Robotics, Artificial Intelligence
and Real-time Systems

Simulation-Based Autonomous Driving in Crowed City

Guided by Ph.D Liguo Zhou

Presented by Hao Yang, Danyang Long, Xinyue Dai

Content

End to End Learning

- **What is it**

A machine learning approach:

Goal:

Aim to achieve a unified learning process from input to output without decomposition into multiple stages.

Feature:

Simplify complex systems by minimizing intermediate steps and allowing models to automatically learn from raw data to a representation and solution of the final task

End to End Learning

- **Advantage**

a) Simplified Complexity

no intermediate steps and components

optimizes all processing steps simultaneously, leading to better performance and smaller systems

b) Automatic Feature Learning

automatically learn useful features from raw data, reducing the need for manual feature engineering

End to End Learning

- **Advantage**

c) End-to-End Optimization

Emphasis on input-to-output optimization, optimizing not just one component, but the performance of the entire system

Traditional Method	End to End Learning
transducers	data collection
Environmental awareness	choosing model
decision-making and planning	training model
control	Evaluation and Adapting
	Deployment

End to End Learning

- **Disadvantage**

a) Big data requirement

No explicit intermediate steps to handle data

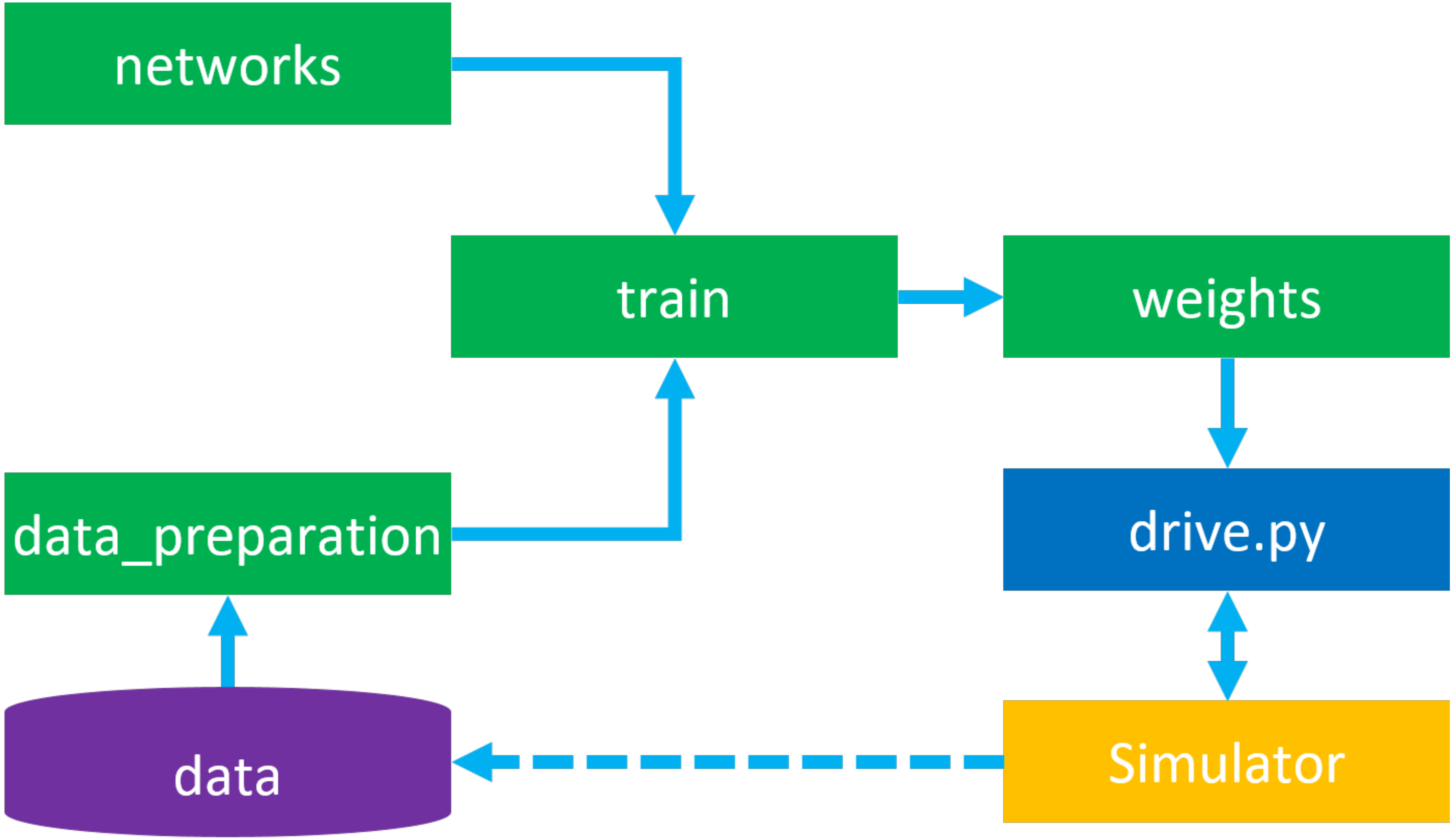
⬇

Large scale data to guarantee generalization

⬇

Need more computing source and time

b)Poor interpretation


networks
train
weights
data_preparation
drive.py
data
Simulator

Structure of Project

- Collection of datasets
- Preprocess of images
- Construction of Networks
- Train: get weights for networks
- Drive.py: combine the output of two networks and send control to simulator

Resnet50

- **Dataset**: recording from simulator (1280x460), only one camera
- **Input**: 224x224
- **Output**: throttle, brake, steering angle
- **Preprocess methods**:

a. def letterbox
b. Add Gaussian noise
c. Random flip
d. Random shadow
e. Random brightness

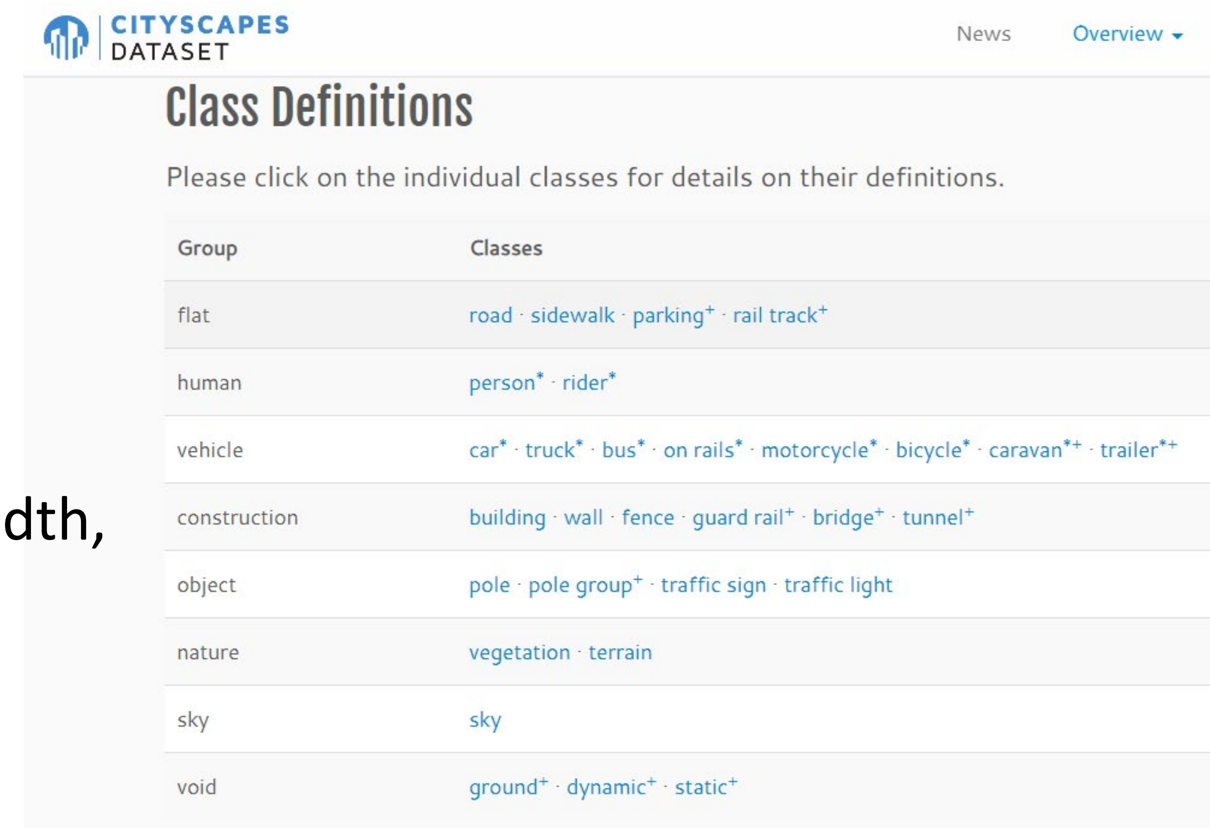
Yolov5

- **Dataset**: Cityscape dataset
- **Class**: car (0), traffic light (1), bus (2)
- **Output**:

a. Matrix: (x_middle_point, y_middle_point, width, height, confidence, class)

b. Rectangles containing detected object

- **Use**: traffic light and vehicle detection



Class Definitions

Please click on the individual classes for details on their definitions.

