序号：83

人工智能程序设计

课程设计报告

题　　目：目标检测模型训练

院　　系：未来书院

班　　级：智创 2402

学生姓名：王站艺

学　　号：20241023039

完成日期：2024 年 12 月 19 日

大连理工大学未来技术学院

一、问题描述

1. 数据来源：

静态图像，8000张，640x480尺寸

2. 目标：

检测小车顶部装置

二、设计思路

1. 模型：

(1)应用yolov8n模型实现

2. 模块：

(1) os模块。允许我与系统交互，数据集划分等操作

(2) Yolo。视觉人工智能，让我快速训练出所需模型。

3. 主要算法：

(1) IOU损失函数：衡量预测的边界与真实边界框之间的重叠程度

(2) 数据增强：

① 数据集只有8000多，采用数据增强，扩大数据集规模

② 增加数据集多样性，有利于提高模型泛化能力

③ 防止模型过于依赖某一项特征，减少过拟合风险

④ 技术：

1)几何变换，颜色变换，增加噪声，裁切缩放，混合技术，透视变换等

(3) 骨干网络：

①借鉴了yolov7，将c3模块　更新了c2f模块，虽然卷积模块变为两个，但是速度更快。没有改变原始架构和梯度传输，使用分组卷积、洗牌和合并的操作组合不同特征，增强模型的学习能力。

(4) DFL函数：

①原因：训练过程中目标边界框不应该是一个确定值，而应该是一个分布，通过它可以减少过拟合现象

②

$$\mathbf{DFL}(\mathcal{S}_i, \mathcal{S}_{i+1}) = -\big((y_{i+1} - y)\log(\mathcal{S}_i) + (y - y_i)\log(\mathcal{S}_{i+1})\big).$$

(5) TaskAlignedAssigner匹配策略：

①根据分类与回归的分数加权的分数选择正样本

② $t = s^{\alpha} + u^{\beta}$，其中 s 是标注类别对应的预测分数值，u 是预测框和 gt 框之间的 iou。计算之后根据分数选择 topK 大的作为正样本，其余为负样本

4. 流程：

(1)配置环境

① 如图所示，python==3.11

Package	Version
Brotli	1.0.9
certifi	2024.12.14
charset-normalizer	3.3.2
colorama	0.4.6
contourpy	1.3.1
cycler	0.12.1
filelock	3.13.1
fonttools	4.55.3
fsspec	2024.10.0
gmpy2	2.1.2
idna	3.7
Jinja2	3.1.4
kiwisolver	1.4.7
MarkupSafe	2.1.3
matplotlib	3.10.0
mkl_fft	1.3.11
mkl_random	1.2.8
mkl-service	2.4.0
mpmath	1.3.0
networkx	3.2.1
numpy	2.0.1
opencv-python	4.10.0.84
packaging	24.2
pandas	2.2.3
pillow	11.0.0
pip	24.2
psutil	6.1.0
py-cpuinfo	9.0.0
pyparsing	3.2.0
PySocks	1.7.1
python-dateutil	2.9.0.post0
pytz	2024.2
PyYAML	6.0.2
requests	2.32.3
scipy	1.14.1
seaborn	0.13.2
setuptools	75.1.0
six	1.17.0
sympy	1.13.1
torch	2.5.1
torchaudio	2.5.1
torchvision	0.20.1
tqdm	4.67.1
typing_extensions	4.11.0
tzdata	2024.2
ultralytics	8.3.51
ultralytics-thop	2.0.13
urllib3	2.2.3
wheel	0.44.0
win-inet-pton	1.1.0

② 下载官方通用yolov8n模型

③ 划分数据集

④ 训练：

1）第一周期。每10epochs，保存并val模型性能，参数设置如下，进行10个伦次共100epochs，先用较小的bach值确保模型稳定

epochs=10, imgsz=640, device=[0,], workers=0, batch=4, cache=True

2）第二周期。调高bach值，加速训练速度，并自定义数据增强，配置如下，以10epochs为一轮，训练6个周期，结束后进行性能评估：

epochs=10, imgsz=640, device=[0,], workers=0, batch=32, cache=True

```
augment:
  flipud: 0.8     # 50% 概率进行垂直翻转
  fliplr: 0.8     # 50% 概率进行水平翻转
  mosaic: 1.0     # 启用 Mosaic 数据增强
  mixup: 0.5      # 启用 Mixup 数据增强
  hsv_h: 0.02     # 色调增强，范围为 ±0.015
  hsv_s: 0.7      # 饱和度增强，范围为 ±0.7
  hsv_v: 0.5      # 亮度增强，范围为 ±0.4
  scale: 0.5      # 随机缩放，范围为 ±50%
  shear: 0.9    # 随机剪切
  perspective: 0.9 # 随机透视变换
```

三、关键代码示例

1. 训练代码：

```
import time
from ultralytics import YOLO
model = YOLO('E:/ultralytics-main/runs/detect/train15/weights/last.pt')
results = model.train(data='E:/ultralytics-main/A_my_data.yaml', epochs=10, imgsz=640, device=[0,], workers=0, batch=32, cache=True)
time.sleep(10)
```

2. val代码

```
from ultralytics import YOLO
model = YOLO('E:/ultralytics-main/runs/detect/train14/weights/best.pt')
validation_results = model.val(data='E:/ultralytics-main/B_my_data.yaml', imgsz=640, batch=4, conf=0.25, iou=0.6, device="0", workers=0)
```

3. 数据集配置代码

A：

```
path: E:/ultralytics-main/data
train:
  - images/train
val:
  - images/val
test:
  - images/test
names:
```

4. car1

```
1: car2
2: car3
3: car4
4: car5
5: car6
```

```
augment:
  flipud: 0.8     # 50% 概率进行垂直翻转
  fliplr: 0.8     # 50% 概率进行水平翻转
  mosaic: 1.0     # 启用 Mosaic 数据增强
  mixup: 0.5      # 启用 Mixup 数据增强
  hsv_h: 0.02     # 色调增强，范围为 ±0.015
  hsv_s: 0.7      # 饱和度增强，范围为 ±0.7
  hsv_v: 0.5      # 亮度增强，范围为 ±0.4
  scale: 0.5      # 随机缩放，范围为 ±50%
  shear: 0.9    # 随机剪切
  perspective: 0.9 # 随机透视变换
```

B

```
path: E:/ultralytics-main/data
train:
  - images/train
val:
  - images/val
test:
  - images/test
```

```
names:
```

5. car1

```
1: car2
2: car3
3: car4
4: car5
5: car6
```

查询类别代码

```
'''
```

```
确定有多少个类别，防止bug
'''

import os
def collect_distinct_labels(label_directory):
    """
    从指定的标签目录中收集所有.txt文件的第一个数字，并返回一个不重复的数字集合。
    参数:
    label_directory (str): 包含标签的目录路径。
    返回:
    set: 包含所有唯一数字的集合。
    """
    distinct_labels = set() # 使用set来存储唯一的标签
    # 遍历标签目录中的所有子目录和文件
    for directory_name in os.listdir(label_directory):
        directory_path = os.path.join(label_directory, directory_name)
        if os.path.isdir(directory_path): # 确保是目录
```

```
            for file_name in os.listdir(directory_path):
                if file_name.endswith('.txt'):  # 筛选.txt文件
                    file_full_path = os.path.join(directory_path,
file_name)
                    with open(file_full_path, 'r') as label_file:
                        for label_line in label_file:
                            line_parts = label_line.strip().split(',')
                            if line_parts and line_parts[0].isdigit():  # 检
查第一个元素是否为数字
                                distinct_labels.add(int(line_parts[0]))  #
添加到集合中
                                break  # 找到第一个数字后即跳出循环
    return distinct_labels
labels_path = "E:/dataset/labels"
all_labels = collect_distinct_labels(labels_path)
print("从.txt文件中提取的唯一标签:")
for label in sorted(all_labels):
    print(label)
```

数据划分代码

```
'''功能
    归一化并且将所有输出统一复制到一个文件夹下。
    便于随机划分，和保存源文件
'''
import os
def normalize_line(line, image_width, image_height):
    """
```

```
        归一化单行标签数据。
[redacted]
        参数:
        line (str): 从.txt文件中读取的一行数据。
        image_width (int): 图像的宽度。
        image_height (int): 图像的高度。
[redacted]
        返回:
        str: 归一化后的字符串，或者如果行格式不正确则返回None。
        """
        parts = line.strip().split(',')
        if len(parts) != 5: # 确保行数据包含5个部分：类别ID，x中心，y中心，宽度，高度
            return None
        class_id, x_center, y_center, w, h = map(float, parts) # 将字符串转换为浮点数
        # 归一化坐标和尺寸，类别ID保持为整数
        norm_x_center = x_center / image_width
        norm_y_center = y_center / image_height
        norm_w = w / image_width
        norm_h = h / image_height
        # 格式化归一化后的字符串，类别ID转换为整数
        return f"{int(class_id)} {norm_x_center:.6f} {norm_y_center:.6f} {norm_w:.6f} {norm_h:.6f}"
    def process(input_folder, output_folder, image_width, image_height):
        """
        处理文件夹中的所有.txt文件，归一化标签数据，并保存到输出文件夹。
```

```
    参数:
    input_folder (str): 包含原始.txt文件的输入文件夹路径。
    output_folder (str): 归一化文件保存的输出文件夹路径。
    image_width (int): 图像的宽度。
    image_height (int): 图像的高度。
    """
    if not os.path.exists(output_folder):
        os.makedirs(output_folder) # 如果输出文件夹不存在，则创建
    file_number = 1 # 初始化文件编号
    for root, dirs, files in os.walk(input_folder): # 遍历输入文件夹
        for file in files:
            if file.endswith('.txt'): # 筛选.txt文件
                file_path = os.path.join(root, file) # 获取文件完整路径
                with open(file_path, 'r') as file: # 打开文件读取
                    lines = file.readlines() # 读取所有行
                normalized_lines = [] # 初始化归一化行列表
                for line in lines:
                    normalized_line = normalize_line(line, image_width, image_height) # 归一化每行
                    if normalized_line: # 如果归一化成功
                        normalized_lines.append(normalized_line) # 添加到列表
                # 保存归一化后的数据到输出文件夹中的新文件
                output_file_name = f'file_{file_number}.txt' # 格式化文件名
```

```
                output_file_path = os.path.join(output_folder, output_file_name) # 获取输出文件路径
                with open(output_file_path, 'w') as file: # 打开文件写入
                    file.writelines('\n'.join(normalized_lines)) # 写入归一化行
                file_number += 1 # 增加文件编号
    # 设置输入输出文件夹路径和图像大小
    input_folder = "E:/dataset/labels"
    output_folder = "E:/datasetguiyi/labels"
    image_width = 640
    image_height = 480
    # 调用函数处理数据
    process(input_folder, output_folder, image_width, image_height)
    print("归一化完成，归一化后的文件已保存到输出文件夹。")
```

图片文件处理代码

```
    '''
```

```
    更改图片名，使其与输出文件名对应，
    并输出到另一文件夹，保存原文件，方便后续操作

    '''

    import os
    import shutil
    def duplicate_and_rename_images(source_directory, target_directory):
        """
```

```
    将源目录中的所有JPEG图片复制到目标目录，并按顺序重命名为'图片_N.jpg'，其中N是递增的整数。

    参数：
    source_directory (str)：包含原始JPEG文件的源目录路径。
    target_directory (str)：复制和重命名后的文件保存的目标目录路径。
    """
    # 如果目标目录不存在，则创建目标目录
    os.makedirs(target_directory, exist_ok=True)
    # 初始化文件计数器，用于生成新的文件名
    file_counter = 1
    # 遍历源目录中的每个子目录
    for directory_name in os.listdir(source_directory):
        directory_path = os.path.join(source_directory, directory_name)
        # 检查路径是否为目录
        if os.path.isdir(directory_path):
            # 在目录中查找所有以.jpg结尾的文件
            jpeg_files = [file for file in os.listdir(directory_path) if file.lower().endswith('.jpg')]
            for jpeg_file in jpeg_files:
                # 构造新的文件名，并带有递增的索引
                formatted_filename = f'图片_{file_counter}.jpg'
                destination_file_path = os.path.join(target_directory, formatted_filename)
                # 将JPEG文件从源目录复制到目标目录，并使用新名称
```

```
                shutil.copy(os.path.join(directory_path, jpeg_file),
destination_file_path)
                # 为下一个文件递增文件计数器
                file_counter += 1
    # 定义源目录和目标目录的路径
    source_dir = "E:/dataset/images"
    target_dir = "E:/datasetguiyi/images"
    # 执行函数复制和重命名图片
    duplicate_and_rename_images(source_dir, target_dir)
    print("图片复制和重命名完成；文件已保存在目标文件夹。")
```

四、效果展示

训练过程效果：

第一周期：

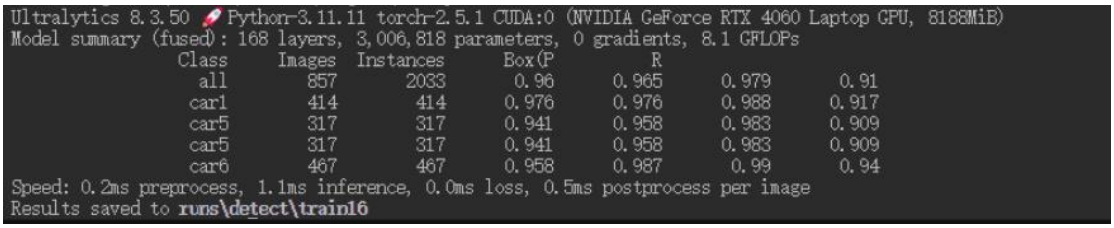
Train1

```
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      0.606G      1.758      3.322      1.425          3        640: 100%|██████████| 1712/1712 [02:02<00:00, 13.93it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:04<00:00, 25.33it/s]
                   all        857       2033      0.427      0.545      0.537      0.376

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/10      0.623G      1.112      1.525      1.073          6        640: 100%|██████████| 1712/1712 [01:58<00:00, 14.45it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:04<00:00, 22.59it/s]
                   all        857       2033      0.717      0.705      0.765      0.578

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/10      0.619G     0.9862      1.125      1.026          8        640: 100%|██████████| 1712/1712 [01:56<00:00, 14.66it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:04<00:00, 25.27it/s]
                   all        857       2033      0.765      0.682      0.779      0.609

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/10      0.617G     0.8989     0.9356     0.9959          3        640: 100%|██████████| 1712/1712 [01:56<00:00, 14.73it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:04<00:00, 22.61it/s]
                   all        857       2033      0.828      0.807      0.876      0.693

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       5/10      0.617G     0.8388     0.8087      0.967          2        640: 100%|██████████| 1712/1712 [01:59<00:00, 14.27it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:04<00:00, 23.81it/s]
                   all        857       2033      0.908      0.828      0.921      0.753

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       6/10      0.617G     0.7634     0.7067     0.9416          1        640: 100%|██████████| 1712/1712 [01:58<00:00, 14.42it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:04<00:00, 23.42it/s]
                   all        857       2033      0.904      0.837      0.929      0.771

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       7/10      0.619G     0.7134     0.6228     0.9237          5        640: 100%|██████████| 1712/1712 [02:12<00:00, 12.90it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:05<00:00, 20.38it/s]
                   all        857       2033       0.91      0.849      0.934      0.789

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       8/10      0.617G     0.6818     0.5748     0.9147          4        640: 100%|██████████| 1712/1712 [02:14<00:00, 12.69it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:05<00:00, 19.52it/s]
                   all        857       2033      0.923      0.887      0.959      0.815

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       9/10      0.617G     0.6341     0.5226     0.8998          4        640: 100%|██████████| 1712/1712 [02:14<00:00, 12.75it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:05<00:00, 19.57it/s]
                   all        857       2033      0.931       0.91      0.965      0.836

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      10/10      0.617G     0.5977     0.4815     0.8872          6        640: 100%|██████████| 1712/1712 [02:13<00:00, 12.85it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:05<00:00, 20.50it/s]
                   all        857       2033      0.941      0.926      0.971      0.848

10 epochs completed in 0.361 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...
Ultralytics 8.3.50 🚀 Python-3.11.11 torch-2.5.1 CUDA:0 (NVIDIA GeForce RTX 4060 Laptop GPU, 8188MiB)
Model summary (fused): 168 layers, 3,006,818 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:04<00:00, 22.03it/s]
                   all        857       2033      0.941      0.926      0.971      0.848
                  car1        414        414      0.965      0.961      0.988       0.87
                  car2        282        282      0.943      0.944      0.969      0.837
                  car3         59         59      0.929      0.881      0.946      0.826
                  car4        494        494      0.961      0.927      0.979      0.842
                  car5        317        317      0.914      0.886      0.959      0.822
                  car6        467        467      0.936      0.955      0.985      0.892
Speed: 0.2ms preprocess, 1.3ms inference, 0.0ms loss, 0.8ms postprocess per image
Results saved to runs\detect\train
```

可以看见第一个10epochs，模型就已经表现出较好性能，从最开始的几乎不能判断到能有正确判断用了10个epochs，速度很快，可能与数据集并不丰富有关，为了模型稳定性，还是决定继续按照原计划进行。期间F1分数大体上上升，模型变得越来越精确。如下图train10的val测试

```
Model summary (fused): 168 layers, 3,006,818 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 108/108 [00:03<00:00, 27.87it/s]
                   all        857       2033      0.972      0.952      0.979      0.899
                  car1        414        414      0.988      0.964      0.986      0.908
                  car2        282        282      0.979      0.973      0.985      0.896
                  car3         59         59      0.946      0.884      0.944      0.868
                  car4        494        494      0.981      0.957      0.985      0.897
                  car5        317        317      0.953      0.952      0.981      0.889
                  car6        467        467      0.984      0.979      0.992      0.933
Speed: 0.2ms preprocess, 1.0ms inference, 0.0ms loss, 0.6ms postprocess per image
```

