

## 1.1 YOLOv8+PaddleOcr

本节主要介绍一种基于 YOLOv8 目标检测与 PaddleOcr 进行车牌识别的实现方法。

要进行车牌识别，主要分为两步。 第一步：进行车辆车牌位置的检测，我们这里使用 yolov8 训练一个车牌检测模型来进行车牌检测，精度为 0.99； 第二步：对第一步检测出的车牌进行识别，直接使用 PaddleOCR 对于车牌进行识别。

### 1.1.1 任务 11-车牌检测-YOLOv8 环境配置

主要基于 YOLOv8 训练了一个车牌检测模型，用于进行车牌位置的检测。YOLOv8 的安装命令如下：

```
pip install ultralytics -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

yolov8 源码地址：

```
https://github.com/ultralytics/ultralytics
```

### 1.1.2 任务 11-车牌检测-数据集准备与处理

训练模型使用的数据集为 CPDD2020 数据集。 数据集下载地址：

```
https://github.com/detectRecog/CCPD
```

CCPD 是一个大型的、多样化的、经过仔细标注的中国城市车牌开源数据集。CCPD 数据集主要分为 CCPD2019 数据集和 CCPD2020(CCPD-Green)数据集。CCPD2019 数据集车牌类型仅有普通车牌(蓝色车牌)，CCPD2020 数据集车牌类型仅有新能源车牌(绿色车牌)。在 CCPD 数据集中，每张图片仅包含一张车牌，车牌的车牌省份主要为皖。

CCPD 中的每幅图像都包含大量的标注信息，但是 CCPD 数据集没有专门的标注文件，每张图像的文件名就是该图像对应的数据标注。标注最困难的部分是注释四个顶点的位置。为了完成这项任务，数据发布者首先在 10k 图像上手动标记四个顶点的位置。然后设计了一个基于深度学习的检测模型，在对该网络进行良好训练后，对每幅图像的四个顶点位置进行自动标注。最后，数据发布者雇用了 7 名兼职工人在两周内纠正这些标注。CCPD 提供了超过 250k 个独特的车牌图像和详细的注释。每张图像的分辨率为 720(宽度)× 1160(高)× 3(通道)。实际上，这种分辨率足以保证每张图像中的车牌清晰可辨，但是该数据有些图片标注可能不准。不过总的来说 CCPD 数据集非常推荐研究车牌识别算法的人员学习使用。

CPDD2020 数据集一共包含 11774 张新能源汽车的车牌数据。部分图片如下：



04-90\_267-158  
&448\_542&55  
3-541&553\_16  
2&551\_158&...



015-90\_260-22  
8&437\_444&5  
07-444&507\_2  
38&502\_228...



015-91\_90-248  
&454\_464&52  
4-464&524\_24  
8&520\_248&...



015-91\_91-282  
&460\_498&53  
0-496&530\_28  
2&526\_282&...



015-92\_268-25  
4&496\_470&5  
66-456&566\_2  
54&564\_258...



025-91\_254-20  
3&499\_523&5  
77-520&577\_2  
22&576\_203...



0075-88\_267-2  
77&482\_421&  
535-421&527\_  
280&535\_277...



0075-88\_270-2  
74&482\_418&  
535-418&529\_  
275&535\_274...



0125-88\_262-2  
29&444\_409&  
514-406&505\_  
229&514\_234...



0125-91\_262-1  
54&383\_346&  
448-346&448\_  
160&441\_154...



0125-91\_262-2  
78&426\_470&  
492-470&492\_  
284&484\_278...



0175-90\_90-31  
3&416\_537&4  
95-537&489\_3  
13&495\_313...



0175-90\_95-21  
9&481\_443&5  
60-438&560\_2  
19&558\_219...



0175-90\_260-2  
43&431\_467&  
509-467&509\_  
253&496\_243...



0275-90\_258-2  
03&408\_491&



0275-90\_268-2  
18&431\_506&



0275-90\_269-2  
46&438\_534&



301-87\_99-293  
&464\_437&53



304-91\_91-135  
&439\_519&54



305-90\_99-135  
&398\_615&50



0325-89\_98-17  
4&462\_462&5

数据集中图片的命名规则如下：

"025-95\_113-154&383\_386&473-386&473\_177&454\_154&383\_363&402-0\_0\_22\_27\_27\_33\_16-37-15.jpg"。

1. 025：车牌区域占整个画面的比例；

2. 95\_113： 车牌水平和垂直角度，水平 95°， 竖直 113°

3. 154&383\_386&473：标注框左上、右下坐标，左上(154, 383)，右下(386, 473)

4. 86&473\_177&454\_154&383\_363&402：标注框四个角点坐标，顺序为右下、左下、左上、右上

5. 0\_0\_22\_27\_27\_33\_16：车牌号码映射关系如下：第一个 0 为省份 对应省份字典 provinces 中的'皖'；第二个 0 是该车所在地的地市一级代码，对应地市一级代码字典 alphabets 的'A'；后 5 位为字母和文字，查看车牌号 ads 字典，如 22 为 Y，27 为 3，33 为 9，16 为 S，最终车牌号码为皖 AY339S

省份：["皖", "沪", "津", "渝", "冀", "晋", "蒙", "辽", "吉", "黑", "苏", "浙", "京", "闽", "赣", "鲁", "豫", "鄂", "湘", "粤", "桂", "琼", "川", "贵", "云", "藏", "陕", "甘", "青", "宁", "新"]

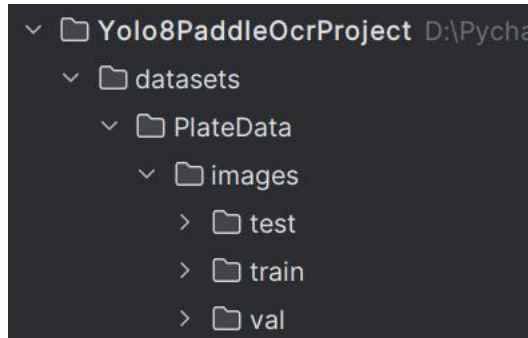
地市：['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']

车牌字典：['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',

```
'W', 'X', 'Y', 'Z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

### 1.1.3 任务 11-车牌检测-制作车牌检测数据集

创建 train、val、test 文件夹，从 CCPD 数据集中选取图片放入对应文件夹 train:56, val:22, test:22，因为我们只是演示，为了快速训练完，所以选取的图片较少。如下图：



这个数据集的检测和识别标签都在图片名中，可以直接通过上述图片的命名规则，从图片读取出来，再写入 txt 文件中即可。

需要安装 OpenCV，安装命令如下：

```
pip install opencv-python
```

制作车牌检测数据集的代码如下：

```
import shutil
import cv2
import os

def txt_translate(path, txt_path):
    print(path)
    print(txt_path)
    for filename in os.listdir(path):
        # print(filename)

        list1 = filename.split("-", 3) # 第一次分割，以减号 '-' 做分割
        subname = list1[2]
        list2 = filename.split(".", 1)
```

```

subname1 = list2[1]
if subname1 == 'txt':
    continue

lt, rb = subname.split("_", 1) # 第二次分割，以下划线'_'做分割
lx, ly = lt.split("&", 1)
rx, ry = rb.split("&", 1)
width = int(rx) - int(lx)
height = int(ry) - int(ly) # bounding box 的宽和高
cx = float(lx) + width / 2
cy = float(ly) + height / 2 # bounding box 中心点

img = cv2.imread(path + filename)
if img is None: # 自动删除失效图片（下载过程有的图片会存在无法读取的
情况）

    print(path + filename)
    os.remove(path + filename)
    continue

width = width / img.shape[1]
height = height / img.shape[0]
cx = cx / img.shape[1]
cy = cy / img.shape[0]

txtname = filename.split(".", 1)
txtfile = txt_path + txtname[0] + ".txt"
# 绿牌是第 0 类，蓝牌是第 1 类
with open(txtfile, "w") as f:
    f.write(str(0) + " " + str(cx) + " " + str(cy) + " " + str(width) + " " +
str(height))

