Automatic Vehicle Classification using a Detection system for mobile Emission Inventory

TOBIAS REINING

SENACYT
Secretaría Nacional de Ciencia, Tecnología e Innovación
CEMCIT-AIP
AIRES
AIR ENGINEERING STUDIES
LABORATORIO
MeTCAp
MECÁNICA TEÓRICA, COMPUTACIONAL Y APLICADA

Motivation

(Semi)-Automate a procedure to count and classify vehicles on a road. Later, this can be used to quantify the emissions data on a given road within a given time-frame.

How? Count cars along a road section, sort into several vehicle classes with known emissions. Each class corresponds to an emission factor.

By simultaneously measuring vehicle speed, the emission level of each vehicle can be further adjusted.

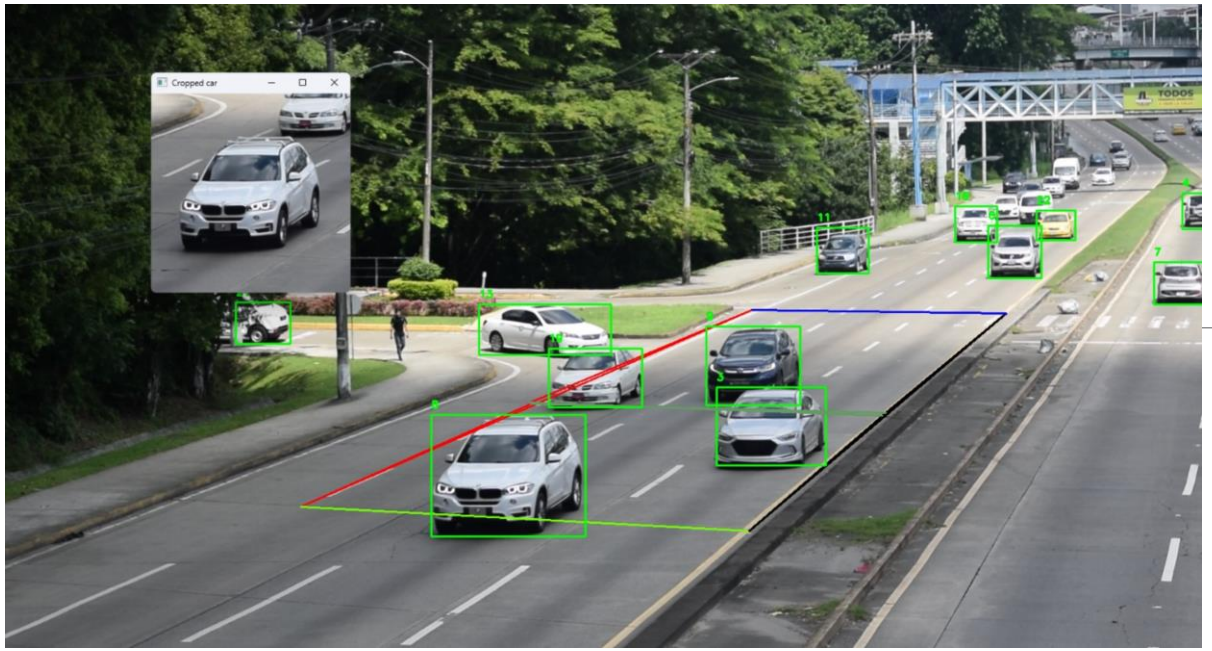
The measurement can be entirely done with a single camera capable of taking high resolution video.

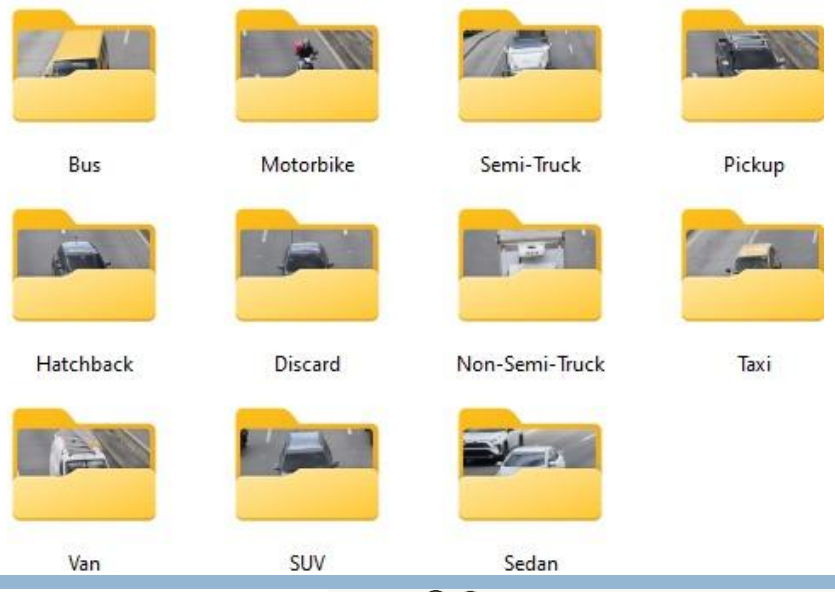
Overview

Video is fed into the program & a default Yolo Model (trained on the default Coco128 dataset) detects vehicles. A frame of each vehicle is saved to the hard drive for later use.

