

2. 使用yolov5训练PASCAL VOC数据集

在进行训练之前首先需要安装相关的环境，包括：

1. CUDA
2. cuDNN
3. anaconda
4. pytorch
5. torchvision
6. yolov5项目中的requirements.txt中包含的依赖项

这里我已经将所有环境都安装好了，就不详细记录过程了，使用的Python版本是3.8，我的CUDA版本是12.5 由于torch官网没有提供12.5的安装命令，我安装的是对应12.2版本的torch。

关于PASCAL VOC数据集

[PASCAL](#) VOC挑战赛（The [PASCAL](#) Visual Object Classes）是一个世界级的[计算机视觉](#)挑战赛，[PASCAL](#)全称：Pattern Analysis, Statical Modeling and Computational Learning，是一个由欧盟资助的网络组织。

很多优秀的计算机视觉模型比如分类，定位，检测，分割，动作识别等模型都是基于PASCAL [VOC](#)挑战赛及其数据集上推出的，尤其是一些目标检测模型（比如大名鼎鼎的R CNN系列，以及后面的YOLO，SSD等）。

PASCAL VOC从2005年开始举办挑战赛，每年的内容都有所不同，从最开始的分类，到后面逐渐增加检测，分割，人体布局，动作识别（Object Classification、Object Detection、Object Segmentation、Human Layout、Action Classification）等内容，数据集的容量以及种类也在不断的增加和改善。该项挑战赛催生出了一大批优秀的计算机视觉模型（尤其是以深度学习技术为主的）。

目前最具有代表性的PASCAL VOC数据集是**PASCAL VOC 2007**和**PASCAL2012数据集**。这两个数据集频频在现在的一些**检测或分割类**的论文当中出现。

PASCALVOC数据集的下载地址：

```
http://host.robots.ox.ac.uk/pascal/VOC/voc2012/VOCtrainval_11-May-2012.tar
```

```
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar
```

```
http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-Nov-2007.tar
```

由于PASCAL VOC2012的测试集没有公开，因此只有2012的训练、验证集和2007的训练、验证集和测试集。

PASCAL VOC数据集的内容

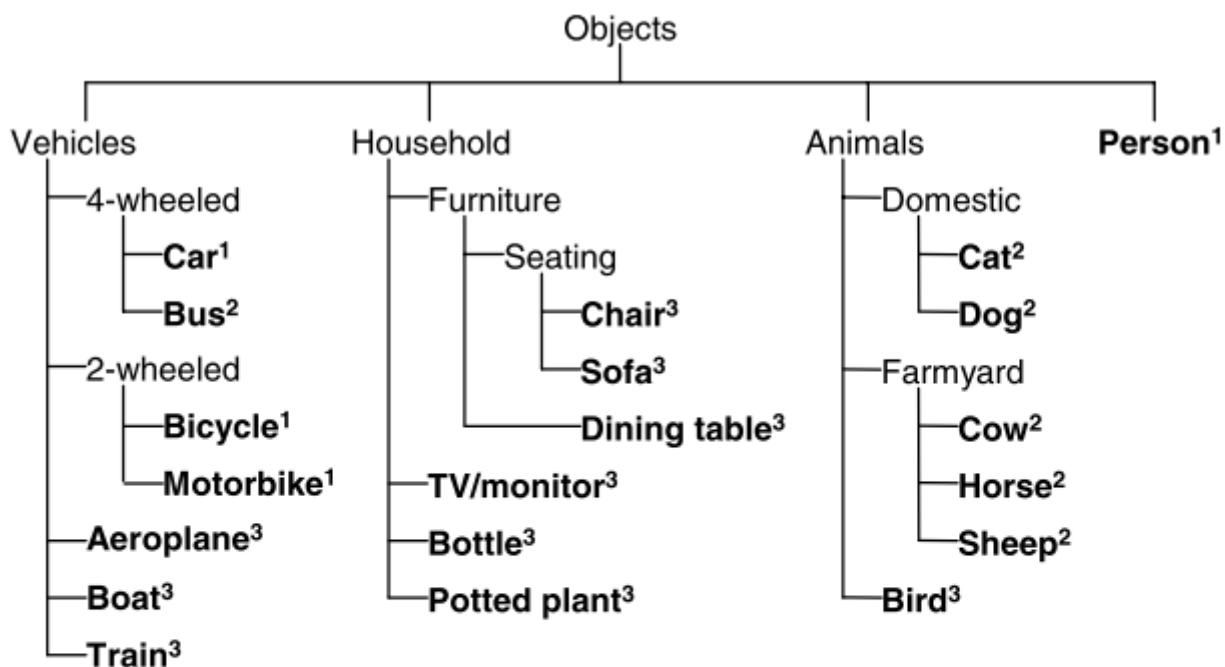


Fig. 2 VOC2007 Classes. Leaf nodes correspond to the 20 classes. The year of inclusion of each class in the challenge is indicated by superscripts: 2005¹, 2006², 2007³. The classes can be considered in a notional taxonomy, with successive challenges adding new branches (increasing the domain) and leaves (increasing detail)