```

```

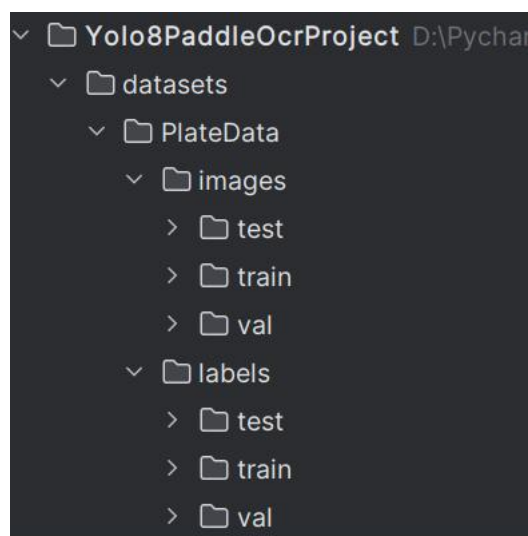
if __name__ == '__main__':
    # det 图片存储地址
    trainDir = r"G:/datasets/CarPlateData/CCPD2020/ccpd_green/train/"
    validDir = r"G:/datasets/CarPlateData/CCPD2020/ccpd_green/val/"
    testDir = r"G:/datasets/CarPlateData/CCPD2020/ccpd_green/test/"

    # det txt 存储地址
    train_txt_path = r"G:/datasets/CarPlateData/CCPD2020/ccpd_green/train_labels/"
    val_txt_path = r"G:/datasets/CarPlateData/CCPD2020/ccpd_green/val_labels/"
    test_txt_path = r"G:/datasets/CarPlateData/CCPD2020/ccpd_green/test_labels/"

    txt_translate(trainDir, train_txt_path)
    txt_translate(validDir, val_txt_path)
    txt_translate(testDir, test_txt_path)

```

制作完成后，如下图所示。



#### 1.1.4 任务 11-车牌检测-目标检测模型训练

图片数据集的存放格式如下，在项目目录中新建 **datasets** 目录，同时将分类的图片分为训练集、验证集与测试集放入 **PlateData** 目录下。

创建 **data.yaml** 文件，用于模型训练，文件内容如下：

```

train: images\train  # train images (relative to 'path') 128 images
val: images\val  # val images (relative to 'path') 128 images

```

```
test: images\test # val images (optional)
```

```
# number of classes
```

```
nc: 1
```

```
# Classes
```

```
names: ['LicensePlate']
```

数据准备完成后，通过调用 **train.py** 文件进行模型训练，**epochs** 参数用于调整训练的轮数，**batch** 参数用于调整训练的批次大小【根据内存大小调整，最小为 **1**】，代码如下：

```
#coding:utf-8

from ultralytics import YOLO

# 加载预训练模型

model = YOLO("yolov8n.pt")

# Use the model

if __name__ == '__main__':

    # Use the model

    results = model.train(data='datasets/PlateData/data.yaml', epochs=300,
batch=4) # 训练模型

    # 将模型转为 onnx 格式

    success = model.export(format='onnx')
```

第一次训练，会自动下载模型：yolov8n.pt。

另外，需要在“C:\Users\liubenjian01\AppData\Roaming\Ultralytics\settings.yaml”中配置数据集路径，才能正确找到训练数据集，如下图所示：

C:\Users\liubenjian01\AppData\Roaming\Ultralytics\settings.yaml - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

```
settings.yaml
1  settings_version: 0.0.4
2  datasets_dir: D:\PycharmProjects\Yolo8PaddleOcrProject
3  weights_dir: weights
4  runs_dir: runs
5  uuid: 619ccd93adc803861b839840276aaa90c7c049a146204c0bc083baee7585031c
6  sync: true
7  api_key: ''
8  openai_api_key: ''
9  clearml: true
10 comet: true
11 dvc: true
12 hub: true
13 mlflow: true
14 neptune: true
15 raytune: true
16 tensorboard: true
17 wandb: true
18
```