The time it takes the vehicle to traverse across a start and finish line is recorded, with a known distance (length between distinct objects in the video), the speed is easily calculated.

Using the saved frames, a custom yolo model can be trained to sort the vehicles into several classes.





SENACYT
Secretaría Nacional de Ciencia, Tecnología e Innovación
CEMCIT-AIP
AIRES
AIR ENGINEERING STUDIES
LABORATORIO
MeTCAp
MECÁNICA TEÓRICA, COMPUTACIONAL Y APLICADA

Why use two Yolo Models?

In theory, a custom Yolo model could be used to detect & classify vehicles in one step.

However, providing training data for this use case means manually drawing boundary boxes and labelling each box is necessary.

Instead, we can leverage existing Yolo models for the detection step, which automates the work of drawing boundary boxes. Now, only the manual work of classification of vehicles remains (for the training data).

Categorisation of vehicles

In theory, it is possible to train a Yolo model to detect individual car models.

But this means that for each possible car model, many examples need to recorded and labelled for the training process.

Additionally, the labelling process becomes very challenging. It might take a human quite a long time to identify a car model from a single image.

Instead, some categories were defined, starting from a classification system in use in Panamá.

Cuadro 3. AUTOMÓVILES EN CIRCULACIÓN EN LA REPÚBLICA, POR CLASE DE PLACA, SEGÚN TIPO: AÑOS 2017-18

Tipo de automóvil	Automóviles en circulación (1)						
	Placa						Variación porcentual
	2017			2018 (P)			
	Total	Comercial	Particular	Total	Comercial	Particular	
TOTAL	**825,093**	**175,275**	**649,818**	**878,762**	**181,472**	**697,290**	**6.5**
Automóviles para el transporte de pasajeros	**644,329**	**81,676**	**562,653**	**685,857**	**81,305**	**604,552**	**6.4**
Automóviles para pasajeros (hasta 13 personas)	**611,643**	**61,809**	**549,834**	**652,011**	**60,719**	**591,292**	**6.6**
Camioneta	**238,653**	3,278	235,375	**261,546**	2,710	258,836	9.6
Cupé	**11,114**	59	11,055	**10,784**	43	10,741	-3.0
Jeep	**7,373**	73	7,300	**7,531**	51	7,480	2.1
Sedán (2)	**354,503**	58,399	296,104	**372,150**	57,915	314,235	5.0
Omnibuses (de 5 personas y más)	**32,686**	**19,867**	**12,819**	**33,846**	**20,586**	**13,260**	**3.5**
Microbús (3)	**23,246**	10,427	12,819	**24,084**	10,824	13,260	3.6
Ómnibus	**9,440**	9,440	-	**9,762**	9,762	-	3.4
Automóviles para el transporte de carga	**180,300**	**93,135**	**87,165**	**192,170**	**99,432**	**92,738**	**6.6**
Camión	**34,441**	34,441	-	**35,963**	35,963	-	4.4
Mula	**8,266**	8,266	-	**8,234**	8,234	-	-0.4
Pick-up	**101,735**	17,372	84,363	**110,462**	20,746	89,716	8.6
Remolque	**16,851**	16,851	-	**18,154**	18,154	-	7.7
Reparto	**19,007**	16,205	2,802	**19,357**	16,335	3,022	1.8
Otros (4)	**464**	464	-	**735**	735	-	58.4

Guidelines for the Detection Code

Video with at least 30 fps and 1920x1080 resolution should be used, otherwise the individual cutouts can get blurry.

A video angle slightly off from the direction the cars travel lets the system collect some information about the side of the vehicles, improving classification.

High video angles (e.g. 90° to the driving direction) are not suitable for the current code.

The code runs on the GPU but can also run on the CPU if the line model.to(‘cuda’) is removed.

Guidelines for the Detection Code

The detection lines (used for speed calculation) need to be set up manually anytime the camera perspective changes.

Distances between visually distinct objects (e.g. two electricity poles) need to be measured out manually.

- Longer distances lead to more accurate results because of the limited video framerate.
- Multiple distances can be measured out, to produce a more accurate average velocity.

Setting up an environment for the code

The setup process is often highly individualistic and can differ from these general guidelines!

Follow the guide on: https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/

To enable GPU support, follow the guide on: https://pytorch.org/get-started/locally/

Pay attention to your GPU driver, CUDA and cuddn versions, and update as necessary

Used Modules:

Used modules:

- Ultralytics: YOLOv5
- cv2 (openCV)
- PIL
- Pytorch (https://pytorch.org/get-started/locally/)

Using the code:

dist: Known distances between objects

tol: Because of the finite framerate, a tolerance (in pixels) needs to be used

target size: square images of this size are saved for later classification.

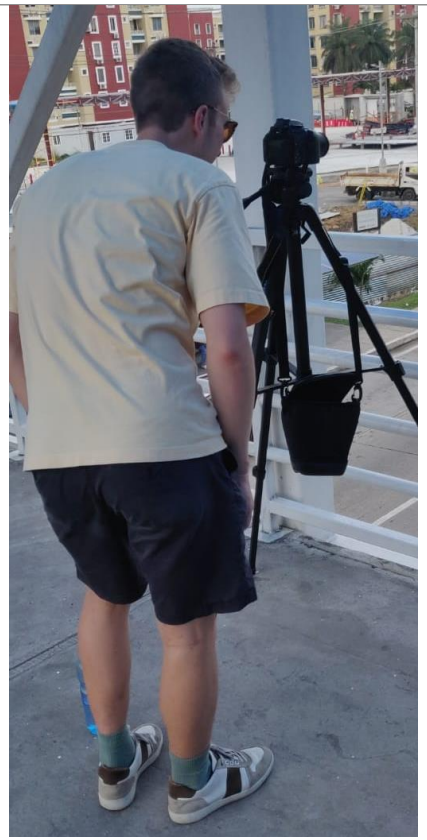
coords: The manually measured coordinates of the detection boxes are inserted here

A short script (boxcreation.py) helps in determining the coordinates.

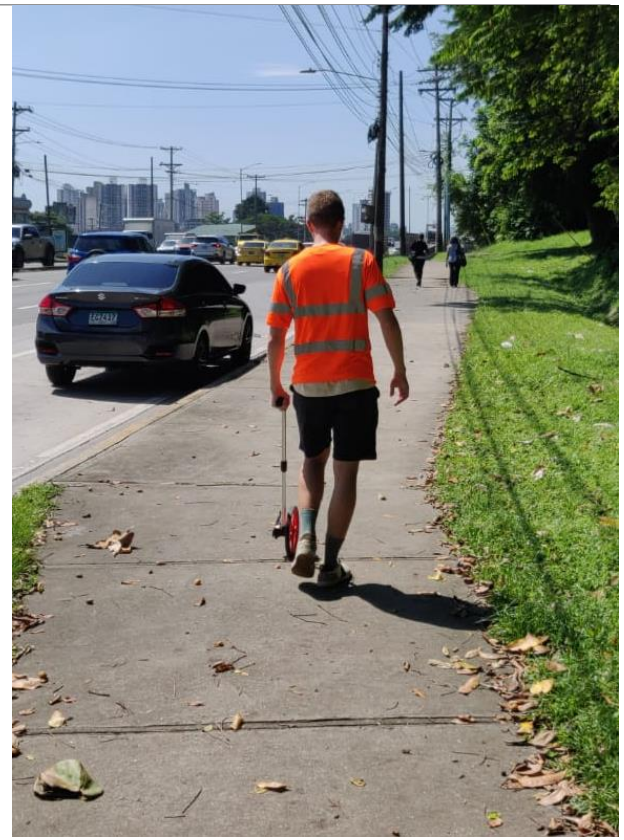
```
dist = [21.95,36.45,26.08] #distances between kn
tol = 6 # Tolerance in pixels
target_size = 288 #to save frames of equal size
car_data={}
cropped_images={}
img_directory="cropped_images"
coords=[[[558,782],[891,633],[1207,817],[1405,64
#top:[bottom left, top left, bottom right, top r
#right]
```

Field Work

Recording Video…



Measuring distances…



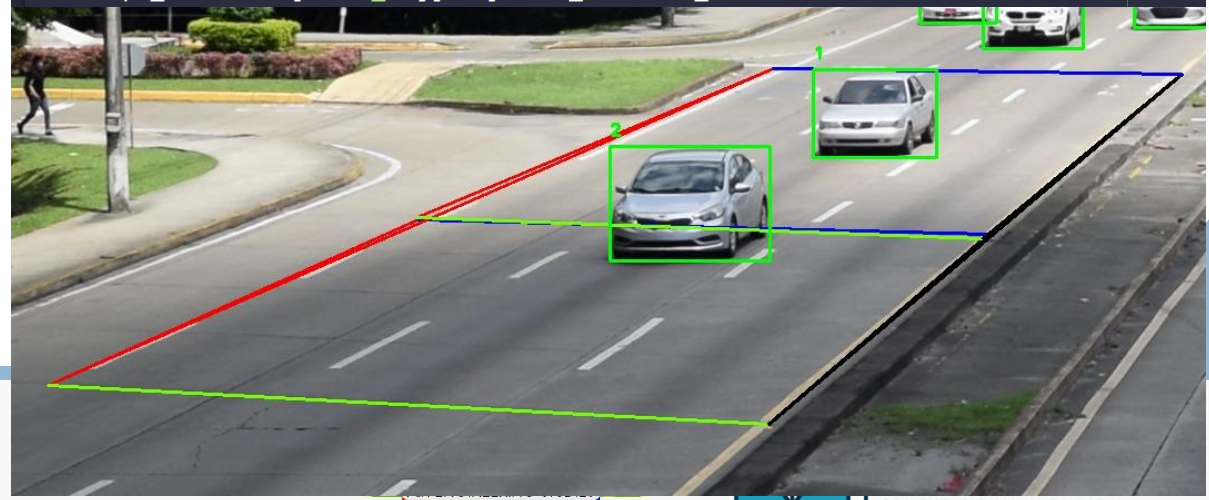
Details about Speed Detection: Slopes

To account for the entry/exit detection lines not being parallel to the horizontal direction, their slopes are calculated. Both for entry and exit: The slope is higher closer to the camera.

The higher the camera angle, the more the speed calculation accuracy improves compared to just using a single, fixed coordinate to determine an entry/exit.

(The slopes in the orthogonal direction can also be considered, but for lower angles, fixed boundaries in the x-direction are used, as the calculation is too rough for the limited framerate)

```
slope_functions={"entry_x":{}, "entry_y": {}, "exit_x": {}, "exit_y": {}}
for idx,box in enumerate(coords):
    if box[0][1]==box[2][1] or box[0][0]==box[2][0]:
        print("Warning: Speed detection lines are parallel to coord. system")
        continue
    slope_entry_y=(box[3][1]-box[1][1])/(box[3][0]-box[1][0])
    slope_exit_y=(box[2][1]-box[0][1])/(box[2][0]-box[0][0])
    slope_entry_x=1/slope_entry_y
    slope_entry_x_alt=(box[3][0]-box[1][0])/(box[3][1]-box[1][1])
    slope_exit_x=1/slope_exit_y
    entry_function_y= lambda x, k=slope_entry_y,d=box[1][1],x0=box[1][0]: k*(x-x0)+d
    exit_function_y= lambda x, k=slope_exit_y,d=box[0][1],x0=box[0][0]: k*(x-x0)+d
    entry_function_x= lambda y, k=slope_entry_x,d=box[1][0],y0=box[1][1]: k*(y-y0)+d
    exit_function_x= lambda y, k=slope_exit_x,d=box[0][0],y0=box[0][1]: k*(y-y0)+d
    slope_functions["entry_y"][idx]=entry_function_y
    slope_functions["exit_y"][idx]=exit_function_y
    slope_functions["entry_x"][idx]=entry_function_x
    slope_functions["exit_x"][idx]=exit_function_x
```



Setting up the detection/tracking

A model trained on the default Yolo Dataset (COCO, 80 classes) is used, yielding the weights/parameters file “best.pt”

The default tracker is used, but other trackers can be enabled.

The GPU is explicitly enabled. The model runs on the CPU if the line “model.to(‘cuda’)” is commented out.

The video file and starting time is specified.

```
model_path = 'best.pt'
model = YOLO(model_path)#, tracker='botsort.yaml')

print("Model device:", model.device)
model.to('cuda')
print("Model device after moving to GPU:", model.device)

video_path = 'rec_11_09_3.MOV'
cap = cv2.VideoCapture(video_path)
start_time_msec = 2 * 60 * 1000 + 5 * 1000
cap.set(cv2.CAP_PROP_POS_MSEC, start_time_msec)
```

Detecting/Tracking objects

In the main loop (while…) each video frame is evaluated.

The confidence threshold and Intersection over union threshold can be modified to improve performance.

To detect vehicles, only the classes [2,3,5,7] (corresponding to car, motorcycle, bus, truck) of the original COCO dataset are detected and tracked. This is specified in the coco.yaml file.

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break #ret==False: End of Video
    frameforscreencap = frame.copy()
    result = model.track(source=frame,persist=True, conf=0.05, iou=0.2, show=False, classes=[2,3,5,7])
    #conf: Confidence threshold for detection; iou: Intersection over union threshold
    result=result[0]
    boxes = result.boxes.xyxy
    ids = result.boxes.id
```

Detecting entry/exit (Speed detection)

An inner loop goes over each detection per frame, and a second inner loop goes over each detection box per detection.

To register an entry/exit, three conditions need to be true:

1) The y-coordinate of the vehicle box lower edge needs to be within the tolerance of the detection line.
2) The x-coordinate needs to be within the limits (e.g.to disregard parked vehicles)
3) The data entry must not already exist.

```
for box, track_id in zip(boxes, ids):
    track_id=int(track_id.item()) #convert tensor to native type
    x1, y1, x2, y2 = map(int, box[:4])
    midpoint = (x1 + x2) // 2, (y1 + y2) // 2
    y = midpoint[1]
    x = midpoint[0]
    if track_id not in car_data:
        car_data[track_id]={}
    for idx, coord in enumerate(coords):
        box_data = car_data[track_id].get(idx,{})
        if abs(y2-slope_functions["entry_y"][idx](x1))<tol and x>coord[1][0] and x<coord[3][0] and
            cv2.line(frame, (coord[1][0], coord[1][1]), (coord[3][0], coord[3][1]), (100, 155, 100
            box_data["entry_time"]=cap.get(cv2.CAP_PROP_POS_MSEC) / 1000.0

        elif abs(y2-slope_functions["exit_y"][idx](x1))<tol and x>coord[0][0] and x<coord[2][0] an
            cv2.line(frame, (coord[0][0], coord[0][1]), (coord[2][0], coord[2][1]), (100, 155, 100
            box_data["exit_time"] = cap.get(cv2.CAP_PROP_POS_MSEC) / 1000.0
```

Speed calculation

At each exit line, the speed is calculated from the recorded time between entry and exit and the known distance.

The location where the frame of each detected car is saved can be changed.

At the last box, the average speed is calculated and saved to the filename