PASCAL数据集中包含如上内容，共四个大类，二十个小类（图中加粗的部分），PASCAL VOC在预测时只输出其中加粗的类别。

翻译后的内容（仅加粗的内容）：

交通工具：

- 四轮车：汽车、公交车
- 两轮车：自行车、摩托车
- 飞机
- 船
- 火车

家庭：

- 椅子
- 沙发
- 餐桌

----- 电视/显示器

----- 瓶子

----- 盆栽

动物:

----- 猫

----- 狗

----- 牛

----- 马

----- 羊

----- 鸟

人类

以下是VOC2007与2012数据集的类别统计:

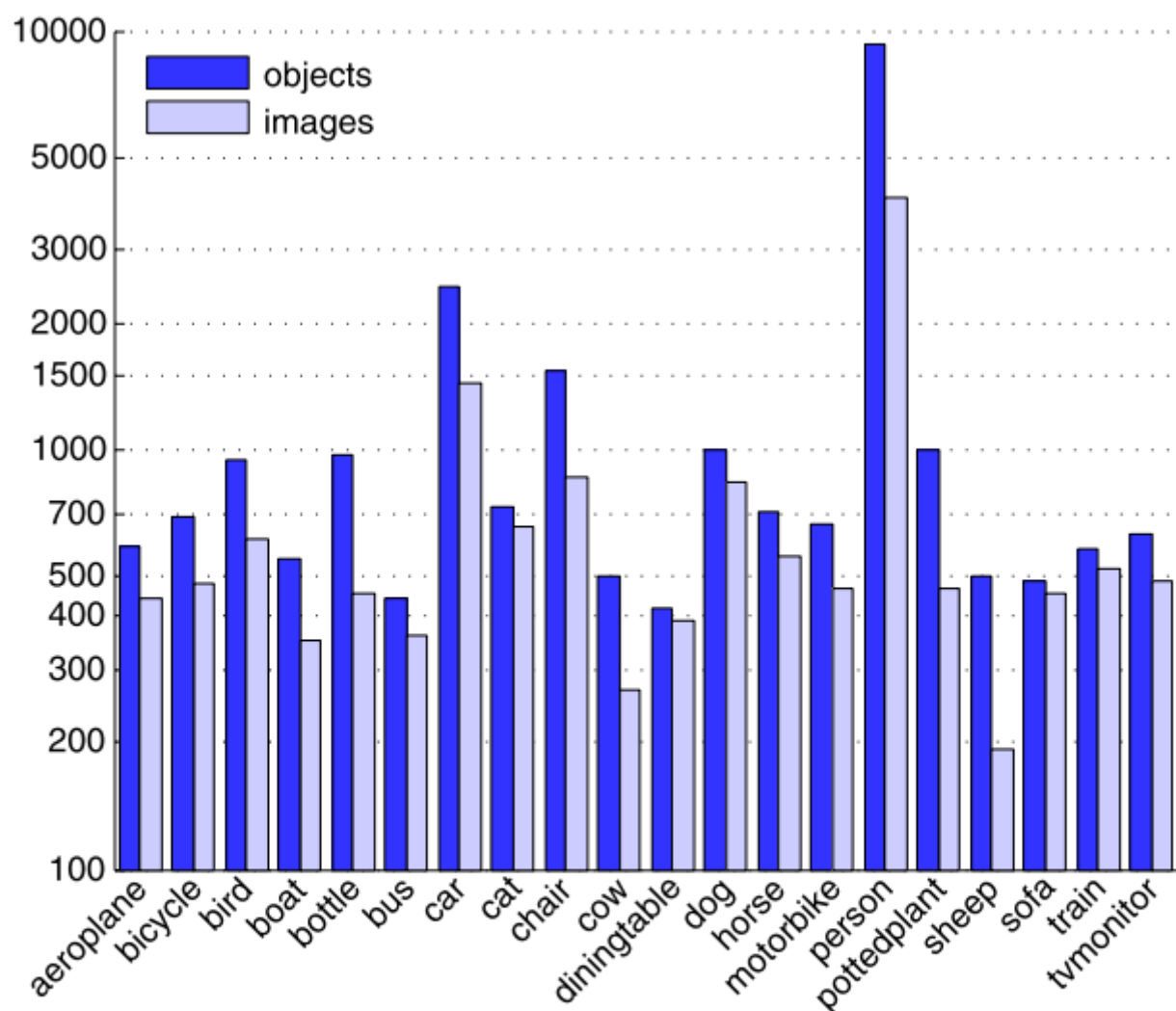


Fig. 4 Summary of the entire VOC2007 dataset. Histogram by class of the number of objects and images containing at least one object of the corresponding class. Note the log scale

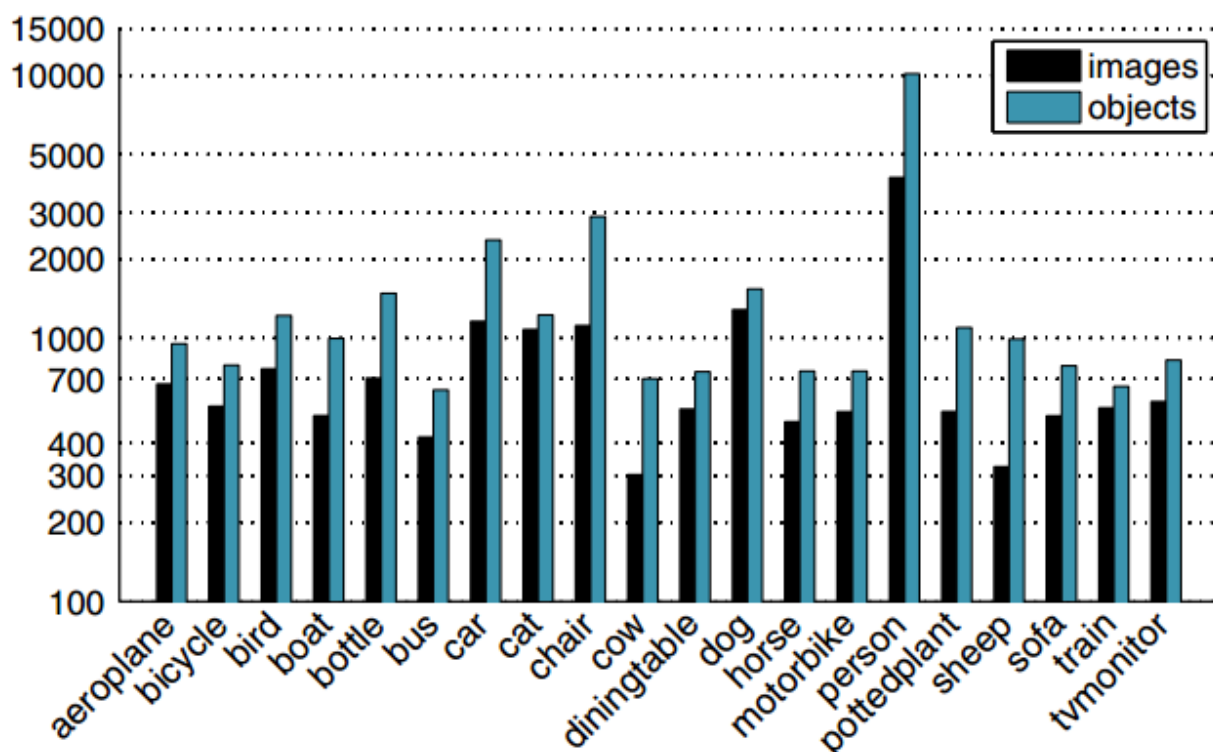


Fig. 2 Summary of the main VOC2012 dataset. Training and validation images only. Histogram by class of the number of objects and images containing at least one object of the corresponding class. Note the log scale on the *vertical axis*. Best viewed in *colour* (Color figure online)

可以看出2007与2012数据集中包含最多的图像都是关于人的图像。

两者对比：

	train		val		trainval		test		数据集总量	
	Images	Objects	Images	Objects	Images	Objects	Images	Objects	Images	Objects
VOC 2007	2501	6301	2510	6307	5011	12608	4952	12032	9963	24640
VOC 2012	5717	13609	5823	13841	11540	27450	11540	27450	23080	54900
求和	8218	19910	8333	20148	16551	40058	16492	39482	33043	79540