Group	Classes
flat	road · sidewalk · parking+ · rail track+
human	person* · rider*
vehicle	car* · truck* · bus* · on rails* · motorcycle* · bicycle* · caravan*+ · trailer*+
construction	building · wall · fence · guard rail+ · bridge+ · tunnel+
object	pole · pole group+ · traffic sign · traffic light
nature	vegetation · terrain
sky	sky
void	ground+ · dynamic+ · static+

+ can output whether there is traffic light and vehicles in front of our car

- Some long bus or car beside our car may also be detected as in front of us

Yolov5

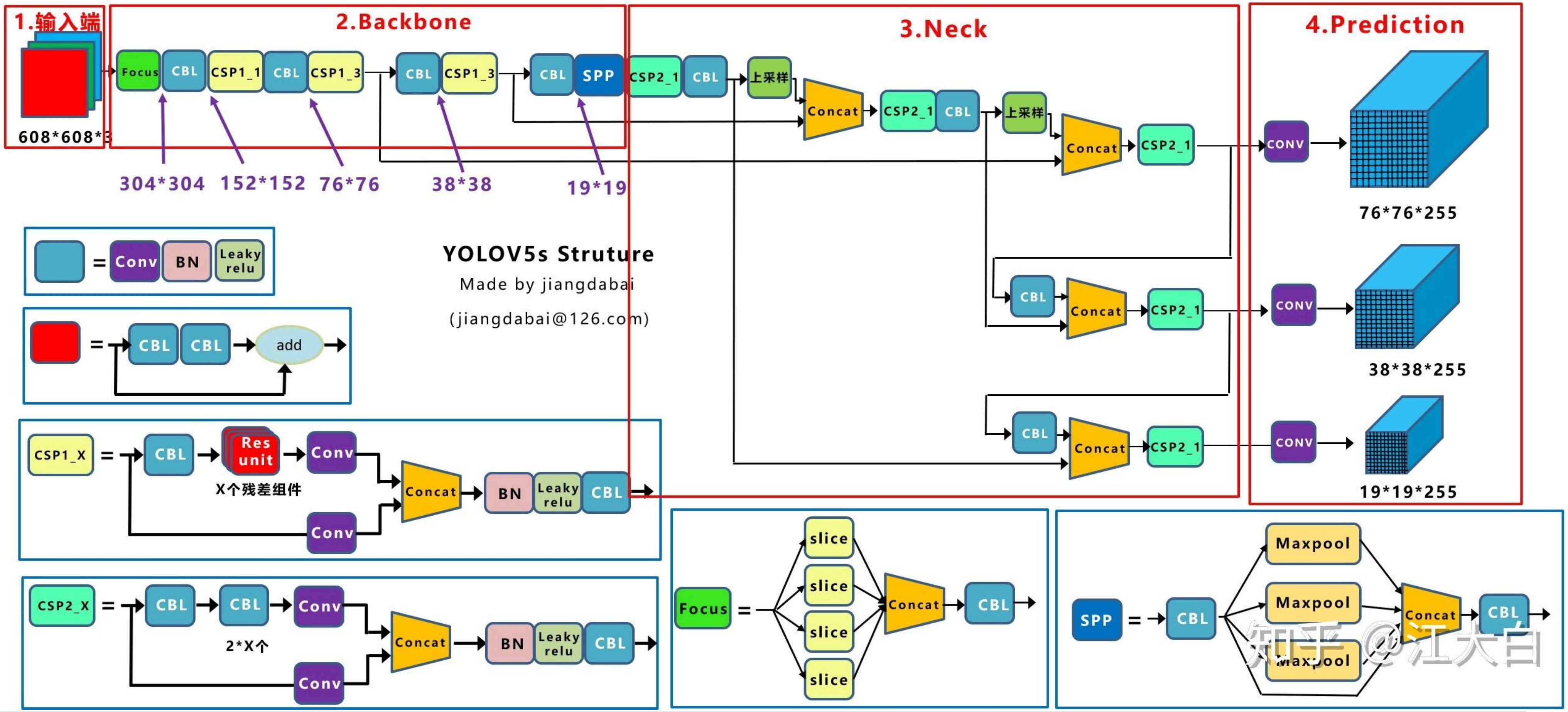
- **Components**:

a. Input

b. Backbone

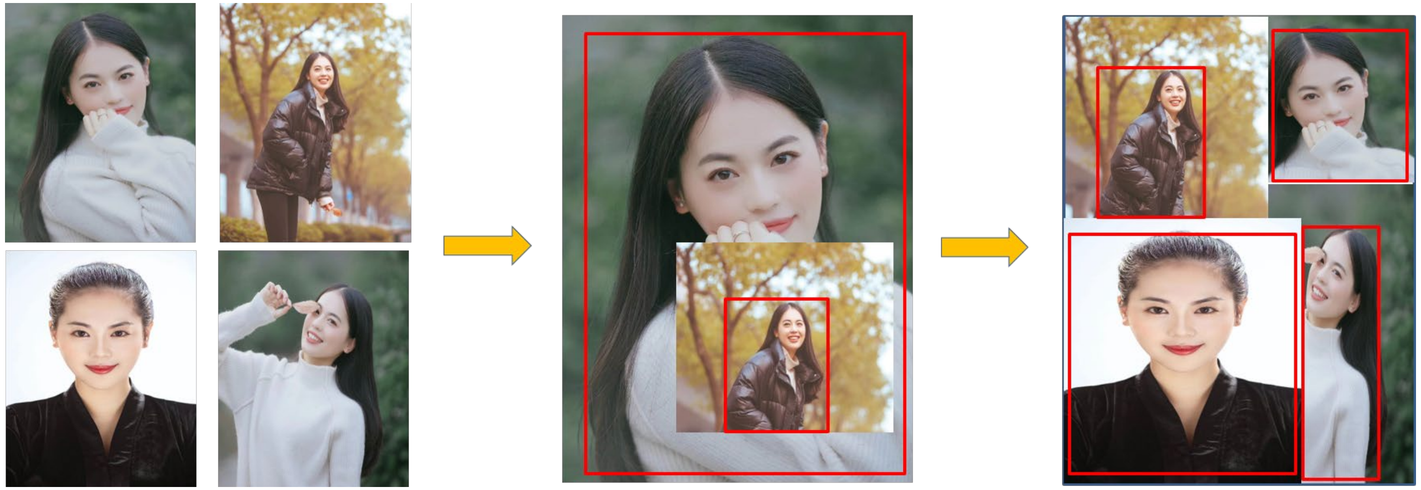
c. Neck

d. Prediction



Yolov5

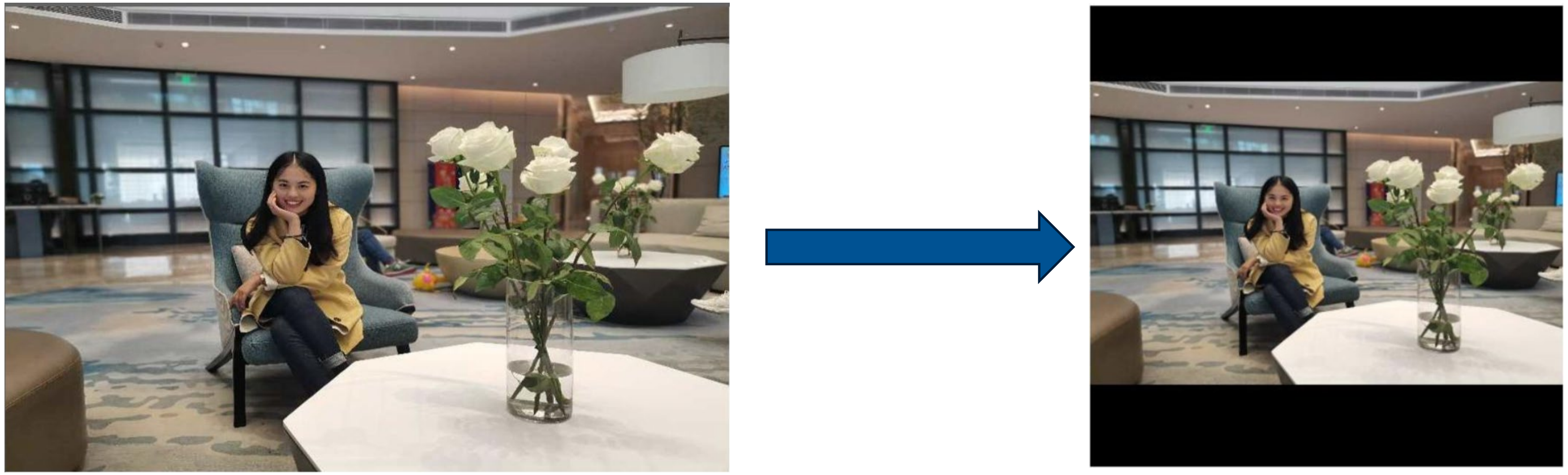
- **Mosaic Augmentation**:
- The Mosaic references the CutMix data augmentation which uses only two images for stitching.
- The Mosaic data augmentation used four images, randomly scaled, randomly cropped, and randomly aligned, for stitching.



Original Pic. (left) CutMix (middle) Mosaic (right) [2]

Yolov5

- **Letterbox function:**
- It’s used to resize the input image into fixed size without changing its original ratio between height and width, which improves inference speed



800x600 (left) to 416x416 (right) [2]

Yolov5

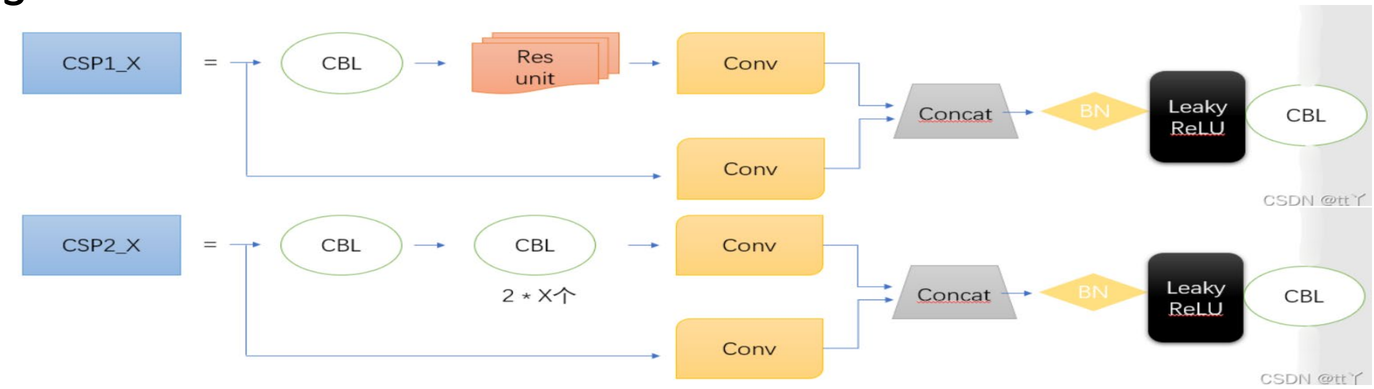
- **CSP Structure**(Cross Stage Paritial Network)[1]: modify Darknet53 and get CSPDarknet53 to construct backbone

a) Split the original input into two branches

b) Each one performs a convolution operation to halve the number of channels

c) Then one branch performing a Bottleneck * N operation

d) Then concatenate the two branches



Two kinds of CSP structures used in yolov5 [4]

Yolov5

- **CSP Structure**(Cross Stage Paritial Network)[1]:

a) It integrates changes in the gradient into the feature map from beginning to end

b) It solves the problem of repeating gradient information for network optimization in Backbone

c) It reduces the number of parameters and FLOPS values of the model

d) It ensures both inference speed and accuracy

e) It reduces the model size.

Yolov5

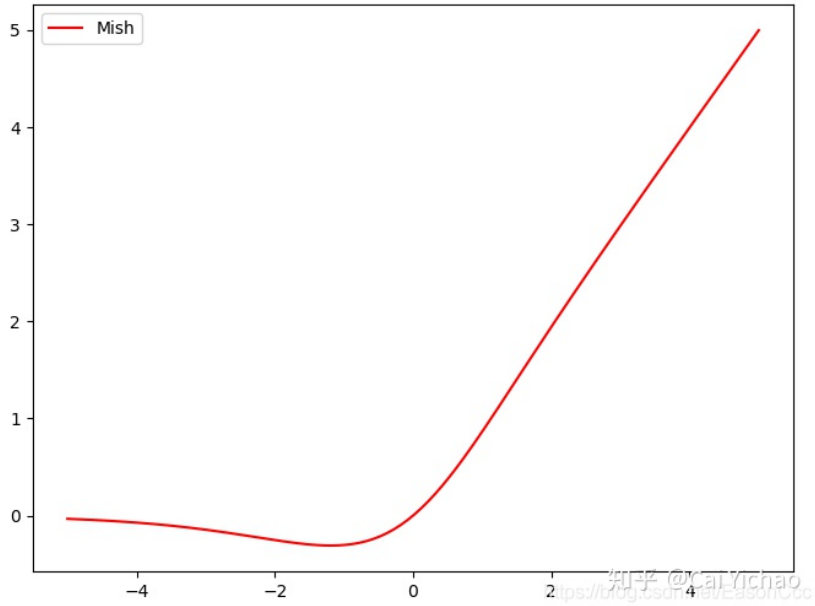
- **Mish Activation**:

a. $y_{mish} = x \ * \tanh(\ln(1 + e^{x}))$

b. No positive boundary, which avoids gradient saturation;

c. The Mish function is smooth everywhere and allows some negativity in the negative region with small absolute values

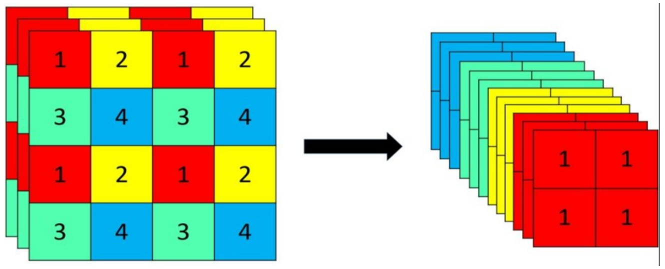
d. Slightly enhance the accuracy



Mish activation [3]

Yolov5

- **Focus Module:**
- FLOPs reduction and speed increase.
- (One focus module replaces 3 yolov3/4 layers)



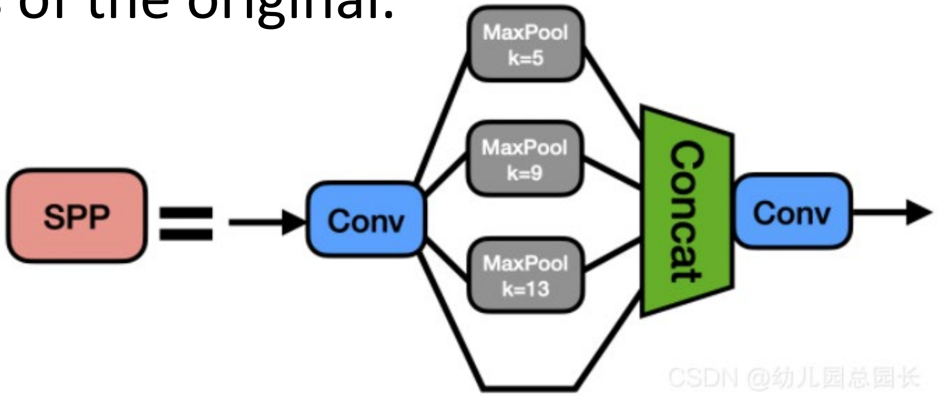
Process of focus module [3]

