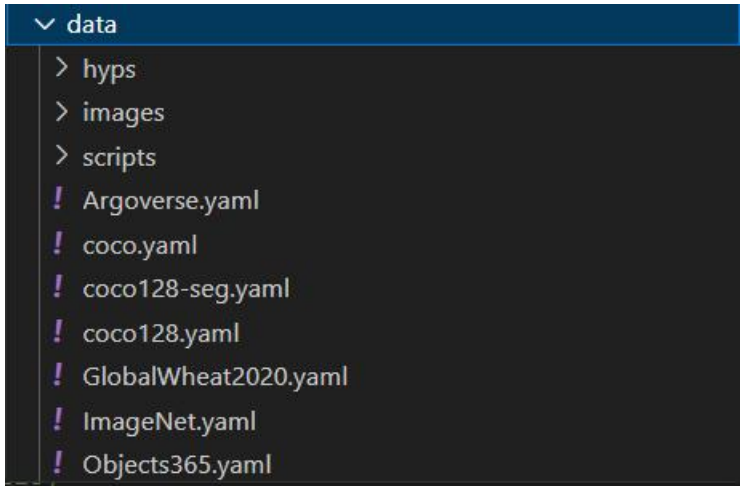
Learning Yolo.V5

By Ruihe An

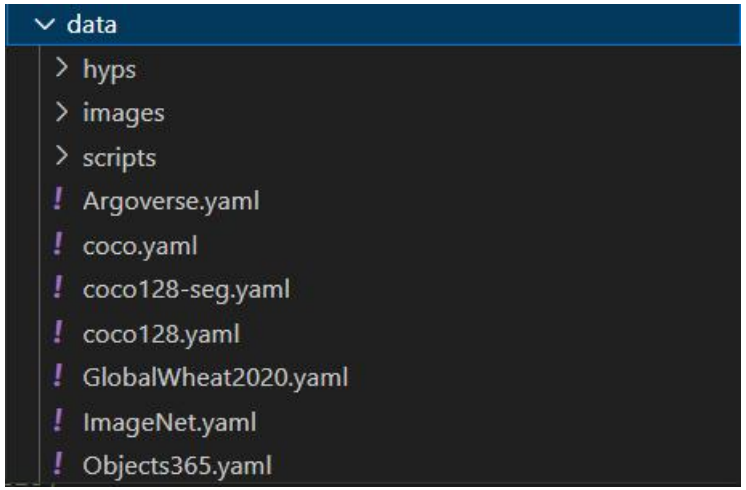
项目结构

- .github
- classify
- data
- models
- segment
- utils
- .dockerignore
- .gitattributes
- .gitignore
- .pre-commit-config.yaml
- benchmarks.py
- CITATION.cff
- CONTRIBUTING.md
- detect.py
- export.py
- hubconf.py
- LICENSE
- README.md
- README.zh-CN.md
- requirements.txt
- setup.cfg
- train.py
- tutorial.ipynb
- val.py