其中VOC2007数据集包含共9k张图像、24K个对象，VOC2012数据集包含23K张图像、54K个对象。

PASCAL VOC数据集的结构

PASCAL VOC数据集的结构如下所示：

PASCAL VOC 2012 数据集

|

└─ VOC2012

```

|   └─ JPEGImages      # 包含所有图像文件
|   |   └─ 2007_000027.jpg
|   |   └─ 2007_000032.jpg
|   |   └─ ...
|   |
|   └─ Annotations     # 包含所有标注文件（XML格式）
|   |   └─ 2007_000027.xml
|   |   └─ 2007_000032.xml
|   |   └─ ...
|   |
|   └─ ImageSets
|   |   └─ Main
|   |       └─ train.txt  # 训练集的图像文件列表
|   |       └─ val.txt   # 验证集的图像文件列表
|   |       └─ test.txt  # 测试集的图像文件列表
|   |
|   └─ SegmentationClass # 语义分割的标注
|   |   └─ 2007_000032.png
|   |   └─ ...
|   |
|   └─ SegmentationObject # 物体分割的标注
|   |   └─ 2007_000032.png
|   |   └─ ...
|   |
|   └─ ...              # 其他可能的子文件夹
|
└─ VOCdevkit
    └─ VOCcode           # 包含用于处理数据集的工具代码
    └─ README

```

实际上我们需要的只有：

1. JPEGImages: 存放所有的图片
2. Annotations: 存放所有的标注信息

需要注意的是，YOLOv5 的要求标注文件后缀为 `.txt`，但 Annotations 中的文件后缀是 `.xml`，所以需要对PASCAL VOC数据集的标注文件进行转化，也就是将原先的xml格式转为yolo要求的.txt格式。

yolo标注文件的内容大致如下：

```
0 0.481719 0.634028 0.690625 0.713278
1 0.741094 0.524306 0.314750 0.933389
2 0.254162 0.247742 0.574520 0.687422
```

其中，每行代表一个物体的标注，每个标注包括五个值，分别是：

<class_id>: 物体的类别标识符。在这里，有三个不同的类别，分别用 0、1 和 2 表示。
<center_x>: 物体边界框的中心点 x 坐标，归一化到图像宽度。这些值的范围应在 0 到 1 之间。
<center_y>: 物体边界框的中心点 y 坐标，归一化到图像高度。同样，这些值的范围应在 0 到 1 之间。
<width>: 物体边界框的宽度，归一化到图像宽度。
<height>: 物体边界框的高度，归一化到图像高度。

以第一行为例：

<class_id> 是 0，表示这个物体属于类别 0。
<center_x> 是 0.481719，这意味着物体边界框的中心点 x 坐标位于图像宽度的 48.17% 处。
<center_y> 是 0.634028，中心点 y 坐标位于图像高度的 63.40% 处。
<width> 是 0.690625，边界框宽度占图像宽度的 69.06%。
<height> 是 0.713278，边界框高度占图像高度的 71.33%

一般而言，yolov5所需要的数据集结构如下所示：

```
-- test
|  |-- images
|  |  |-- 000000000036.jpg
|  |  `-- 000000000042.jpg
|  `-- labels
|      |-- 000000000036.txt
|      `-- 000000000042.txt
-- train
|  |-- images
|  |  |-- 000000000009.jpg
|  |  `-- 000000000025.jpg
|  `-- labels
|      |-- 000000000009.txt
|      `-- 000000000025.txt
`-- val
    |-- images
    |  |-- 000000000030.jpg
    |  `-- 000000000034.jpg
```

```
`-- labels
    |-- 0000000000030.txt
    |-- 0000000000034.txt
```

其中test指测试集、train指训练集、val指验证集，每个目录下包括两个子目录，其中images是图像文件，labels是图像标注的标签文件，标签文件名和图像文件名对应。