其中寻找数据集的路径是 `datasets_dir` 配置路径、`data.yaml` 所在路径、`data.yaml` 中配置路径组合起来的完整路径。

训练完后，如果想导出 `onnx` 格式模型（`best`），需要安装 `onnx` 模块，命令如下：

```
pip install onnx
```

下面是训练运行过程相关截图：



```
Project: Yolo8PaddleOcrProject
Run: train
main.py datasets.py train.py data.yaml

D:\PycharmProjects\Yolo8PaddleOcrProject\venv\Scripts\python.exe D:\PycharmProjects\Yolo8PaddleOcrProject\train.py
Ultraalytics YOLOv8.2.28 Python-3.8.18 torch-2.3.0+cpu CPU (13th Gen Intel Core(TM) i7-1355U)
engine\trainer: task=detect, mode=train, model=yolov8n.pt, data=datasets\PlateData\data.yaml, epochs=10
Overriding model.yaml nc=80 with nc=1

from n params module arguments
0 -1 1 464 ultralytics.nn.modules.conv.Conv [3, 16, 3, 2]
1 -1 1 4672 ultralytics.nn.modules.conv.Conv [16, 32, 3, 2]
2 -1 1 7360 ultralytics.nn.modules.block.C2f [32, 32, 1, True]
3 -1 1 18560 ultralytics.nn.modules.conv.Conv [32, 64, 3, 2]
4 -1 2 49664 ultralytics.nn.modules.block.C2f [64, 64, 2, True]
5 -1 1 73984 ultralytics.nn.modules.conv.Conv [64, 128, 3, 2]
6 -1 2 197632 ultralytics.nn.modules.block.C2f [128, 128, 2, True]
7 -1 1 295424 ultralytics.nn.modules.conv.Conv [128, 256, 3, 2]
8 -1 1 460288 ultralytics.nn.modules.block.C2f [256, 256, 1, True]
9 -1 1 164608 ultralytics.nn.modules.block.SPPF [256, 256, 5]
10 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
11 [-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
12 -1 1 148224 ultralytics.nn.modules.block.C2f [384, 128, 1]
13 -1 1 0 torch.nn.modules.upsampling.Upsample [None, 2, 'nearest']
14 [-1, 4] 1 0 ultralytics.nn.modules.conv.Concat [1]
15 -1 1 37248 ultralytics.nn.modules.block.C2f [192, 64, 1]
16 -1 1 36992 ultralytics.nn.modules.conv.Conv [64, 64, 3, 2]
17 [-1, 12] 1 0 ultralytics.nn.modules.conv.Concat [1]
18 -1 1 123648 ultralytics.nn.modules.block.C2f [192, 128, 1]
19 -1 1 147712 ultralytics.nn.modules.conv.Conv [128, 128, 3, 2]
20 [-1, 9] 1 0 ultralytics.nn.modules.conv.Concat [1]
21 -1 1 493056 ultralytics.nn.modules.block.C2f [384, 256, 1]
22 [15, 18, 21] 1 751507 ultralytics.nn.modules.head.Detect [1, [64, 128, 256]]

Model summary: 225 layers, 3011043 parameters, 3011027 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights
Freezing layer 'model.22.dfl.conv.weight'
train: Scanning D:\PycharmProjects\Yolo8PaddleOcrProject\datasets\PlateData\labels\train.cache... 56 images
val: Scanning D:\PycharmProjects\Yolo8PaddleOcrProject\datasets\PlateData\labels\val.cache... 22 images
Plotting labels to runs\detect\train\labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr=0.01' and 'momentum=0.937' and determining best 'optimizer'
0%| 0/28 [00:00<?, ?it/s]optimizer: AdamW(lr=0.002, momentum=0.9) with parameter groups 5
Image sizes 640 train, 640 val
Using 0 dataloader workers
Logging results to runs\detect\train
Starting training for 10 epochs...
Closing dataloader mosaic

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
1/10 0G 1.053 3.946 1.091 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 0.00333 1 0.895 0.554
0%| 0/28 [00:00<?, ?it/s]
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
2/10 0G 1.026 2.998 1.012 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 1 0.73 0.984 0.614
0%| 0/28 [00:00<?, ?it/s]
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
3/10 0G 1.112 2.45 1.091 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 0.00683 1 0.741 0.551
0%| 0/28 [00:00<?, ?it/s]
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
4/10 0G 0.9407 2.802 1.013 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 1 0.947 0.99 0.786
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
5/10 0G 1.014 2.296 1.045 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 1 0.947 0.99 0.786
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
6/10 0G 0.8976 1.99 0.9988 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 0.991 0.955 0.993 0.769
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
7/10 0G 0.8535 1.918 0.9479 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 0.995 1 0.995 0.813
0%| 0/28 [00:00<?, ?it/s]
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
8/10 0G 0.8593 1.88 0.9607 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 0.992 1 0.995 0.841
0%| 0/28 [00:00<?, ?it/s]
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
9/10 0G 0.7592 1.767 0.9161 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 0.994 1 0.995 0.844
0%| 0/28 [00:00<?, ?it/s]
Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
10/10 0G 0.7837 1.799 0.8993 2 640: 100%| 28/28
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 0.995 1 0.995 0.85

10 epochs completed in 0.167 hours.
Optimizer stripped from runs\detect\train\weights\last.pt, 6.2MB
Optimizer stripped from runs\detect\train\weights\best.pt, 6.2MB

Validating runs\detect\train\weights\best.pt...
Ultraalytics YOLOv8.2.28 Python-3.8.18 torch-2.3.0+cpu CPU (13th Gen Intel Core(TM) i7-1355U)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
Class Images Instances Box(P R mAP50 mAP50-95): 100%|
all 22 22 0.995 1 0.995 0.85
Speed: 2.9ms preprocess, 278.5ms inference, 0.0ms loss, 4.6ms postprocess per image
Results saved to runs\detect\train
Ultraalytics YOLOv8.2.28 Python-3.8.18 torch-2.3.0+cpu CPU (13th Gen Intel Core(TM) i7-1355U)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs

PyTorch: starting from 'runs\detect\train\weights\best.pt' with input shape (1, 3, 640, 640) BCHW and
onnx: starting export with onnx 1.16.1 opset 17...
onnx: export success 1.3s, saved as 'runs\detect\train\weights\best.onnx' (11.7 MB)

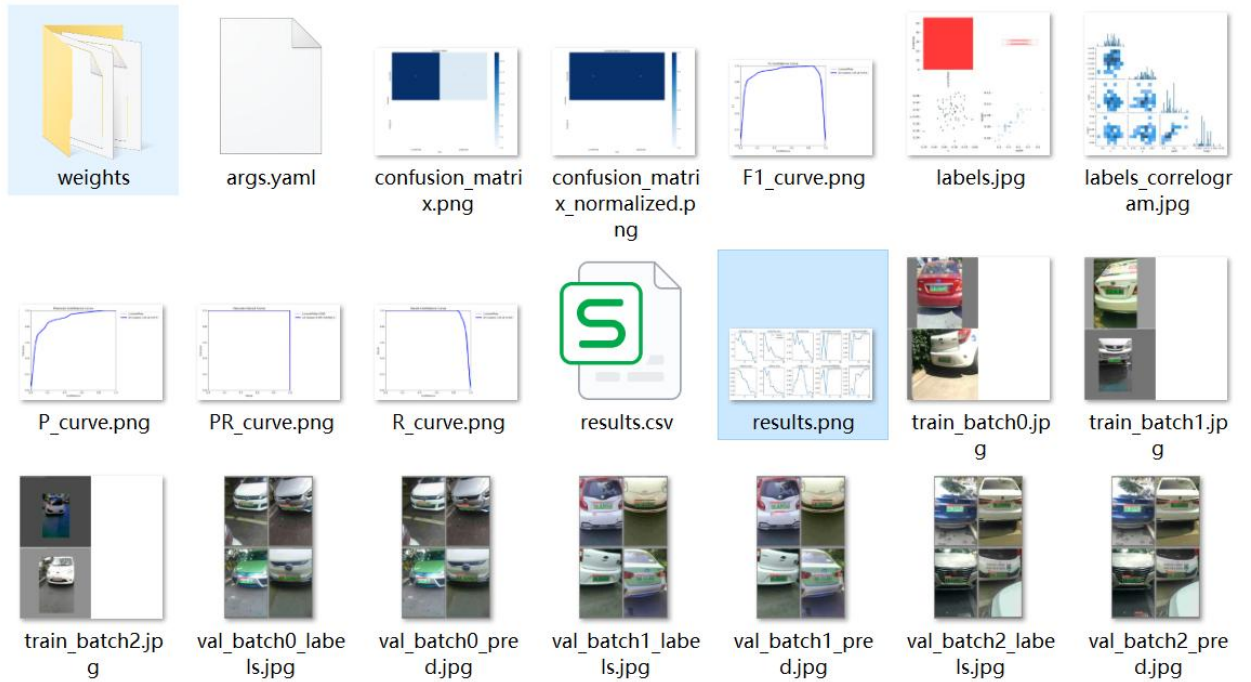
Export complete (4.0s)
Results saved to D:\PycharmProjects\Yolo8PaddleOcrProject\runs\detect\train\weights
Predict: yolo predict task=detect model=runs\detect\train\weights\best.onnx imgsz=640
Validate: yolo val task=detect model=runs\detect\train\weights\best.onnx imgsz=640 data=dataset1
Visualize: https://netron.app

Process finished with exit code 0
```



### 1.1.5 任务 11-车牌检测-训练结果评估

在深度学习中，我们通常用损失函数下降的曲线来观察模型训练的情况。YOLOv8 在训练时主要包含三个方面的损失：定位损失(**box\_loss**)、分类损失(**cls\_loss**)和动态特征损失(**dfl\_loss**)，在训练结束后，可以在 `runs/detect/train` 目录下找到训练过程及结果文件，如下所示：