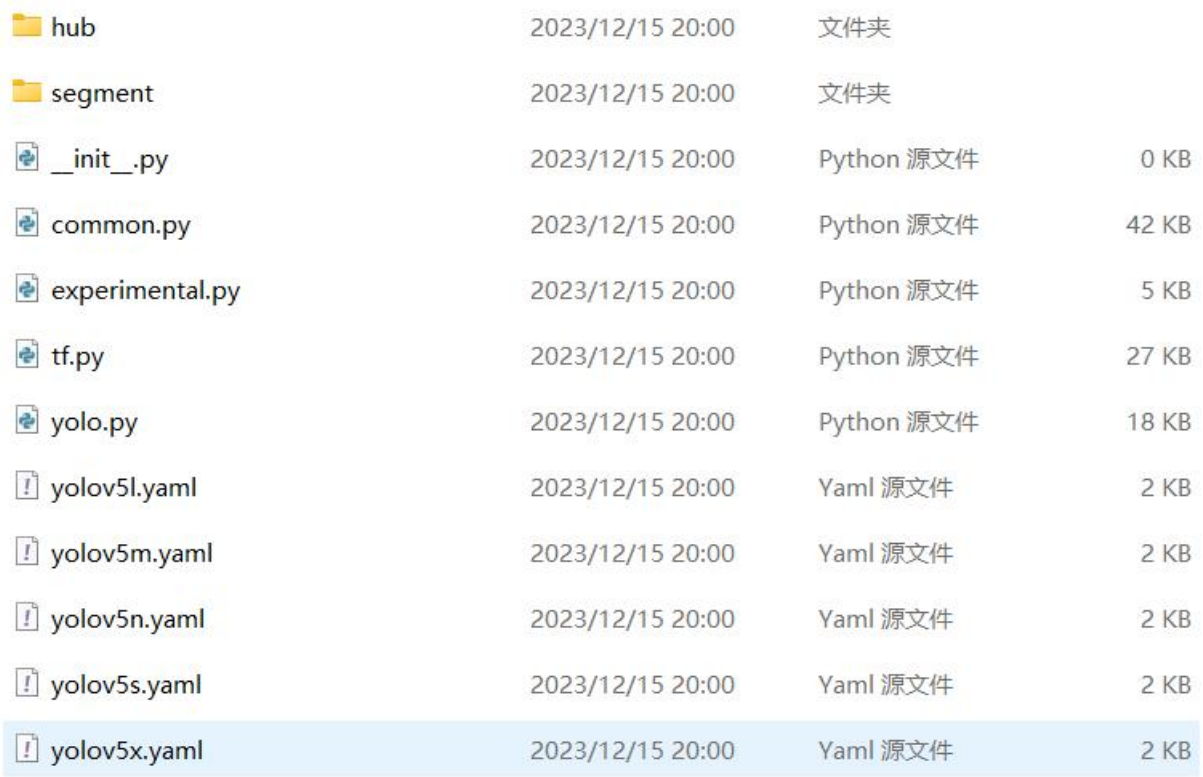


data

- data文件夹主要是存放一些超参数的配置文件(如.yaml文件是用来配置训练集和测试集还有验证集的路径的，其中还包括目标检测的种类数和种类的名称:还有一些测试数据集（imageNet、coco,etc.）。



models



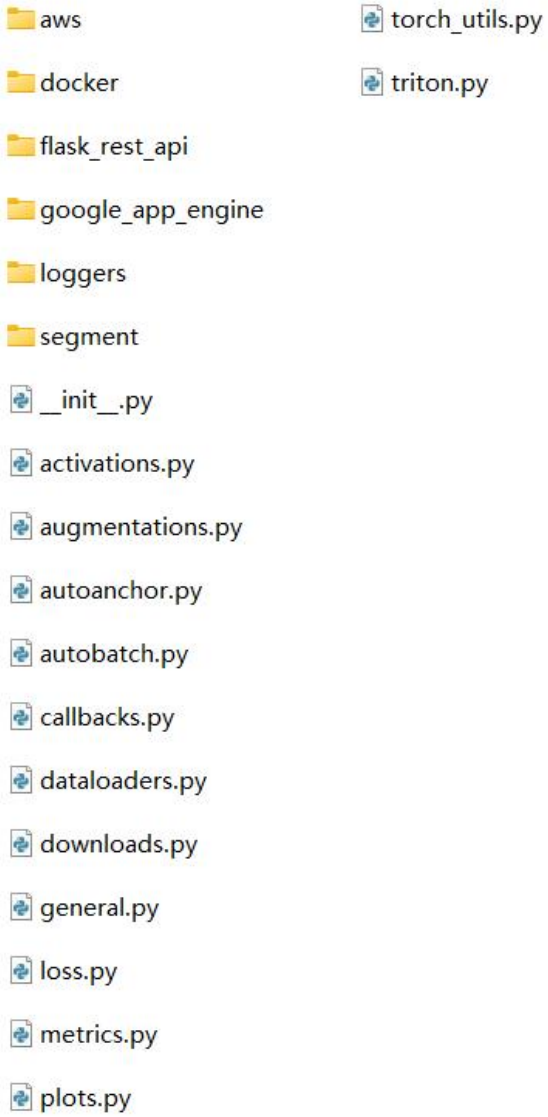
hub	2023/12/15 20:00	文件夹	
segment	2023/12/15 20:00	文件夹	
__init__.py	2023/12/15 20:00	Python 源文件	0 KB
common.py	2023/12/15 20:00	Python 源文件	42 KB
experimental.py	2023/12/15 20:00	Python 源文件	5 KB
tf.py	2023/12/15 20:00	Python 源文件	27 KB
yolo.py	2023/12/15 20:00	Python 源文件	18 KB
yolov5l.yaml	2023/12/15 20:00	Yaml 源文件	2 KB
yolov5m.yaml	2023/12/15 20:00	Yaml 源文件	2 KB
yolov5n.yaml	2023/12/15 20:00	Yaml 源文件	2 KB
yolov5s.yaml	2023/12/15 20:00	Yaml 源文件	2 KB
yolov5x.yaml	2023/12/15 20:00	Yaml 源文件	2 KB