PASVOCAL VOC数据集yolov5数据集

使用如下代码（偷的）：

```
"""
本脚本有两个功能：
    1. 将 voc 数据集标注信息(.xml)转为 yolo 标注格式(.txt)，并将图像文件复制到相应文件夹
    2. 根据 json 标签文件，生成对应 names 标签(my_data_label.names)
    3. 兼容 YOLOv3 和 YOLOv5"""
import os
from tqdm import tqdm
from lxml import etree
import json
import shutil
import argparse
from tqdm import tqdm
from prettytable import PrettyTable
from sklearn.model_selection import train_test_split

def args_table(args):
    # 创建一个表格
    table = PrettyTable(["Parameter", "Value"])
    table.align["Parameter"] = "l" # 使用 "l" 表示左对齐
    table.align["Value"] = "l" # 使用 "l" 表示左对齐

    # 将args对象的键值对添加到表格中
    for key, value in vars(args).items():
        # 处理列表的特殊格式
        if isinstance(value, list):
            value = ', '.join(map(str, value))
        table.add_row([key, value])

    # 返回表格的字符串表示
    return str(table)

def generate_train_and_val_txt(args):
    target_train_file = args.train_txt_path
```

```

target_val_file = args.val_txt_path

# 获取源文件夹中的所有文件
files = os.listdir(args.voc_images_path)

# 划分训练集和验证集
train_images, val_images = train_test_split(files,
test_size=args.val_size, random_state=args.seed)

# 打开目标文件以写入模式
with open(target_train_file, 'w', encoding='utf-8') as f:
    # 使用tqdm创建一个进度条, 迭代源文件列表
    for file in tqdm(train_images, desc=f"\033[1;33mProcessing Files for train\033[0m"):
        file_name, _ = os.path.splitext(file)
        # 写入文件名
        f.write(f'{file_name}\n')

with open(target_val_file, 'w', encoding='utf-8') as f:
    # 使用tqdm创建一个进度条, 迭代源文件列表
    for file in tqdm(val_images, desc=f"\033[1;33mProcessing Files for val\033[0m"):
        file_name, _ = os.path.splitext(file)
        # 写入文件名
        f.write(f'{file_name}\n')

print(f"\033[1;32m文件名已写入到 {target_train_file} 和 {target_val_file} 文件中!\033[0m")

def parse_args():
    # 创建解析器
    parser = argparse.ArgumentParser(description="将 .xml 转换为 .txt")

    # 添加参数
    parser.add_argument('--voc_root', type=str, default="VOCdevkit",
                        help="PASCAL VOC路径(之后的所有路径都在voc_root下)")
    parser.add_argument('--voc_version', type=str, default="VOC2012-Lite",
                        help="VOC 版本")
    parser.add_argument('--save_path', type=str, default="VOC2012-YOLO",
                        help="转换后的保存目录路径")
    parser.add_argument('--train_list_name', type=str, default="train.txt",
                        help="训练图片列表名称")
    parser.add_argument('--val_list_name', type=str, default="val.txt",
                        help="验证图片列表名称")
    parser.add_argument('--val_size', type=float, default=0.1, help="验证集比例")
    parser.add_argument('--seed', type=int, default=42, help="随机数种子")
    parser.add_argument('--num_classes', type=int, default=20, help="数据集类别数(用于校验)")

```



```

        parser.add_argument('--classes', help="数据集具体类别数(用于生成
classes.json 文件)",
                             default=['aeroplane', 'bicycle', 'bird', 'boat',
'bubble', 'bus', 'car', 'cat',
                                     'chair', 'cow', 'diningtable', 'dog',
'horse', 'motorbike', 'person',
                                     'pottedplant', 'sheep', 'sofa', 'train',
'tvmonitor'])

    return parser.parse_args()

def configure_path(args):
    # 转换的训练集以及验证集对应txt文件
    args.train_txt = "train.txt"
    args.val_txt = "val.txt"

    # 转换后的文件保存目录
    args.save_file_root = os.path.join(args.voc_root, args.save_path)

    # 生成json文件
    # label标签对应json文件
    args.label_json_path = os.path.join(args.voc_root, "classes.json")

    # 创建一个将类别与数值关联的字典
    class_mapping = {class_name: index + 1 for index, class_name in
enumerate(args.classes)}
    with open(args.label_json_path, 'w', encoding='utf-8') as json_file:
        json.dump(class_mapping, json_file, ensure_ascii=False, indent=4)

    print(f'\033[1;31m类别列表已保存到 {args.label_json_path}\033[0m')

    # 拼接出voc的images目录, xml目录, txt目录
    args.voc_images_path = os.path.join(args.voc_root, args.voc_version,
"JPEGImages")
    args.voc_xml_path = os.path.join(args.voc_root, args.voc_version,
"Annotations")
    args.train_txt_path = os.path.join(args.voc_root, args.voc_version,
args.train_txt)
    args.val_txt_path = os.path.join(args.voc_root, args.voc_version,
args.val_txt)

    # 生成对应的 train.txt 和 val.txt    generate_train_and_val_txt(args)

    # 检查文件/文件夹都是否存在
    assert os.path.exists(args.voc_images_path), f"VOC images path not
exist...({args.voc_images_path})"
    assert os.path.exists(args.voc_xml_path), f"VOC xml path not exist...
({args.voc_xml_path})"
    assert os.path.exists(args.train_txt_path), f"VOC train txt file not