```
elif abs(y2-slope_functions["exit_y"][idx](x1))<tol and
    cv2.line(frame, (coord[0][0], coord[0][1]), (coord[
    box_data["exit_time"] = cap.get(cv2.CAP_PROP_POS_MS
    if "entry_time" in box_data and "exit_time" in box_
        speed = (3.6 * dist[idx]) / (box_data["exit_tim
        print(f"Vehicle {track_id}: Speed in box {idx}:
        box_data["Speed"]=speed
    if idx == 0: #take the picture at the top box end
        width=(x2-x1)/target_size #normalized width. fo
        height=(y2-y1)/target_size
        half_size=target_size/2
        if width > 1 or height > 1:
            half_size = half_size*2 # if vehicles are b
            width=min(width/2,1)
            height=min(height/2,1)
        x1_new = int(max(x - half_size, 0))
        x2_new = int(min(x + half_size, frameforscreenc
        y1_new = int(max(y - half_size, 0))
        y2_new = int(min(y + half_size, frameforscreenc
        cropped_image = frameforscreencap[y1_new:y2_new
        cropped_images[track_id]=cropped_image #save it
        cv2.imshow('Cropped car', cropped_image)
        base_name = os.path.basename(video_path)
        file_name, file_extension = os.path.splitext(ba
        save_path = f"cropped_images/{track_id}_{file_n
        if not os.path.exists(img_directory):
            os.makedirs(img_directory)
        cv2.imwrite(save_path, cropped_images[track_id]
    if idx == 1: # calculate avg speed at the last box
        avg_speed=0
        for i in car_data[track_id]:
            speed = car_data[track_id][i].get("Speed")
            if speed is not None:
                avg_speed+=speed
        avg_speed=avg_speed/len(car_data[track_id])
```

Saving the frames

The frames of the detected objects are saved in a larger, standard-size frame (target_size). For large objects (trucks), the target size is doubled.

The size of the detected frame in the larger frame is normalized and saved in the filename (for later extraction).

This semi-automates the labelling process (the box sizes are now known, only the class is missing).

(The frame should not be saved too far towards the edge of the video, otherwise the target_size frame will exceed the available space. The final saved image will have the vehicle off-center → the label will be incorrect)