models是模型文件夹。里面主是一些网络构建的配置文件和函数，其中包含了该项目的四个不同的版本，分别为是s、m、1、X。

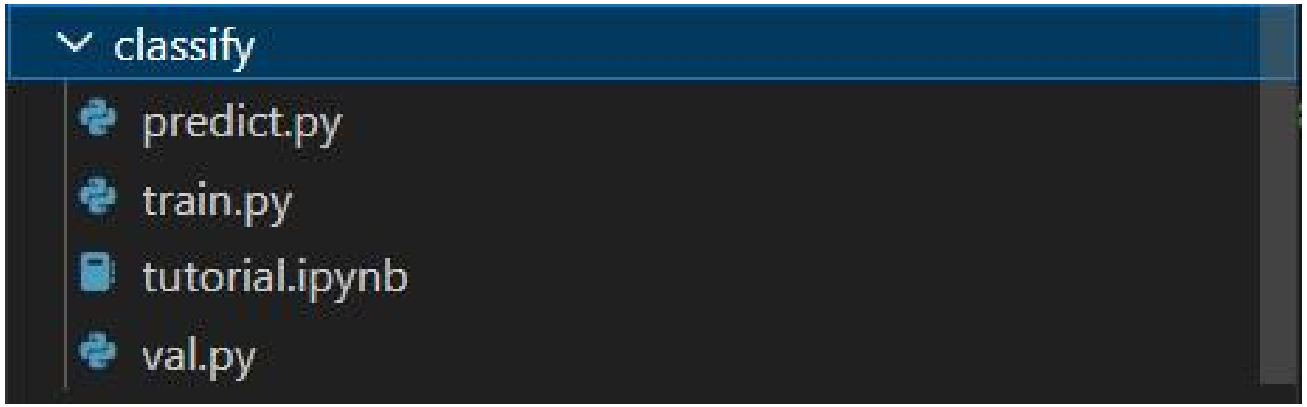
从名字就可以看出，分别代表了这几个版本的大小。

utils

存放工具函数的文件夹



代码主要组成：推理，训练，验证



predict-推理部分代码

predict-预测文件

- '''========================二、设置main函数==================================='''
- def main(opt):
- 	# 检查环境/打印参数,主要是requirement.txt的包是否安装，用彩色显示设置的参数
- 	check_requirements(exclude=('tensorboard', 'thop'))
- 	# 执行run()函数
- 	run(**vars(opt))
-
-
- # 命令使用
- if __name__ == "__main__":
- 	opt = parse_opt() # 解析参数
- 	main(opt) # 执行主函数

opt:设置参数

```python
def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s-seg.pt', help='model path(s)')
    parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/predict-seg', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
    parser.add_argument('--retina-masks', action='store_true', help='whether to plot masks in native resolution')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt
```

这段代码是一个 Python 脚本中的一个函数，用于解析命令行参数并返回这些参数的值.主要功能是为模型进行推理时提供参数。下面是每个参数的作用和默认值:

- --weights: 训练的权重路径
- --data:配置数据文件路径，包括image/label/classes等信息
- --conf-thres:置信度阀值，默认为 0.50
- -jou-thres:非极大抑制时的 loU 值，默认为 0.45
- --max-det.保留的最大检测框数量，每张图片中检测目标的个数最多为1000类
- --view-img:是否展示预测之后的图片/视频，默认False
- --save-crop是否保存裁剪预测框图片，默认为False
- --classes: 仅检测指定类别，默认为 None
- --agnostic-nms:是否使用类别不敏感的非极大抑制 (即不考虑类别信息)，默认为 False
- --augment:是否使用数据增强进行推理，默认为 False
- .--visualize:是否可视化特征图，默认为 False.
- --update :如果为True，则对所有模型进行strip optimizer操作，去除pt文件中的优化器等信息，默认为False
- --line-thickness: 画 bounding box 时的线条宽度，默认为 3
- --hakf:是否使用 FP16 半精度进行推理，默认为False.
- --dnn:是否使用 OpenCV DNN 进行 ONNX 推理，默认为 False
- --vid-stride 指定视频处理的帧率
- --retina-masks 如果运行脚本时提供了“--retina-masks”，则相应的布尔变量将设置为“True”，表示应以本机分辨率绘制蒙版。

执行run函数

```python
@smart_inference_mode()
def run(
    weights=ROOT / 'yolov5s-seg.pt',  # model.pt path(s)
    source=ROOT / 'data/images',  # file/dir/URL/glob/screen/0(webcam)
    data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
    imgsz=(640, 640),  # inference size (height, width)
    conf_thres=0.25,  # confidence threshold
    iou_thres=0.45,  # NMS IOU threshold
    max_det=1000,  # maximum detections per image
    device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
    view_img=False,  # show results
    save_txt=False,  # save results to *.txt
    save_conf=False,  # save confidences in --save-txt labels
    save_crop=False,  # save cropped prediction boxes
    nosave=False,  # do not save images/videos
    classes=None,  # filter by class: --class 0, or --class 0 2 3
    agnostic_nms=False,  # class-agnostic NMS
    augment=False,  # augmented inference
    visualize=False,  # visualize features
    update=False,  # update all models
    project=ROOT / 'runs/predict-seg',  # save results to project/name
    name='exp',  # save results to project/name
    exist_ok=False,  # existing project/name ok, do not increment
    line_thickness=3,  # bounding box thickness (pixels)
    hide_labels=False,  # hide labels
    hide_conf=False,  # hide confidences
    half=False,  # use FP16 half-precision inference
    dnn=False,  # use OpenCV DNN for ONNX inference
    vid_stride=1,  # video frame-rate stride
    retina_masks=False,
):
```

这段代码主要用于处理输入来源。定义了一些布尔值区分输入是图片、视频、网络流还是摄像头

```
'''==========================2.初始化配置============================='''
    # 输入的路径变为字符串
    source = str(source)
    # 是否保存图片和txt文件，如果nosave(传入的参数)为false且source的结尾不是txt则保存图片
    save_img = not nosave and not source.endswith('.txt')  # save inference images
    # 判断source是不是视频/图像文件路径
    # Path()提取文件名。suffix：最后一个组件的文件扩展名。若source是"D://YOLOv5/data/1.jpg"， 则Path(source).suffix是".jpg"，Path(source).suffix[1:]是"jpg"
    # 而IMG_FORMATS 和 VID_FORMATS两个变量保存的是所有的视频和图片的格式后缀。
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    # 判断source是否是链接
    # .lower()转化成小写 .upper()转化成大写 .title()首字符转化成大写，其余为小写，.startswith('http://')返回True or Flase
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    # 判断是source是否是摄像头
    # .isnumeric()是否是由数字组成，返回True or False
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not is_file)
    if is_url and is_file:
        # 返回文件。如果source是一个指向图片/视频的链接,则下载输入数据
        source = check_file(source)  # download
```

