1. Introduction to the Problem

1.1 Problem Description and Project Change

As society rapidly urbanised, the roads are becoming congested with lots of vehicles, requiring the traffic management. On specific roads, certain types of vehicles should be restricted for safety and efficient traffic flow. For example, on highways, the small motorcycles are dangerous to drive and also increase the risk of accidents and on bridges, heavy trucks must be restricted.

To address these issues, we introduced an object detection model which can detect vehicles entering and driving on the road in real-time. This model can also classify the types of vehicle, and automatically determine which vehicles are allowed to pass and which should be restricted.

We expect to enhance road safety, traffic flow, and increasing efficiency of urban traffic management using this model.

We started our study with the intention of categorising and identifying automobiles in photos using both YOLOv5 and Faster R-CNN, as stated in the "Project Modification" portion of our report. On the other hand, we faced some difficulties with the Faster R-CNN model, such as limited resources, long training durations, and low accuracy. We strategically decided to shift our main attention to the YOLOv5 model in response to these difficulties, which allowed for more effective resource allocation and prompt outcomes. We shortened our project by focusing on YOLOv5, demonstrating the agility and dedication of our team to meeting our project objectives. In addition to providing comprehensive details on YOLO v5s setup and training outcomes, our research assesses the difficulties faced by Faster R-CNN and explains why YOLOv5 was chosen as the default model. We also describe our methodology for managing projects, individual contributions, team meetings, and references for more details. In the end, switching to YOLOv5 helped us get beyond the difficulties with Faster R-CNN and accomplish the vehicle categorization and detection goals of our project.

1.2 Vehicle Classification in real life

Our research focuses on how important it is to classify and detect vehicles in real-world applications including surveillance systems, autonomous driving, and traffic monitoring. Improving the safety and effectiveness of these applications depends on the precise and effective identification and classification of vehicles. At first, we intended to use Faster R-CNN in addition to YOLOv5 to tackle these issues. But because of constraints like limited resources, long training cycles, and labour-intensive data annotation, we had to shift our primary focus back to YOLOv5. We wanted to hasten project progress and deploy resources efficiently, which is why we made this important adaptation. Essentially, the goal of our project is to address the increasing demand for vehicle classification and detection in real-world scenarios by utilising advanced object detection models, specifically YOLOv5. This will ultimately improve safety and efficacy in domains such as autonomous driving and surveillance.

1.3 Use MATLAB and Colab for vehicle classification

While Google Colab offers GPU resources and serves as a backup platform for model training, particularly Faster R-CNN, we choose MATLAB for the implementation and fine-tuning of the YOLOv5 model in our vehicle object recognition project. Utilizing 627 photographs of different vehicle classes divided into training, validation, and test sets, we leverage the Roboflow open images dataset. A modest architecture serves as the foundation for YOLOv5, which is trained across 30 epochs with a batch size of 16. We halt when the loss varies, suggesting possible overfitting, and save the training after 10 epochs. The assessment findings demonstrate different performances in terms of class; the test dataset produced a mAP50 of 0.803, whereas the validation dataset produced a mAP50 of 0.675 and a mAP50-95 of 0.525.

Significant difficulties with the Faster R-CNN model's implementation, such as lengthy training periods and the time-consuming human annotation process, led to the model's removal. A comparison study highlights YOLOv5's advantages, especially with regard to training effectiveness and precision. A tiny dataset, insufficient GPU resources, insufficient model training expertise, and the requirement for data augmentation were some of the challenges the project had to overcome. Regular meetings and the use of a Swimlane Diagram for process visualization and individual contribution tracking were part of the team's project management methodology. Despite these difficulties, the study effectively illustrates the accuracy and effectiveness of YOLOv5-based vehicle classification.

1.4 Dataset Introduction

We first introduce our main dataset, the Roboflow open pictures dataset, which includes a wide range of photos from different vehicle classes, in our "Dataset Introduction " section (1.4). The dataset was painstakingly divided into three subsets: test, validation, and training. Each subset has a distinct function in the creation and assessment of the model. We decided against data augmentation in favour of the original photos because of resource constraints, such as GPU capacity and training time. With an emphasis on developing accurate object detection models for common vehicle types like automobiles, buses, motorbikes, trucks, and ambulances, our dataset contains information on these vehicles. The data division maintains consistent 416 x 416 pixel sizes for all images, allocating 70% to training, 20% to validation, and 10% to the test subset. Class numbers, label names, training and validation paths, and other crucial dataset information are all contained in the data.yaml file. We'll go into further detail about our dataset, its organisation, and our method of training the YOLOv5 and Faster-RCNN models. We'll also talk about why we decided against doing data augmentation because of resource limitations.

2. High Level Description of the Code

2.0 Dataset

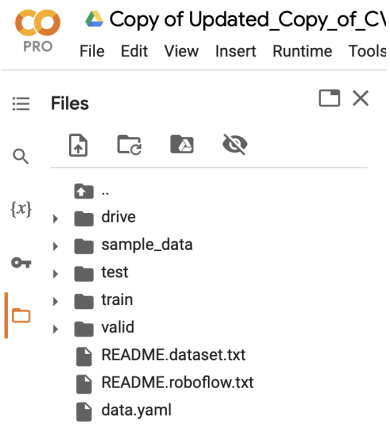
We used the roboflow open images dataset

```
!pip install -q roboflow
```

```
%cd /content
!curl -L "https://public.roboflow.com/ds/toVJdWBuRp?key=I1KM5dFkuo" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