```
elif abs(y2-slope_functions["exit_y"][idx](x1))<tol and
    cv2.line(frame, (coord[0][0], coord[0][1]), (coord[
    box_data["exit_time"] = cap.get(cv2.CAP_PROP_POS_MS
    if "entry_time" in box_data and "exit_time" in box_
        speed = (3.6 * dist[idx]) / (box_data["exit_tim
        print(f"Vehicle {track_id}: Speed in box {idx}:
        box_data["Speed"]=speed
    if idx == 0: #take the picture at the top box end
        width=(x2-x1)/target_size #normalized width. fo
        height=(y2-y1)/target_size
        half_size=target_size/2
        if width > 1 or height > 1:
            half_size = half_size*2 # if vehicles are b
            width=min(width/2,1)
            height=min(height/2,1)
        x1_new = int(max(x - half_size, 0))
        x2_new = int(min(x + half_size, frameforscreenc
        y1_new = int(max(y - half_size, 0))
        y2_new = int(min(y + half_size, frameforscreenc
        cropped_image = frameforscreencap[y1_new:y2_new
        cropped_images[track_id]=cropped_image #save it
        cv2.imshow('Cropped car', cropped_image)
        base_name = os.path.basename(video_path)
        file_name, file_extension = os.path.splitext(ba
        save_path = f"cropped_images/{track_id}_{file_n
        if not os.path.exists(img_directory):
            os.makedirs(img_directory)