Yolov5

- **SPP Module** (Spatial Pyramid Pooling):
- SPP first halves the input channels by a standard convolution module
- Then it does maxpooling with kernel-sizes of 5, 9, and 13 respectively (for different kernel sizes, the padding is adaptive)
- The result of the three maxpooling is concatenated with the data without pooling operation
- The final number of channels after merging is 2 times of the original.

a. Extract and combine local and full features of image

b. Enhance the detection accuracy



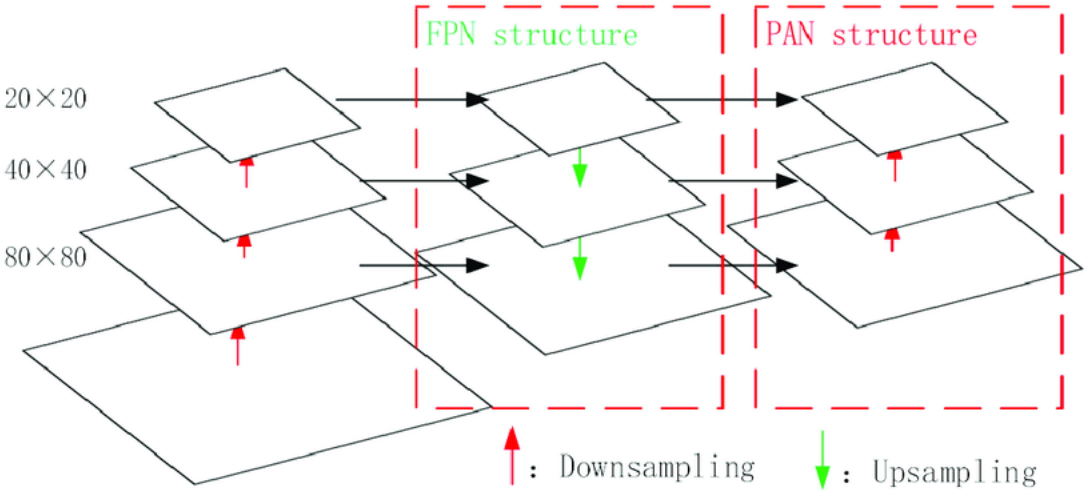
Process of SPP [2]

Yolov5

- **FPN+PAN**:

a. The FPN structure delivers strong semantic features from top to bottom

b. The PAN structure delivers strong positioning features from the bottom upwards.



Structure of FPN and PAN [4]

Yolov5

- **NMS** (Non-Max Suppression):
- Pick out all boxes with same classification
- Compare the IoU (Intersection over Union), if IoU > threshold (e.g., 0.5), delete one with smaller confidence
- The saved boxes with same classification have low IoU and high confidence

a. Keep only one box for each object

b. Anchor is also used to fit object in different shapes

Traffic Light Detection and Collision Detection

Traffic light detection:

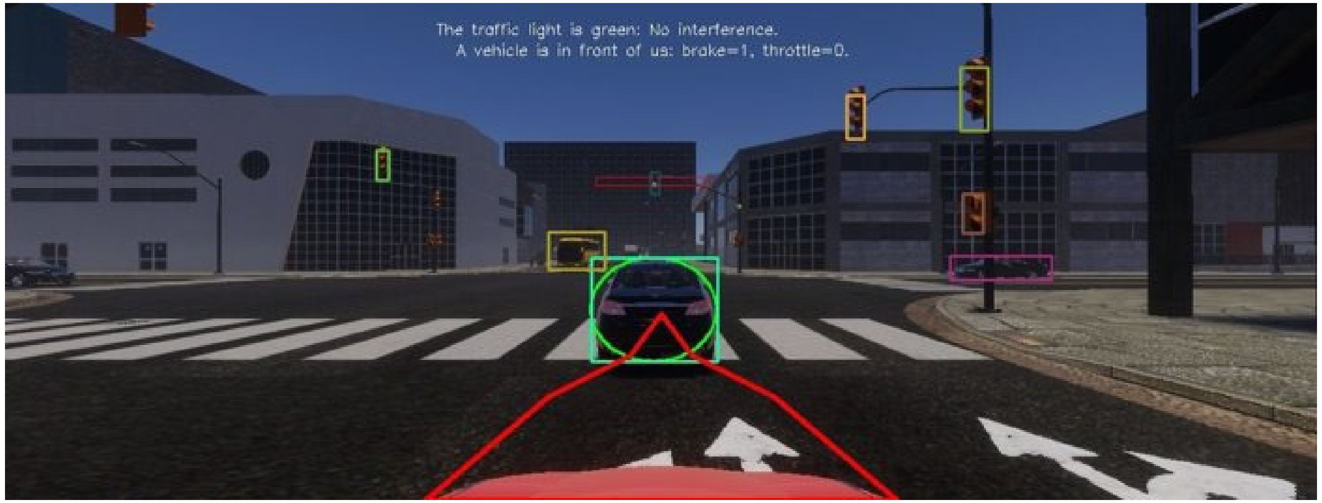
1. Is the center of traffic light in our red rectangle? (both width and height)
2. If yes, is it a red light?

Collision detection:

1. Is a car near the polygon detection area?
2. If yes, draw the ellipse area. Does it intersect with our polygon?

Overall detection logic:

1. Traffic light detection
2. Collision detection



Traffic Lights Detection – Logic

- Due to the poor performance of using only Method 2, we use the combination of both.
- Label 1 is the result of improved Method 1, Label 3 is the result of Method 2.
- In case the traffic light is red and our detection is green, we do a red_mask_feature.

```
def red_detector(rgb_image):
    label1 = create_feature(rgb_image) # Method 1
    label3 = rgb_detector(rgb_image) # Method 2
    h,s,v= red_mask_feature(rgb_image)

    if label1 != [1,0,0]  and h>0:
        if h>0:
            label = [1,0,0]
    elif label1 == label3 and h == 0.0:
        label = label1
    else:
        label = label3

    return label
```

Traffic Lights Detection – Methods

Method 1 (inspired by [5])

1. Do red, yellow, green masks
2. Slice the image into up, middle, and down
3. Assign labels based on the value of brightness

```
# Find all misclassified images in a given test set
MISCLASSIFIED = get_misclassified_images(STANDARDIZED_TEST_LIST)

# Accuracy calculations
total = len(STANDARDIZED_TEST_LIST)
num_correct = total - len(MISCLASSIFIED)
accuracy = num_correct/total

print('Accuracy: ' + str(accuracy))
print("Number of misclassified images = " + str(len(MISCLASSIFIED)) +' out of '+ str(total))
```

```
Accuracy: 0.9292929292929293
Number of misclassified images = 21 out of 297
```

Method 2 (inspired by [6]):

1. Give the color range of green, yellow, and red
2. Calculate pixel by pixel
3. Do average

```
# Find all misclassified images in a given test set
MISCLASSIFIED = get_misclassified_images(STANDARDIZED_TEST_LIST)

# Accuracy calculations
total = len(STANDARDIZED_TEST_LIST)
num_correct = total - len(MISCLASSIFIED)
accuracy = num_correct/total

print('Accuracy: ' + str(accuracy))
print("Number of misclassified images = " + str(len(MISCLASSIFIED)) +' out of '+ str(total))
```

✓ 1.5s

```
Accuracy: 0.9764309764309764
Number of misclassified images = 7 out of 297
```

Based on Dataset from [5]

Traffic Lights Detection – Misclassification

In our own dataset, we have 135, 60, and 132 images in our own testing dataset, respectively for red, yellow, and green traffic lights. We did a test on the dataset from [5] and on our own dataset. We got a better result.

```
print('Accuracy: ' + str(accuracy))
print("Number of misclassified images = " + str(len(MISCLASSIFIED)) +' out of '+ str(total))
```

✓ 1.5s

```
Accuracy: 0.9831649831649831
Number of misclassified images = 5 out of 297
```