加载模型

- # Load model
- device = select_device(device)
- model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
- stride, names, pt = model.stride, model.names, model.pt
- imgsz = check_img_size(imgsz, s=stride) # check image size

这段代码主要是用于选择设备、初始化模型和检查图像大小

推理部分：

- # Inference
- 	# 可视化文件路径。如果为True则保留推理过程中的特征图，保存在runs文件夹中
- 	visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
- 	# 推理结果，pred保存的是所有的bound_box的信息，
- 	pred, proto = model(im, augment=augment, visualize=visualize)[:2]

  	# 使用model函数对图像im进行预测，augmen和visualize参数用于指示是否应该在预测时使用数据增强和可视化。

NMS去除多余框

- # NMS
- # 执行非极大值抑制，返回值为过滤后的预测框
- with dt[2]:
- pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det, nm=32) '''
- pred: 网络的输出结果
- conf_thres： 置信度阈值
- iou_thres： iou阈值
- classes: 是否只保留特定的类别 默认为None
- agnostic_nms： 进行nms是否也去除不同类别之间的框
- max_det: 检测框结果的最大数量 默认1000
- '''

执行非最大值抑制(NMS)的步骤，用于筛选预测结果

使用一个循环来遍历检测结果列表中的每个物体，并对每个物体进行处理

```
# Process predictions
for i, det in enumerate(pred):  # per image 每次迭代处理一张图片
    '''
    i: 每个batch的信息
    det:表示5个检测框的信息
    '''
    seen += 1 #seen是一个计数的功能
    if webcam:  # batch_size >= 1
        # 如果输入源是webcam则batch_size>=1 取出dataset中的一张图片
        p, im0, frame = path[i], im0s[i].copy(), dataset.count
        s += f'{i}: '# s后面拼接一个字符串i
    else:
        p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

    '''
        大部分我们一般都是从LoadImages流读取本都文件中的照片或者视频 所以batch_size=1
            p: 当前图片/视频的绝对路径 如 F:\yolo_v5\yolov5-U\data\images\bus.jpg
            s: 输出信息 初始为 ''
            im0: 原始图片 letterbox + pad 之前的图片
            frame: 视频流,此次取的是第几张图片
    '''
```

判断有没有框

```
        if len(det):
            if retina_masks:
                # scale bbox first the crop masks
            # 将预测信息映射到原图
            # 将标注的bounding_box大小调整为和原图一致（因为训练时原图经过了放缩）此时坐标格式为xyxy
                det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()
# rescale boxes to im0 size
                masks = process_mask_native(proto[i], det[:, 6:], det[:, :4], im0.shape[:2])  # HWC
            else:
                masks = process_mask(proto[i], det[:, 6:], det[:, :4], im.shape[2:], upsample=True)  # HWC
                det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()
# rescale boxes to im0 size
```

train-预测部份代码

载入参数

```
''' ======================1.载入参数和初始化配置信息==========================  '''
    '''
    1.1 载入参数
    '''
def train(hyp,  # 超参数 可以是超参数配置文件的路径或超参数字典 path/to/hyp.yaml or hyp
          opt,  # main中opt参数
          device,  # 当前设备
          callbacks  # 用于存储Loggers日志记录器中的函数，方便在每个训练阶段控制日志的记录情况
          ):
    # 从opt获取参数。日志保存路径，轮次、批次、权重、进程序号(主要用于分布式训练)等
    save_dir, epochs, batch_size, weights, single_cls, evolve, data, cfg, resume, noval, nosave, workers, freeze,
= \
        Path(opt.save_dir), opt.epochs, opt.batch_size, opt.weights, opt.single_cls, opt.evolve, opt.data, opt.cfg, \
        opt.resume, opt.noval, opt.nosave, opt.workers, opt.freeze
# 该段代码接收传来的参数
```

读取超参配置信息

```
# Hyperparameters 加载超参数
if isinstance(hyp, str): # isinstance()是否是已知类型。 判断hyp是字典还是字符串
    # 若hyp是字符串，即认定为路径，则加载超参数为字典
    with open(hyp, errors='ignore') as f:
        # 加载yaml文件
        hyp = yaml.safe_load(f)  # load hyps dict 加载超参信息
# 打印超参数 彩色字体
LOGGER.info(colorstr('hyperparameters: ') + ', '.join(f'{k}={v}' for k, v in hyp.items()))
```

