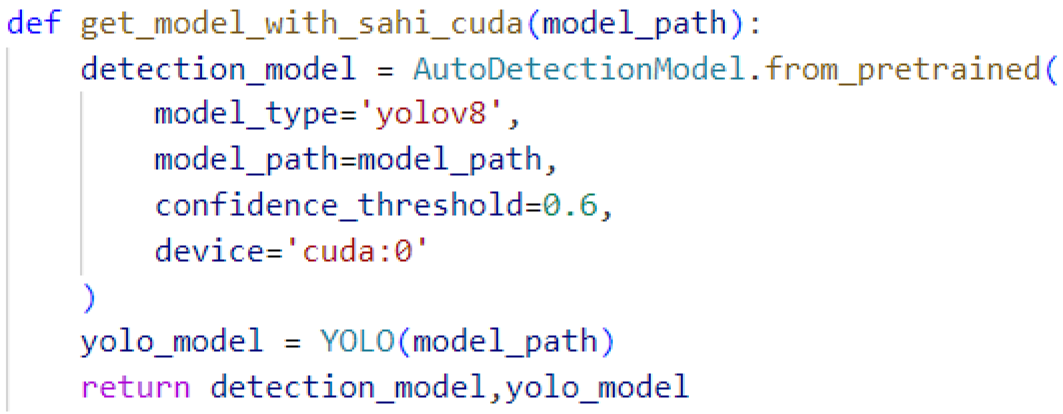
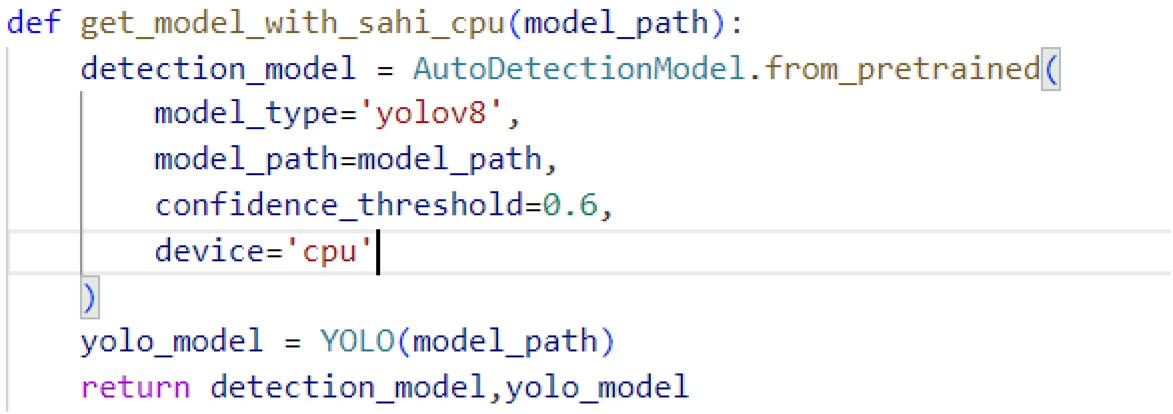
Working with yolov11

this file directory includes the work to finetune yolov11 and detect vehicles with `sahi` technics.

make sure your work direction is `MOP-Code`, otherwise all the file path in the code should be changed!



```python
def get_model_with_sahi_cuda(model_path):
    detection_model = AutoDetectionModel.from_pretrained(
        model_type='yolov8',
        model_path=model_path,
        confidence_threshold=0.6,
        device='cuda:0'
    )
    yolo_model = YOLO(model_path)
    return detection_model,yolo_model
```



```python
def get_model_with_sahi_cpu(model_path):
    detection_model = AutoDetectionModel.from_pretrained(
        model_type='yolov8',
        model_path=model_path,
        confidence_threshold=0.6,
        device='cpu'
    )
    yolo_model = YOLO(model_path)
    return detection_model,yolo_model
```