```

```

exist...({args.train_txt_path})"
    assert os.path.exists(args.val_txt_path), f"VOC val txt file not
exist...({args.val_txt_path})"
    assert os.path.exists(args.label_json_path), f"label_json_path does not
exist...({args.label_json_path})"
    if os.path.exists(args.save_file_root) is False:
        os.makedirs(args.save_file_root)
        print(f"创建文件夹: {args.save_file_root}")

def parse_xml_to_dict(xml):
    """
    将xml文件解析成字典形式, 参考tensorflow的recursive_parse_xml_to_dict
    Args:
        xml: xml tree obtained by parsing XML file contents using lxml.etree
    Returns:
        Python dictionary holding XML contents.
    """
    if len(xml) == 0: # 遍历到底层, 直接返回tag对应的信息
        return {xml.tag: xml.text}

    result = {}
    for child in xml:
        child_result = parse_xml_to_dict(child) # 递归遍历标签信息
        if child.tag != 'object':
            result[child.tag] = child_result[child.tag]
        else:
            if child.tag not in result: # 因为object可能有多个, 所以需要放入列表里
                result[child.tag] = []
            result[child.tag].append(child_result[child.tag])
    return {xml.tag: result}

def translate_info(file_names: list, save_root: str, class_dict: dict,
train_val='train', args=None):
    """
    将对应xml文件信息转为yolo中使用的txt文件信息
    :param file_names:
    :param save_root:
    :param class_dict:
    :param
train_val:
    :return:
    """
    save_txt_path = os.path.join(save_root,
train_val, "labels")
    if os.path.exists(save_txt_path) is False:
        os.makedirs(save_txt_path)
    save_images_path = os.path.join(save_root, train_val, "images")
    if os.path.exists(save_images_path) is False:
        os.makedirs(save_images_path)

    for file in tqdm(file_names, desc="translate {}
file..."format(train_val)):
        # 检查下图像文件是否存在
        img_path = os.path.join(args.voc_images_path, file + ".jpg")
        assert os.path.exists(img_path), "file:{} not

```

```

exist...".format(img_path)

# 检查xml文件是否存在
xml_path = os.path.join(args.voc_xml_path, file + ".xml")
assert os.path.exists(xml_path), "file:{} not
exist...".format(xml_path)

# read xml
with open(xml_path) as fid:
    xml_str = fid.read()
xml = etree.fromstring(xml_str)
data = parse_xml_to_dict(xml)["annotation"]
img_height = int(data["size"]["height"])
img_width = int(data["size"]["width"])

# write object info into txt
assert "object" in data.keys(), "file: '{}' lack of object
key.".format(xml_path)
if len(data["object"]) == 0:
    # 如果xml文件中没有目标就直接忽略该样本
    print("Warning: in '{}' xml, there are no
objects.".format(xml_path))
    continue

with open(os.path.join(save_txt_path, file + ".txt"), "w") as f:
    for index, obj in enumerate(data["object"]):
        # 获取每个object的box信息
        xmin = float(obj["bndbox"]["xmin"])
        xmax = float(obj["bndbox"]["xmax"])
        ymin = float(obj["bndbox"]["ymin"])
        ymax = float(obj["bndbox"]["ymax"])
        class_name = obj["name"]

        class_index = class_dict[class_name] - 1 # 目标id从0开始

        # 进一步检查数据, 有的标注信息中可能有w或h为0的情况, 这样的数据会导致
        计算回归loss为nan
        if xmax <= xmin or ymax <= ymin:
            print("Warning: in '{}' xml, there are some bbox w/h
<=0".format(xml_path))
            continue

        # 将box信息转换到yolo格式
        xcenter = xmin + (xmax - xmin) / 2
        ycenter = ymin + (ymax - ymin) / 2
        w = xmax - xmin
        h = ymax - ymin

        # 绝对坐标转相对坐标, 保存6位小数
        xcenter = round(xcenter / img_width, 6)

```

```

        ycenter = round(ycenter / img_height, 6)
        w = round(w / img_width, 6)
        h = round(h / img_height, 6)

        info = [str(i) for i in [class_index, xcenter, ycenter, w,
h]]

        if index == 0:
            f.write(" ".join(info))
        else:
            f.write("\n" + " ".join(info))

        # copy image into save_images_path
        path_copy_to = os.path.join(save_images_path, img_path.split(os.sep)
[-1])

        if os.path.exists(path_copy_to) is False:
            shutil.copyfile(img_path, path_copy_to)

def create_class_names(class_dict: dict, args):
    keys = class_dict.keys()
    with open(os.path.join(args.voc_root, "my_data_label.names"), "w") as w:
        for index, k in enumerate(keys):
            if index + 1 == len(keys):
                w.write(k)
            else:
                w.write(k + "\n")

def main(args):
    # read class_indict
    json_file = open(args.label_json_path, 'r')
    class_dict = json.load(json_file)

    # 读取train.txt中的所有行信息，删除空行
    with open(args.train_txt_path, "r") as r:
        train_file_names = [i for i in r.read().splitlines() if
len(i.strip()) > 0]
    # voc信息转yolo，并将图像文件复制到相应文件夹
    translate_info(train_file_names, args.save_file_root, class_dict,
"train", args=args)

    # 读取val.txt中的所有行信息，删除空行
    with open(args.val_txt_path, "r") as r:
        val_file_names = [i for i in r.read().splitlines() if len(i.strip())
> 0]
    # voc信息转yolo，并将图像文件复制到相应文件夹
    translate_info(val_file_names, args.save_file_root, class_dict, "val",
args=args)

```

```

# 创建my_data_label.names文件
create_class_names(class_dict, args=args)

if __name__ == "__main__":
    args = parse_args()
    configure_path(args)

    # 美化打印 args    print(f"\033[1;34m{args_table(args)}\033[0m")

    # 执行 .xml 转 .txt    main(args)

```

保存文件名为voc2yolo.py，然后执行：

```
python voc2yolo.py --voc_root ./VOCdevkit --voc_version VOC2007
```

其中参数--voc-root指VOC数据集的地址，--voc_version指定数据集的版本。

这里我首先使用的是VOC2007的数据集，阅读代码之后发现，这里的VOC数据集根目录的参数是一个父级目录，VOC版本应当是数据集根目录下的子目录，所有的遍历都在这个目录下进行。

同时还有其他的参数可选：

- --save_path 转换后的保存目录路径
- --train_list_name 训练图片列表名称
- --val_list_name 验证集图片列表名称
- --val_size 验证集的比例
- --seed 随机数种子
- --num_classes 数据集类别数
- --classes 数据集具体类别

这里默认的参数值就是VOC的参数，因此其他的保持默认即可，另外再指定输出目录：

```
python voc2yolo.py --voc_root ./voc_set --voc_version VOC2007 --save_path
VOC_2007_YOLO
```

需要注意的是，**输出之后的文件也是以指定的voc根目录为父级目录。**

转换完成之后还会额外数据两个文件：

- classes.json 各个类别的比例文件
- my_data_label.names 各个类别的名称

修改yolov5关于VOC的配置文件：

1. 修改数据集配置文件

在yolo项目下的data目录中找到VOC.yaml文件，复制一份重命名为voc_007.yaml，只需要保留：

```
path: ../voc_set/VOC_2007_YOLOV5
train: # train images (relative to 'path') 16551 images
  - train/images
val: # val images (relative to 'path') 4952 images
  - val/images

# Classes
names:
  0: aeroplane
  1: bicycle
  2: bird
  3: boat
  4: bottle
  5: bus
  6: car
  7: cat
  8: chair
  9: cow
  10: diningtable
  11: dog
  12: horse
  13: motorbike
  14: person
  15: pottedplant
  16: sheep
  17: sofa
  18: train
  19: tvmonitor
```

其中路径是数据集的路径，train为数据集下的测试集的图像路径，val为数据集的验证集的图像路径，names为数据集中包含的类别名称。

其他的配置项关于测试集和下载项是可选的，暂时也不需要。

需要注意的是：**yolov5是支持多个文件夹的，也就是在train、val、test等配置项中可以指定多个值。**

例如：

```
train:
  - partA/train2017.txt
```






```

- partB/train2017.txt
- partC/train2017.txt
val:
- partA/val2017.txt
- partB/val2017.txt
- partC/val2017.txt
test:
- partA/test-dev2017.txt
- partB/test-dev2017.txt