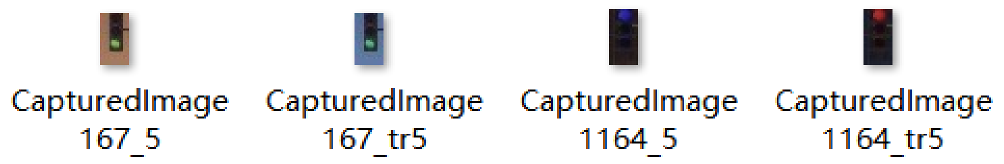
✓ 1.7s

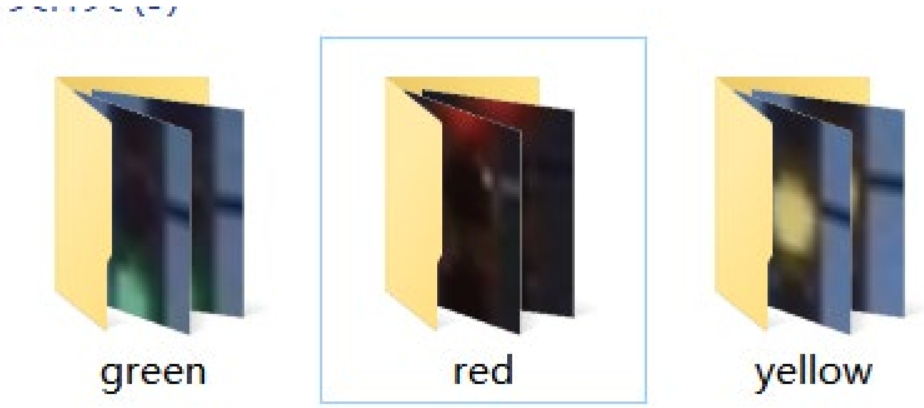
```
Accuracy: 0.9847094801223242
Number of misclassified images = 5 out of 327
```

Own dataset

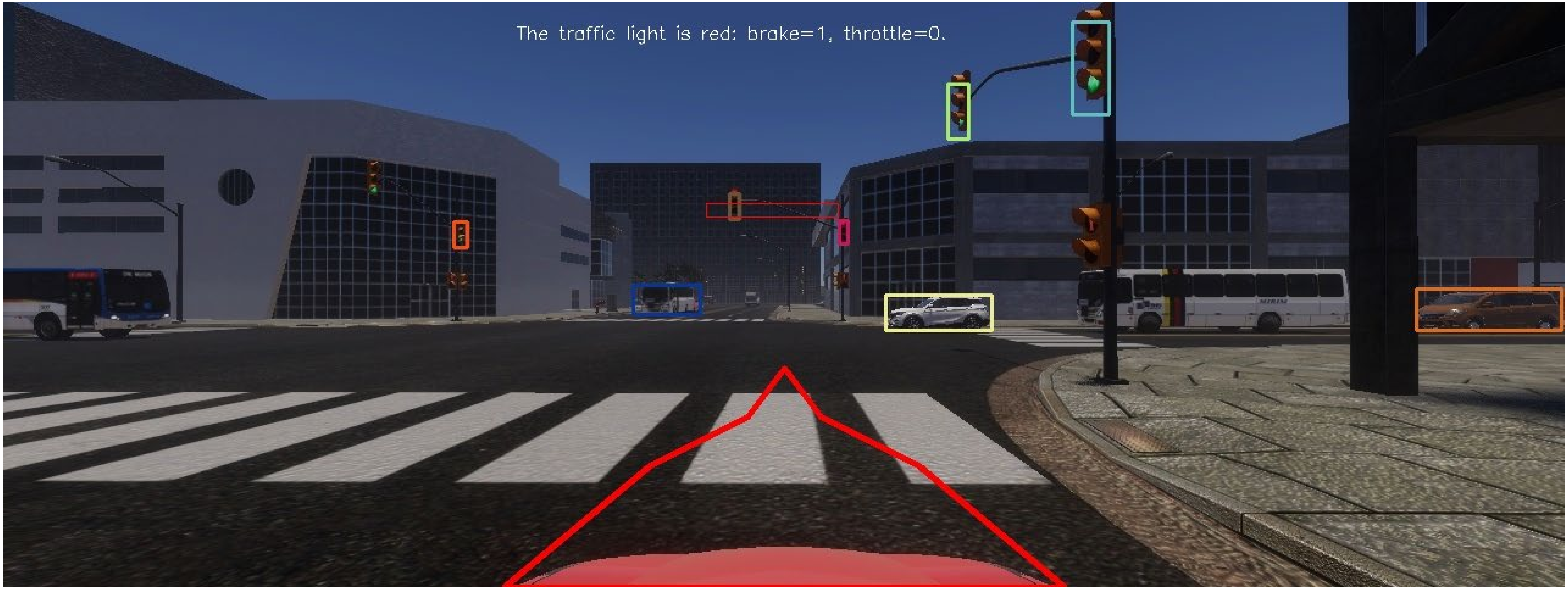
Traffic Lights Detection – Data Preprocessing

1. LoadImages
2. Yolov5 detection
3. Get predictions
4. Set a range for traffic lights
5. Crop the original image
6. Output traffic light images to a folder under primary Dataset

