一、背景描述

1. 项目算法要达到的目的：非机动车乱停放识别算法主要用于建城管场景如沿街，小区，广场的非停车区域，可对进入非停车区域进行自动识别。

二、目标描述

1. 需求边界定义：没有人骑(或扶着/倚着/推动)的非机动车；
2. 算法报警的业务逻辑：识别到没人骑(或扶着/倚着/推动)的非机动车，停留在非停车区域进行报警。
3. 识别场景：街道
4. 识别对象：自行车、电动车、人力三轮车、电动三轮车、摩托车
5. 环境要求：室外，光照充足。
6. 其他补充：无（默认）

三、算法内容

1. 算法输入：单个视频流
2. 算法输入设备：固定摄像头
3. 算法输出：callback
4. 1080P算法最小识别像素：最小像素80\*80
5. 算法实时性：实时性高
6. 算法部署：本地部署
7. 精确率目标：90%

四、数据需求

标签：

（1）bicycle：自行车、电动自行车，不管是否有人骑（如果无人骑时，此标注与none\_person\_bicycle标签范围相同）

（2）motorbike：男式摩托车，不管是否有人骑（如果无人骑时，此标注与none\_person\_motorbick标签范围相同）

（3）electric\_scooter：电瓶车、电动车、女式摩托车，不管是否有人骑（如果无人骑时，此标注与none\_person\_electric\_scooter标签范围相同）

（4）tricycle：人力三轮车，不管是否有人骑（如果无人骑时，此标注与none\_person\_tricycle标签范围相同）

（5）auto\_tricycle：电动三轮小货车、电动三轮载客车，，不管是否有人骑（如果无人骑时，此标注与none\_person\_auto\_tricycle标签范围相同）

（6）none\_person\_bicycle：无人的自行车、电动自行车

（7）none\_person\_motorbike：无人的男式摩托车

（8）none\_person\_electric\_scooter：无人的电瓶车、电动车、女式摩托车

（9）none\_person\_tricycle：无人的人力三轮车

（10）none\_person\_auto\_tricycle：无人的电动三轮小货车、电动三轮载客车

（11）unknown\_person\_bicycle：未知有人的自行车、电动自行车**（此目标不会计入成绩）**

（12）unknown\_person\_motorbike：未知有人的男式摩托车**（此目标不会计入成绩）**

（13）unknown\_person\_electric\_scooter：未知有人的电瓶车、电动车、女式摩托车**（此目标不会计入成绩）**

（14）unknown\_person\_tricycle：未知有人的的人力三轮车**（此目标不会计入成绩）**

（15）unknown\_person\_auto\_tricycle：未知有人的电动三轮小货车、电动三轮载客车**（此目标不会计入成绩）**

（16）person：行人，路边的人、有推车动作的人等

（17）rider：骑车的人、坐在车上面的人  
（18）bicycle\_person：有人骑的自行车，一个大框框住人+车

（19）motorbike\_person：有人骑的摩托车，一个大框框住人+车，

（20）electric\_scooter\_person：有人骑的电瓶车、电动车、女式摩托车，一个大框框住人+车

（21）tricycle\_person：有人骑的人力三轮车，一个大框框住人+车

（22）auto\_tricycle\_person：有人骑的电动三轮小货车、电动三轮载客车，一个大框框住人+车

五、自动测试方法

* **实战榜测试逻辑：**当model\_data中的几种非机动车目标，在预设的一段时限内内，在某一个摄像头cid下的某个roi区域内，如果一直只有 无人机动车 出现，则需要进行报警，并需要在algorithm\_data中输出对应的roi区域坐标(roi区域坐标会由args参数传入，具体参加接口代码示例)。本榜单需要考虑的无人机动车标签如下：

none\_person\_bicycle：无人的自行车、电动自行车

none\_person\_motorbike：无人的男式摩托车

none\_person\_electric\_scooter：无人的电瓶车、电动车、女式摩托车

none\_person\_tricycle：无人的人力三轮车

none\_person\_auto\_tricycle：无人的电动三轮小货车、电动三轮载客车

也就是说，如果roi区域中长时间只出现上述标签，则需要报警，且报警目标为该roi区域

* **实战榜测试方法：**

**以下业务逻辑理解起来可能稍复杂，您可以参考实战榜python接口进行代码编写**

1. 业务测试的数据集由多个图片文件夹组成，每个图片文件夹模拟一个假想的摄像头，文件夹名字作为摄像头的cid命名。文件夹中的图片代表视频帧，并按照视频帧顺序进行命名。测试程序轮询各个文件夹，每次轮询到同一个文件夹时，取出下一帧图片，传入算法。测试程序每次传入一张图片给算法时，需要将此文件夹的名字（摄像头cid）作为ji\_calc\_image的args参数的一个字段传入算法。

2. 每个文件夹中附带一个表示本文件夹（摄像头）的roi区域的标注文件，只有处于roi区域中的无人非机动车车辆才应该被考虑是否属于违停。测试程序需要将roi区域作为ji\_calc\_image的args参数的一个字段传入算法(args为json字符串)。

3. 测试程序提供一个配置参数，来设置一个roi区域内连续只出现无人车辆达到多少帧时视为违停。只要roi区域内出现人或者有人的车辆，则整个roi区域不算做违停。程序需要为每1个文件夹（cid）中的每1个roi区域各自建立1个帧数记录变量，以记录该roi区域内的车辆信息。如果roi区域内的车辆信息符合以下2点，则变量记录的帧数需要 +1：

               a. 此roi区域内出现 无人非机动车，即以下标签中任意，：

               none\_person\_bicycle

               none\_person\_motorbike

               none\_person\_electric\_scooter

               none\_person\_tricycle

               none\_person\_auto\_tricycle

               b. 此roi区域内没有出现 以下任一标签：

                        person

                        rider

                        bicycle\_person

                        motorbike\_person

                        electric\_scooter\_person

                        tricycle\_person

                        auto\_tricycle\_person

4、如果轮询到某个文件夹的某张图片时，图片中某个roi对应的帧数记录变量超过了预设的违停帧数限制，则应该报警，而如果算法此时没有报警，则视为错误。

5、算法在json输出的algorithm\_data下面的target\_info中，输出需要报警的roi区域信息，roi的坐标直接从args中获得即可。测试程序将按照roi的坐标，将算法的json输出与ground truth进行比对，并以roi为测试单位计算算法的准确率。

* **实战榜测试指标**：报警准确的roi数目/总roi数目（其中 roi总数 为每一帧的roi数目之和）
* **模型榜测试描述：**

1. 统计所有标签的precision, recall, f1-score, 并在测试日志中输出

2. 除了unknown开头的目标以外的2d框标签的f1-score，作为模型榜的排名

六、算法输出接口

**SDK封装输出规范**

Python接口

**模型榜接口**

用户需要按照要求实现如下函数接口，发起测试时，系统会调用文件/project/ev\_sdk/src/ji.py，并将测试图片逐次送入process\_image接口，需要实现的程序接口：

import argparse

import numpy as np

import cv2

import json

def init():

"""

Initialize model

Returns: model

"""

return {}

def process\_image(net, input\_image, args=None):

"""

Do inference to analysis input\_image and get output

Attributes:

handle: algorithm handle returned by init()

input\_image (numpy.ndarray): image to be process, format: (h, w, c), BGR

args:

模型榜时, 此参数为空。

Returns: process result

"""

fake\_result = {}

fake\_result["algorithm\_data"] = {

"is\_alert": False,

"target\_count": 0,

"target\_info": []

}

fake\_result["model\_data"] = {}

fake\_result["model\_data"]["objects"] = [

{

"x": 1622,

"y": 677,

"width": 57,

"height": 69,

"confidence": 0.800463,

"name": "bicycle\_person"

}

]

return json.dumps(fake\_result, indent=4)

实战榜接口（以下代码供参考）

import logging as log

import json

from shapely.geometry import Polygon

log.basicConfig(level=log.DEBUG)

class\_names = ["none\_person\_bicycle", "none\_person\_motorbike", "none\_person\_electric\_scooter", "none\_person\_tricycle",

"none\_person\_auto\_tricycle"] # 无人非机动车的类别名称（使得roi需要报警的类别名称）

other\_class\_names = ['person', 'rider',

'bicycle\_person', 'motorbike\_person', 'electric\_scooter\_person', 'tricycle\_person',

'auto\_tricycle\_person'] # 人或者有人的非机动车，这些目标出现在roi区域内时，使得整个roi不需要报警

camera\_id\_num\_contain = {}

def is\_contain(r, boxs, width, height):

'''

输入r表示roi，输入boxs表示包含多个检测框的列表

函数的作用是判断roi里是否包含检测框

'''

roi = r.replace('POLYGON((', '').replace('))', '')

polygona = [(float(x.split(' ')[0]) \* width,

float(x.split(' ')[1]) \* height) for x in roi.split(',')]

ploy\_a = Polygon(polygona)

flag = False

for box in boxs:

poly\_b = Polygon([(box[0], box[1]), (box[2], box[1]),

(box[2], box[3]), (box[0], box[3])])

if ploy\_a.contains(poly\_b):

flag = True

return flag

return flag

def init():

model = "您的深度学习模型" # 开发者自己编写

return model

def process\_image(net, input\_image, args=None):

args = json.loads(args) # 把字符串转成字典

'''

args里的信息是这样的

{'alert\_count\_threshold': 3, 'is\_last': False, 'polygon\_1': ['POLYGON

0.36363636363636365 0.665,0.0015151515151515152 0.885,0.14242424242424243 0.9975,0.4681818181818182 0.7225

', 'POLYGON

0.7893939393939394 0.5825,0.7621212121212121 0.7,0.8575757575757575 0.7175,0.8712121212121212 0.59

'], 'cid': 'task\_non\_vehicles\_000003'}

您不需要用到 is\_last 字段

'''

log.info(f'args:{args}')

# roi包含非机动车的连续帧数的报警阈值

alert\_count\_threshold = args['alert\_count\_threshold']

camera\_id = args['cid'] # 摄像头的id

# 输入图像里的roi，有可能有多个roi的，因此它是一个列表。并且一个camera\_id里的每一帧的roi都是固定的，roi里的坐标值是归一化过后的

rois = args['polygon\_1']

if camera\_id not in camera\_id\_num\_contain:

camera\_id\_num\_contain[camera\_id] = {roi\_name: 0 for roi\_name in

rois} # 初始化camera\_id, 给camera\_id里的每个roi初始化数值，记录roi连续出现非机动车的帧数，初始值是0

height, width = input\_image.shape[:2]

results = net.detect(input\_image)

detect\_objs = []

# 记录输入图像里检测到的 其他非报警目标 和 报警目标(无人非机动车) 的检测框坐标

other\_boxs, not\_vehicle\_boxs = [], []

for k, det in enumerate(results):

x, y, width, height, name, score = det

obj = {

'name': name,

'x': x,

'y': y,

'width': width,

'height': height,

'confidence': float(score)

}

detect\_objs.append(obj)

if name in class\_names:

not\_vehicle\_boxs.append([x, y, width, height]) # 当前目标是无人非机动车(报警目标)

elif name in other\_class\_names:

other\_boxs.append([x, y, width, height]) # 当前目标是人或者有人的非机动车(非报警目标)

for roi\_name in rois:

contain\_not\_vehicle = is\_contain(

roi\_name, not\_vehicle\_boxs, width, height) # 判断当前roi里是否包含无人非机动车

contain\_others = not is\_contain(

roi\_name, other\_boxs, width, height) # 判断当前roi里是否包含其他非报警目标

if contain\_not\_vehicle and contain\_others: # 当前roi 包含无人非机动车 且 不包含其他非报警目标

camera\_id\_num\_contain[camera\_id][roi\_name] += 1 # 连续帧的数值加1

else:

camera\_id\_num\_contain[camera\_id][roi\_name] = 0 # 连续帧的数值设置0

target\_info = [] # 记录报警roi

for roi\_name in camera\_id\_num\_contain[camera\_id]:

# 如果当前roi的包含非机动的连续帧大于报警阈值

if camera\_id\_num\_contain[camera\_id][roi\_name] > alert\_count\_threshold:

# 报警roi的信息添加到target\_info里，其中time\_elapsed不纳入精度计算里

target\_info.append({'roi': roi\_name, 'time\_elapsed': 1})

result = {'algorithm\_data': {'is\_alert': True, 'target\_info': target\_info},

'model\_data': {"objects": detect\_objs}}

return json.dumps(result)

'''

测试指标 = 报警准确的roi数目 / 测试集里的总roi数目

'''

C++接口

请参考 /usr/local/ev\_sdk/include/ji.h

ev\_sdk 4.0接口如下

JiErrorCode ji\_init(int argc, char \*\*argv);

void\* ji\_create\_predictor(JiPredictorType pdtype);

JiErrorCode ji\_calc\_image(void\* predictor, const JiImageInfo\* pInFrames, const unsigned int nInCount, const char\* args, JiImageInfo \*\*pOutFrames, unsigned int & nOutCount, JiEvent &event);

接⼝的返回值 event.json 需要满⾜算法输出规范。

七、json输出规范

注意，报警的逻辑为：

在某个目标roi区域内，一直只存在无人的非机动车车辆，并且时间超过预设的时限，才进行报警，并且报警目标为该区域，而不是车辆本身。

示例

不告警情形

{

"algorithm\_data": {

"is\_alert": false,

"target\_count": 0,

"target\_info": []

},

"model\_data": {

"objects": [

{

"x": 716,

"y": 716,

"height": 646,

"width": 233,

"confidence": 0.999660,

"name": "bicycle\_person"

}

]

}

}

告警情形示例

{

"algorithm\_data": {

"is\_alert": true,

"target\_count": 2,

"target\_info": [

{

"roi": "POLYGON((0.36363636363636365 0.665,0.0015151515151515152 0.885,0.14242424242424243 0.9975,0.4681818181818182 0.7225))",

"time\_elapsed": 1

},

{

"roi": "POLYGON((0.7893939393939394 0.5825,0.7621212121212121 0.7,0.8575757575757575 0.7175,0.8712121212121212 0.59))",