```
extracting: valid/labels/b39ce222d4077e92_jpg.rf.dG9MelQmthewmJnOGXBa.txt
extracting: valid/labels/b3aa29cc695d8bed_jpg.rf.8cb112823e07ea8cfb21bf775cbcc0b0.txt
extracting: valid/labels/b3aa29cc695d8bed_jpg.rf.OFMbBOuzaGaWCLoBRIkw.txt
extracting: valid/labels/b5cd0b99c2f9bf15_jpg.rf.EyGI7jJzcPDYodN9oSj5.txt
extracting: valid/labels/b5cd0b99c2f9bf15_jpg.rf.b20d8ff6435c5057b5a739a7ec596088.txt
extracting: valid/labels/b7ca8f95ad08cdda_jpg.rf.3751acce55be3b308e918cec95e3ac5e.txt
extracting: valid/labels/b7ca8f95ad08cdda_jpg.rf.S6zSOtEuKBgZ5xemd06l.txt
extracting: valid/labels/be613907a0a28d0a_jpg.rf.36ef2c64ce15b8493c719b28b592b114.txt
```

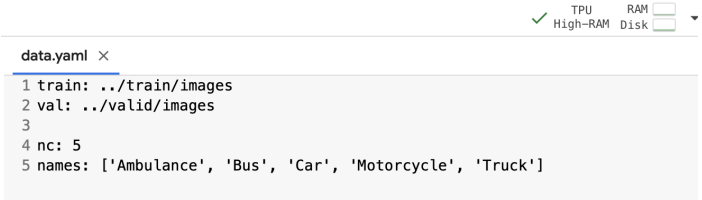
This code is for downloading a dataset from Roboblow, unzipping the image files.



This dataset comprises 627 images of various vehicle classes for object detection.
The vehicle classes include:

- Car
- Bust
- Motorcycle
- Truck
- Ambulance

The entire image dataset is splitted train, valid, test dataset, following 70%, 20%, and 10% respectively.
Data augmentation is not applied to reduce running time and the size of each image is 416*416.



This is a screenshot of the data.yaml file which includes paths for train and validation dataset, total number of classes, and the names of each label.

2.1 YOLOv5 Model

```
%cd /content
! git clone https://github.com/ultralytics/yolov5.git
```

```
/content
Cloning into 'yolov5'...
remote: Enumerating objects: 16026, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 16026 (delta 33), reused 41 (delta 25), pack-reused 15967
Receiving objects: 100% (16026/16026), 14.68 MiB | 21.54 MiB/s, done.
Resolving deltas: 100% (10999/10999), done.
```

This code clones the YOLOv5 repository from GitHub, to use the pre-trained models, and other python scripts for object detection.
As our dataset is small, we used transfer learning, adapting this pre-trained model weight to our specific vehicle dataset for further fine-tuning.

yolov5s.yaml

```
# YOLOv5 🚀 by Ultralytics, AGPL-3.0 license

# Parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
```

This is the yaml file of the YOLOv5 small model. This defines the architecture and parameters of the pre-trained model. For the parameters, the model is configured to identify 80 classes, with specific depth(0.33) and width(0.5). The model uses 3 anchors to detect various scales objets effectively.

The backbone structure extracts the feature map from the images, comprising a series of convolutional layers, C3, and SPPF.

- C3 contains three convolutional layers for capturing more complex patterns by splitting and merging feature maps.
- SSPF is designed for multi-scale feature extraction, generating consistent output size.
- ‘nc: 80’: defines the number of categories for target detection. There are 80 categories, which usually correspond to COCO datasets.
- ‘depth_multiple: 0.33’: multiple of the depth of the model, which affects the number of layers of the model. A value less than 1 indicates that this is a "compressed" version of the model.
- ‘width_multiple: 0.50’: the multiple of the layer channel. This affects the width of the network layer or the number of channels. Similarly, a value less than 1 means that this is a lightweight model.
- ‘anchors’: defines 3 sets of preset bounding box sizes that will be used on 3 different scales of feature maps.

The head focused to predict object detection, upsampling concating multiple backbone features. The last detect layer predicts objects based on the extracted three different scale features.

```
hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, ob
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5 🚀 runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
100% 755k/755k [00:00<00:00, 14.4MB/s]
Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt to yolov5s.pt...
100% 14.1M/14.1M [00:00<00:00, 120MB/s]

Overriding model.yaml nc=80 with nc=5

                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  2    115712  models.common.C3                        [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  3    625152  models.common.C3                        [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   1182720  models.common.C3                        [512, 512, 1]
  9                -1  1    656896  models.common.SPPF                      [512, 512, 5]
 10                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 12           [-1, 6]  1         0  models.common.Concat                    [1]
 13                -1  1    361984  models.common.C3                        [512, 256, 1, False]
 14                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 16           [-1, 4]  1         0  models.common.Concat                    [1]
 17                -1  1     90880  models.common.C3                        [256, 128, 1, False]
 18                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 19          [-1, 14]  1         0  models.common.Concat                    [1]
 20                -1  1    296448  models.common.C3                        [256, 256, 1, False]
 21                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 22          [-1, 10]  1         0  models.common.Concat                    [1]
 23                -1  1   1182720  models.common.C3                        [512, 512, 1, False]
 24      [17, 20, 23]  1     26970  models.yolo.Detect                      [5, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256,
YOLOv5s summary: 214 layers, 7033114 parameters, 7033114 gradients, 16.0 GFLOPs

Transferred 342/349 items from yolov5s.pt
optimizer: SGD(lr=0.01) with parameter groups 57 weight(decay=0.0), 60 weight(decay=0.0005), 60 bias
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01), CLAHE(p=0.01, clip_limit=(1, 4.0), tile_grid_size=(8, 8))
```

The hyperparameters set above affect model performance directly.
These need to be set before training the model, as opposed to the internal parameters of the model (which are automatically learned and adjusted in the training process).

Main hyperparameter Setting:

- lr0 = 0.01 (Initial learning rate)
- lrf = 0.01 (Final learning rate)
- momentum = 0.937: used to accelerate convergence in gradient declines and reduce oscillations.
- Weight_decay = 0.0005: weight attenuation, also known as L2 regularisation, is used to prevent overfitting.
- Optimizer = SGD(Stochastic Gradient Descent)

- ‘warmup_epochs=3.0’ and warmup_momentum=0.8 and warmup_bias_lr=0.1: the cycle, momentum and deviation learning rate of the warm-up phase, which is used for the steady increase of learning rate in the early stage of model training.
- ‘box=0.05’, ‘cls=0.5’, ‘obj=1.0’: loss function weights for boxes, categories, and targets.
- ‘cls_pw=1.0’ and ‘obj_pw=1.0’: the positive and negative weights of the classification and target loss function.
- ‘Iou_t=0.2’: the threshold of IoU (crossover ratio).
- ‘anchor_t=4.0’: the threshold of the anchor point.
- ‘fl_gamma=0.0’: the gamma value of Focal loss.

Data enhancement parameters: such as hsv_h, hsv_s, hsv_v, control the adjustment range of hue, saturation and brightness. Degrees, translate, scale, shear, perspective, flipud, fliplr, mosaic, mixup, copy_paste and other parameters control image rotation, translation, zoom, cut, perspective, up and down flip, left and right flip, mosaic, mixing, copy and paste and other data enhancement techniques.

Training 1: Training model for 10 epochs