```

2. 修改模型配置文件

在修改完数据集配置文件之后，还需要修改模型配置文件，选择使用的预训练模型。在选择模型之前看一下不同模型的对比：

				
Nano	Small	Medium	Large	XLarge
YOLOv5n	YOLOv5s	YOLOv5m	YOLOv5l	YOLOv5x
4 MB _{FP16} 6.3 ms _{V100} 28.4 mAP _{COCO}	14 MB _{FP16} 6.4 ms _{V100} 37.2 mAP _{COCO}	41 MB _{FP16} 8.2 ms _{V100} 45.2 mAP _{COCO}	89 MB _{FP16} 10.1 ms _{V100} 48.8 mAP _{COCO}	166 MB _{FP16} 12.1 ms _{V100} 50.7 mAP _{COCO}

这里通过三个指标和可视化的网络结构来直观展现出不同预训练模型的区别，其中：
第一个指标表示使用FP16（16位浮点数精度）的情况下的存储大小。这个也就是模型预训练权重文件的大小。

第二个指标是在使用V100GPU的情况下的推理时间，反应模型的推理速度。

第三个指标是在COCO数据集下的平均平均精确度，反应模型预测的正确率。

这里我选择yolov5s模型来对训练数据集。

首先在yolov5的github仓库中选择当前发行版本的tag，在tag列表中下载yolov5s.pt文件，存放到项目根目录下的weights目录下、

然后修改配置文件models/yolov5s.yaml，这里我复制了一份并且重命名为yolov5s_voc.yaml。

这里只需要修改nc参数为20即可，表示数据集的类别，VOC数据集一共有20个类别所以填写20。

3. 开始训练

执行：

```
python train.py --data data/voc_2007.yaml --cfg models/yolov5s_voc.yaml --weights weights/yolov5s.pt --batch-size 16 --epochs 200
```

这里的参数在train.py第544行的parse_opt函数中有相关介绍以及其他更多可用的参数。

4. 故障排除

在我训练的过程中产生许多意料之外的错误，好在最后的——排除了。

首先是控制台警告，是原先的一个AMP广播机制的函数将被弃用，更换为新的支持的函数即可。

然后在训练过程中出现内存溢出的问题，这里花费很长时间解决，直接导致这个问题的原因是我的C盘空间不足了，由于我的内存只有16GB，因此在训练过程中需要使用大量的虚拟内存，这会导致在磁盘中产生很大的虚拟内存文件，我这里是保存在C盘的，虽然我在接下来将虚拟内存的页面文件修改为在D盘保存但是没有产生效果，C盘一直处于爆满的状态，因此我将系统备份之后，迁移到另一个容量更大的固态硬盘中，更换硬盘之后就没有这个问题了。

训练结果

训练结果会自动记录在 Tensorboard 和 CSV 日志记录器中，保存在 runs/train 目录下，每次新的训练都会创建一个新的实验目录，例如 runs/train/exp2、runs/train/exp3 等。

该目录包含了训练和验证的统计数据、马赛克图像、标签、预测结果、以及经过增强的马赛克图像，还包括 Precision-Recall (PR) 曲线和混淆矩阵等度量和图表。

同时有两个参数文件：

- hyp.yaml 训练时的超参数
- opt.yaml 训练时的配置参数

训练完成之后可以通过tensorborad对训练过程可视化：

```
tensorboard --logdir=./runs
```

tensorboard模块需要通过pip命令安装。

```
pip install tensorboard
```

如果运行产生报错，将tensorboard降级：


```
pip install tensorboard==2.12.0
```

。。。我在训练的时候没有启用tensorboard，彻底废了。
由于没有.event文件，没办法用tensorboard可视化训练过程了，不过在训练结果目录下的result.txt和各种统计指标的图像也可以用于分析训练过程。

测试训练的模型

1. 使用模型进行预测

训练完成之后的模型保存在runs/trains/expX/weights目录中，其中有两个模型：

- last.pt 最后一次训练的模型
- best.pt 训练结果中表现最好的模型

在控制台运行：

```
python detect.py --weights runs/trains/exp13/weights/best.pt --source  
data/images/test.png
```

其中更多的参数在detect.py中的Parse_opt函数中有说明。

预测结果在runs/detect/expX目录下。



2. 性能统计

在项目目录下执行：

```
python val.py --data data/voc_2007.yaml --weights  
runs/train/exp13/weights/best.pt --batch-size 16
```

需要指定训练时的数据集配置文件和训练产生的权重文件以及训练的是batch_size。更多的参数在val.py第470行的函数中有说明。

注意：在最新的yolov5项目中，原先用于测试性能的test.py文件已经更名为val.py