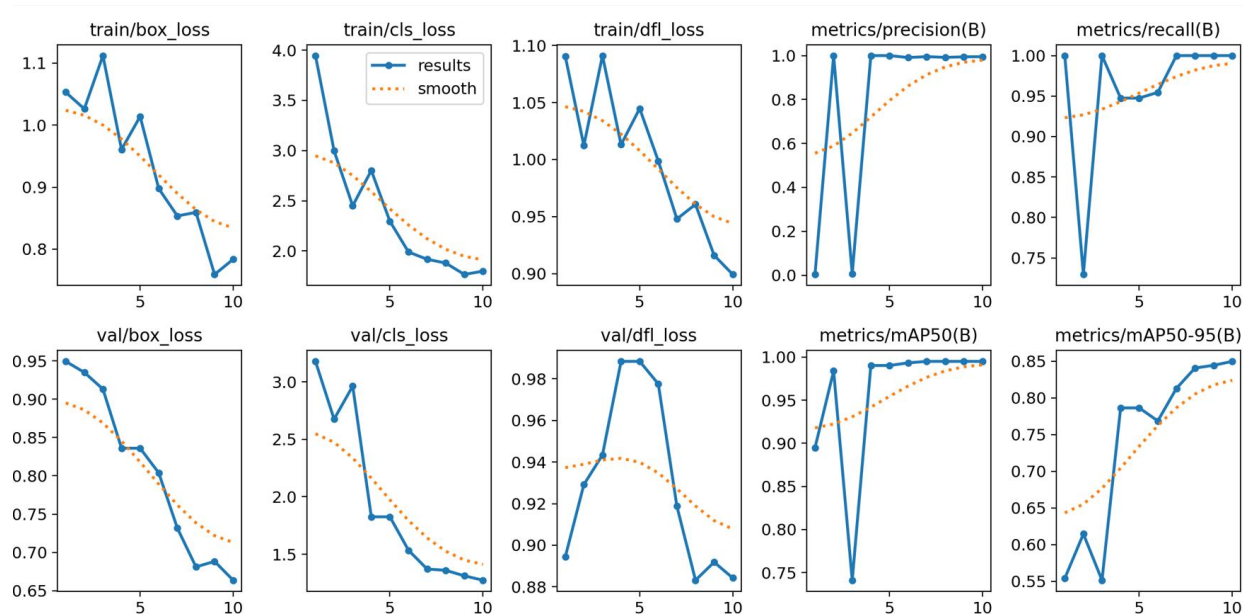


定位损失 **box\_loss**: 预测框与标定框之间的误差 (**GIoU**)，越小定位得越准；

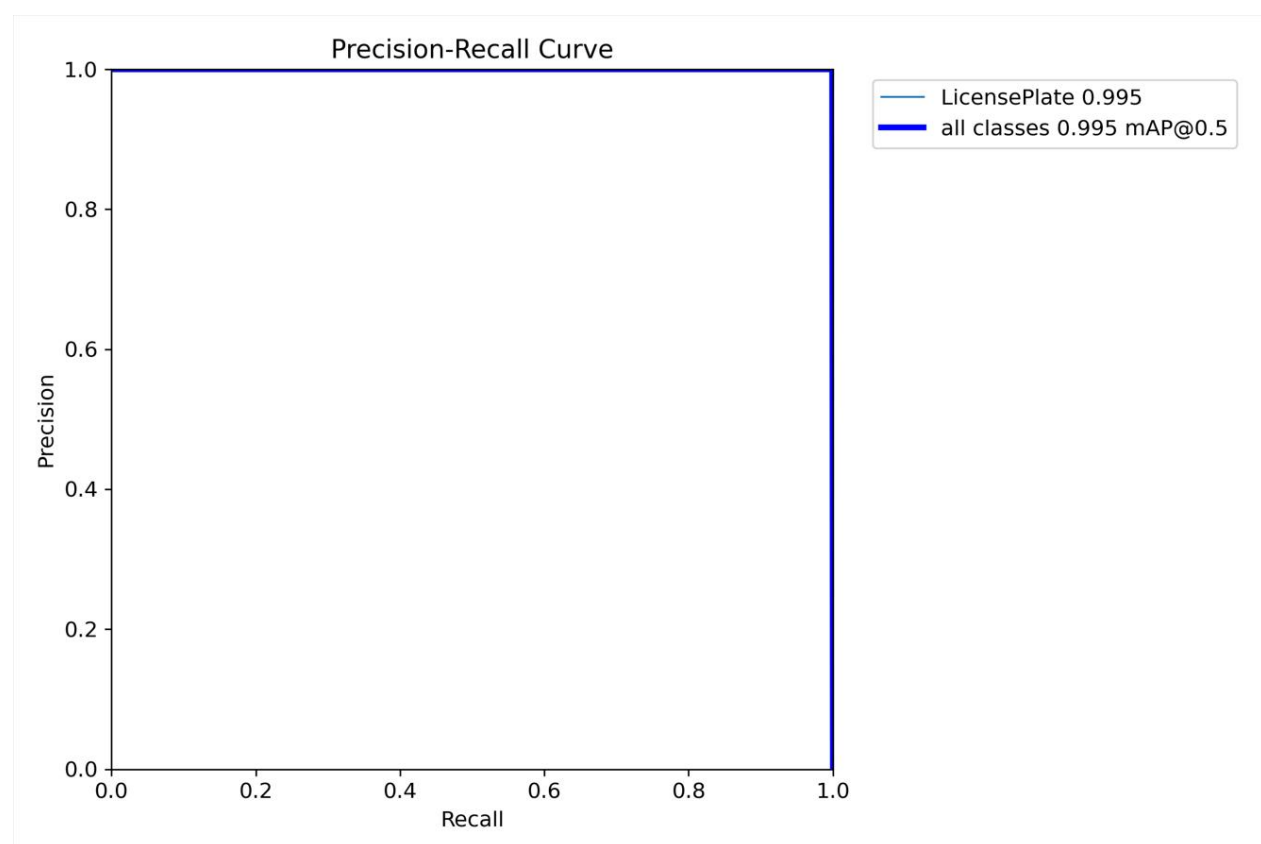
分类损失 **cls\_loss**: 计算锚框与对应的标定分类是否正确，越小分类得越准；

动态特征损失 (**dfl\_loss**): **DFLLoss** 是一种用于回归预测框与目标框之间距离的损失函数。

在计算损失时，目标框需要缩放到特征图尺度，即除以相应的 **stride**，并与预测的边界框计算 **Ciou Loss**，同时与预测的 **anchors** 中心点到各边的距离计算回归 **DFLLoss**。这个过程是 YOLOv8 训练流程中的一部分，通过计算 **DFLLoss** 可以更准确地调整预测框的位置，提高目标检测的准确性。 本文训练结果如下：



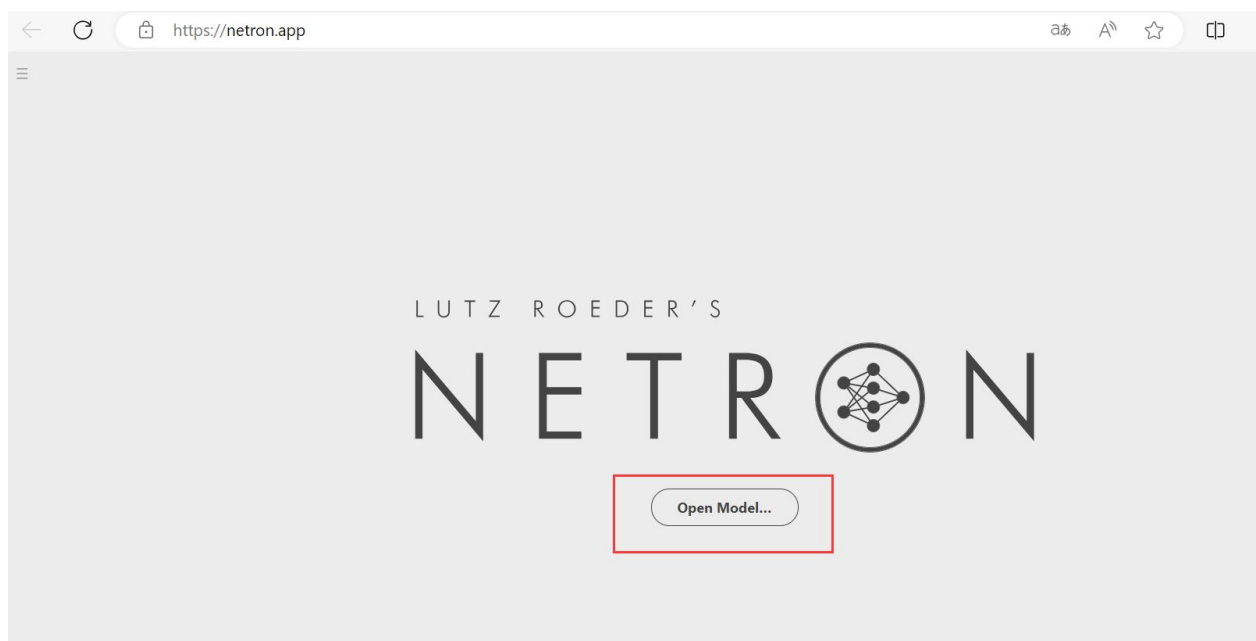
我们通常用 PR 曲线来体现精确率和召回率的关系，本文训练结果的 PR 曲线如下。mAP 表示 Precision 和 Recall 作为两轴作图后围成的面积，m 表示平均，@后面的数表示判定 iou 为正负样本的阈值。mAP@.5: 表示阈值大于 0.5 的平均 mAP，可以看到本模型目标检测的 mAP@0.5 平均值为 0.995，结果相当不错。