        cv2.imwrite(save_path, cropped_images[track_id]
    if idx == 1: # calculate avg speed at the last box
        avg_speed=0
        for i in car_data[track_id]:
            speed = car_data[track_id][i].get("Speed")
            if speed is not None:
                avg_speed+=speed
        avg_speed=avg_speed/len(car_data[track_id])
```

Boxcreation.py

rec_11_09_3

Main File:
yoloVideorec11_09_3.py

22_rec_11_09_3_59.73kmh_wi0.62_he0.49

16_rec_11_09_3_40.06kmh_wi0.74_he0.56

37_rec_11_09_3_40.86kmh_wi0.74_he0.54

38_rec_11_09_3_41.10kmh_wi0.69_he0.53

22_rec_11_09_3_59.73kmh_wi0.62_he0.49

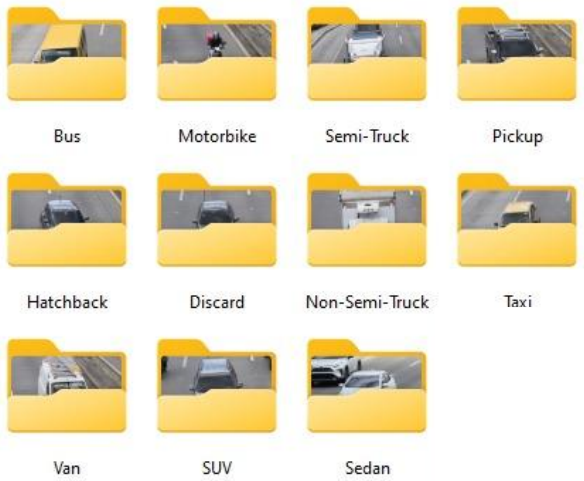
vehicle number

speed

width and height of vehicle in the image

Sort Images manually

Use ImageSort.py for easy sorting



SorttoTrainVal.py

labelcreation.py

Use a pretrained model.
Smaller (yolov5s): faster, less accurate
Bigger (yolov5l) slower, more accurate

Train model with:
python train.py --img 288 --epochs 300 --data cars.yaml --weights yolov5l.pt –batch-size -1

Training Yolo

Follow the instructions at:
https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/

Images are split into 80% training data, 20% validation data. (Use sorttoTrainVal.py)

A label (.txt) needs to be created for each image, stored in a parallel folder structure (Use labelcreation.py)