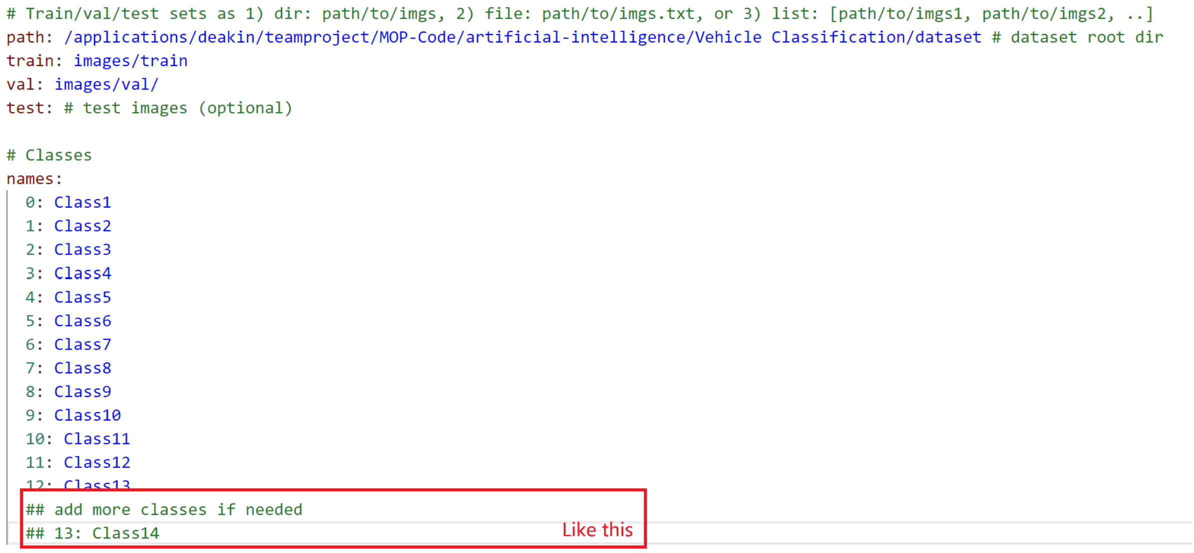
finetune yolov11

prepare dataset

First you need to use the code `merge_dataset.py`, you will build a `dataset` file directory in `../dataset/`, which is re-formed by the dataset provided in the path `../`(such as the pictures in `../vehicle_class_1`, `../vehicle_class_10`).

the `train.yaml` has been written as the dataset claim, so you don't need to change it.

If you want to add train pictures, I guess you just need to put these images into `../dataset/`,and add their respoding classes in `train.yaml`(if there is new class)



```
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: /applications/deakin/teamproject/MOP-Code/artificial-intelligence/Vehicle Classification/dataset # dataset root dir
train: images/train
val: images/val/
test: # test images (optional)

# Classes
names:
  0: Class1
  1: Class2
  2: Class3
  3: Class4
  4: Class5
  5: Class6
  6: Class7
  7: Class8
  8: Class9
  9: Class10
  10: Class11
  11: Class12
  12: Class13
  ## add more classes if needed
  ## 13: Class14
```

Like this

Train

what you need to do is just run the `train.py` in your computer. But remember, if your computer don't support `cuda`, you may need to add a parameter in function `train`, just to make your device is your cpu

```
# Train the mode`l
results = model.train(data=os.path.join(now_path_MyTest,"train.yaml"), epochs=100, imgsz=640)
# no cuda
# results = model.train(data=os.path.join(now_path_MyTest,"train.yaml"), epochs=100, imgsz=640, device='cpu')
```

And the train results will be saved in `**/MOP-code/runs/`.