```
%cd /content/yolov5/
!python train.py --img 416 \
                 --batch 16 \
                 --epochs 10 \
                 --data /content/data.yaml \
                 --cfg /content/yolov5/models/yolov5s.yaml \
                 --weights yolov5s.pt \
                 --name vehicle_yolov5s_results
```

These codes are for setting the parameters of the model training. The image size is set to 416*416 which is the same as original image size, and 16 batch size, which indicates using 16 images simultaneously to update the model weights. Considering the limited resources(RAM and GPU), we chose 16 for batch size and ran 10 epochs first, which still took approximately 2hrs. The data and model configuration file have been explained above, and pre-trained weights of small YOLOv5 used to avoid excessively long training times, as other larger, complex models might take very long to train. The training results are saved under “vehicle_yolov5s_results” directory.

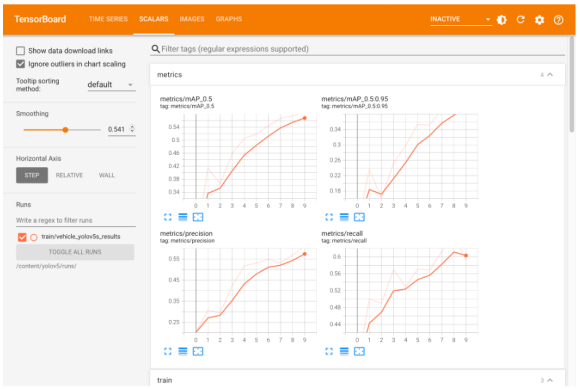
```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        7/9         0G    0.04079    0.02131    0.01404         50        416: 100% 55/55 [10:46<00:00, 11.76s/it]
                  Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [01:11<00:00,  8.98s/it]
                    all        250        454      0.532      0.613      0.567      0.394

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        8/9         0G    0.03761    0.01998    0.01412         31        416: 100% 55/55 [10:48<00:00, 11.80s/it]
                  Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [01:13<00:00,  9.15s/it]
                    all        250        454      0.569      0.645      0.575      0.402

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        9/9         0G    0.03573    0.01989    0.01404         44        416: 100% 55/55 [10:49<00:00, 11.81s/it]
                  Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [01:10<00:00,  8.82s/it]
                    all        250        454       0.61      0.594      0.584      0.418

10 epochs completed in 2.020 hours.
Optimizer stripped from runs/train/vehicle_yolov5s_results/weights/last.pt, 14.3MB
Optimizer stripped from runs/train/vehicle_yolov5s_results/weights/best.pt, 14.3MB

Validating runs/train/vehicle_yolov5s_results/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
                  Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [01:06<00:00,  8.36s/it]
                    all        250        454      0.611      0.594      0.583      0.416
              Ambulance        250         64      0.727       0.75      0.805      0.657
                    Bus        250         46      0.822      0.696      0.715      0.498
                    Car        250        238      0.575      0.504      0.521      0.345
             Motorcycle        250         46      0.643      0.652      0.666      0.437
                  Truck        250         60      0.287      0.367      0.208      0.144
Results saved to runs/train/vehicle_yolov5s_results
```

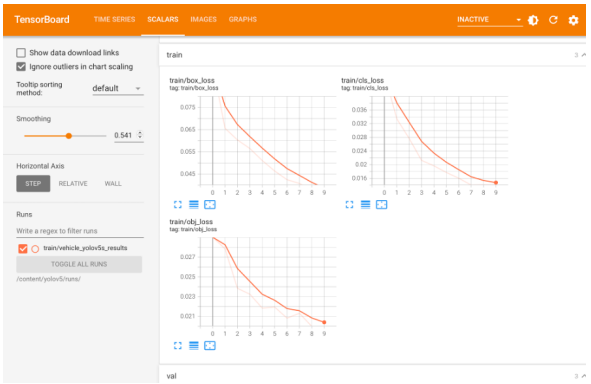
Results for first 10 epochs

```
1 shutil.copy('/content/yolov5/runs/train/vehicle_yolov5s_results/weights/last.pt', de
2 shutil.copy('/content/yolov5/runs/train/vehicle_yolov5s_results/weights/best.pt', de
```

```
'/content/drive/MyDrive/Colab_Models/best.pt'
```

Saved the last and best model weights of the first 10 epochs for continued training.





Based on training logs in the above TensorBoard, we concluded that the model requires additional epochs for training. This is due to the consistent upward trend in training mAP, precision, recall and simultaneously steady decline in all loss values. We delved deeper into these aspects in the evaluation section below.

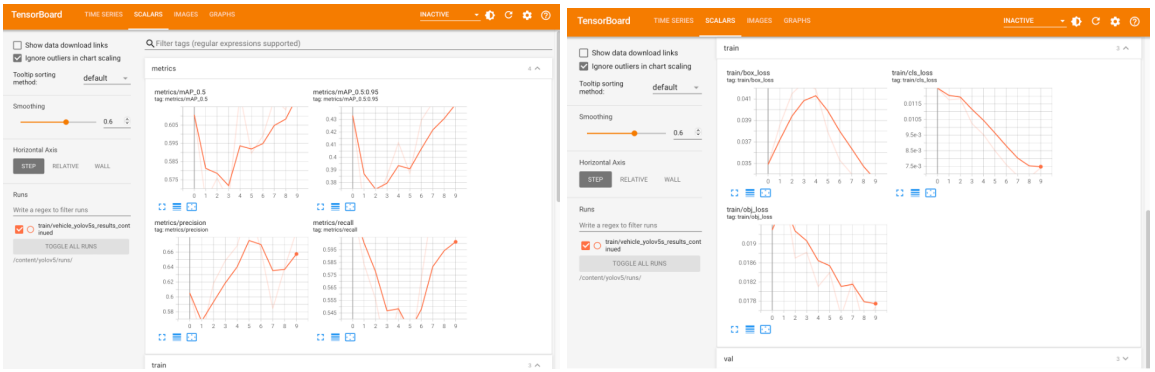
Training 2

```
1 %cd /content/yolov5/
2
3 !python train.py --img 416 --batch 16 --epochs 10 --data /content/data.yaml --cfg /c
4
```

This code is for training another 10 epochs. This time we used last.pt weights which are already trained for 10 epochs and all settings unchanged. The results saved it as name under vehicle_yolov5s_results_continued

```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        7/9         0G    0.03403    0.01822   0.006879         50        416: 100% 55/55 [09:08<00:00,  9.97s/it]
                  Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:57<00:00,  7.21s/it]
                    all        250        454      0.584      0.631      0.619      0.443

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        8/9         0G    0.03228    0.01725   0.006796         31        416: 100% 55/55 [08:59<00:00,  9.82s/it]
                  Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [01:01<00:00,  7.72s/it]
                    all        250        454      0.639      0.613      0.613      0.444

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        9/9         0G    0.03157    0.01769   0.007406         44        416: 100% 55/55 [09:00<00:00,  9.82s/it]
                  Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [01:01<00:00,  7.65s/it]
                    all        250        454      0.688      0.612      0.639      0.459

10 epochs completed in 1.676 hours.
Optimizer stripped from runs/train/vehicle_yolov5s_results_continued/weights/last.pt, 14.3MB
Optimizer stripped from runs/train/vehicle_yolov5s_results_continued/weights/best.pt, 14.3MB

Validating runs/train/vehicle_yolov5s_results_continued/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
                  Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:56<00:00,  7.05s/it]
                    all        250        454      0.688      0.611      0.639      0.456
              Ambulance        250         64      0.796      0.781       0.85      0.684
                    Bus        250         46      0.811      0.744      0.738      0.585
                    Car        250        238      0.648      0.513      0.565      0.373
             Motorcycle        250         46      0.558      0.739      0.679      0.396
                  Truck        250         60      0.627       0.28      0.362      0.243
Results saved to runs/train/vehicle_yolov5s_results_continued
```

This is the result of extra 10 epochs so we trained 20 epochs so far.



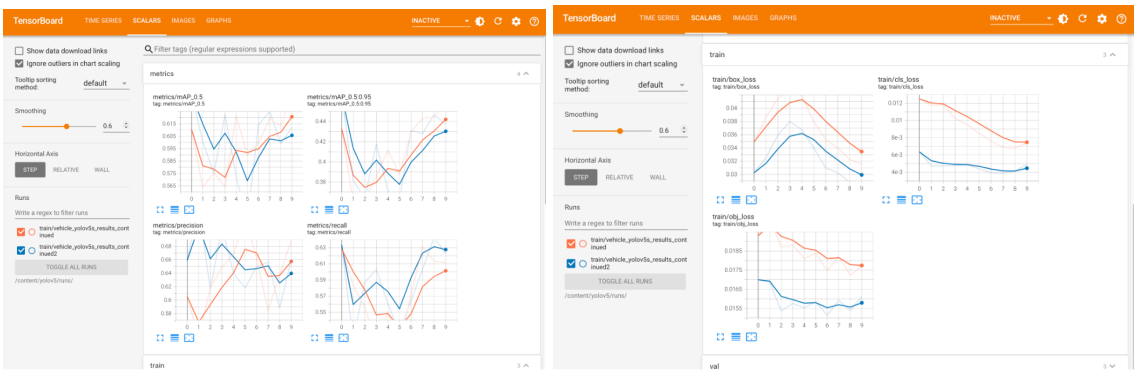
Unlike graphs in the first TensorBoard, the training mAP, precision and recall values start fluctuating in the second TensorBoard. The values decreased at the first few epochs but changed to increasing and the losses still decreased. Thus, we think the model is not overfitted to the train dataset and can be trained using the same train dataset. We kept the same configuration setting and trained the model using last.pt weight for 10 more epochs.

Training 3

```
%cd /content/yolov5/

!python train.py --img 416 --batch 16 --epochs 10 --data /content/data.yaml --cfg /content/yolov5/models/yolov5s.yaml --weigh

```

```
               all        250        454      0.65      0.648      0.614      0.451

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        7/9         0G    0.03025    0.01593   0.003862         50        416: 100% 55/55 [08:58<00:00,  9.80s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [01:02<00:00,  7.86s/it]
                   all        250        454      0.657      0.668      0.625      0.428

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        8/9         0G    0.02889    0.01539   0.004128         31        416: 100% 55/55 [08:51<00:00,  9.66s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:57<00:00,  7.17s/it]
                   all        250        454      0.588      0.643      0.599      0.447

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
        9/9         0G    0.02861    0.01613   0.004935         44        416: 100% 55/55 [08:53<00:00,  9.70s/it]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:58<00:00,  7.26s/it]
                   all        250        454      0.661      0.622      0.612      0.437

10 epochs completed in 1.655 hours.
Optimizer stripped from runs/train/vehicle_yolov5s_results_continued2/weights/last.pt, 14.3MB
Optimizer stripped from runs/train/vehicle_yolov5s_results_continued2/weights/best.pt, 14.3MB

Validating runs/train/vehicle_yolov5s_results_continued2/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [00:54<00:00,  6.79s/it]
                   all        250        454      0.656      0.632      0.639      0.466
             Ambulance        250         64      0.669      0.844      0.822      0.667
                   Bus        250         46      0.768      0.739       0.76       0.61
                   Car        250        238      0.615      0.538      0.574      0.362
            Motorcycle        250         46      0.602      0.739      0.614      0.396
                 Truck        250         60      0.627        0.3      0.424      0.293
Results saved to runs/train/vehicle_yolov5s_results_continued2
```

So this is the final results of training models.



Concluding our training, the model has been trained for 30 epochs in total.
In the third TensorBoard, there are two lines. The blue graphs are the results of the model which trained for 30 epochs and red lines are the results for the second training model which has completed 20 epochs. Although the mAP, Precision, Recalls are higher and all the losses are lower than the previous trained model, we observed the remarkable fluctuation of mAP, Precision, and Recall and staying flat flow in loss values, signalling the overfitting and meaningless for further training. Therefore we decided to finish model training and use the best.pt weight for the final model testing.

2.2 Faster-RCNN Model

Load Dataset

The first step is to import the data set file into Matlab. This step is done with the codes in Figure 1. First, the “load” function is used to load the data file named “matlab.mat”, and then the data named “T” is stored in the “Dataset” variable, so that the data can be manipulated and analysed by accessing the “Dataset” later.

```
data = load('matlab.mat');
Dataset = data.T;
```

After importing the data set file, we split the data set in order to train and evaluate the model. First, the seed of the random number generator is set to 0 through the code “rng(0)” to ensure that the same random arrangement results are obtained in different runs, so as to facilitate the repeatability of the experiment. The line “idx = floor(0.8 * height(Dataset))” calculates the number of samples that will be used for the training set, which is 80% of the sample size. The line “trainingDataTbl = Dataset(shuffledIdx(1:idx),:)” uses a randomly arranged index to select the first 80% of the samples from the "Dataset" as the training set. The line “testDataTbl = Dataset(shuffledIdx(idx+1:end),:)” uses a randomly arranged index to select the remaining 20% of the sample from the "Dataset" as the test set.

```
rng(0)
shuffledIdx = randperm(height(Dataset));
idx = floor(0.8 * height(Dataset));
trainingDataTbl = Dataset(shuffledIdx(1:idx),:);
testDataTbl = Dataset(shuffledIdx(idx+1:end),:);
```

The “imdsTrain = imageDatastore(trainingDataTbl{:,'Path'});” code creates an image data store object named “imdsTrain”. It uses the “Path” column in the training data table “trainingDataTbl”, which contains the image file path for the training data sample. The imageDatastore" function loads these image file paths into the “datastore” object for later use in model training. “bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));” This line of code creates a labeled data store object named “bldsTrain”. It uses all the columns in the training data table “trainingDataTbl” except the “Path” column, which contains the label information from the object detection task. The “boxLabelDatastore” function loads this label data into the “datastore” object for association with the corresponding image data. In the same way, the above method is also used for test sets. These data storage objects will help you effectively organize and manage image data and corresponding label data, and can often be used for object detection tasks in deep learning toolkits.

```
imdsTrain = imageDatastore(trainingDataTbl{:,'Path'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));

imdsTest = imageDatastore(testDataTbl{:,'Path'});
bldsTest = boxLabelDatastore(testDataTbl(:,2:end));
```

What the following codes do is combine the image data store object and the label data store object, effectively associating the input data with its corresponding label, so that the object detection model can be trained and tested.

```
trainingData = combine(imdsTrain,bldsTrain);

testData = combine(imdsTest,bldsTest);
```

The purpose of the following codes are to read a sample from the training data, display the image, and draw an object detection box on the image in order to visualize the object detection task in the training data. This method is used to check and verify the correctness of the training data. data includes picture data, frame coordinates, and categories. By opening the “data” structure, you can see that 1 represents the image and 2 represents the coordinates. Use the “insertShape” function to draw the object detection box on the image “I”.

```
data = read(trainingData);
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,'Rectangle',bbox);
annotatedImage = imresize(annotatedImage,2);
figure;
imshow(annotatedImage);
```

Create a Faster-RCNN network

These two lines of code define two important variables: the size of the input image and the number of categories in the task. The “inputSize” variable represents the size of the input image for the neural network. Where "[224 224 3]" indicates that the input image is 224x224 pixels in size and has 3 channels of RGB. This is a typical CNN input size and is suitable for many image classification and object detection tasks. The “numClasses” variable represents the number of categories in the task, which is determined by querying the number of columns in the Dataset. Where “-1” is because the first column in the dataset is the image path instead of the category label. This variable is often used in classification tasks to specify the number of nodes in the output layer of the neural network to ensure correct category predictions.

```
inputSize = [224 224 3];
numClasses = width(Dataset)-1;
```

The purpose of these code are to prepare the data for the target detection task, including the pre-processing of the training data and the estimation of the anchor box used to generate the candidate target region. Anchor frame is an important concept in object detection, which is used to provide multi-scale and multi-aspect ratio candidate regions to help models locate and classify target objects.

```
preprocessedTrainingData = transform(trainingData, @(data)preprocessData(data,inputSize));
numAnchors = 4;
anchorBoxes = estimateAnchorBoxes(preprocessedTrainingData,numAnchors)
```

After that, we use Resnet50 as the backbone network for feature extraction.

```
featureExtractionNetwork = resnet50;
```

The following codes set up a feature extraction layer.

```
featureLayer = 'activation_40_relu';
```

The following codes pass the above parameters to the “fasterRCNNLayers” function to create a layer diagram of the Faster R-CNN model that includes feature extraction, region suggestion, target

classification, and bounding box regression for object detection tasks. Specific functions can be queried in the Matlab official website.

```
lgraph = fasterRCNNLayers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer);
```

Train Faster-RCNN

This codes define some training options for model training. These options are passed to the deep learning model's training function to specify the configuration and parameters of the training. MaxEpochs indicates the maximum number of iteration rounds of training, which is set to 10. In each round, the entire training set is used to update the weights of the model. “MiniBatchSize” represents the size of each small batch, indicating that only one sample is used each time model parameters are updated. InitialLearnRate is the initial value of the learning rate, which is set to 0.001. The learning rate is a hyperparameter used to control the weight updating step length, and an appropriate learning rate can ensure the stability and convergence of model training. The configuration of the training is adjusted according to the characteristics of the task and the data set to ensure the best training results.

```
options = trainingOptions('sgdm',...
    'MaxEpochs',10,...
    'MiniBatchSize',1,...
    'InitialLearnRate',1e-3,...
    'CheckpointPath',tempdir);
```

The code “doTrainingAndEval = true;” defines a logical variable that controls whether the model is trained and evaluated. If set to true, model training and evaluation is performed; If it is set to false, the pre-trained model is loaded and the training step is skipped.

```
doTrainingAndEval = true;
if doTrainingAndEval
    [detector, info] = trainFasterRCNNObjectDetector(trainingData,lgraph,options, ...
        'NegativeOverlapRange',[0 0.3], ...
        'PositiveOverlapRange',[0.6 1]);
    save('net.mat','detector')
    save('netInfo.mat','info')
else
    pretrained = load('net.mat');
    detector = pretrained.detector;
end
```

This codes are used to detect the target object in the test image using the trained Faster R-CNN model, and then return the detected bounding box and confidence score. First, the image to be detected is loaded into variable “I”, and then the image size is adjusted to match the input size during model training, so as to facilitate the accuracy of prediction. After object detection is performed on the trained model, a rectangular region array representing the location of the object and a detection confidence score are returned. Where the array represents the object location, and the higher the confidence score, the more reliable the detected object.

```
I = imread(testDataTbl.imageFilename{10});
I = imresize(I,inputSize(1:2));
[bboxes,scores] = detect(detector,I);
```

The following codes are used to draw the detected object bounding box on the image, add the detected object information to the image in the form of a comment, and then display the annotated image. Draw rectangles around each detected object on the image, and add a label next to each rectangle to detect the confidence score. Through these operations, the object detection results of the model can be visually seen on the image, so as to further analyze and evaluate the performance of the model.

```
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)
```

Evaluate results with test sets

The following codes preprocess the test data. With this operation, the images in the test data are adjusted to the same input size as when trained in order to input them into the model for prediction. This is to ensure the compatibility of the test data with the model in order to obtain accurate predictive results.

```
testData = transform(testData,@(data)preprocessData(data,inputSize));
```

This codes are used to perform the object detection task and generate the detection results. The value of the variable “doTrainingAndEval” determines whether to perform the test using the trained model or load the pre-trained test results. If the test is performed using a trained model, the small batch size is set to 4 per process, and the test results include the bounding box of the object and the test confidence score. With this condition, you can choose whether to perform object detection or use existing detection results as needed, which allows you to perform object detection tasks without retraining the model.

```
doTrainingAndEval = true
if doTrainingAndEval
    detectionResults = detect(detector,testData,'MinibatchSize',4);
else
    pretrained = load('net.mat');
    detectionResults = pretrained.detectionResults;
end
```

The following codes evaluate the performance of the object detection task, including calculating Precision, Recall, and Average Precision (AP). These performance metrics are very helpful for understanding model accuracy and recall performance, especially in multi-class object detection tasks.

```
classID = 1;
metrics = evaluateObjectDetection(detectionResults,bldsTest);
precision = metrics.ClassMetrics.Precision{classID};
recall = metrics.ClassMetrics.Recall{classID};
[ap, recall, precision] = evaluateDetectionPrecision(detectionResults,testData);
```

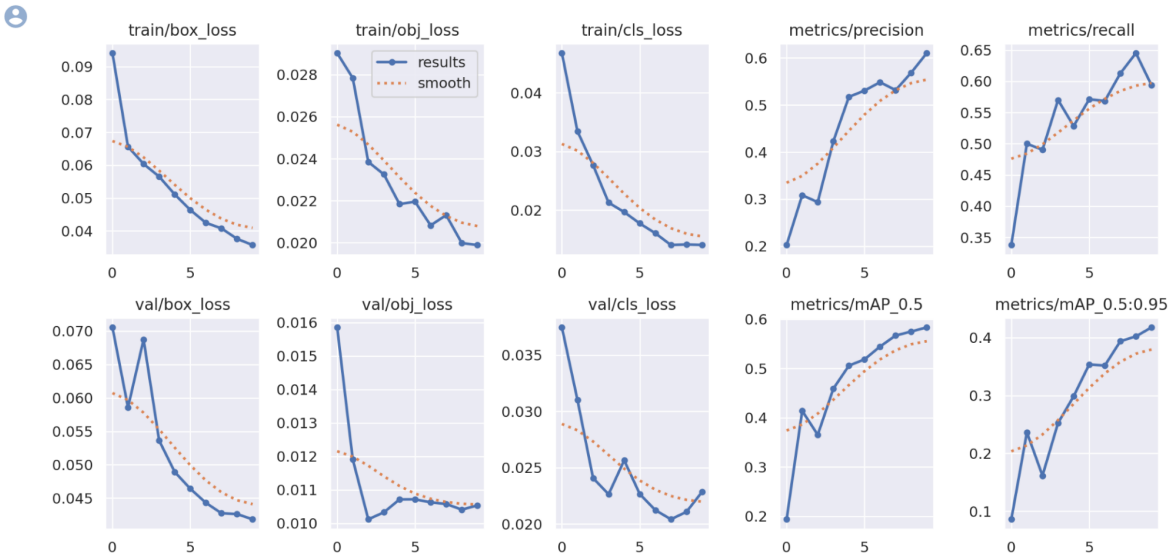
The following codes are used to plot the recall - accuracy curve and display the average precision (AP) value on the chart. With this chart, you can visualize the tradeoff between the recall rate and accuracy of the model, as well as the overall performance of the model. Average precision is usually an important performance indicator to measure the accuracy of a model under different recall rates, and the higher the value, the better the performance of the model.