A .yaml file has to be created (e.g. in Notepad) to transfer info (classes, folder structure) for the training process.

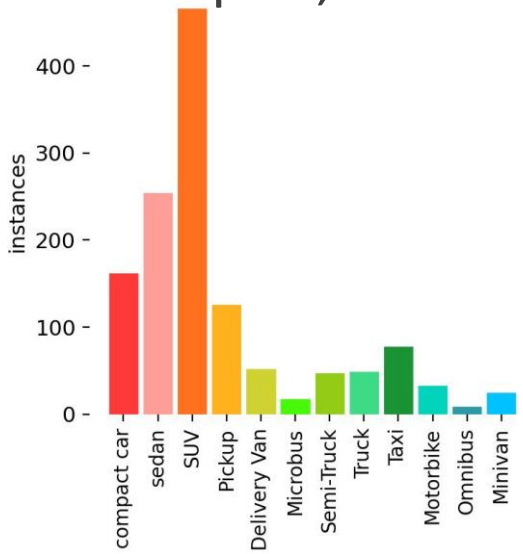
Use 300 epochs, automatic batch size (–batch-size -1) and img=target_size for a start

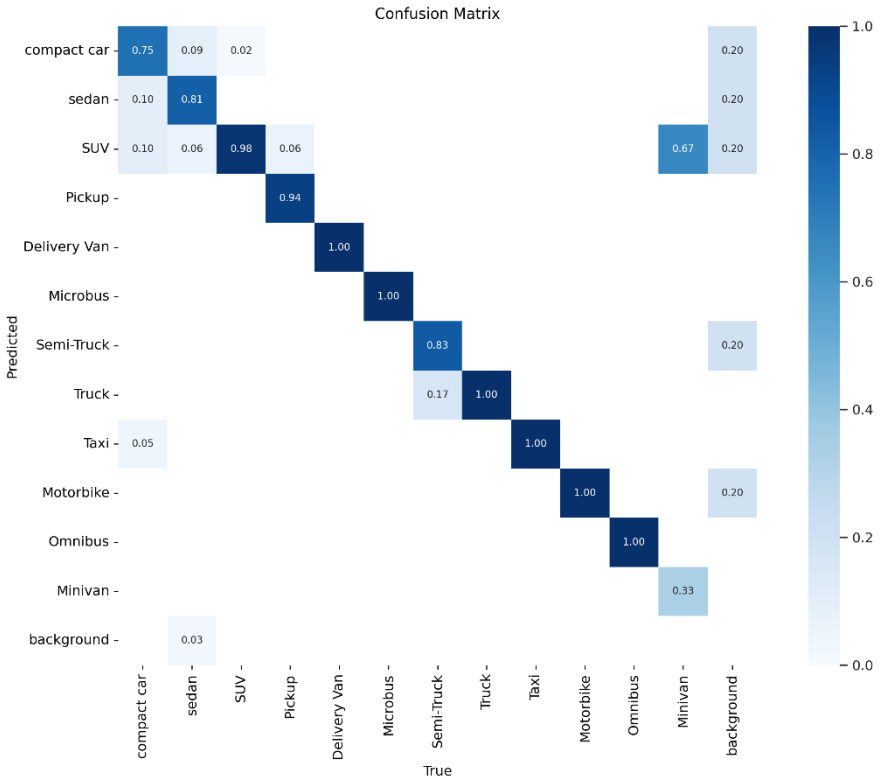
Results:

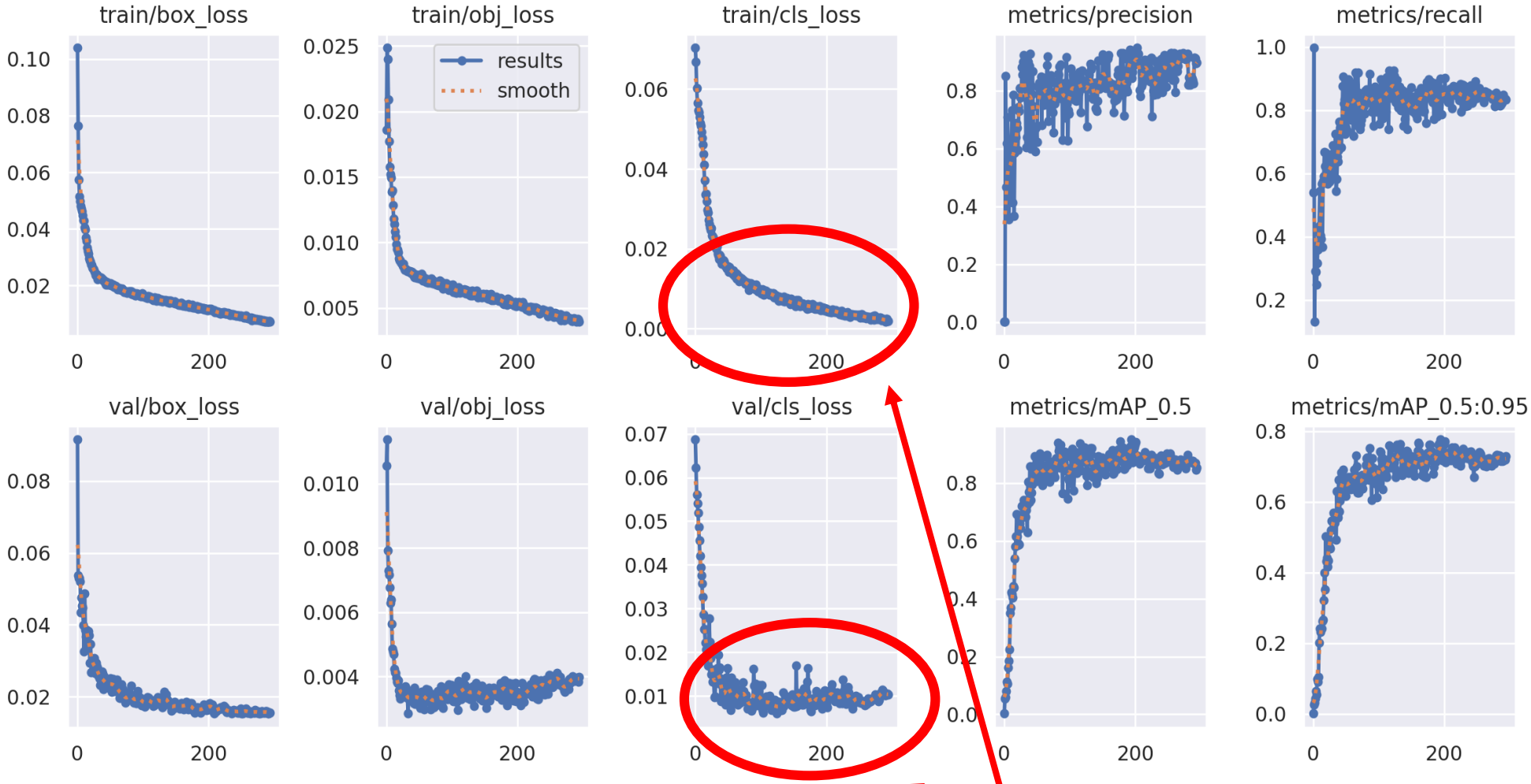
A training run with data from about 1600 vehicles (about 4 x 10min recordings) was completed, using a pre-trained model: yolov5l.pt

For those categories with more examples, the Yolo net has reasonable accuracy.

The fewer examples in a category, the lower the accuracy (e.g. Minivans with very few examples).



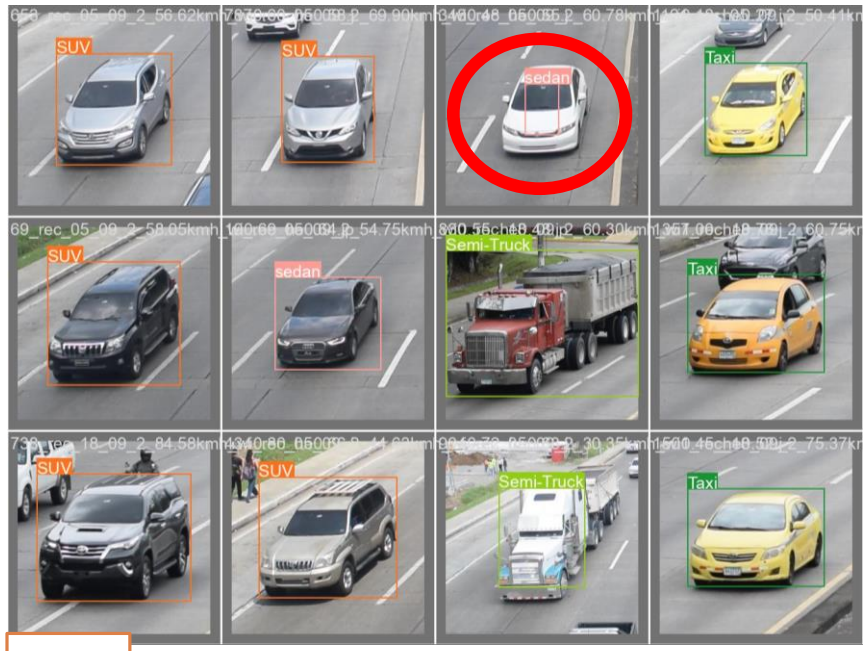




Overfitting is beginning: training loss goes down, while validation loss stays flat/goes up. Not much improvement after ca. 150 epochs

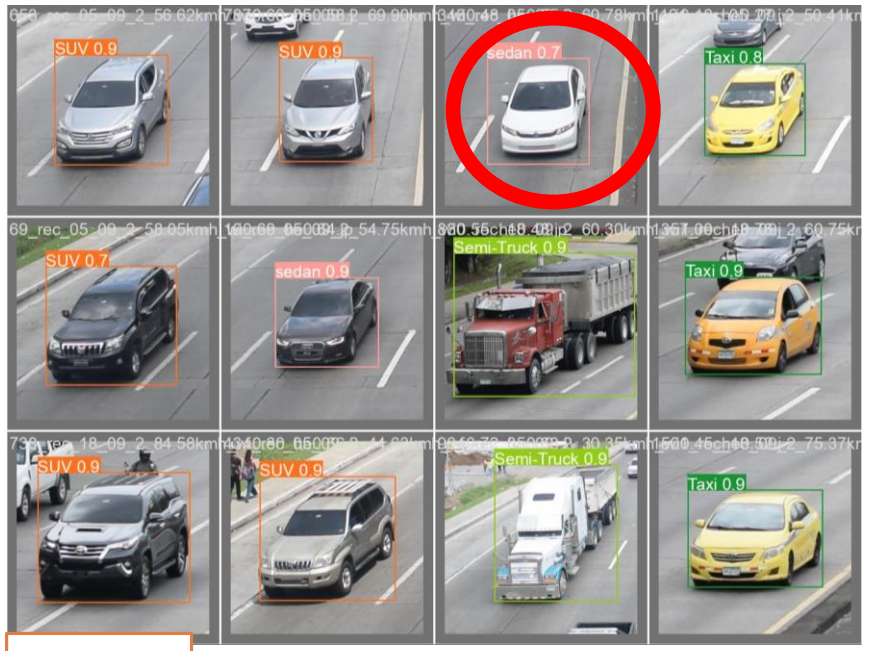
Results:

Sometimes the labels can be wrong, this reduces the accuracy.



Labels

Sometimes, the predictions are better then the label



Predictions

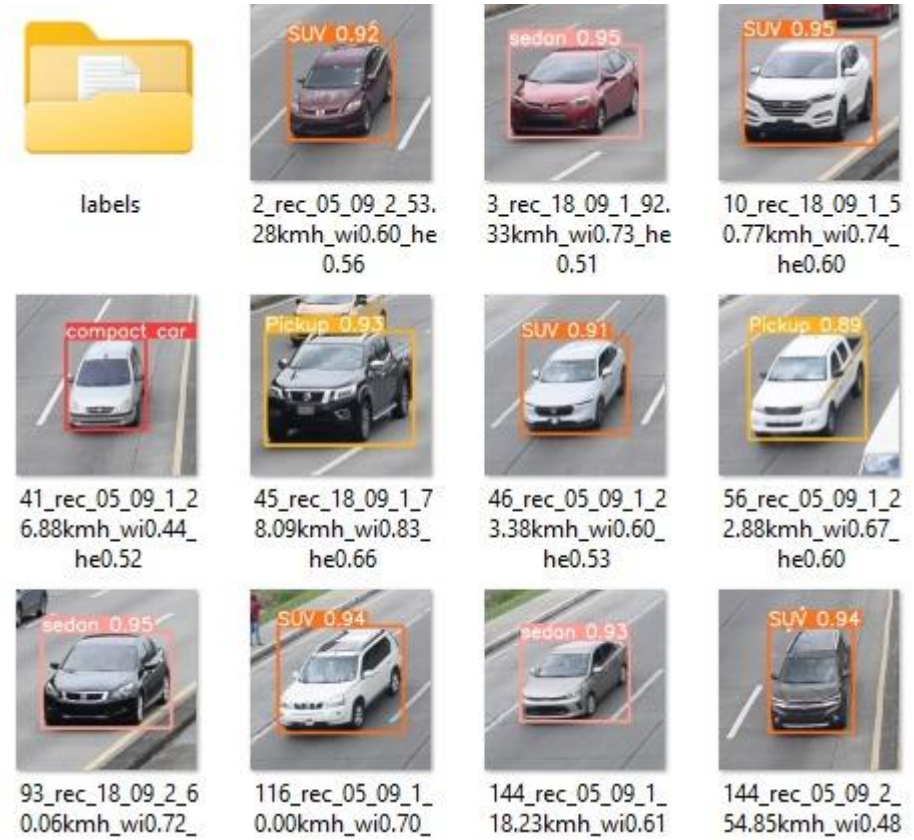
Using the model:

Use the model with:

Python detect.py –weights “path” –img “target_size” –conf 0.25 – source “path” –save-txt

Images with predictions are stored, along with a .txt file each.

Using the .txt files, the amount and type of cars is easily established.



Thank you for your attention!

QUESTIONS?