接下来自定义数据增强训练6个10epochs



性能确有提升，但并不是期间最优模型，最优模型出现在train14中

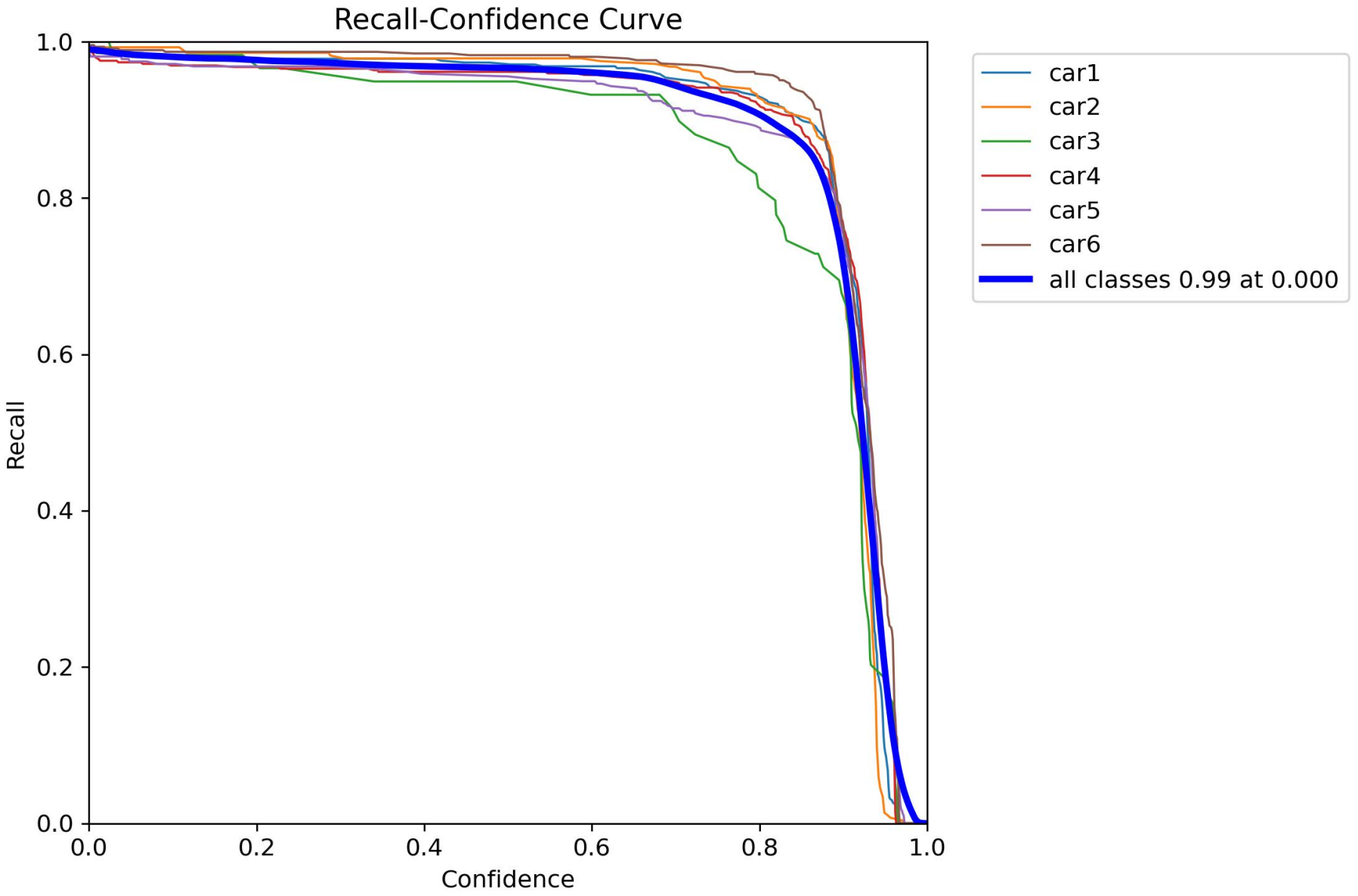
故认为，短期训练已经接近最优解，长期训练成本太高，故应该考虑消融等

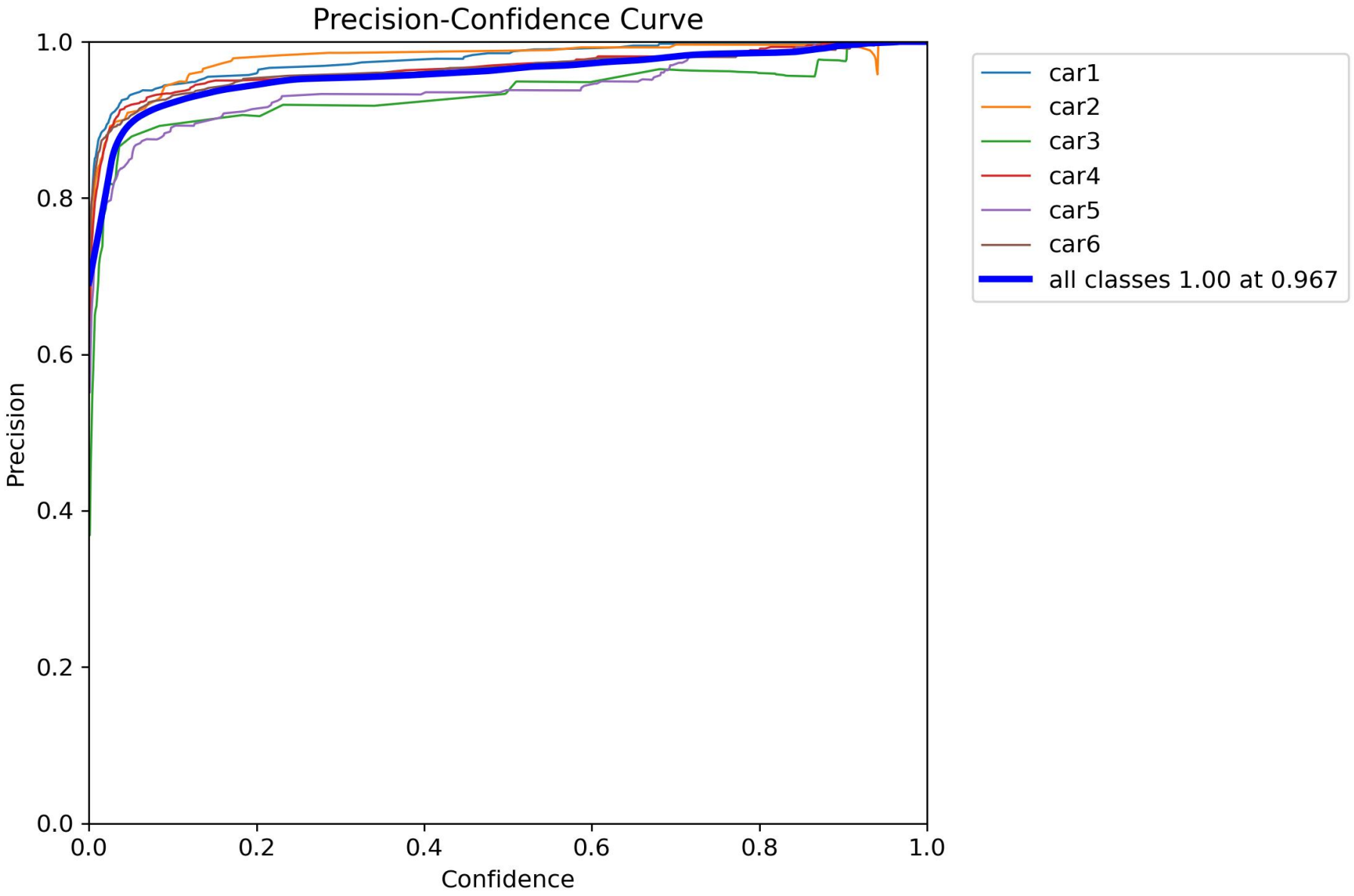
。

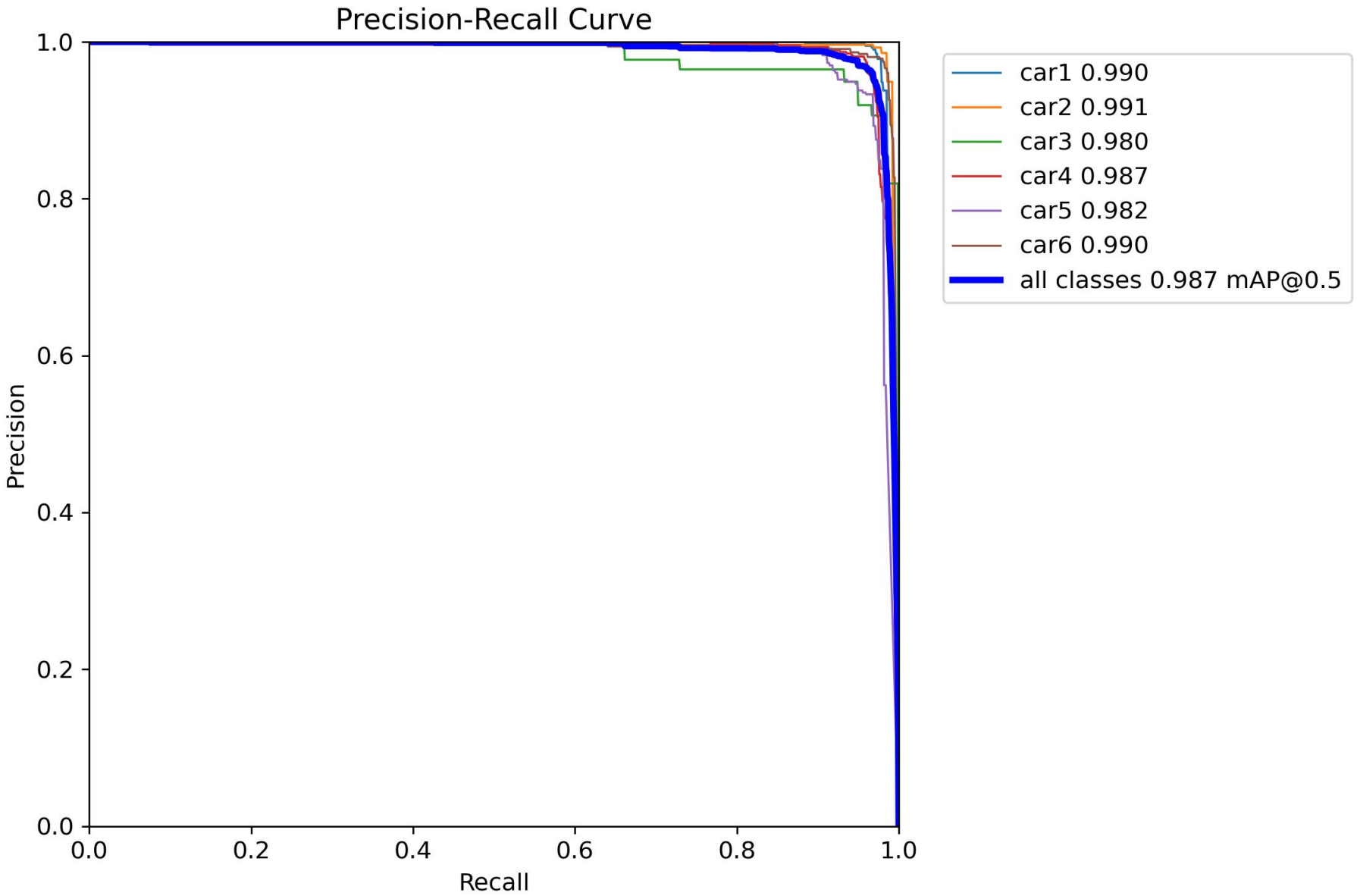
目标检测效果图：

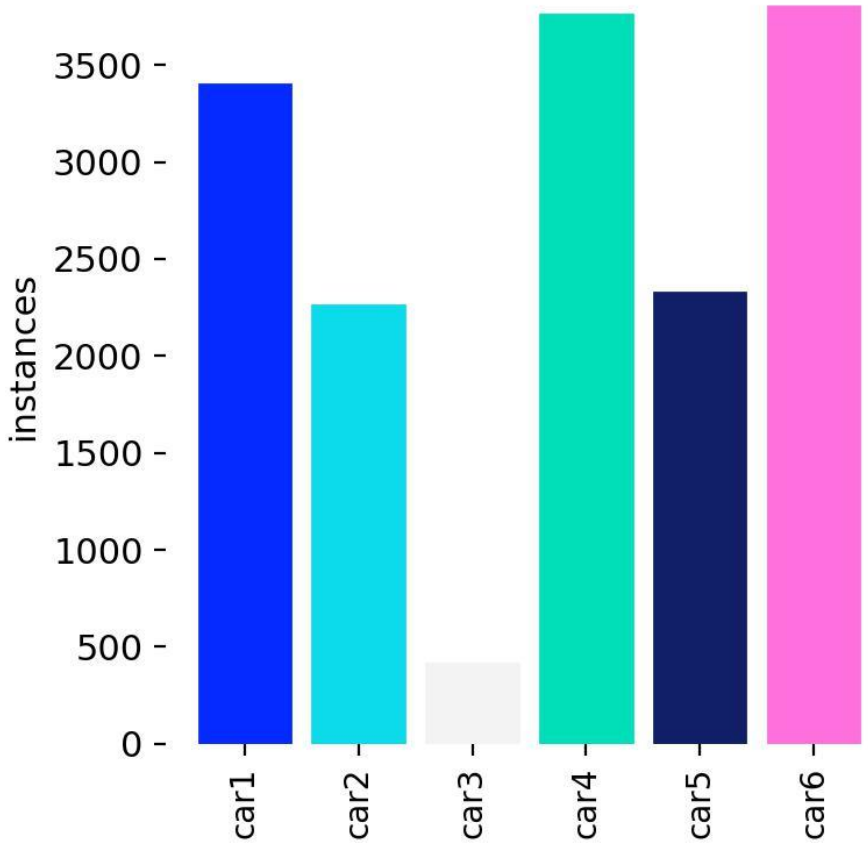
最优模型效果图：