```
figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))
```

3. Results and Evaluation Method

3.1 Evaluation of Yolov5

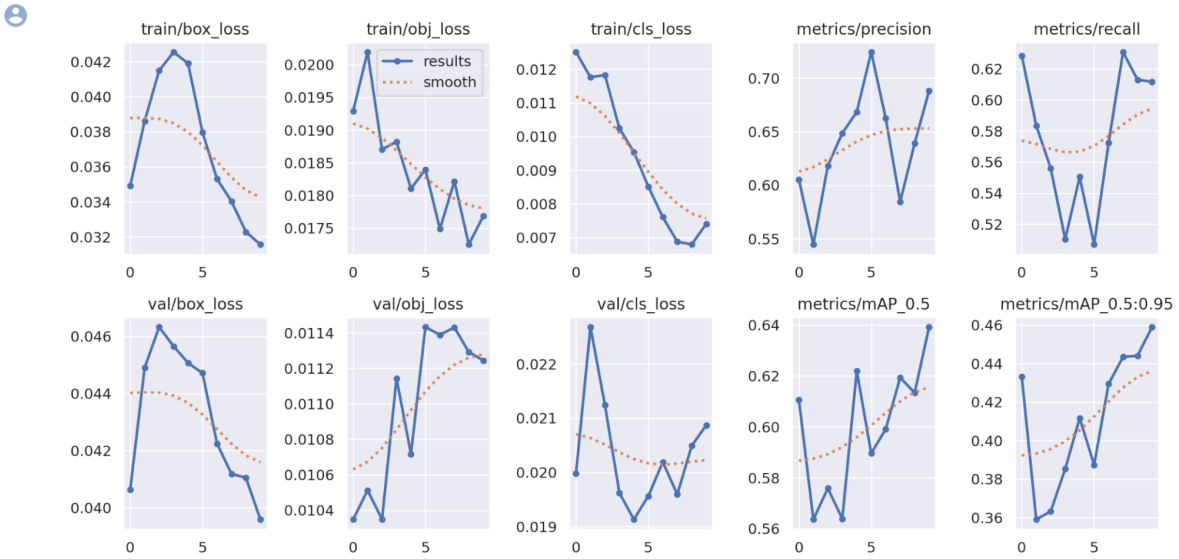
First 10 epoch



<fig> first 10 epoch

For the initial 10 epochs of training the YOLOv5 model, we observed a decrease in the overall loss, which is an encouraging sign. However, during this period, we noticed a decline in performance metrics, specifically precision, recall, mAP50, and mAP50-95. Therefore, we have made the decision to extend the training process to further enhance model performance.

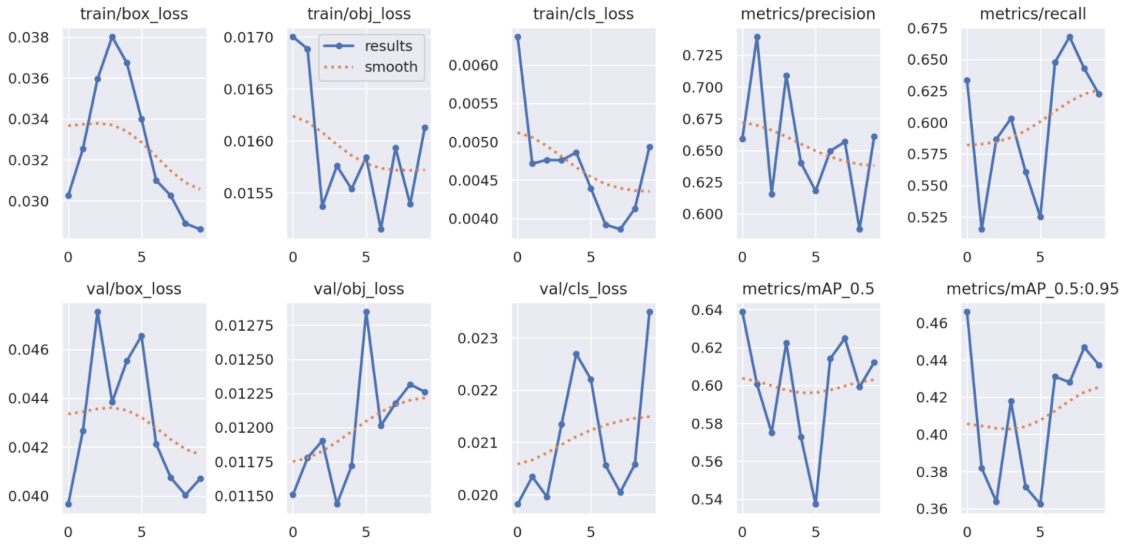
Second 10 epoch



<fig> second 10 epoch

During the subsequent 10 epochs of training, we observed fluctuations in both loss and performance metrics, including precision, recall, and mAP. Despite these fluctuations, we believe that there is room for further improvement in the model. Consequently, we have chosen to continue training to achieve better results.

Last 10 epoch



<fig>last 10 epoch

In the last 10 epochs of training, we observed significant fluctuations across the entire epochs. In light of this instability, we made the decision to exit the training process at this juncture.

Evaluation using Validation dataset

```
/content/drive/MyDrive/Colab_Models/best.pt --data /content/data.yaml --img-size 416 --conf-thres 0.25 --iou-thres 0.45 --task val
```

```
/content/yolov5
val: data=/content/data.yaml, weights=['/content/drive/MyDrive/Colab_Models/best.pt'], batch_size=32, imgsz=416, conf_thres=0.25, iou_thr
WARNING ⚠ confidence threshold 0.25 > 0.001 produces invalid results
YOLOv5 🚀 v7.0-230-g53efd07 Python-3.10.12 torch-2.1.0+cu118 CPU

Fusing layers...
YOLOv5s summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
val: Scanning /content/valid/labels.cache... 250 images, 0 backgrounds, 0 corrupt: 100% 250/250 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 8/8 [01:20<00:00, 10.10s/it]
                   all        250        454      0.674      0.638      0.675      0.525
             Ambulance        250         64      0.683      0.875      0.825       0.69
                   Bus        250         46      0.773      0.739       0.79      0.669
                   Car        250        238       0.64      0.538      0.627      0.432
            Motorcycle        250         46       0.63      0.739       0.66       0.46
                 Truck        250         60      0.643        0.3      0.472      0.375
Speed: 4.6ms pre-process, 313.0ms inference, 1.1ms NMS per image at shape (32, 3, 416, 416)
Results saved to runs/val/exp5
```

<fig> result for task validation

In task validation, the model demonstrated an overall precision of 0.674 and recall of 0.638, resulting in a mean average precision (mAP50) of 0.675 and mAP50-95 of 0.525 for the entire dataset. Specific vehicle classes varied in performance, with Ambulance and Bus showing higher precision and recall.

Evaluation using Test dataset

```
/content/drive/MyDrive/Colab_Models/best.pt --data /content/data.yaml --img-size 416 --conf-thres 0.25 --iou-thres 0.45 --task test
```

```
/content/yolov5
val: data=/content/data.yaml, weights=['/content/drive/MyDrive/Colab_Models/best.pt'], batch_size=32, imgsz=416, conf_thres=0.25, iou_thre
WARNING ⚠ confidence threshold 0.25 > 0.001 produces invalid results
YOLOv5 🚀 v7.0-230-g53efd07 Python-3.10.12 torch-2.1.0+cu118 CPU