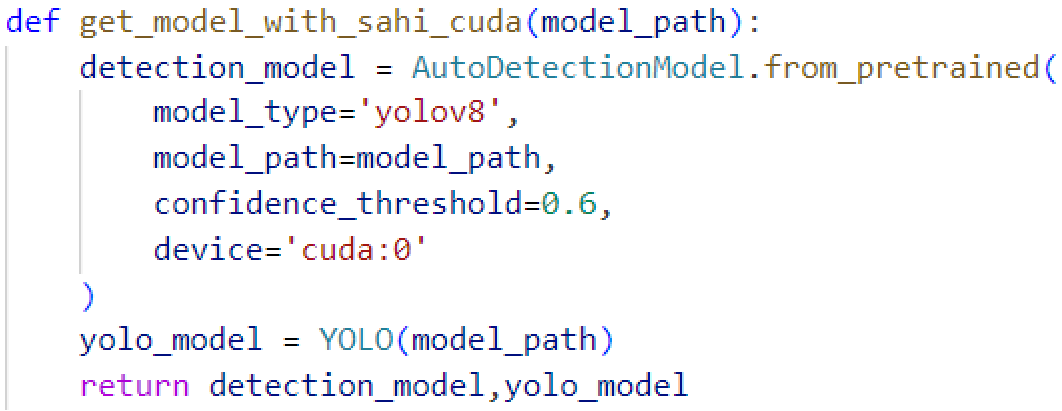
My train result also provided in the `./yolov11/`, which was trained in nividia 3070 gpu for 100 epoches.

Working with `SAHI`

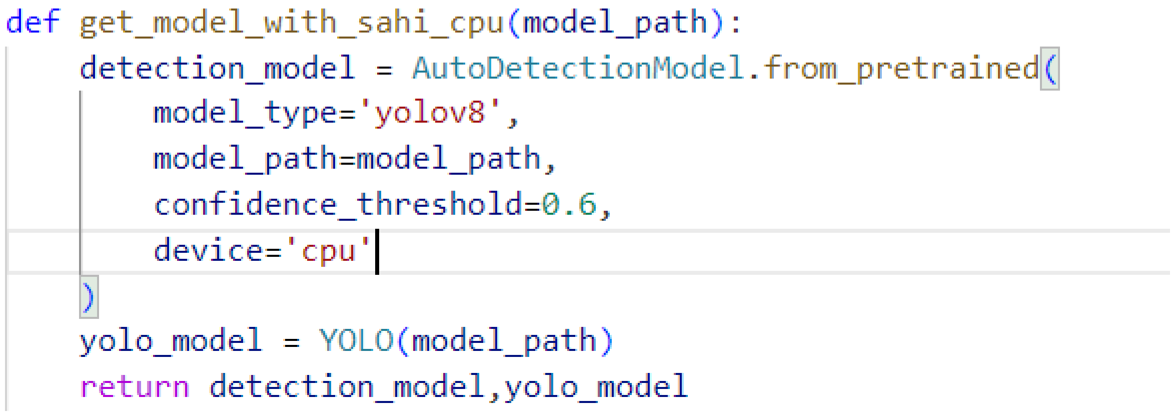
Build your detect model

Using the function in `./sahi/utils.py`, you can build your detect model with `SAHI`.

`model_path` is your YOLO path, and choose one function depending on your device.



```
def get_model_with_sahi_cuda(model_path):
    detection_model = AutoDetectionModel.from_pretrained(
        model_type='yolov8',
        model_path=model_path,
        confidence_threshold=0.6,
        device='cuda:0'
    )
    yolo_model = YOLO(model_path)
    return detection_model,yolo_model
```



```
def get_model_with_sahi_cpu(model_path):
    detection_model = AutoDetectionModel.from_pretrained(
        model_type='yolov8',
        model_path=model_path,
        confidence_threshold=0.6,
        device='cpu'
    )
    yolo_model = YOLO(model_path)
    return detection_model,yolo_model
```

Detect

Using the function in `./sahi/detect.py`, you can choose three different ways to detect. They are detected by yolo only, yolo with sahi, or yolo with multi-scale sahi.

all these function is easy to read, but if you want to figure out what do some parameters mean, you can also refer to the usage with [SAHI Tiled Inference - Ultralytics YOLO Docs]

The `multi-scale SAHI` involves applying the SAHI technique with various sizes on the same photographs, which can lead to the model detecting an excessive number of errors due to inappropriate SAHI sizes. **My solution involves merging these detected areas and selecting the most frequent label as the final label, drawing inspiration from the concept that “multiple weak classifiers can form a strong classifier.”**

Test

the test function is also provided in `./sahi/detect.py`. This function `test_performance` will merge 3 different result produced by responding methods, and merge them into a picture to compare difference better. Like this: