【Page-1】-【[深度学习] 基于切片辅助超推理库SAHI优化小目标识别-CSDN博客】-【2023-11-29】-【 https://blog.csdn.net/LuohenYJ/article/details/128538834 】

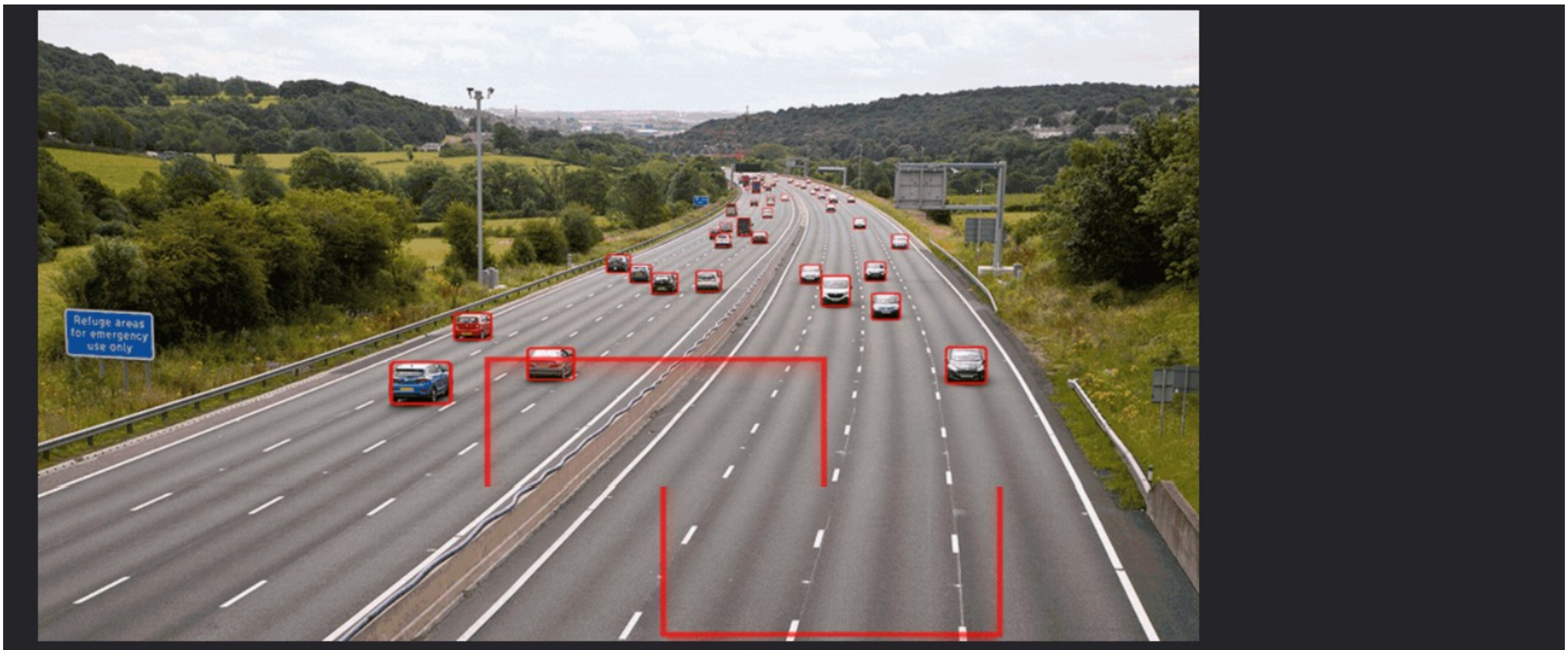
[深度学习] 基于切片辅助超推理库SAHI优化小目标识别

原创 落痕的寒假 已于 2023-01-03 20:34:45 修改 阅读量4.6k 收藏 79 点赞数 18 版权

分类专栏: 深度学习 图像处理 Python 文章标签: 深度学习 计算机视觉 人工智能

对象检测是迄今为止计算机视觉中最重要的应用领域。然而，小物体的检测和大图像的推理仍然是实际使用中的主要问题，这是因为小目标物体有效特征少，覆盖范围少。小目标物体的定义通常有两种方式。一种是绝对尺度定义，即以物体的像素尺寸来判断是否为小目标，如在COCO数据集中，尺寸小于32×32像素的目标被判定为小目标。另外一种是相对尺度定义，即以物体在图像中的占比面积比例来判断是否为小目标，例如国际光学工程学会SPIE定义，若目标尺寸小于原图的0.12%则可以判定成小目标。

SAHI: Slicing Aided Hyper Inference（切片辅助超推理）通过图像切片的方式来检测小目标。SAHI检测过程可以描述为：通过滑动窗口将图像切分成若干区域，各个区域分别进行预测，同时也对整张图片进行推理。然后将各个区域的预测结果和整张图片的预测结果合并，最后用NMS（非极大值抑制）进行过滤。用动图表示该识别过程如下：



SAHI的官方仓库地址为：sahi。关于SAHI的使用可以阅读官方demo和官方文档：sahi-demo和sahi-docs。如果想进一步了解SAHI具体工作性能和原理，可以阅读官方发表的论文：Slicing Aided Hyper Inference and Fine-Tuning for Small Object Detection。
SAHI安装指令如下：

```
pip install sahi
```

本文所有算法展示效果和代码见：

github: Python-Study-Notes

文章目录

1 SAHI使用

```
import sahi
# 打印sahi版本
print(sahi.__version__)
```

```
0.11.6
```

1.1 图像切片

SAHI提供了封装好的函数接口，以切分输入图像和其标注数据。切分后的子图及其标注数据可以用于识别，或者保存为本地数据以供模型训练。

1.1.1 单张图像切片

SAHI提供slice_image函数以切分单张图片及其标注文件（仅支持coco标注文件），slice_image函数接口介绍如下：

```
# 返回SAHI的图像分片结果类SliceImageResult
def slice_image(
    image: Union[str, Image.Image], # 单张图像地址或单个pillow image对象，必填参数
    coco_annotation_list: Optional[CocoAnnotation] = None, # coco标注文件
    output_file_name: Optional[str] = None, # 输出文件名前缀
    output_dir: Optional[str] = None, # 输出文件地址
    slice_height: int = None, # 子图切分高度
    slice_width: int = None, # 子图切分宽度
    overlap_height_ratio: float = None, # 子图高度间的重叠率
    overlap_width_ratio: float = None, # 子图宽度间的重叠率
    auto_slice_resolution: bool = True, # 如果没有设置slice_height和slice_width，则自动确定slice_height、slice_width、overlap_
height_ratio、overlap_width_ratio
    min_area_ratio: float = 0.1, # 子图中标注框小于原始标注框占比，则放弃该标注框
    out_ext: Optional[str] = None, # 图像后缀格式
    verbose: bool = False, # 是否打印详细信息
)
```

slice_image函数源代码位于sahi/slicing.py中，这段代码可以单步调试看看怎么运行的，主要逻辑如下：

1. 获得pillow image图像对象
2. 调用get_slice_bboxes函数切分图像
    - 获得切分参数

```
if slice_height and slice_width:
    # 计算重叠像素
    y_overlap = int(overlap_height_ratio * slice_height)
    x_overlap = int(overlap_width_ratio * slice_width)
elif auto_slice_resolution:
    x_overlap, y_overlap, slice_width, slice_height = get_auto_slice_params(height=image_height, width=image_width)
```

    - 循环切分图像

```
# 行循环
while y_max < image_height:
    # 设置起始切分坐标
    x_min = x_max = 0
    y_max = y_min + slice_height
    # 列循环
    while x_max < image_width:
        x_max = x_min + slice_width
        # 如果图像不够切分，框往左或往上移动
        if y_max > image_height or x_max > image_width:
            xmax = min(image_width, x_max)
            ymax = min(image_height, y_max)
            xmin = max(0, xmax - slice_width)
            ymin = max(0, ymax - slice_height)
            slice_bboxes.append([xmin, ymin, xmax, ymax])
        else:
            slice_bboxes.append([x_min, y_min, x_max, y_max])
        # 下一次切分从本次切分图像x_max-x_overlap开始
        x_min = x_max - x_overlap
    y_min = y_max - y_overlap
```

3. 保存图片结果和标注结果，并包装返回SliceImageResult对象

以下代码演示了对单张图片进行切片，并将切分后的子图保存到本地。

展示原图

```
# 展示输入图片
from PIL import Image
# 图像地址: https://github.com/obss/sahi/tree/main/demo/demo_data
image_path = "image/small-vehicles1.jpeg"
img = Image.open(image_path).convert('RGB')
img
```

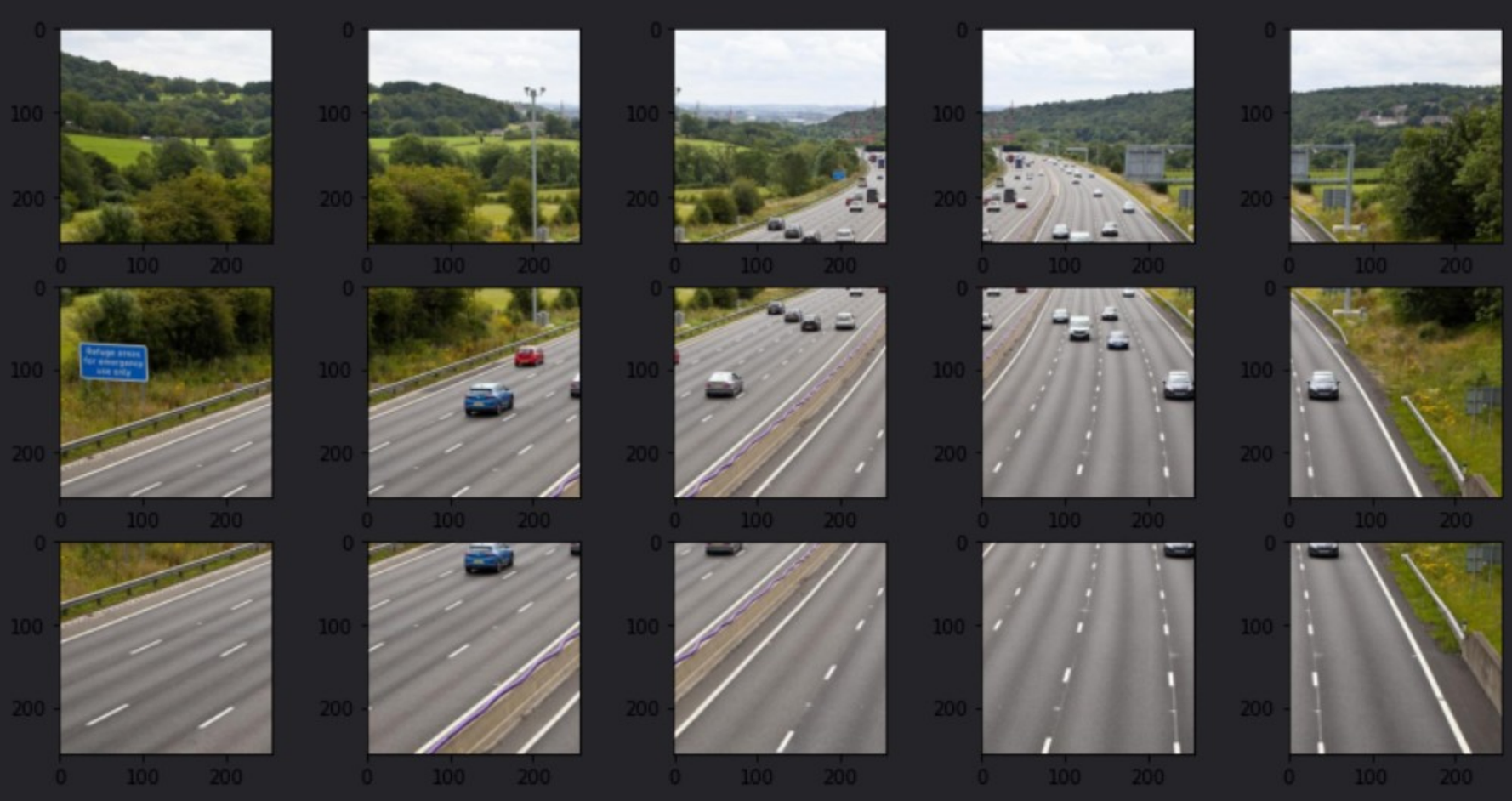
切分图片

```
from sahi.slicing import slice_image

# 输出文件名前缀
output_file_name = "slice"
# 输出文件夹
output_dir = "result"

# 切分图像
slice_image_result = slice_image(
    image=image_path,
    output_file_name=output_file_name,
    output_dir=output_dir,
    slice_height=256,
    slice_width=256,
    overlap_height_ratio=0.2,
    overlap_width_ratio=0.2,
    verbose=False,
)
print("原图宽{}，高{}".format(slice_image_result.original_image_width, slice_image_result.original_image_height))
# 切分后的子图以形式：图像前缀_所在原图顶点坐标来保存文件
print("切分子图{}张".format(len(slice_image_result.filenames)))

```

```
原图宽1068，高580
切分子图15张
```

展示切分后的子图

```
import matplotlib.pyplot as plt
from PIL import Image
import math
import os

axarr_row = 3
axarr_col = math.ceil(len(slice_image_result.filenames)/axarr_row)
f, axarr = plt.subplots(axarr_row, axarr_col, figsize=(14,7))
for index, file in enumerate(slice_image_result.filenames):
    img = Image.open(os.path.join(slice_image_result.image_dir,file))
    axarr[int(index/axarr_col), int(index%axarr_col)].imshow(img)
```



1.1.2 COCO数据集切片

SAHI提供slice_coco函数以切分coco数据集（仅支持coco数据集）。slice_coco函数接口介绍如下：

```
# 返回切片后的coco标注字典文件，coco文件保存地址
def slice_coco(
    coco_annotation_file_path: str, # coco标注文件
    image_dir: str, # coco图像集地址
    output_coco_annotation_file_name: str, # 输出coco标注集文件名，不需要加文件类型后缀
    output_dir: Optional[str] = None, # 输出文件地址
    ignore_negative_samples: bool = False, # 是否忽略没有标注框的子图
    slice_height: int = 512, # 切分子图高度
    slice_width: int = 512, # 切分子图宽度
    overlap_height_ratio: float = 0.2, # 子图高度之间的重叠率
    overlap_width_ratio: float = 0.2, # 子图宽度之间的重叠率
    min_area_ratio: float = 0.1, # 如果没有设置slice_height和slice_width，则自动确定slice_height、slice_width、overlap_height_
ratio、overlap_width_ratio
    out_ext: Optional[str] = None,  # 保存图像的扩展
    verbose: bool = False, # 是否打印详细信息
)
```

slice_coco函数源代码位于sahi/slicing.py中，这段代码可以单步调试看看怎么做的，主要逻辑如下：

1. 读取coco文件和图片信息
2. 循环读取coco数据集的图片，每张图片调用get_slice_bboxes函数切分图像
3. 创建coco dict结果并保存文件

以下代码演示了对coco数据集进行切片，并将切分后的子图和标注文件保存到本地。coco数据集可以包含若干张图片，但是以下代码示例中只包含一张图片，方便演示。

【Page-6】-【[深度学习] 基于切片辅助超推理库SAHI优化小目标识别-CSDN博客】-【2023-11-29】-【 https://blog.csdn.net/LuohenYJ/article/details/128538834 】

展示数据集

```
# 展示图像
from PIL import Image, ImageDraw
from sahi.utils.file import load_json
import matplotlib.pyplot as plt
import os

# coco图像集地址
image_path = "image"
# coco标注文件
coco_annotation_file_path="image/terrain2_coco.json"
# 加载数据集
coco_dict = load_json(coco_annotation_file_path)

f, axarr = plt.subplots(1, 1, figsize=(8, 8))
# 读取图像
img_ind = 0
img = Image.open(os.path.join(image_path,coco_dict["images"][img_ind]["file_name"])).convert('RGBA')
# 绘制标注框
for ann_ind in range(len(coco_dict["annotations"])):
    xywh = coco_dict["annotations"][ann_ind]["bbox"]
    xyxy = [xywh[0], xywh[1], xywh[0] + xywh[2], xywh[1] + xywh[3]]
    ImageDraw.Draw(img, 'RGBA').rectangle(xyxy, width=5)
axarr.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x210a7583250>
```

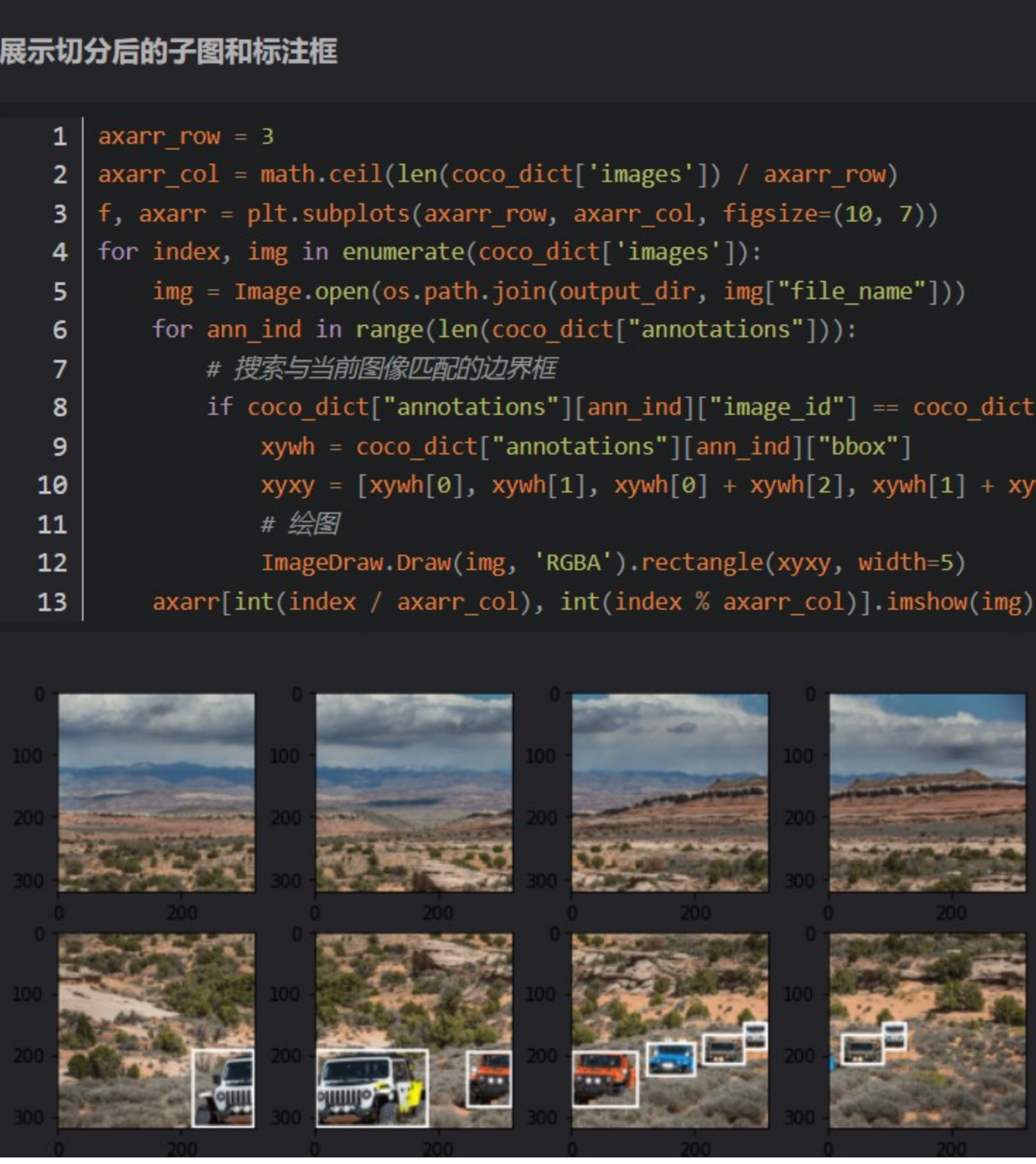
切分数据集

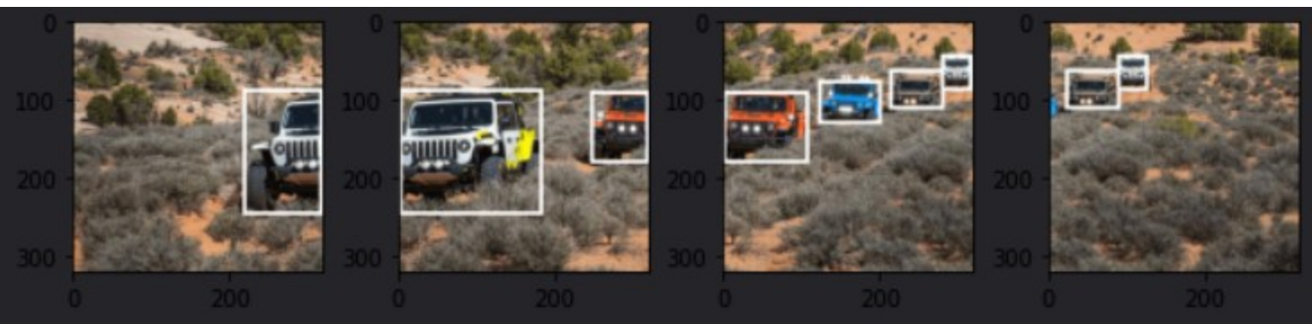
```python
from sahi.slicing import slice_coco

# 保存的coco数据集标注文件名
output_coco_annotation_file_name="sliced"
# 输出文件夹
output_dir = "result"

# 切分数据集
coco_dict, coco_path = slice_coco(
    coco_annotation_file_path=coco_annotation_file_path,
    image_dir=image_path,
    output_coco_annotation_file_name=output_coco_annotation_file_name,
    ignore_negative_samples=False,
    output_dir=output_dir,
    slice_height=320,
    slice_width=320,
    overlap_height_ratio=0.2,
    overlap_width_ratio=0.2,
    min_area_ratio=0.2,
    verbose=False
)

print("切分子图{}张".format(len(coco_dict['images'])))
print("获得标注框{}个".format(len(coco_dict['annotations'])))
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|| 1/1 [00:00<00:00, 334.21it/s]
100%|| 1/1 [00:00<00:00, 11.80it/s]

切分子图12张
获得标注框18个
```

展示切分后的子图和标注框

```python
axarr_row = 3
axarr_col = math.ceil(len(coco_dict['images']) / axarr_row)
f, axarr = plt.subplots(axarr_row, axarr_col, figsize=(10, 7))
for index, img in enumerate(coco_dict['images']):
    img = Image.open(os.path.join(output_dir, img["file_name"]))
    for ann_ind in range(len(coco_dict["annotations"])):
        # 搜索与当前图像匹配的边界框
        if coco_dict["annotations"][ann_ind]["image_id"] == coco_dict["images"][index]["id"]:
            xywh = coco_dict["annotations"][ann_ind]["bbox"]
            xyxy = [xywh[0], xywh[1], xywh[0] + xywh[2], xywh[1] + xywh[3]]
            # 绘图
            ImageDraw.Draw(img, 'RGBA').rectangle(xyxy, width=5)
    axarr[int(index / axarr_col), int(index % axarr_col)].imshow(img)
```





1.2 图像预测

1.2.1 接口介绍

SHAI提供了图像切片预测的封装接口，具体的函数接口如下：

AutoDetectionModel类

SAHI基于AutoDetectionModel类的from_pretrained函数加载深度学习模型。目前支持YOLOv5 models, MMDetection models, Detectron2 models和HuggingFace object detection models等深度学习模型库，如果想支持新的模型库,可以参考sahi/models目录下的模型文件，新建模型检测类。

模型预测

- 基于get_prediction函数调用模型预测单张图片，也就是直接调用AutoDetectionModel类提供的模型，直接推理单张图片。
- 基于get_sliced_prediction函数以切分图片的方式进行预测。在get_sliced_prediction函数内部会先切分图片，然后对每个子图单独进行模型推理；如果设置了对整张原图进行推理，那么也会整合原图推理的结果以增加模型精度。最后对所有的预测结果进行nms整合，相近的两个预测框也会进行合并。get_sliced_prediction函数接口如下：

```
def get_sliced_prediction(
    image,
    detection_model=None,
    slice_height: int = None,
    slice_width: int = None,
    overlap_height_ratio: float = 0.2,
    overlap_width_ratio: float = 0.2,
    perform_standard_pred: bool = True, # 是否单独对原图进行识别
    postprocess_type: str = "GREEDYNMM", # 合并结果的方式，可选'NMM', 'GRREDYNMM',  'NMS'
    postprocess_match_metric: str = "IOS", # NMS匹配方式IOU或者IOS
    postprocess_match_threshold: float = 0.5, # 匹配置信度
    postprocess_class_agnostic: bool = False, # 在合并结果时，是否将不同类别的检测框放在一起处理
    verbose: int = 1,
    merge_buffer_length: int = None, # 低配设备使用，以加快处理
    auto_slice_resolution: bool = True,
)
```

- 基于predict函数进行批处理，predict函数进一步封装了识别代码，如果想使用该函数，阅读predict源代码参数接口即可。

1.2.2 应用实例

直接预测图片

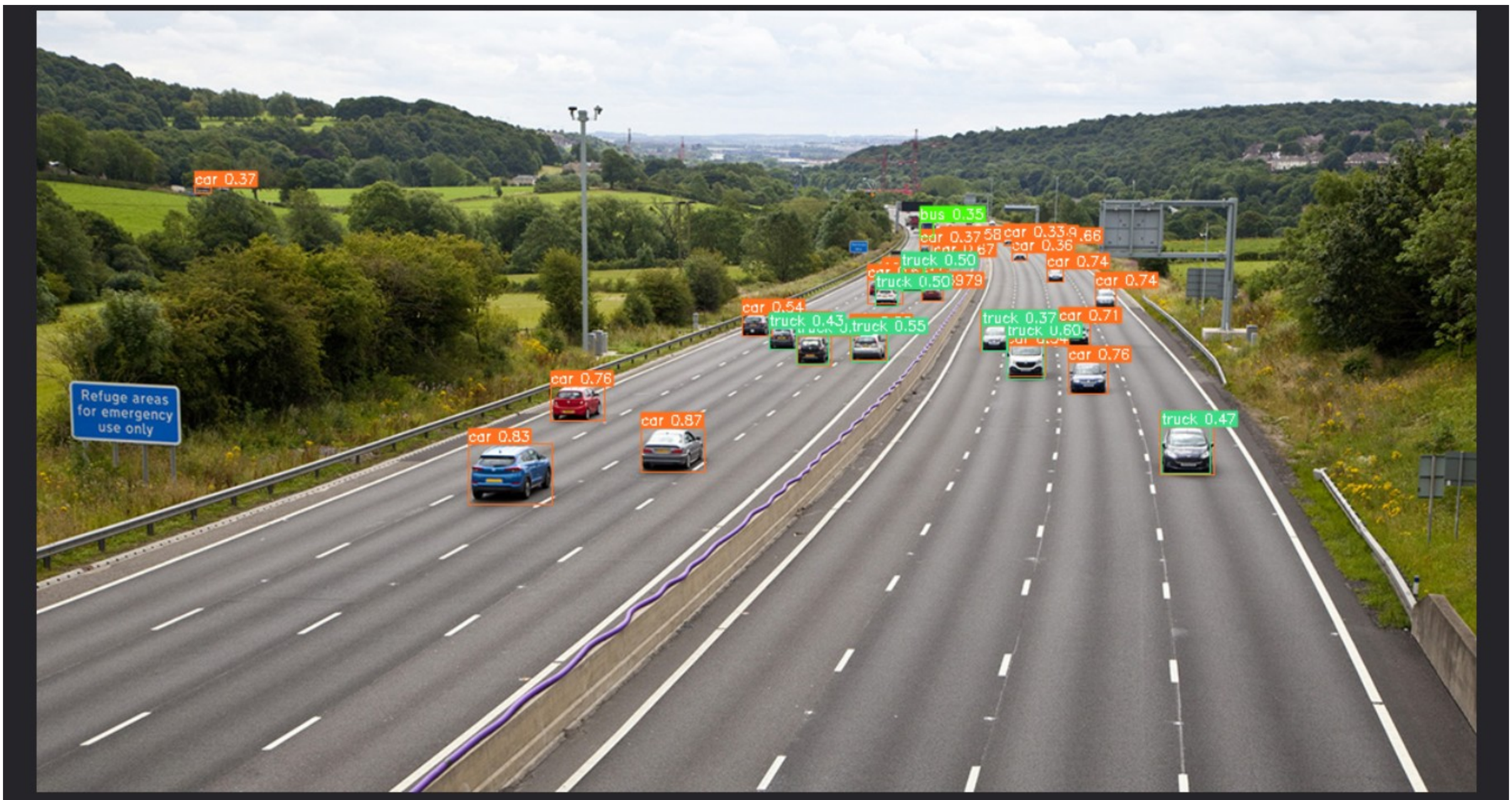
```python
from sahi import AutoDetectionModel
from sahi.predict import get_prediction

# 初始化检测模型，缺少yolov5代码，pip install yolov5即可
detection_model = AutoDetectionModel.from_pretrained(
    model_type='yolov5', # 模型类型
    model_path='./yolov5n.pt', # 模型文件路径
    confidence_threshold=0.3, # 检测阈值
    device="cpu",  # or 'cuda:0'
);
image = 'image/small-vehicles1.jpeg'

# 获得模型直接预测结果
result = get_prediction(image, detection_model)

# result是SAHI的PredictionResult对象，可获得推理时间，检测图像，检测图像尺寸，检测结果
# 查看标注框，可以用于保存为其他格式
for pred in result.object_prediction_list:
    bbox = pred.bbox  # 标注框BoundingBox对象，可以获得边界框的坐标、面积
    category = pred.category  # 类别Category对象，可获得类别id和类别名
    score = pred.score.value  # 预测置信度

# 保存文件结果
export_dir = "result"
file_name = "res"
result.export_visuals(export_dir=export_dir, file_name=file_name)

# 展示结果
from PIL import Image
import os
image_path = os.path.join(export_dir,file_name+'.png')
img = Image.open(image_path).convert('RGB')
img
```

【Page-10】-【[深度学习] 基于切片辅助超推理库SAHI优化小目标识别-CSDN博客】-【2023-11-29】-【 https://blog.csdn.net/LuohenYJ/article/details/128538834 】

切片预测图片

```python
from sahi import AutoDetectionModel
from sahi.predict import get_sliced_prediction

# 初始化检测模型
detection_model = AutoDetectionModel.from_pretrained(
    model_type='yolov5',
    model_path='yolov5n.pt',
    confidence_threshold=0.3,
    device="cpu",  # or 'cuda:0'
)
image = 'image/small-vehicles1.jpeg'


result = get_sliced_prediction(
    image,
    detection_model,
    slice_height = 256,
    slice_width = 256,
    overlap_height_ratio = 0.2,
    overlap_width_ratio = 0.2,
    perform_standard_pred = True,
)

# result是SAHI的PredictionResult对象，可获得推理时间，检测图像，检测图像尺寸，检测结果
# 查看标注框，可以用于保存为其他格式
for pred in result.object_prediction_list:
    bbox = pred.bbox  # 标注框BoundingBox对象，可以获得边界框的坐标、面积
    category = pred.category  # 类别Category对象，可获得类别id和类别名
    score = pred.score.value  # 预测置信度

# 保存文件结果
export_dir = "result"
file_name = "res"
result.export_visuals(export_dir=export_dir, file_name=file_name)
# 结果导出为coco标注形式
coco_anno = result.to_coco_annotations()
# 结果导出为coco预测形式
coco_pred = result.to_coco_predictions()

# 展示结果
from PIL import Image
import os
image_path = os.path.join(export_dir,file_name+'.png')
img = Image.open(image_path).convert('RGB')
img

```

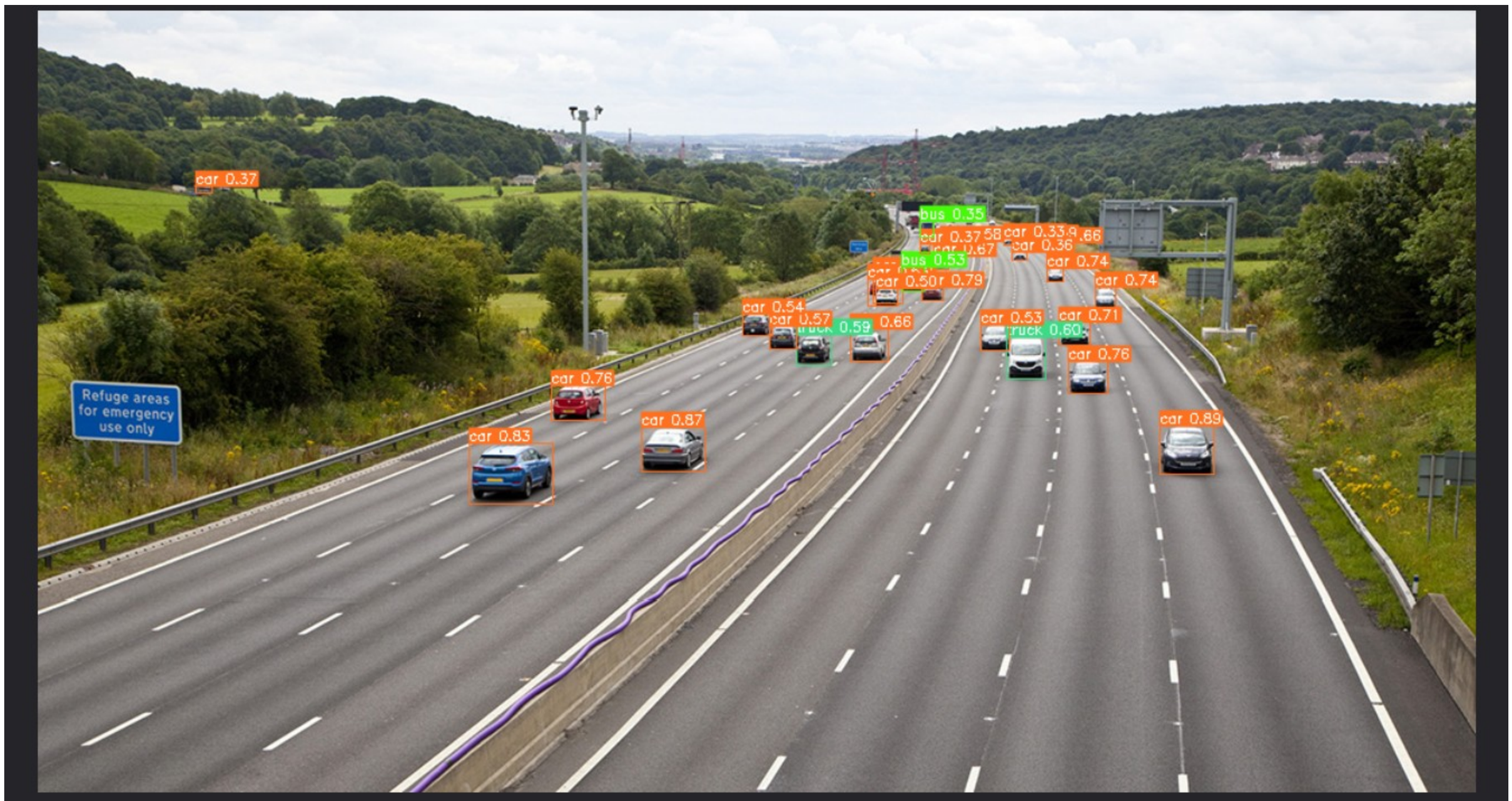
```
Performing prediction on 15 number of slices.
```



相对单张图片直接识别，通过切片的方式能够识别到更多的小目标。由于使用的模型是yolov5n，可以看到一些识别结果不正确，比如同一辆车在不同子图被分别识别为卡车或汽车，一种好的解决办法是将postprocess_class_agnostic参数设置为True，将不同类别的检测框放在一起进行合并，同时降低 postprocess_match_threshold以滤除结果。

```
image = 'image/small-vehicles1.jpeg'


result = get_sliced_prediction(
    image,
    detection_model,
    slice_height = 256,
    slice_width = 256,
    overlap_height_ratio = 0.2,
    overlap_width_ratio = 0.2,
    perform_standard_pred = True,
    postprocess_match_threshold = 0.2,
    postprocess_class_agnostic = True,
)


# 保存文件结果
export_dir = "result"
file_name = "res"
result.export_visuals(export_dir=export_dir, file_name=file_name)

# 展示结果
from PIL import Image
import os
image_path = os.path.join(export_dir,file_name+'.png')
img = Image.open(image_path).convert('RGB')
img
```

```
Performing prediction on 15 number of slices.
```



1.3 SAHI工具函数

SAHI提供多个工具函数以处理COCO数据集，具体使用可以阅读sahi-docs-coco。

1.3.1 coco数据集制作与精度分析

以下代码创建了coco标注数据，并保存到本地

```python
from sahi.utils.file import save_json
from sahi.utils.coco import Coco, CocoCategory, CocoImage, CocoAnnotation,CocoPrediction


# 创建coco对象
coco = Coco()

# 添加类
coco.add_category(CocoCategory(id=0, name='human'))
coco.add_category(CocoCategory(id=1, name='vehicle'))

# 循环遍历图像
for i in range(3):
    # 创建单个图像
    coco_image = CocoImage(
        file_name="image{}.jpg".format(i), height=1080, width=1920)

    # 添加图像对应的标注
    coco_image.add_annotation(
        CocoAnnotation(
            # [x_min, y_min, width, height]
            bbox=[0, 0, 200, 200],
            category_id=0,
            category_name='human'
        )
    )
    coco_image.add_annotation(
        CocoAnnotation(
            bbox=[200, 100, 300, 300],
            category_id=1,
            category_name='vehicle'
        )
    )

```

```
    # 添加图像预测数据
    coco_image.add_prediction(
      CocoPrediction(
        score=0.864434,
        bbox=[0, 0, 150, 150],
        category_id=0,
        category_name='human'
      )
    )
    coco_image.add_prediction(
      CocoPrediction(
        score=0.653424,
        bbox=[200, 100, 250, 200],
        category_id=1,
        category_name='vehicle'
      )
)
    # 将图像添加到coco对象
    coco.add_image(coco_image)

# 提取json标注数据，不会保存图像预测结果
coco_json = coco.json

# 将json标注数据保存为json本地文件
save_json(coco_json, "coco_dataset.json")

# 提取预测结果json文件，并保存到本地
predictions_array = coco.prediction_array
save_json(predictions_array, "coco_predictions.json")
```

当我们获得了预测数据，我们可以基于pycocotools工具分析预测数据的精度，pycocotools是目标检测必备工具，官方仓库地址为cocoapi，结果分析代码如下：

```
# 需要单独安装pycocotools
from pycocotools.cocoeval import COCOeval
from pycocotools.coco import COCO

coco_ground_truth = COCO(annotation_file="coco_dataset.json")
coco_predictions = coco_ground_truth.loadRes("coco_predictions.json")

coco_evaluator = COCOeval(coco_ground_truth, coco_predictions, "bbox")
# 进行匹配计算
coco_evaluator.evaluate()
# 进行结果的累加
coco_evaluator.accumulate()
# 输出结果
coco_evaluator.summarize()
```

```
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Loading and preparing results...
DONE (t=0.00s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.00s).
Accumulating evaluation results...
DONE (t=0.01s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.200
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.200
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.200
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.200
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.200
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.200
```

统计数据集标注信息

```
from sahi.utils.coco import Coco

coco = Coco.from_coco_dict_or_path("coco_dataset.json")

# 获得数据集状态，指标说明看字段名就能懂
stats = coco.stats
stats
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 1504.59it/s]




{'num_images': 3,
 'num_annotations': 6,
 'num_categories': 2,
 'num_negative_images': 0,
 'num_images_per_category': {'human': 3, 'vehicle': 3},
 'num_annotations_per_category': {'human': 3, 'vehicle': 3},
 'min_num_annotations_in_image': 2,
 'max_num_annotations_in_image': 2,
 'avg_num_annotations_in_image': 2.0,
 'min_annotation_area': 40000,
 'max_annotation_area': 90000,
 'avg_annotation_area': 65000.0,
 'min_annotation_area_per_category': {'human': 40000, 'vehicle': 90000},
 'max_annotation_area_per_category': {'human': 40000, 'vehicle': 90000}}
```

预测结果过滤

```python
from sahi.utils.file import save_json
from sahi.utils.coco import remove_invalid_coco_results

# 去除预测结果中的无效边界框，如边界框坐标为负的结果
coco_results = remove_invalid_coco_results("coco_predictions.json")

save_json(coco_results, "fixed_coco_result.json")

# 根据数据集实际标注信息，进一步去除边界框坐标超过图像长宽的结果
coco_results = remove_invalid_coco_results("coco_predictions.json", "coco_dataset.json")
```

1.3.2 coco数据集处理

切分数据集

```python
from sahi.utils.coco import Coco

# 指定coco文件
coco_path = "coco_dataset.json"

# 初始coco对象
coco = Coco.from_coco_dict_or_path(coco_path)

# 拆分数据集为训练集和验证集，训练集图像占比0.85
result = coco.split_coco_as_train_val(
  train_split_rate=0.85
)

# 保存训练集和验证集
save_json(result["train_coco"].json, "train_split.json")
save_json(result["val_coco"].json, "val_split.json")
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 3005.95it/s]
```

修改标注类别

```python
from sahi.utils.coco import Coco
from sahi.utils.file import save_json


coco = Coco.from_coco_dict_or_path("coco_dataset.json")
print("标注类别：{}".format(coco.category_mapping))

# 修改数据集类别
# 将标注中human类的索引改为3，将原先vehicle类的标注删除
# 新加big_vehicle类和car类
desired_name2id = {
  "big_vehicle": 1,
  "car": 2,
  "human": 3
}
# 更新标注类别
coco.update_categories(desired_name2id)

print("修改后标注类别：{}".format(coco.category_mapping))

# 保存结果
save_json(coco.json, "updated_coco.json")
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 1002.78it/s]

标注类别：{0: 'human', 1: 'vehicle'}
修改后标注类别：{1: 'big_vehicle', 2: 'car', 3: 'human'}
```

按照标注框面积过滤数据集

```python
from sahi.utils.coco import Coco
from sahi.utils.file import save_json

# 打开标注数据
coco = Coco.from_coco_dict_or_path("coco_dataset.json")

# 过滤包含标注框面积小于min的图像
area_filtered_coco = coco.get_area_filtered_coco(min=50000)
# 过滤标注框面积不在[min,max]的图像
area_filtered_coco = coco.get_area_filtered_coco(min=50, max=80000)
# 筛选同时符合多个类别面积要求的图像
intervals_per_category = {
  "human": {"min": 20, "max": 30000},
  "vehicle": {"min": 50, "max": 90000},
}
area_filtered_coco = coco.get_area_filtered_coco(intervals_per_category=intervals_per_category)

# 导出数据
save_json(area_filtered_coco.json, "area_filtered_coco.json")
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 1503.69it/s]
```

过滤无标注的图片

```python
from sahi.utils.coco import Coco
from sahi.utils.file import save_json
# 去除无标注框的图片
coco = Coco.from_coco_dict_or_path("coco_dataset.json", ignore_negative_samples=True)
# 导出数据
# save_json(coco.json, "coco_ignore_negative.json")
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 3007.39it/s]
```

裁剪标注框

```python
from sahi.utils.coco import Coco
from sahi.utils.file import save_json


coco_path = "coco_dataset.json"

# 将溢出边界框剪裁为图像宽度和高度
coco = Coco.from_coco_dict_or_path(coco_path, clip_bboxes_to_img_dims=True)

# 对已有coco对象，将溢出边界框剪裁为图像宽度和高度
coco = coco.get_coco_with_clipped_bboxes()

save_json(coco.json, "coco.json")
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 1007.04it/s]
```

合并coco数据集

```python
# from sahi.utils.coco import Coco
# from sahi.utils.file import save_json

# coco_1 = Coco.from_coco_dict_or_path("coco1.json", image_dir="images_1/")
# coco_2 = Coco.from_coco_dict_or_path("coco2.json", image_dir="images_2/")

# # 合并数据集
# coco_1.merge(coco_2)

# # 保存
# save_json(coco_1.json, "merged_coco.json")
```

下采样数据集

```python
from sahi.utils.coco import Coco
from sahi.utils.file import save_json
coco_path = "coco_dataset.json"

coco = Coco.from_coco_dict_or_path(coco_path)

# 用1/10的图像创建Coco对象
# subsample_ratio表示每10张图像取1张图像
subsampled_coco = coco.get_subsampled_coco(subsample_ratio=10)

# 仅对包含标注框为category_id的图像进行下采样,category_i=-1时表示负样本
subsampled_coco = coco.get_subsampled_coco(subsample_ratio=10, category_id=0)

# 保存数据集
save_json(subsampled_coco.json, "subsampled_coco.json")
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 1512.19it/s]
```

上采样数据集

```python
from sahi.utils.coco import Coco
from sahi.utils.file import save_json
coco_path = "coco_dataset.json"

coco = Coco.from_coco_dict_or_path(coco_path)

# 每个样本重复10次
upsampled_coco = coco.get_upsampled_coco(upsample_ratio=10)


# 仅对包含标注框为category_id的图像进行采样,category_i=-1时表示负样本
subsampled_coco = coco.get_upsampled_coco(upsample_ratio=10, category_id=0)


# 导出数据集
save_json(upsampled_coco.json, "upsampled_coco.json")
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 1503.51it/s]
```

1.3.3 coco数据集转换

导出为yolov5格式并分割数据集

```python
from sahi.utils.coco import Coco

# 注意image_dir路径
coco = Coco.from_coco_dict_or_path("coco_dataset.json", image_dir="images/")

# 导出为yolov5数据集格式, train_split_rate设置训练集数据比例
# coco.export_as_yolov5(
#   output_dir="output/",
#   train_split_rate=0.85
# )
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 3/3 [00:00<00:00, 1002.22it/s]
```

将训练集和验证集导出为yolov5格式

```python
from sahi.utils.coco import Coco, export_coco_as_yolov5

# 注意image_dir路径
train_coco = Coco.from_coco_dict_or_path("train_split.json", image_dir="images/")
val_coco = Coco.from_coco_dict_or_path("val_split.json", image_dir="images/")

# 导出数据集
# data_yml_path = export_coco_as_yolov5(
#   output_dir="output",
#   train_coco=train_coco,
#   val_coco=val_coco
# )
```

```
indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 2/2 [00:00<00:00, 1002.34it/s]


indexing coco dataset annotations...


Loading coco annotations: 100%|██████████| 1/1 [00:00<00:00, 1003.42it/s]
```

1.4 总结

目标检测过程中，通过对高分辨率小目标图像进行滑动窗口切片，能够有效提高大分辨率小目标图像的识别精度。但是滑动切片识别有需要注意的地方：

- 需要图像数据集是否符合通用的高分辨小目标图像标准，如果对普通数据集进行切片识别容易拆分已有目标物体，这样做浪费推理时间也会导致最终检测结果精度不高。
- 滑动切片对识别模型的精度有一定的要求，一般来说模型越大精度越高，但是切片识别所花费的推理时间也越长。所以需要平衡模型精度和模型推理时间，而且也要确定滑动切片的尺度。
- 滑动切片识别在识别目标类别较少的任务中，识别精度更高，因为后处理能过滤很多重复识别检测框。

如果想了解其他的小目标识别方案，可以看看paddle家的paddledetection-smalldet。paddle提供了基于原图和基于切图的小目标识别方案，也提供了统计数据集尺寸分布的代码（该统计代码对某些特定的数据集效果不好，具体原因看看代码）。推荐看看PaddleDetection的小目标识别方案，做的很不错。

2 参考

- sahi
- sahi-demo
- sahi-docs
- Slicing Aided Hyper Inference and Fine-Tuning for Small Object Detection
- sahi/slicing.py
- sahi/models
- sahi-docs-coco
- cocoapi
- paddledetection-smalldet

显示推荐内容