"time\_elapsed": 1

}

]

},

"model\_data": {

"objects": [

{

"x": 397,

"y": 397,

"height": 488,

"width": 215,

"confidence": 0.978979,

"name": "none\_person\_bicycle"

},

{

"x": 716,

"y": 716,

"height": 646,

"width": 233,

"confidence": 0.999660,

"name": "none\_person\_electric\_scooter"

}

]

}

}

说明

target\_info的值类型为列表，列表可以包括n个元素；  
每个元素必须包含：矩形目标框、目标类别；  
如果包含置信度，键名为confidence；  
目标类别名称使用name表示，且其值必须可以唯一区分当前目标所属的类别。

字段详解

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 字段名称 | 类型 | 备注 |
| 1 | algorithm\_data | Object | 业务相关的输出信息 |
| 2 | is\_alert | bool | 报警标识, true告警, false不告警 |
| 3 | target\_count | int | target\_info中对象数目 |
| 4 | target\_info | Array | 报警信息 |
| 5 | model\_data | Object | 模型相关的输出信息 |
| 6 | objects | Array | 模型输出信息 |

target\_info字段详解:

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 字段名称 | 类型 | 备注 |
| 1 | roi | string | 目标限制非机动车违停区域（roi区域）的坐标，格式为：  "POLYGON((0.36363636363636365 0.665,0.0015151515151515152 0.885,0.14242424242424243 0.9975,0.4681818181818182 0.7225))"  可以直接从输入参数args中将字段值复制过来。 |
| 2 | time\_elapsed | string | 以容易读的形式返回违停的时间：  该区域内一直包含无人违停非机动车（且不包含非报警目标）的时间长度，精确到分钟。  **该输出不会作为测试标准。** |

objects字段详解:

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 字段名称 | 类型 | 备注 |
| 1 | x | int | 识别框左上角X坐标 |
| 2 | y | int | 识别框左上角Y坐标 |
| 3 | width | int | 识别框宽度 |
| 4 | height | int | 识别框高度 |
| 5 | confidence | double | 置信度 |
| 6 | name | String | 模型检测到的目标名称（与标注对应）:  （1）bicycle：自行车、电动自行车，不管是否有人骑（如果无人骑时，此标注与none\_person\_bicycle标签范围相同）  （2）motorbike：男式摩托车，不管是否有人骑（如果无人骑时，此标注与none\_person\_motorbick标签范围相同）  （3）electric\_scooter：电瓶车、电动车、女式摩托车，不管是否有人骑（如果无人骑时，此标注与none\_person\_electric\_scooter标签范围相同）  （4）tricycle：人力三轮车，不管是否有人骑（如果无人骑时，此标注与none\_person\_tricycle标签范围相同）  （5）auto\_tricycle：电动三轮小货车、电动三轮载客车，，不管是否有人骑（如果无人骑时，此标注与none\_person\_auto\_tricycle标签范围相同）  （6）none\_person\_bicycle：无人的自行车、电动自行车  （7）none\_person\_motorbike：无人的男式摩托车  （8）none\_person\_electric\_scooter：无人的电瓶车、电动车、女式摩托车  （9）none\_person\_tricycle：无人的人力三轮车  （10）none\_person\_auto\_tricycle：无人的电动三轮小货车、电动三轮载客车  （11）unknown\_person\_bicycle：未知有人的自行车、电动自行车**（此目标不会计入成绩）**  （12）unknown\_person\_motorbike：未知有人的男式摩托车**（此目标不会计入成绩）**  （13）unknown\_person\_electric\_scooter：未知有人的电瓶车、电动车、女式摩托车**（此目标不会计入成绩）**  （14）unknown\_person\_tricycle：未知有人的的人力三轮车**（此目标不会计入成绩）**  （15）unknown\_person\_auto\_tricycle：未知有人的电动三轮小货车、电动三轮载客车**（此目标不会计入成绩）**  （16）person：行人，路边的人、有推车动作的人等  （17）rider：骑车的人、坐在车上面的人 （18）bicycle\_person：有人骑的自行车，一个大框框住人+车  （19）motorbike\_person：有人骑的摩托车，一个大框框住人+车，  （20）electric\_scooter\_person：有人骑的电瓶车、电动车、女式摩托车，一个大框框住人+车  （21）tricycle\_person：有人骑的人力三轮车，一个大框框住人+车  （22）auto\_tricycle\_person：有人骑的电动三轮小货车、电动三轮载客车，一个大框框住人+车 |

备注:  
1. 其中algorithm\_data字段对应的是业务数据，只有当检测到有至少1个告警目标时才会使得  
target\_info不为空集。  
2. model\_data字段对应的是模型数据，只要模型检测到目标，则objects不为空；否则为空[]