加载预训练模型

```
''' ====================2.model：加载网络模型=========================  '''
# Model 载入模型
# 检查文件后缀是否是.pt
check_suffix(weights, '.pt')  # check weights
# 加载预训练权重 yolov5提供了5个不同的预训练权重，可以根据自己的模型选择预训练权重
pretrained = weights.endswith('.pt')

if pretrained:
    # 使用预训练的话：
    # torch_distributed_zero_first(RANK): 用于同步不同进程对数据读取的上下文管理器
    with torch_distributed_zero_first(LOCAL_RANK):
        weights = attempt_download(weights)  # download if not found locally
    # ============加载模型以及参数================= #
    ckpt = torch.load(weights, map_location=device)  # load checkpoint
    """
    两种加载模型的方式：opt.cfg / ckpt['model'].yaml
    这两种方式的区别：区别在于是否是使用resume
    如果使用resume-断点训练：
    将opt.cfg设为空，选择ckpt['model']yaml创建模型，且不加载anchor。
    这也影响了下面是否除去anchor的key(也就是不加载anchor)，如果resume则不加载anchor
    原因：
    使用断点训练时,保存的模型会保存anchor,所以不需要加载，
    主要是预训练权重里面保存了默认coco数据集对应的anchor，
    如果用户自定义了anchor，再加载预训练权重进行训练，会覆盖掉用户自定义的anchor。
    """
```

分别使用预训练权重参数文件与不使用预训练权重参数文件加载模型

```
    # ***加载模型*** #
    model = Model(cfg or ckpt['model'].yaml, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device)  # create

    # ***以下三行是获得anchor*** #
    # 若cfg 或 hyp.get('anchors')不为空且不使用中断训练 exclude=['anchor'] 否则 exclude=[]
    exclude = ['anchor'] if (cfg or hyp.get('anchors')) and not resume else []  # exclude keys
    # 将预训练模型中的所有参数保存下来，赋值给csd
    csd = ckpt['model'].float().state_dict()  # checkpoint state_dict as FP32
    # 判断预训练参数和新创建的模型参数有多少是相同的
    # 筛选字典中的键值对，把exclude删除
    csd = intersect_dicts(csd, model.state_dict(), exclude=exclude)  # intersect

    # ***模型创建*** #
    model.load_state_dict(csd, strict=False)  # load
    # 显示加载预训练权重的的键值对和创建模型的键值对
    # 如果pretrained为ture 则会少加载两个键对（anchors, anchor_grid）
    LOGGER.info(f'Transferred {len(csd)}/{len(model.state_dict())} items from {weights}')  # report
else:
    # #直接加载模型，ch为输入图片通道
    model = Model(cfg, ch=3, nc=nc, anchors=hyp.get('anchors')).to(device)  # create
```

设置优化器

- # Optimizer 优化器
- nbs = 64 # nominal batch size
- """
- nbs = 64
- batchsize = 16
- accumulate = 64 / 16 = 4
- 模型梯度累计accumulate次之后就更新一次模型 相当于使用更大batch_size
- """
- accumulate = max(round(nbs / batch_size), 1) # accumulate loss before optimizing
- # 根据accumulate设置权重衰减参数，防止过拟合
- hyp['weight_decay'] *= batch_size * accumulate / nbs # scale weight_decay
- # 打印缩放后的权重衰减超参数
- optimizer = smart_optimizer(model, opt.optimizer, hyp['lr0'], hyp['momentum'], hyp['weight_decay'])

nbs指的是nominal batch size,名义上的batch size。这里的nbs跟命令行参数中的batch size不同，命令行中的batch size默认为16，nbs设置为64。
accumulate 为累计次数，在这里 nbs/batch size (64/16) 计算出opt.batch size输入多少批才达到nbs的水平。简单来说，nbs为64，代表想要达到的batch size，这里的数值是64; batch size为opt.batch size，这里的数值是16，64/16等于4,也就是opt.batch size需要输入4批才能达到nbs,accumulate等于4。(round表示四舍五入取整数，而max表示accumulate不能低于1。
当给模型喂了4批图片数据后，将四批图片数据得到的梯度值，做累积。当每累积到4批数据时，才会对参数做更新，这样就实现了与batch size=64时相同的效果
最后还要做权重参数的缩放，因为batch size发生了变化，所有权重参数也要做相应的缩放。

设置学习率

```
# Scheduler  设置学习率策略:两者可供选择，线性学习率和余弦退火学习率
if opt.linear_lr:
    # 使用线性学习率
    lf = lambda x: (1 - x / (epochs - 1)) * (1.0 - hyp['lrf']) + hyp['lrf']  # linear
else:
    # 使用余弦退火学习率
    lf = one_cycle(1, hyp['lrf'], epochs)  # cosine 1->hyp['lrf']
# 可视化 scheduler
scheduler = lr_scheduler.LambdaLR(optimizer, lr_lambda=lf)  # plot_lr_scheduler(optimizer, scheduler, epochs)
```

在训练过程中变更学习率可能会让训练效果更好，YOLOv5提供了两种学习率变化的策略:
一种是linear lr (线性学习率)，是通过线性插值的方式调整学习率
另一种则是One Cyce(余孩浪火学习率)，即周期性学习率调整中，周期被设置为1。在一周期策略中，最大学习率被设置为LR
Range test 中可以找到的最高值，最小学习率比最大学习率小几个数量级，这里默认one cvcle。

EMA

- # EMA
- ema = ModelEMA(model) if RANK in {-1, 0} else None

- EMA为指数加权平均或滑动平均。其将前面模型训练权重，偏差进行保存，在本次训练过程中，假设为第次，将第一次到第n-1次以指数权重进行加和，再加上本次的结果，且越远离第n次，指数系数越大，其所占的比重越小