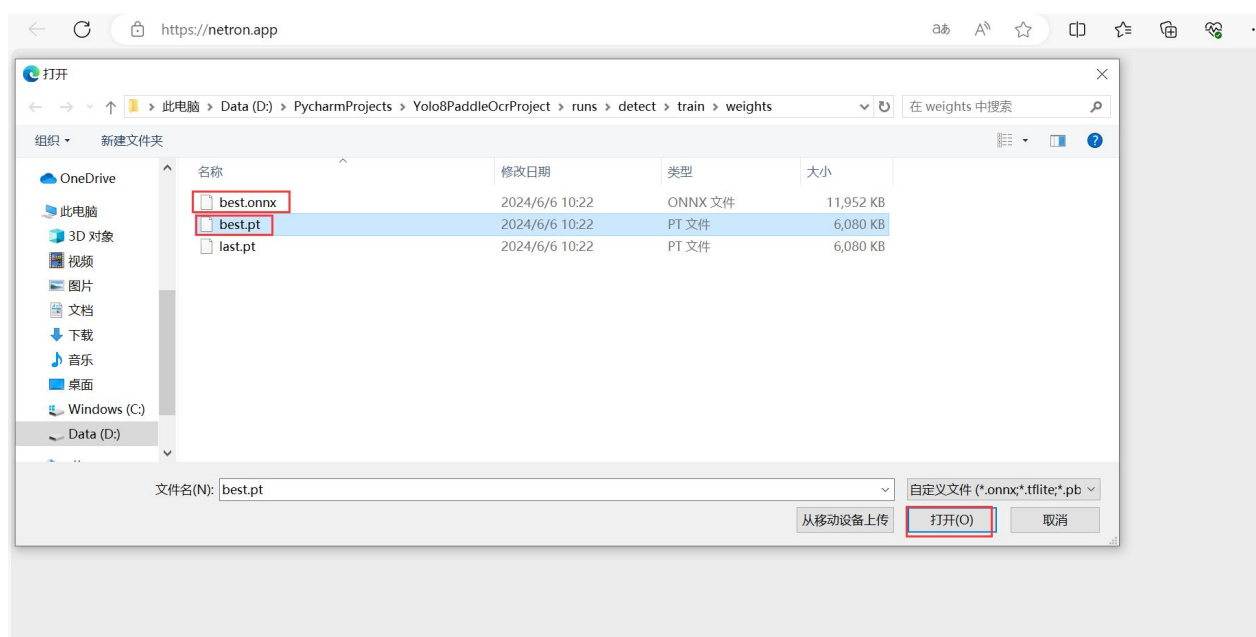
如果想可视化训练好的模型，可以打开以下网址：

<https://netron.app/>

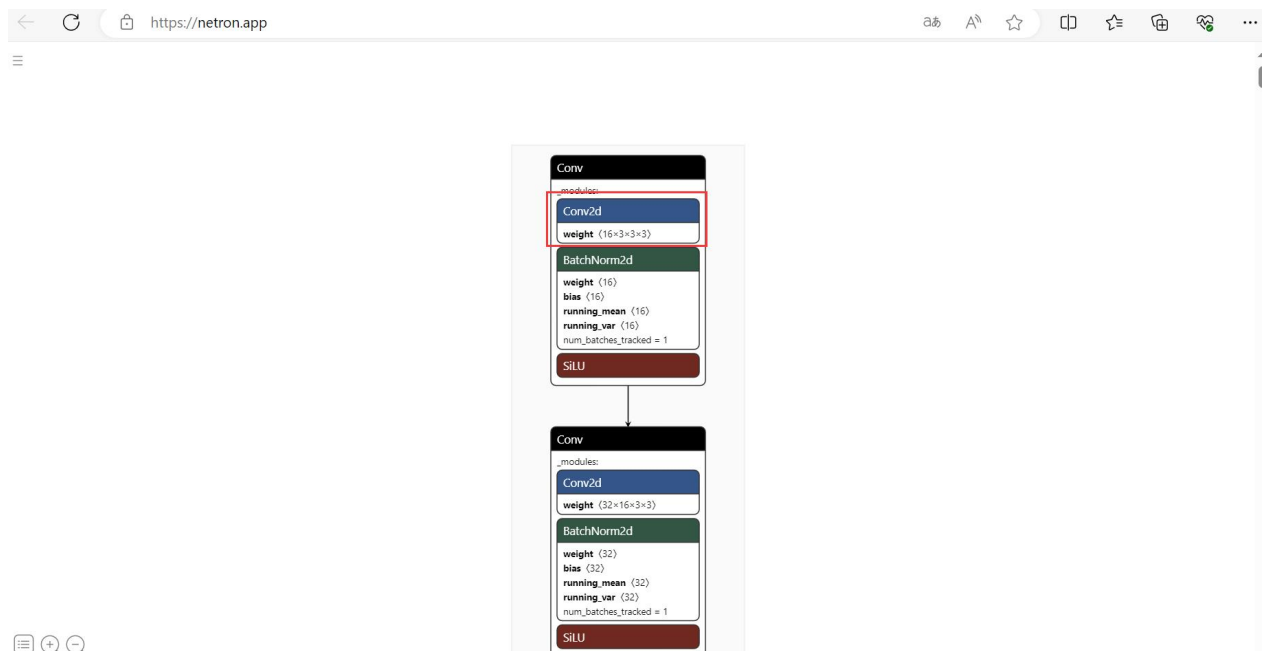
界面如下：



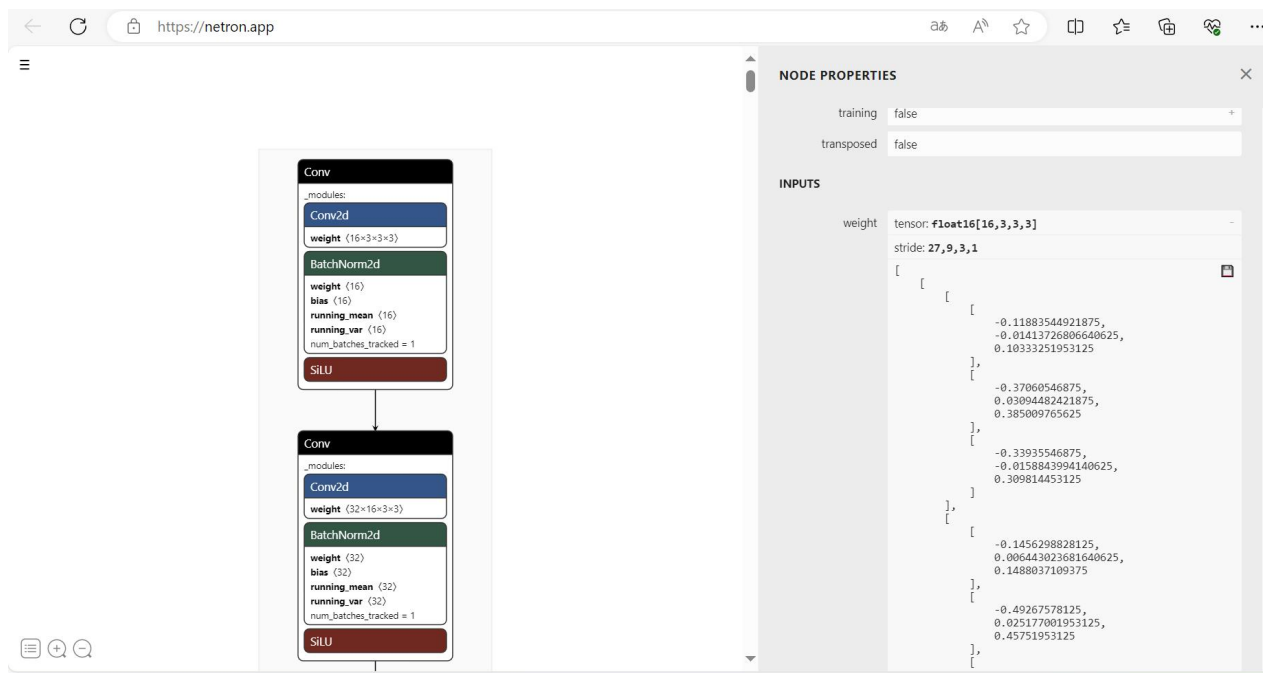
点击“Open Model...”，选择训练好的模型，也可以是转换的 onnx 格式的，然后点击“打开”：