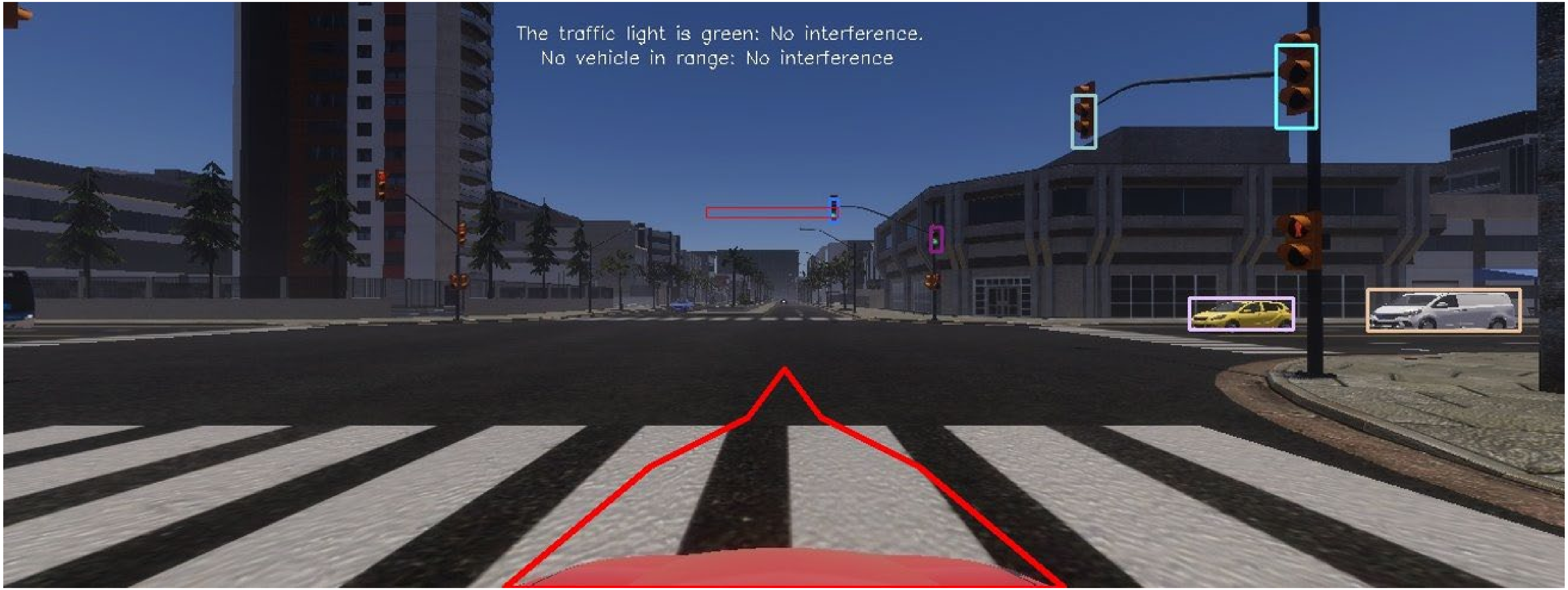




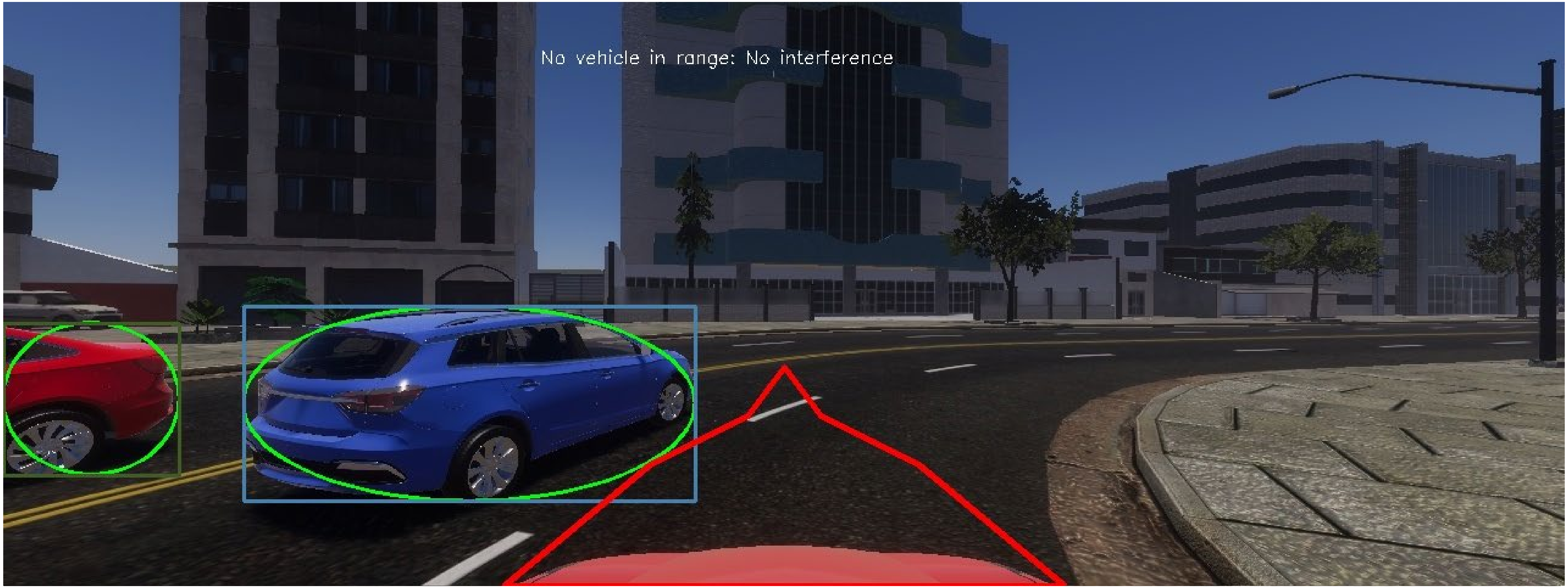
Traffic Lights Detection – Results



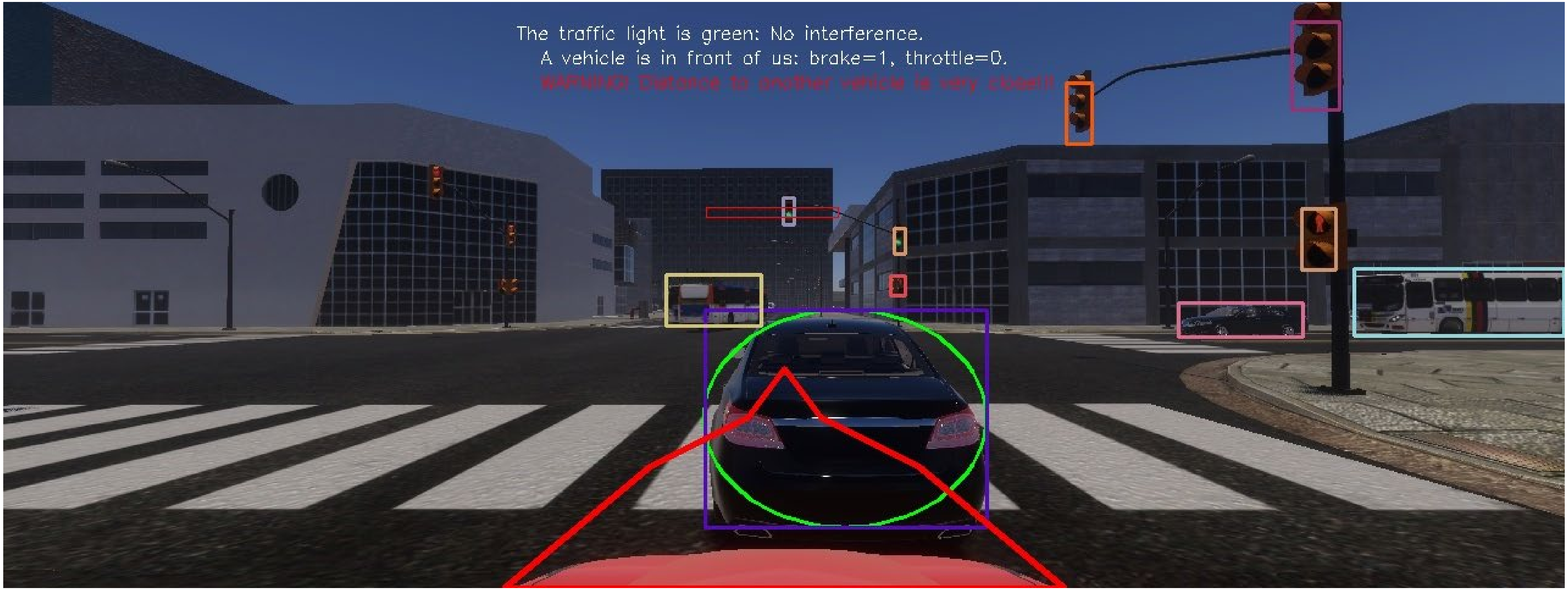
Collision Detection - Results



Collision Detection – Extreme Case



Detection Results



Detections

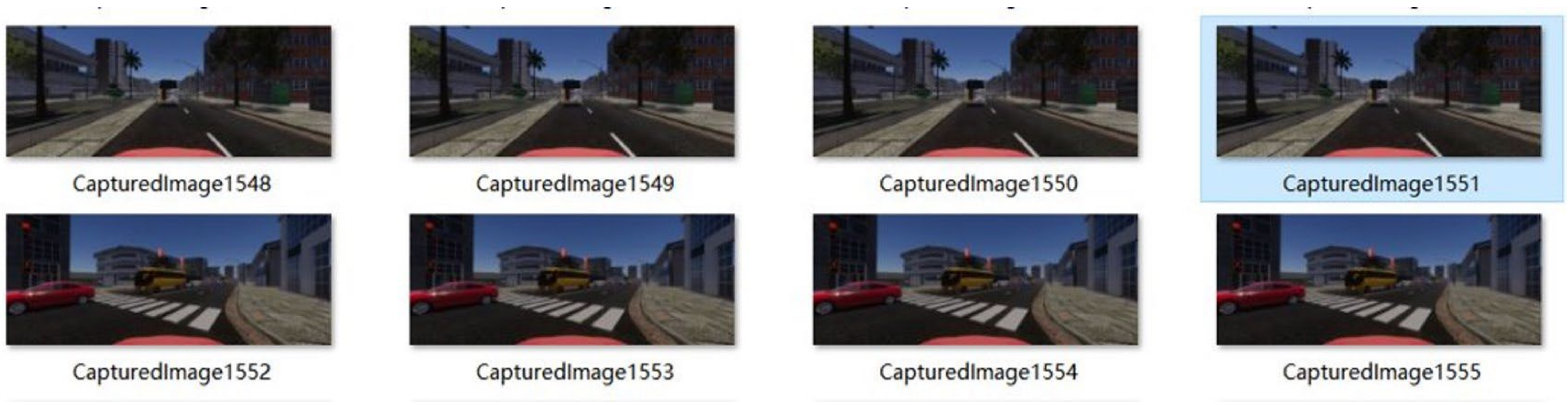
```
# for detecting bus and car
if pred[5] == 2 or pred[5] == 0:
    if 470 > h_max > h_min_polygon:
        center = (int((w_min+ w_max)/2), int((h_min+ h_max)/2))
        width = abs(w_max - w_min)
        height = abs(h_min- h_max)

        ellipse = Ellipse(center, width, height, angle=0)
        vertices = ellipse.get_verts()     # get the vertices from the ellipse object
        ellipse = Polygon(vertices)
        exterior_coords2 = list(ellipse.exterior.coords)
        contour2 = np.array(exterior_coords2, dtype=np.int32)
        cv2.drawContours(image, [contour2], 0, (0, 255, 0), thickness=1)

        if polygon.intersects(ellipse):
            veh_detect = True
```