Resume

- # Resume
- best_fitness, start_epoch = 0.0, 0
- if pretrained:
- if resume:
- best_fitness, start_epoch, epochs = smart_resume(ckpt, optimizer, ema, weights, epochs, resume)
- del ckpt, csd

- 断点续训；可以理解为把上次中断结束时的模型，作为新的预训练模型，然后从中获取上次训练时的参数，并恢复训练状态。

创建数据集

```
181     # Trainloader 训练集数据加载
182 >   train_loader, dataset = create_dataloader(…
201     '''
202       返回一个训练数据加载器，一个数据集对象:
203       训练数据加载器是一个可迭代的对象，可以通过for循环加载1个batch_size的数据
204       数据集对象包括数据集的一些参数，包括所有标签值、所有的训练数据路径、每张图片的尺寸等等
205     '''
206     labels = np.concatenate(dataset.labels, 0)
207     mlc = int(labels[:, 0].max())  # max label class 标签符号最大值
208     assert mlc < nc, f'Label class {mlc} exceeds nc={nc} in {data}. Possible class labels are 0-{nc - 1}'
209     #  如果类别总数小于类别数则表示有问题
210     # Process 0 验证集数据集加载
211     if RANK in {-1, 0}:# 加载验证集数据加载器
212 >       val_loader = create_dataloader(val_path,…
226
227         if not resume:
228             if not opt.noautoanchor:
229                 check_anchors(dataset, model=model, thr=hyp['anchor_t'], imgsz=imgsz)  # run AutoAnchor
230             model.half().float()  # pre-reduce anchor precision
231
232             if plots:# plots画出标签信息
233                 plot_labels(labels, names, save_dir)
234         # callbacks.run('on_pretrain_routine_end', labels, names)
```

- 首先，通过create dataloader()函数得到两个对象。一个为train loader，另一个为dataset,
- train loader为训练数据加载器，可以通过for循环遍历出每个batch的训练数据
- dataset为数据集对象，包括所有训练图片的路径，所有标签，每张图片的大小，图片的配置，超参数等等
- 然后将所有样本的标签拼接到一起，统计后做可视化，同时获得所有样本的类别，根据上面的统计对所有样本的类别，中心点Xy位置，长宽wh做可视化。

```
for epoch in range(start_epoch, epochs):  # epoch ------------------------------------------------------------------
    '''
    告诉模型现在是训练阶段 因为BN层、DropOut层、两阶段目标检测模型等
    训练阶段阶段和预测阶段进行的运算是不同的，所以要将二者分开
    model.eval()指的是预测推断阶段
    '''
    model.train()

    # Update image weights (optional, single-GPU only)  更新图片的权重
    if opt.image_weights: # 获取图片采样的权重
        # 经过一轮训练，若哪一类的不精确度高，那么这个类就会被分配一个较高的权重，来增加它被采样的概率
        cw = model.class_weights.cpu().numpy() * (1 - maps) ** 2 / nc  # class weights
        # 将计算出的权重换算到图片的维度，将类别的权重换算为图片的权重
        iw = labels_to_image_weights(dataset.labels, nc=nc, class_weights=cw)  # image weights
        # 通过random.choices生成图片索引indices从而进行采样，这时图像会包含一些难识别的样本
        dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n)  # rand weighted idx

    # Update mosaic border (optional)
    # b = int(random.uniform(0.25 * imgsz, 0.75 * imgsz + gs) // gs * gs)
    # dataset.mosaic_border = [b - imgsz, -b]  # height, width borders

    # 初始化训练时打印的平均损失信息
```

- 这段代码主要是释放训练开始命令和更新权重
- 首先训练过程走起，通过model.train函数告诉模型已经进入了训练阶段。因为有些层或模型在训练阶段与预测阶段进行的操作是不一样的，所以要通过model.train()函数用来声明，接下来是训练。
- 然后是更新图片的权重。训练时有些类的准确率可能比较难以识别，准确率并不会很高。在更新图片权重时就会把这些难以识别的类挑出来，并为这个类产生一些权重高的图片，以这种方式来增加识别率低的类别的数据量。提高准确率。

scheduler

- # Scheduler 进行学习率衰减
- lr = [x['lr'] for x in optimizer.param_groups] # for loggers
- # 根据前面设置的学习率更新策略更新学习率
- scheduler.step()

后续内容就是保存训练结果和算出mAp

- Q：Why no Anchor?

Validation-验证

1、保存预测信息

首先获取图片的w和h，也就是对应的宽高，然后把每个图片的预测信息分别存入save dir/labels下的xxx.txt中接着将bbox的左上角点、右下角点坐标的格式，转换为bbox中心点 +bbox的w,h的格式，并进行归一化。即: xyxy (左上右下)xywh (中心宽高)
最后，将预测的类别和坐标值保存到对应图片image name.txt文中

```
'''======================1.保存预测信息到txt文件======================'''
def save_one_txt(predn, save_conf, shape, file):
    # Save one txt result
    # gn = [w, h, w, h] 对应图片的宽高  用于后面归一化
    gn = torch.tensor(shape)[[1, 0, 1, 0]]  # normalization gain whwh
    # 将每个图片的预测信息分别存入save_dir/labels下的xxx.txt中 每行: class_id + score + xywh
    for *xyxy, conf, cls in predn.tolist():
        # 将xyxy(左上角+右下角)格式转为xywh(中心点+宽高)格式，并归一化，转化为列表再保存
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        # line的形式是：  "类别 xywh"，若save_conf为true，则line的形式是："类别 xywh 置信度"
        line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
        # 将上述test得到的信息输出保存 输出为xywh格式 coco数据格式也为xywh格式
        with open(file, 'a') as f:
            # 写入对应的文件夹里，路径默认为“runs\detect\exp*\labels”
            f.write(('%g ' * len(line)).rstrip() % line + '\n')
```