加载模型后，界面如下：



可以点击对应神经网络结果，查看相关信息。



### 1.1.6 任务 11-车牌检测-检测车牌（图片）

模型训练完成后，我们可以得到一个最佳的训练结果模型 **best.pt** 文件，在 **runs/trian/weights** 目录下。我们可以使用该文件进行后续的推理检测。图片检测代码如下：

```
# coding:utf-8

from ultralytics import YOLO
```

```
import cv2

# 所需加载的模型目录
path = 'runs/detect/train/weights/best.pt'

# 需要检测的图片地址
img_path = "datasets/PlateData/images/test/038125-90_91-182&469_548&574-548&574_193&572_182&476_547&469-0_0_3_24_32_32_33_30-95-94.jpg"

# 加载预训练模型
model = YOLO(path, task='detect')

# 检测图片
results = model(img_path)
res = results[0].plot()
cv2.imshow("YOLOv8 PlateDetection", res)
cv2.waitKey(0)
```

执行上述代码后，会将执行的结果直接标注在图片上，结果如下：



可以发现,该模型能够很好的检测出车牌区域。下面我们需要对检测出的车牌进行识别。

### 1.1.7 任务 12-车牌识别-PaddleOCR 介绍

车牌识别直接使用开源的 PaddleOCR 检测模型。地址如下:

<https://github.com/PaddlePaddle/PaddleOCR>

PaddleOCR 旨在打造一套丰富、领先、且实用的 OCR 工具库,助力开发者训练出更好的模型,并应用落地。

PaddleOCR 由 PMC 监督。Issues 和 PRs 将在尽力的基础上进行审查。欲了解 PaddlePaddle 社区的完整概况,请访问 community。

注意: Issues 模块仅用来报告程序 Bug,其余提问请移步 Discussions 模块提问。如所提 Issue 不是 Bug,会被移到 Discussions 模块。

支持多种 OCR 相关前沿算法,在此基础上打造产业级特色模型 PP-OCR、PP-Structure 和 PP-ChatOCRv2,并打通数据生产、模型训练、压缩、预测部署全流程。



PP-OCR 系列模型列表：

模型简介	模型名称	推荐场景	检测模型	方向分类器	识别模型
中英文超轻量 PP-OCRv4 模型 (15.8M)	ch_PP-OCRv4_xx	移动端&服务器端	<a href="#">推理模型</a> / <a href="#">训练模型</a>	<a href="#">推理模型</a> / <a href="#">训练模型</a>	<a href="#">推理模型</a> / <a href="#">训练模型</a>
中英文超轻量 PP-OCRv3 模型 (16.2M)	ch_PP-OCRv3_xx	移动端&服务器端	<a href="#">推理模型</a> / <a href="#">训练模型</a>	<a href="#">推理模型</a> / <a href="#">训练模型</a>	<a href="#">推理模型</a> / <a href="#">训练模型</a>
英文超轻量 PP-OCRv3 模型 (13.4M)	en_PP-OCRv3_xx	移动端&服务器端	<a href="#">推理模型</a> / <a href="#">训练模型</a>	<a href="#">推理模型</a> / <a href="#">训练模型</a>	<a href="#">推理模型</a> / <a href="#">训练模型</a>

1.1.8 任务 12-车牌识别-PaddleOCR 环境配置

配置环境命令如下：

```
pip install paddlepaddle paddleocr shapely
```

1.1.9 任务 12-车牌识别-PaddleOCR 模型使用

我们可以从测试集中，手动截图车牌，测试 PaddleOCR 模型使用。



使用代码如下：

```
from paddleocr import PaddleOCR, draw_ocr

# Paddleocr 目前支持的多语言语种可以通过修改 lang 参数进行切换
# 例如 `ch`, `en`, `fr`, `german`, `korean`, `japan`

cls_model_dir = 'PaddleOCR/cls_infer'
rec_model_dir = 'PaddleOCR/rec_infer'

ocr = PaddleOCR(use_angle_cls=True, lang="ch", det=False,
cls_model_dir=cls_model_dir,
rec_model_dir=rec_model_dir) # need to run only once to
download and load model into memory

img_path = 'plate.png'
result = ocr.ocr(img_path, cls=True)
```

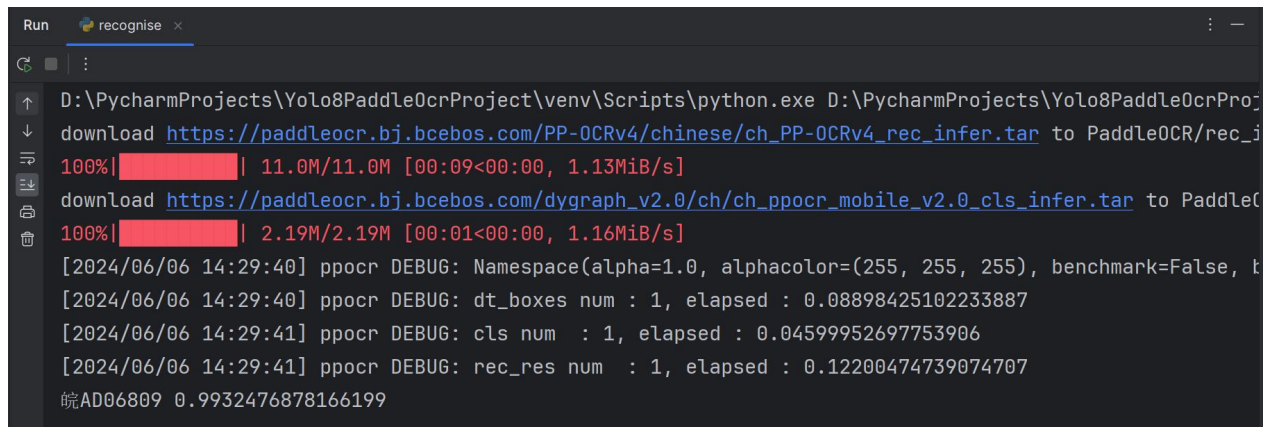
```
license_name, conf = result[0][0][1]

if '.' in license_name:

    license_name = license_name.replace('.', '')

print(license_name, conf)
```