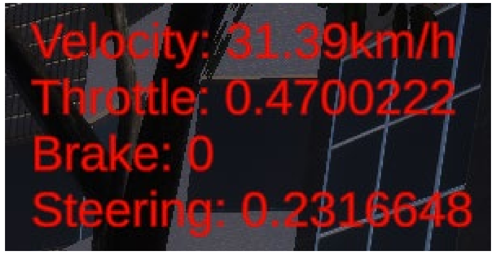
Problem we met: during the traning session

- The traffic light is unreasonable long time to wait, therefore larger dataset and unuseful traning data.
- Pieces of a dataset are not continuous.



Problem we met: during the traning session

- Only integer inputs for keyboard users, but the brake sometimes does not work for the controller users

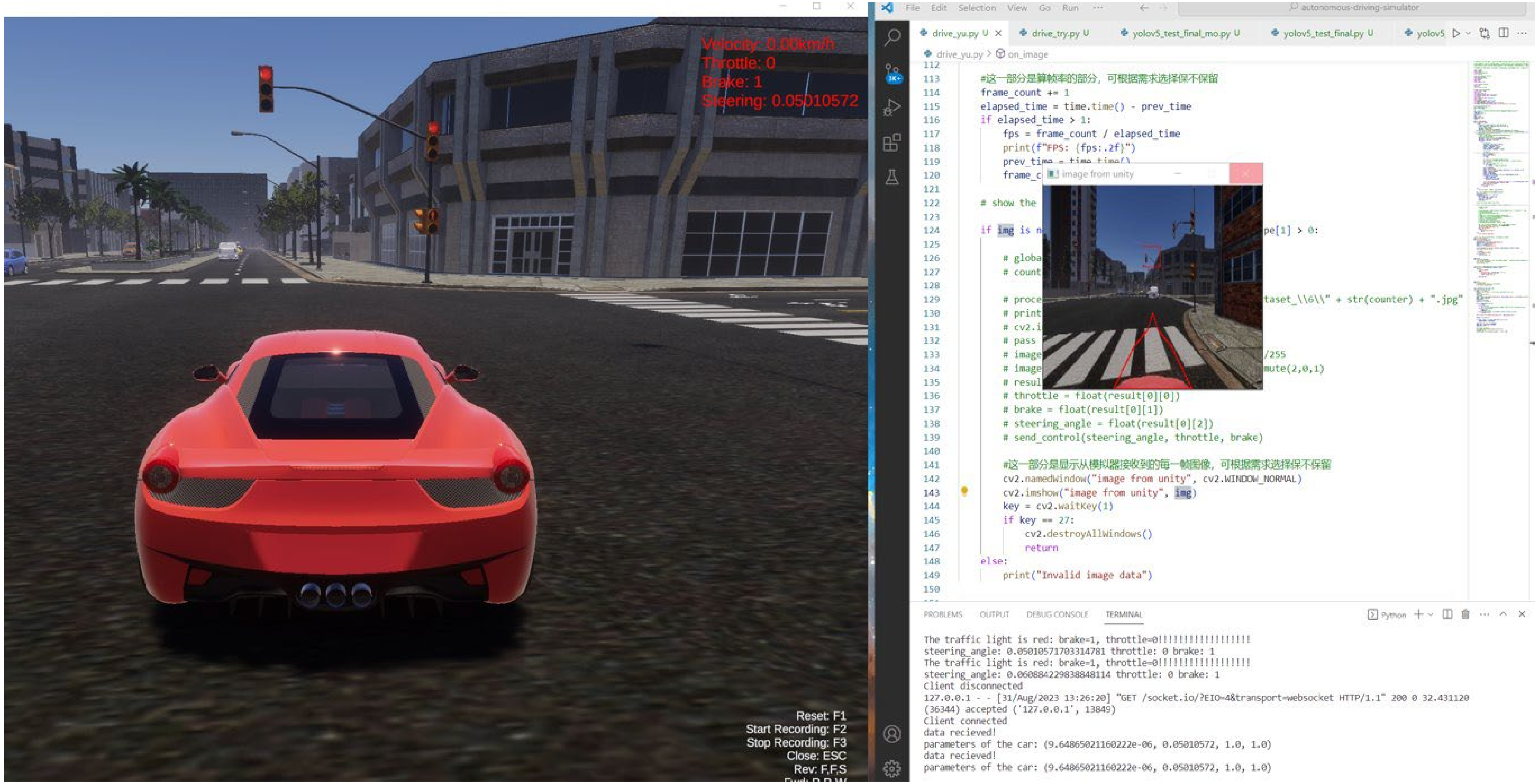


Problem we met: during the traning session

- Traffic rules do not exist. (video)

Problem we met: during the test session

- Servere lag



Problem we met: during the test session

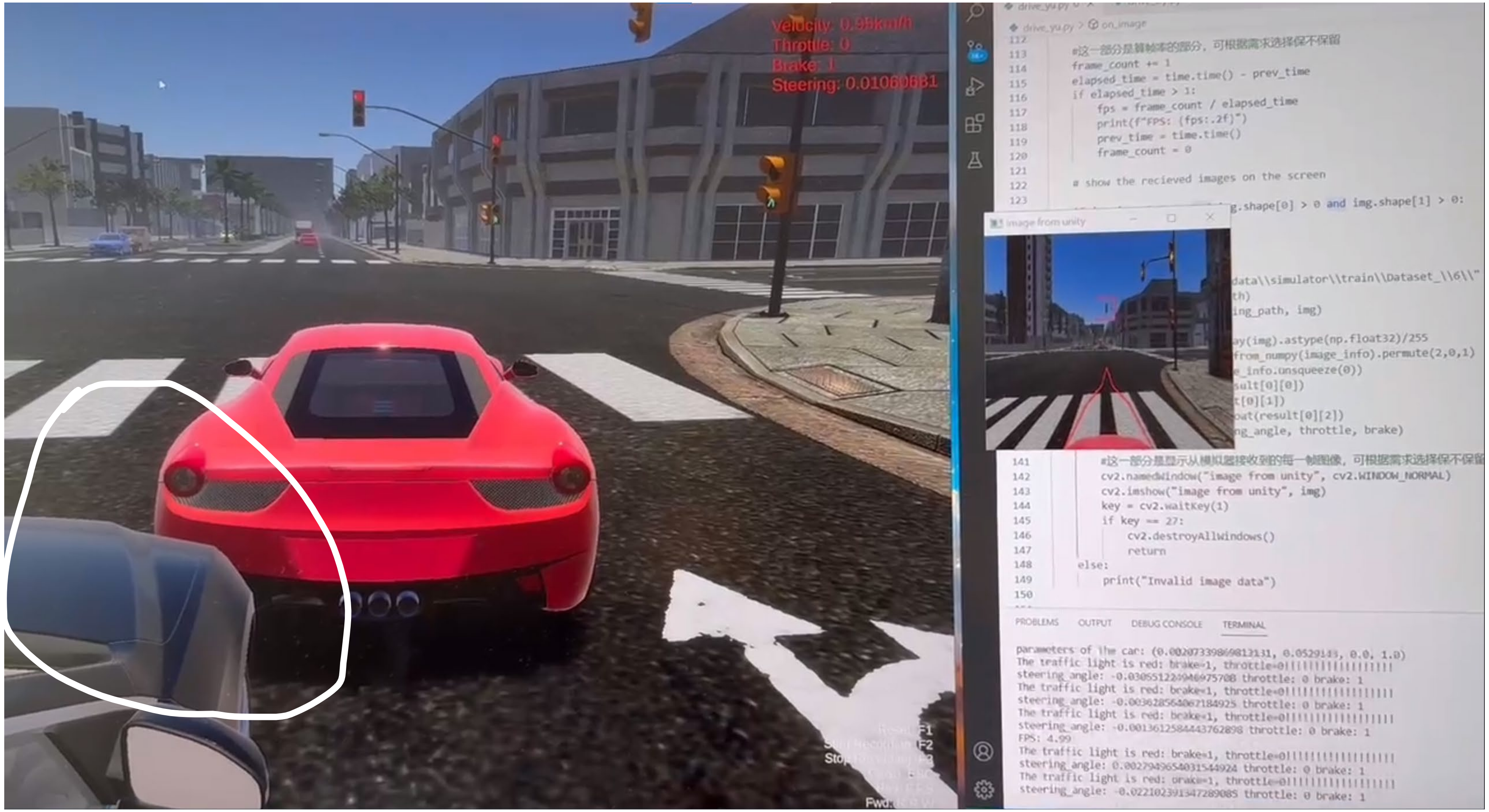
- Unstable connection

```
steering_angle: 0.12275790423154831 throttle: 0 brake: 1
data\simulator\train\Dataset_\5\718.jpg
The traffic light is red: brake=1, throttle=0!!!!!!!!!!!!!!!!!
steering_angle: 0.1255977302789688 throttle: 0 brake: 1
data\simulator\train\Dataset_\5\719.jpg
The traffic light is red: brake=1, throttle=0!!!!!!!!!!!!!!!!!
steering_angle: 0.1311405599117279 throttle: 0 brake: 1
data\simulator\train\Dataset_\5\720.jpg
The traffic light is red: brake=1, throttle=0!!!!!!!!!!!!!!!!!
steering_angle: 0.12756134569644928 throttle: 0 brake: 1
FPS: 5.86
data\simulator\train\Dataset_\5\721.jpg
data recieved!
parameters of the car: (0.00231157906819135, 0.1382471, 0.0, 1.0)
No interference.
steering_angle: 0.13492977619171143 throttle: 0.9593274593353271 brake: -0.0278029665350914
data\simulator\train\Dataset_\5\722.jpg
The traffic light is red: brake=1, throttle=0!!!!!!!!!!!!!!!!!
steering_angle: 0.1392003297805786 throttle: 0 brake: 1
data\simulator\train\Dataset_\5\723.jpg
No interference.
steering_angle: 0.13653279840946198 throttle: 0.9599994421005249 brake: -0.03159686177968979
data\simulator\train\Dataset_\5\724.jpg
No interference.
steering_angle: 0.1301477551460266 throttle: 0.9532437324523926 brake: -0.026919439435005188
```