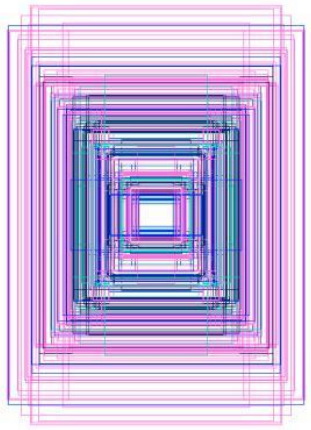

car1 0.9
car4 0.9
car6 0.9
car4 0.8
car5 0.9
car3 0.9
file_2366.jpg

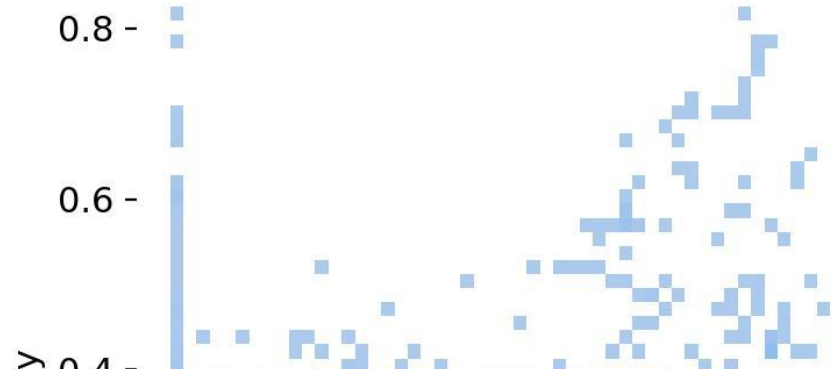

Recall-Confidence Curve
Recall
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
car1
car2
car3
car4
car5
car6
all classes 0.99 at 0.000

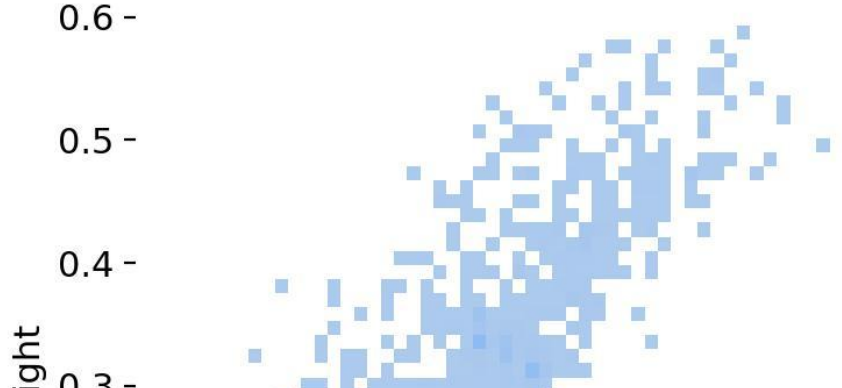

Precision-Confidence Curve
Precision
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
car1
car2
car3
car4
car5
car6
all classes 1.00 at 0.967


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
car1 0.990
car2 0.991
car3 0.980
car4 0.987
car5 0.982
car6 0.990
all classes 0.987 mAP@0.5


instances
3500
3000
2500
2000
1500
1000
500
0
car1
car2
car3
car4
car5
car6




0.8
0.6


0.6
0.5
0.4

其中可以看见car3类别数量少，证明模型14还存在来自于数据集上的缺陷，下一步应该数据增强生成car3类别的数据用于训练。

五、存在的问题及展望

一. 不足

并没有进行消融算法训练，数据增强并没有使用混合手段，导致模型存在某些car3类别，或者性能上的瓶颈，第10次训练到第16次训练性能其实变化不大。

二. 未来将要的扩充功能

开发图形化界面，接受用户输入图片，直接标出目标位置。