运行结果如下：



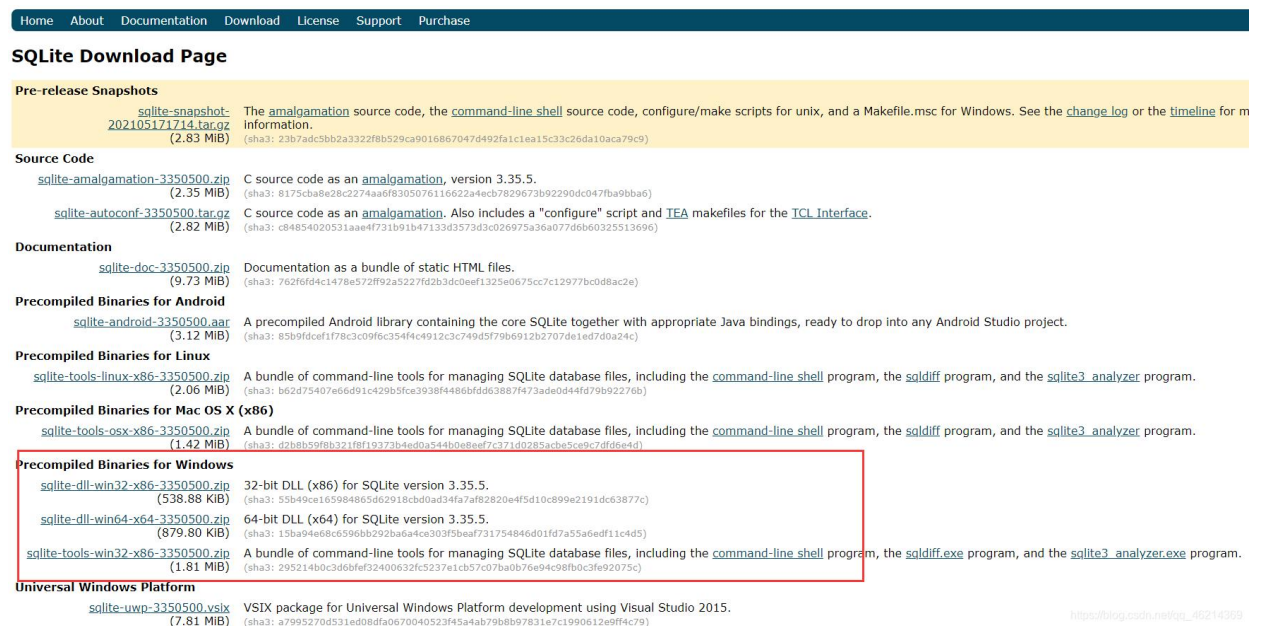
```
Run recognise x
D:\PycharmProjects\Yolo8Paddle0crProject\venv\Scripts\python.exe D:\PycharmProjects\Yolo8Paddle0crProject\venv\Scripts\python.exe
download https://paddleocr.bj.bcebos.com/PP-0CRv4/chinese/ch_PP-0CRv4_rec_infer.tar to PaddleOCR/rec_infer.tar
100% |██████████| 11.0M/11.0M [00:09<00:00, 1.13MiB/s]
download https://paddleocr.bj.bcebos.com/dygraph_v2.0/ch/ch_ppocr_mobile_v2.0_cls_infer.tar to PaddleOCR/cls_infer.tar
100% |██████████| 2.19M/2.19M [00:01<00:00, 1.16MiB/s]
[2024/06/06 14:29:40] ppocr DEBUG: Namespace(alpha=1.0, alphacolor=(255, 255, 255), benchmark=False, t
[2024/06/06 14:29:40] ppocr DEBUG: dt_boxes num : 1, elapsed : 0.08898425102233887
[2024/06/06 14:29:41] ppocr DEBUG: cls num : 1, elapsed : 0.04599952697753906
[2024/06/06 14:29:41] ppocr DEBUG: rec_res num : 1, elapsed : 0.12200474739074707
皖AD06809 0.9932476878166199
```

如果出现以下问题：

ImportError: DLL load failed while importing \_sqlite3: 找不到指定的模块

去 SQLite 官网去下载缺失的 DDL 文件，网址：

<https://www.sqlite.org/download.html>



Home About Documentation Download License Support Purchase

### SQLite Download Page

**Pre-release Snapshots**

[sqlite-snapshot-202105171714.tar.gz](#) (2.83 MiB) The [amalgamation](#) source code, the [command-line shell](#) source code, configure/make scripts for unix, and a Makefile.msc for Windows. See the [change log](#) or the [timeline](#) for more information. (sha3: 23b7adc5bb2a3322f8b529ca9016867047d492fa1c1ea15c33c26da10aca79c9)

**Source Code**

[sqlite-amalgamation-3350500.zip](#) (2.35 MiB) C source code as an [amalgamation](#), version 3.35.5. (sha3: 8175cba8e28c2274aa6f8305076116622a4ecb7829673b92290dc047fba9bba6)

[sqlite-autoconf-3350500.tar.gz](#) (2.82 MiB) C source code as an [amalgamation](#). Also includes a "configure" script and [TEA](#) makefiles for the [TCL](#) Interface. (sha3: c84854020531aae4f731b91b4713d3573d3c026975a36a077d6b60325513696)

**Documentation**

[sqlite-doc-3350500.zip](#) (9.73 MiB) Documentation as a bundle of static HTML files. (sha3: 762f6f04c1478e572f92a5227f02b3dc0eef1325e0675cc7c12977bc0d8ac2e)

**Precompiled Binaries for Android**

[sqlite-android-3350500.aar](#) (3.12 MiB) A precompiled Android library containing the core SQLite together with appropriate Java bindings, ready to drop into any Android Studio project. (sha3: 85b9f0cef1f783c09f6c354f4c4912c3c749d5f79b6912b2707de1ed7d0a24c)

**Precompiled Binaries for Linux**

[sqlite-tools-linux-x86-3350500.zip](#) (2.06 MiB) A bundle of command-line tools for managing SQLite database files, including the [command-line shell](#) program, the [sqlite3](#) program, and the [sqlite3\\_analyzer](#) program. (sha3: b62d75407e86d91c429b5fca3938f4486b6dd638879473ade0d44fd79b92276b)

**Precompiled Binaries for Mac OS X (x86)**

[sqlite-tools-osx-x86-3350500.zip](#) (1.42 MiB) A bundle of command-line tools for managing SQLite database files, including the [command-line shell](#) program, the [sqlite3](#) program, and the [sqlite3\\_analyzer](#) program. (sha3: d2b8b59f8b321f8f19373b4ed0a544b0e8eef7c371d0285acba5ce9c7dfde4d)

**Precompiled Binaries for Windows**

[sqlite-dll-win32-x86-3350500.zip](#) (538.88 KiB) 32-bit DLL (x86) for SQLite version 3.35.5. (sha3: 55b49ce165984865d62918cbd0ad34fa7af82820e4f5d10c899e2191dc63877c)

[sqlite-dll-win64-x64-3350500.zip](#) (879.80 KiB) 64-bit DLL (x64) for SQLite version 3.35.5. (sha3: 15ba94e08c6596bb292ba6a4ce303f5beaf731754846d01fd7a55a6edf11c4d5)

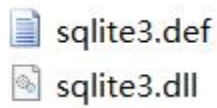
[sqlite-tools-win32-x86-3350500.zip](#) (1.81 MiB) A bundle of command-line tools for managing SQLite database files, including the [command-line shell](#) program, the [sqlite3.exe](#) program, and the [sqlite3\\_analyzer.exe](#) program. (sha3: 295214b0c3d0b6f52400632c5237e1cb57cd7ba0b76e94c98f0c3fe92075c)

**Universal Windows Platform**

[sqlite-uwp-3350500.vsix](#) (7.81 MiB) VSIX package for Universal Windows Platform development using Visual Studio 2015. (sha3: a7995270d531ed08dfa670040523f45a4ab79b8b97831e7c1990612e9ff4c79) <https://aka.ms/sqlite-net/02214399>