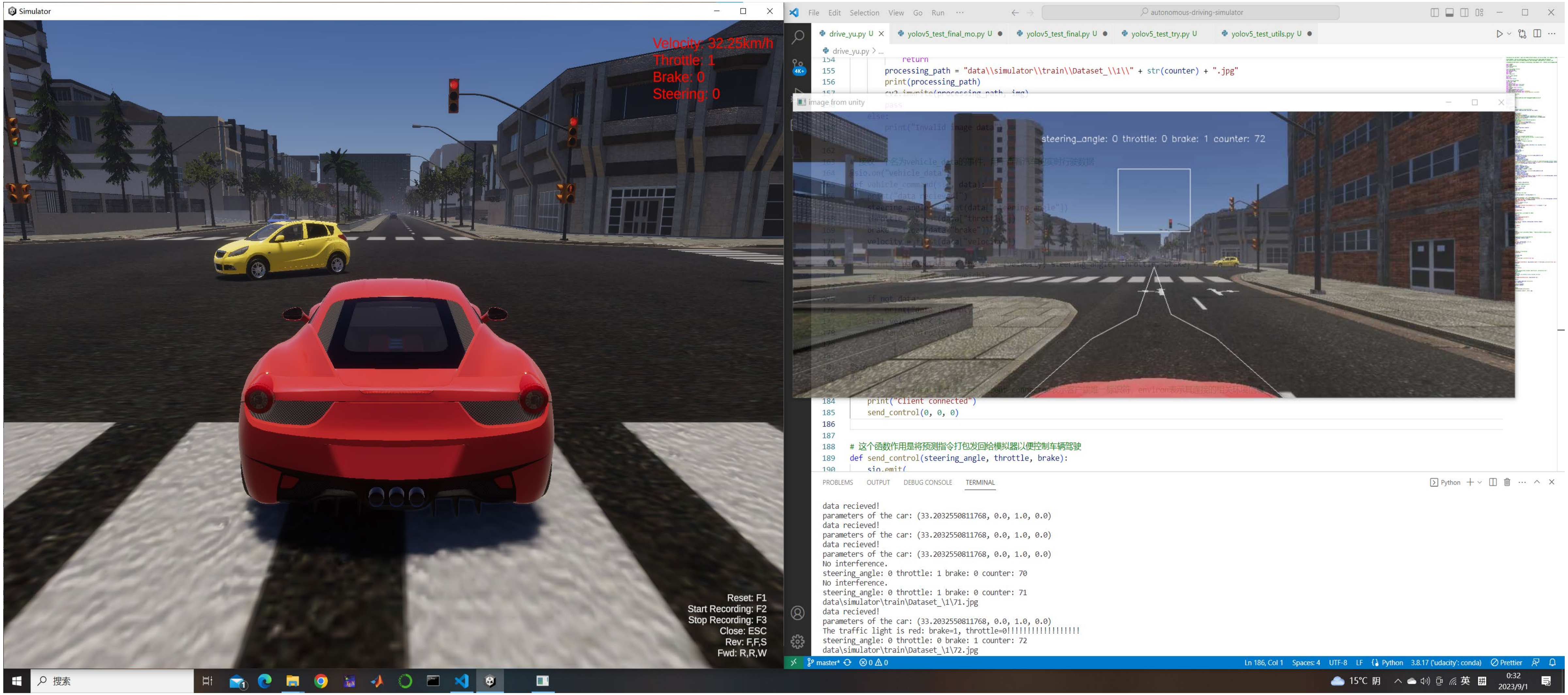
Then, the traffic light is out of my detection area :(

Problem we met: during the test session

- Even with good GPU: RTX 3070 Ti, the simulator lags.
- The car after us does not stop (video).



(video)


Simulator
Velocity: 32.25km/h
Throttle: 1
Brake: 0
Steering: 0
Reset: F1
Start Recording: F2
Stop Recording: F3
Close: ESC
Rev: F,F,S
Fwd: R,R,W
image from unity
steering_angle: 0 throttle: 0 brake: 1 counter: 72
processing_path = "data\\simulator\\train\\Dataset_\\1\\" + str(counter) + ".jpg"
print(processing_path)
print("Client connected")
send_control(0, 0, 0)
# 这个函数作用是将预测指令打包发回给模拟器以便控制车辆驾驶
def send_control(steering_angle, throttle, brake):
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
data recieved!
parameters of the car: (33.2032550811768, 0.0, 1.0, 0.0)
data recieved!
parameters of the car: (33.2032550811768, 0.0, 1.0, 0.0)
data recieved!
parameters of the car: (33.2032550811768, 0.0, 1.0, 0.0)
No interference.
steering_angle: 0 throttle: 1 brake: 0 counter: 70
No interference.
steering_angle: 0 throttle: 1 brake: 0 counter: 71
data\simulator\train\Dataset_\1\71.jpg
data recieved!
parameters of the car: (33.2032550811768, 0.0, 1.0, 0.0)
The traffic light is red: brake=1, throttle=0!!!!!!!!!!!!!!!!!!
steering_angle: 0 throttle: 0 brake: 1 counter: 72
data\simulator\train\Dataset_\1\72.jpg

Simulator Demonstration

Reference

1. https://link.zhihu.com/?target=https%3A//arxiv.org/pdf/1911.11929.pdf
2. https://zhuanlan.zhihu.com/p/143747206
3. https://www.cnblogs.com/boligongzhu/p/15508249.html
4. https://blog.csdn.net/weixin_55073640/article/details/122614176
5. https://github.com/ianleongg/Traffic-Light-Classifier
6. https://blog.csdn.net/weixin_51390582/article/details/130948872?spm=1001.2014.3001.5506