保存的信息：
cls:图片类别
xywh:坐标+宽高
conf:置信度

计算指标

```python
def process_batch(detections, labels, iouv, pred_masks=None, gt_masks=None, overlap=False, masks=False):
    """
    Return correct prediction matrix
    Arguments:
        detections (array[N, 6]), x1, y1, x2, y2, conf, class
        labels (array[M, 5]), class, x1, y1, x2, y2
    Returns:
        correct (array[N, 10]), for 10 IoU levels
    """
    if masks:
        if overlap:
            nl = len(labels)
            index = torch.arange(nl, device=gt_masks.device).view(nl, 1, 1) + 1
            gt_masks = gt_masks.repeat(nl, 1, 1)  # shape(1,640,640) -> (n,640,640)
            gt_masks = torch.where(gt_masks == index, 1.0, 0.0)
        if gt_masks.shape[1:] != pred_masks.shape[1:]:
            gt_masks = F.interpolate(gt_masks[None], pred_masks.shape[1:], mode='bilinear', align_corners=False)[0]
            gt_masks = gt_masks.gt_(0.5)
        iou = mask_iou(gt_masks.view(gt_masks.shape[0], -1), pred_masks.view(pred_masks.shape[0], -1))
    else:  # boxes
        iou = box_iou(labels[:, 1:], detections[:, :4])

    correct = np.zeros((detections.shape[0], iouv.shape[0])).astype(bool)
    correct_class = labels[:, 0:1] == detections[:, 5]
    for i in range(len(iouv)):
        x = torch.where((iou >= iouv[i]) & correct_class)  # IoU > threshold and classes match
        if x[0].shape[0]:
            matches = torch.cat((torch.stack(x, 1), iou[x[0], x[1]][:, None]), 1).cpu().numpy()  # [label, detect, iou]
            if x[0].shape[0] > 1:
                matches = matches[matches[:, 2].argsort()[::-1]]
                matches = matches[np.unique(matches[:, 1], return_index=True)[1]]
                # matches = matches[matches[:, 2].argsort()[::-1]]
                matches = matches[np.unique(matches[:, 0], return_index=True)[1]]
            correct[matches[:, 1].astype(int), i] = True
    return torch.tensor(correct, dtype=torch.bool, device=iouv.device)
```

- 这段代码主要是计算correct，来获取匹配预测框的iou信息

这个函数主要有两个作用:

- 。作用1:对预测框与gt进行匹配
- 。作用2:对匹配上的预测框进行iou数值判断，用True来填充，其余没有匹配上的预测框的所以行数全部设置为False
- 对于每张图像的预测框，需要筛选出能与gt匹配的框来进行相关的iou计算，设置了iou从0.5-0.95的10个梯度，如果匹配的预测框iou大于相对于的阈值，则在对应位置设置为True，否则设置为False；而对于没有匹配上的预测框全部设置为False。

执行run函数-参数设置

```
def run(data, # 数据集配置文件地址 包含数据集的路径、类别个数、类名、下载地址等信息 train.py时传入data_dict
        weights=None,  # 模型的权重文件地址 运行train.py=None 运行test.py=默认weights/yolov5s
        batch_size=32,  # 前向传播的批次大小 运行test.py传入默认32 运行train.py则传入batch_size // WORLD_SIZE * 2
        imgsz=640,  # 输入网络的图片分辨率 运行test.py传入默认640 运行train.py则传入imgsz_test
        conf_thres=0.001,  # object置信度阈值 默认0.001
        iou_thres=0.6,  # 进行NMS时IOU的阈值 默认0.6
        task='val',  # 设置测试的类型 有train, val, test, speed or study几种 默认val
        device='',  # 执行 val.py 所在的设备 cuda device, i.e. 0 or 0,1,2,3 or cpu
        single_cls=False,  # 数据集是否只有一个类别 默认False
        augment=False,  # 测试时增强
        verbose=False,  # 是否打印出每个类别的mAP 运行test.py传入默认Fasle 运行train.py则传入nc < 50 and final_epoch
        save_txt=False,  # 是否以txt文件的形式保存模型预测框的坐标 默认True
        save_hybrid=False,  # 是否保存预测每个目标的置信度到预测txt文件中 默认True
        save_conf=False,  # 保存置信度
        save_json=False,  # 是否按照coco的json格式保存预测框，并且使用cocoapi做评估（需要同样coco的json格式的标签），
                          #运行test.py传入默认Fasle 运行train.py则传入is_coco and final_epoch(一般也是False)
        project=ROOT / 'runs/val',  # 验证结果保存的根目录 默认是 runs/val
        name='exp',  # 验证结果保存的目录 默认是exp  最终: runs/val/exp
        exist_ok=False,  # 如果文件存在就increment name，不存在就新建  默认False(默认文件都是不存在的)
        half=True,  # 使用 FP16 的半精度推理
        dnn=False,  # 在 ONNX 推理时使用 OpenCV DNN 后段端
        model=None,  # 如果执行val.py就为None 如果执行train.py就会传入( model=attempt_load(f, device).half() )
        dataloader=None, # 数据加载器 如果执行val.py就为None 如果执行train.py就会传入testloader
        save_dir=Path(''), # 文件保存路径 如果执行val.py就为‘’ , 如果执行train.py就会传入save_dir(runs/train/expn)
        plots=True, # 是否可视化 运行val.py传入，默认True
        callbacks=Callbacks(),  # 回调函数
        compute_loss=None, # 损失函数 运行val.py传入默认None 运行train.py则传入compute_loss(train)
        ):
```