根据系统对应的版本下载相应 DDL 文件。

解压之后得到如下两个文件：



直接将 sqlite3.def, sqlite3.dll 解压至所用 Python 环境的 DDLs 目录下即可。

### 1.1.10 任务 12-车牌识别-YOLOv8+PaddleOCR

完整代码如下：

```
# coding:utf-8

import os

os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

from ultralytics import YOLO
import cv2
import utils
from PIL import ImageFont
from paddleocr import PaddleOCR

def get_license_result(ocr, image):
    """
    image:输入的车牌截取照片
    输出，车牌号与置信度
    """
    result = ocr.ocr(image, cls=True)[0]
    if result:
        license_name, conf = result[0][1]
        if '.' in license_name:
            license_name = license_name.replace('.', '')
```

```

        return license_name, conf

    else:

        return None, None

# 需要检测的图片地址

img_path = "datasets/PlateData/images/test/0171875-90_264-274&421_514&493-514&493_280&490_274&421_506&421-0_0_3_24_30_32_32_29-145-34.jpg"

now_img = utils.img_cvread(img_path)

fontC = ImageFont.truetype("platech.ttf", 50, 0)

# 加载 ocr 模型

cls_model_dir = 'PaddleOCR/cls_infer'

rec_model_dir = 'PaddleOCR/rec_infer'

ocr = PaddleOCR(use_angle_cls=False, lang="ch", det=False,
cls_model_dir=cls_model_dir, rec_model_dir=rec_model_dir)

# 所需加载的模型目录

path = 'runs/detect/train/weights/best.pt'

# 加载预训练模型

model = YOLO(path, task='detect')

# 检测图片

results = model(img_path)[0]

location_list = results.bboxes.xyxy.tolist()

if len(location_list) >= 1:

    location_list = [list(map(int, e)) for e in location_list]

    # 截取每个车牌区域的照片

    license_imgs = []

```

```

for each in location_list:
    x1, y1, x2, y2 = each
    cropImg = now_img[y1:y2, x1:x2]
    license_imgs.append(cropImg)
    cv2.imshow('Detection', cropImg)
    cv2.waitKey(0)

# 车牌识别结果
lisence_res = []
conf_list = []

for each in license_imgs:
    license_num, conf = get_license_result(ocr, each)
    if license_num:
        lisence_res.append(license_num)
        conf_list.append(conf)
    else:
        lisence_res.append('无法识别')
        conf_list.append(0)

for text, box in zip(lisence_res, location_list):
    now_img = utils.drawRectBox(now_img, box, text, fontC)

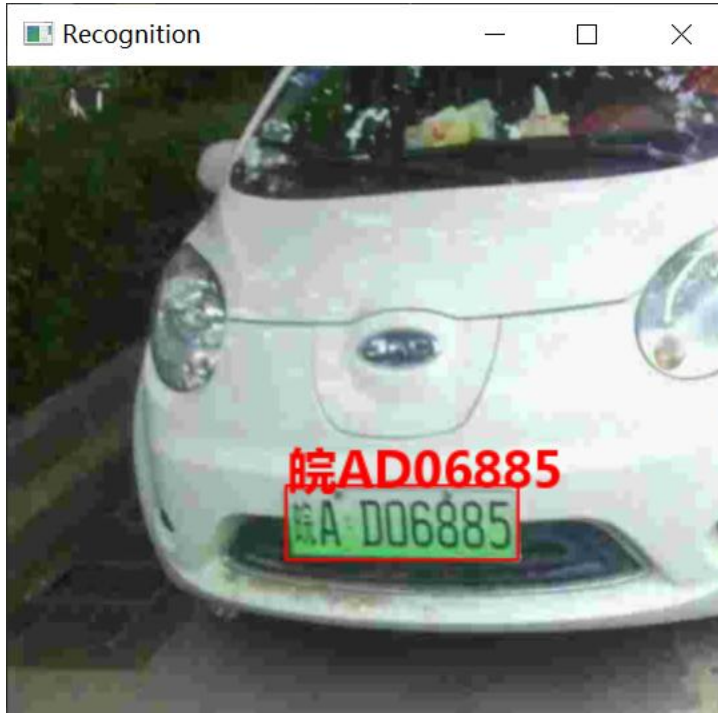
now_img = cv2.resize(now_img, dsize=None, fx=0.5, fy=0.5,
interpolation=cv2.INTER_LINEAR)

cv2.imshow("Recognition", now_img)
cv2.waitKey(0)

```

运行结果如下：





如果遇到如下问题:

OMP: Error #15: Initializing libiomp5md.dll, but found libiomp5md.dll already initialized.

OMP: Hint This means that multiple copies of the OpenMP runtime have been linked into the program. That is dangerous, since it can degrade performance or cause incorrect results. The best thing to do is to ensure that only a single OpenMP runtime is linked into the process, e.g. by avoiding static linking of the OpenMP runtime in any library. As an unsafe, unsupported, undocumented workaround you can set the environment variable KMP\_DUPLICATE\_LIB\_OK=TRUE to allow the program to continue to execute, but that may cause crashes or silently produce incorrect results. For more information, please see <http://www.intel.com/software/products/support/>.

究其原因其实是, Python 的环境下存在两个 libiomp5md.dll 文件。所以直接去虚拟环境的路径下搜索这个文件, 可以看到在环境里有两个 dll 文件。请在以上代码前面, 加入如下代码:

```
import os
```

```
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
```

### 1.1.11 任务 13-视频车牌识别

完整代码如下:



```
# coding:utf-8

# coding:utf-8

import os

os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

import cv2
from ultralytics import YOLO
import utils
from PIL import ImageFont
from paddleocr import PaddleOCR

def get_license_result(ocr, image):
    """
    image:输入的车牌截取照片
    输出, 车牌号与置信度
    """
    result = ocr.ocr(image, cls=True)[0]
    if result:
        license_name, conf = result[0][1]
        if '.' in license_name:
            license_name = license_name.replace('.', '')
        return license_name, conf
    else:
        return None, None

fontC = ImageFont.truetype("platech.ttf", 50, 0)
```

```
# 加载 ocr 模型

cls_model_dir = 'PaddleOCR/cls_infer'

rec_model_dir = 'PaddleOCR/rec_infer'

ocr = PaddleOCR(use_angle_cls=False, lang="ch", det=False,
cls_model_dir=cls_model_dir, rec_model_dir=rec_model_dir)


# 所需加载的模型目录

path = 'runs/detect/train/weights/best.pt'

# 加载预训练模型

# conf 0.25 object confidence threshold for detection

# iou 0.7 intersection over union (IoU) threshold for NMS

model = YOLO(path, task='detect')


# 需要检测的图片地址

video_path = "plate.mp4"


cap = cv2.VideoCapture(video_path)

# Loop through the video frames

while cap.isOpened():

    # Read a frame from the video

    success, frame = cap.read()


    if success:

        # Run YOLOv8 inference on the frame

        results = model(frame)[0]


        location_list = results.boxes.xyxy.tolist()

        if len(location_list) >= 1:

            location_list = [list(map(int, e)) for e in location_list]

            # 截取每个车牌区域的照片
```

```

        license_imgs = []
        for each in location_list:
            x1, y1, x2, y2 = each
            cropImg = frame[y1:y2, x1:x2]
            license_imgs.append(cropImg)
        # 车牌识别结果
        lisence_res = []
        conf_list = []
        for each in license_imgs:
            license_num, conf = get_license_result(ocr, each)
            if license_num:
                lisence_res.append(license_num)
                conf_list.append(conf)
            else:
                lisence_res.append('无法识别')
                conf_list.append(0)
        for text, box in zip(lisence_res, location_list):
            frame = utils.drawRectBox(frame, box, text, fontC)

    frame = cv2.resize(frame, dsize=None, fx=0.5, fy=0.5,
interpolation=cv2.INTER_LINEAR)
    cv2.imshow("PlateRecognition", frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
else:
    # Break the loop if the end of the video is reached
    break

```

```
# Release the video capture object and close the display window
```

```
cap.release()
```

```
cv2.destroyAllWindows()
```

运行效果如下：