加载val数据集

```
'''======================加载val数据集======================'''
    # Dataloader
    if not training:
        if pt and device.type != 'cpu':
            # 创建一张全为0的图片（四维张量）
            model(torch.zeros(1, 3, imgsz, imgsz).to(device).type_as(next(model.model.parameters()))) # warmup
        pad = 0.0 if task == 'speed' else 0.5
        task = task if task in ('train', 'val', 'test') else 'val'  # path to train/val/test images
        # 调用datasets.py文件中的create_dataloader函数创建dataloader
        dataloader = create_dataloader(data[task], imgsz, batch_size, stride, single_cls, pad=pad, rect=pt,
                                       prefix=colorstr(f'{task}: '))[0]
```

- 这段代码主要是加载val数据集
- 判断是否是训练。如果不是训练——执行val.py调用run函数，就调用create dataloader生成dataloader。如果是训练执行train.py调用run函数，就不需要生成dataloader 可以直接从参数中传过来testloader。
- 训练时 (train.py) 调用: 加载val数据集
- 验证时 (val.py) 调用: 不需要加载val数据集 直接从train.py 中传入testloader

计算loss

- '''===计算损失==='''
- # Loss
- # compute_loss不为空 说明正在执行train.py 根据传入的compute_loss计算损失值
- if compute_loss:
- # loss 包含bounding box 回归的GIoU、object和class 三者的损失
- loss += compute_loss([x.float() for x in train_out], targets)[1] # box, obj, cls

这段代码主要是计算验证集损失
判断compute loss是否为空，不为空则说明正在执行train.py，根据传入的compute loss计算损失值.loss 包含bounding box 回归的GioU、object和class 三者的损失
分类损失(cls loss): 该损失用于判断模型是否能够准确地识别出图像中的对象，并将其分类到正确的类别中。
置信度损失(obj loss): 该损失用于衡量模型预测的框(即包含对象的矩形)与真实框之间的差异.
边界框损失(box loss): 该损失用于衡量模型预测的边界框与真实边界框之间的差异，这有助于确保模型能够准确地定位对象

运行NMS，删除冗余预测框

首先将真实框target的xywh (因为 target 是在 labelimg 中做了归一化的)映射到真实的图像尺寸
然后，在NMS之前将数据集标签 targets 添加到模型预测中，这允许在数据集中自动标记(for autlabelling)其它对象且mAP反映了新的混合标签。
最后调用general.py中的函数，进NMS操作，并计算NMS过程所需要的时间

```
'''===6.4 NMS获得预测框==='''
        # NMS
        # targets: [num_target, img_index+class_index+xywh] = [31, 6]
        targets[:, 2:] *= torch.Tensor([width, height, width, height]).to(device)  # to pixels
        # 提取bach中每一张图片的目标的label
        # lb: {list: bs} 第一张图片的target[17, 5] 第二张[1, 5] 第三张[7, 5] 第四张[6, 5]
        lb = [targets[targets[:, 0] == i, 1:] for i in range(nb)] if save_hybrid else []  # for autolabelling
        # 计算NMS过程所需要的时间
        t3 = time_sync()
        # 调用general.py中的函数 进行非极大值抑制操作
        out = non_max_suppression(out, conf_thres, iou_thres, labels=lb, multi_label=True,
agnostic=single_cls)
        # 累计NMS时间
        dt[2] += time_sync() - t3
```

画出前三个batch图片的gt和pred框

```
# Plot images
if plots and batch_i < 3:
    if len(plot_masks):
        plot_masks = torch.cat(plot_masks, dim=0)
    plot_images_and_masks(im, targets, masks, paths, save_dir / f'val_batch{batch_i}_labels.jpg', names)
    plot_images_and_masks(im, output_to_target(preds, max_det=15), plot_masks, paths,
                          save_dir / f'val_batch{batch_i}_pred.jpg', names)  # pred

# callbacks.run('on_val_batch_end')
```

计算指标

```
# Compute metrics
stats = [torch.cat(x, 0).cpu().numpy() for x in zip(*stats)]  # to numpy
if len(stats) and stats[0].any():
    results = ap_per_class_box_and_mask(*stats, plot=plots, save_dir=save_dir, names=names)
    metrics.update(results)
nt = np.bincount(stats[4].astype(int), minlength=nc)  # number of targets per class
```

- 这段代码主要用于计算评判分类结果发各种指标

Thank You!

PPT部分内容参考了csdn上一位大佬的解读及注释：
http://t.csdnimg.cn/ItiSg