Fusing layers...
YOLOv5s summary: 157 layers, 7023610 parameters, 0 gradients, 15.8 GFLOPs
test: Scanning /content/test/labels.cache... 126 images, 0 backgrounds, 0 corrupt: 100% 126/126 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 4/4 [00:41<00:00, 10.35s/it]
                   all        126        258      0.855      0.694      0.803        0.6
             Ambulance        126         18        0.9          1      0.995      0.862
                   Bus        126         38      0.923      0.789      0.879      0.651
                   Car        126        150      0.825       0.63      0.775       0.51
            Motorcycle        126         32      0.784      0.453      0.637      0.383
                 Truck        126         20      0.842        0.6      0.732      0.592
Speed: 4.7ms pre-process, 316.2ms inference, 0.6ms NMS per image at shape (32, 3, 416, 416)
Results saved to runs/val/exp9
```

<fig> result for task test

In the task test, it achieved an overall precision of 0.855 and recall of 0.694, resulting in a mean average precision (mAP50) of 0.803. Class-specific results varied, with Ambulance displaying outstanding accuracy.

Visualise the Performance

This image below is a load model.

Load Model

```
[ ] !python detect.py --weights /content/drive/MyDrive/Colab_Models/best.pt \
                      --img-size 416 \
                      --conf-thres 0.25 \
                      --iou-thres 0.45 \
                      --source /content/test/images \
                      --project /content/drive/MyDrive/YOLOv5_Results \
                      --name exp \
                      --exist-ok
```

```
image 71/126 /content/test/images/95bf6406c152cf40_jpg.rf.2KlhzCSeifqpfUPfwyW9.jpg: 416x416 1 Car, 1 Truck, 179.6ms
image 72/126 /content/test/images/95bf6406c152cf40_jpg.rf.58a7d5b1c4309ad003108642c51ae8b3.jpg: 416x416 1 Car, 1 Truck, 195.6ms
image 73/126 /content/test/images/9876c285fe7bc3a0_jpg.rf.7aa68726bc91a7c26d99c23dd79cc833.jpg: 416x416 2 Ambulances, 192.8ms
image 74/126 /content/test/images/9876c285fe7bc3a0_jpg.rf.iBBVQWEpeWVbehE5MJ6Q.jpg: 416x416 2 Ambulances, 190.9ms
image 75/126 /content/test/images/9ca8e20e0869f289_jpg.rf.4bb7001185dba72f41b354c6235e79cb.jpg: 416x416 (no detections), 172.8ms
image 76/126 /content/test/images/9ca8e20e0869f289_jpg.rf.ScBMQ9VslO942N7rh8rz.jpg: 416x416 (no detections), 186.3ms
image 77/126 /content/test/images/9d2d424099458956_jpg.rf.9cc91fddce7c7441e0f085d139b62aac.jpg: 416x416 1 Ambulance, 175.0ms
image 78/126 /content/test/images/9d2d424099458956_jpg.rf.vihT0YRoLD0QuODDlBz7.jpg: 416x416 1 Ambulance, 178.9ms
image 79/126 /content/test/images/9fdf0bf1160fca8e_jpg.rf.183a3d9dfcbbd5ad78f2148712430881.jpg: 416x416 1 Motorcycle, 185.2ms
image 80/126 /content/test/images/9fdf0bf1160fca8e_jpg.rf.ERTEMjgijaUbIUeFSoNX.jpg: 416x416 1 Motorcycle, 180.8ms
image 81/126 /content/test/images/a3cffcdbff959432_jpg.rf.1f1f8cfcdfe052a8ec72e63b11889b66.jpg: 416x416 1 Car, 1 Truck, 179.7ms
image 82/126 /content/test/images/a3cffcdbff959432_jpg.rf.4MQ1NmvMx33nwLe9QSvu.jpg: 416x416 1 Car, 1 Truck, 182.6ms
image 83/126 /content/test/images/a7674f15d9b1f35f_jpg.rf.7bbd38887231affc55a0083cfd2f422e.jpg: 416x416 1 Ambulance, 287.4ms
image 84/126 /content/test/images/a7674f15d9b1f35f_jpg.rf.pL09u0Oz8DRvTB3AnBXT.jpg: 416x416 1 Ambulance, 291.6ms
image 85/126 /content/test/images/aed83d879a22a6c4_jpg.rf.ada6c87c88d32689f79e3a03c9ebd314.jpg: 416x416 (no detections), 298.7ms
image 86/126 /content/test/images/aed83d879a22a6c4_jpg.rf.jXuDPuNwGM96NxEjG3PY.jpg: 416x416 (no detections), 328.6ms
image 87/126 /content/test/images/b3736bcf6bec5308_jpg.rf.988ed2199cddfaecb34a83859649121b.jpg: 416x416 1 Bus, 302.0ms
```

<fig> load model

```
from IPython.display import Image, display
import os

results_dir = "/content/drive/MyDrive/YOLOv5_Results/exp"
for img_file in os.listdir(results_dir):
    if img_file.endswith(".jpg") or img_file.endswith(".png"):
        display(Image(filename=os.path.join(results_dir, img_file)))
```

<fig> best model weight

The image above represents the best model weight, and it was employed to evaluate the model's performance using test images. The image also depicts code used in this evaluation.

The code identifies image files with ".jpg" and ".png" extensions in the defined directory and randomly selects 10 of them for display. It utilises the IPython library to showcase the chosen images, making it a handy tool for visually inspecting and analysing a subset of images from a larger collection stored in the specified directory.

As shown below, these are random 10 images that are detected by the model.

<fig1>

<fig2>

The first figure is a car, and this accuracy is 88% . The second figure is a truck, and this accuracy is 91%.

<fig3>

<fig4>

The third figure is car and this accuracy is 76%. The fourth figure is a car, and this accuracy is 95%.

<fig5>

<fig6>

The fifth figure is the car, and this accuracy is 85%. The sixth figure is a bus, and this accuracy is 90%.

<fig7>

<fig8>

The seventh figure is a bus. And this accuracy is 88%. The eighth figure is a car, and this accuracy is 28.

<fig9>

<fig10>

The ninth figure is a car. And this accuracy is 71%. The tenth figure is a car, and this accuracy is 90%.

3.2 Evaluation of Faster-RCNN

In the first Faster Rcnn model training, we made the following settings

```
options = trainingOptions('sgdm',...
    'MaxEpochs',2,...
    'MiniBatchSize',1,...
    'InitialLearnRate',1e-3,...
    'CheckpointPath',tempdir);
```

We set MaxEpochs to 2 and MiniBatchSize to 1 and training the model, the training result is below:

```
Initializing input data normalization.
|=========================================================================================================================================
====|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  RPN Mini-batch  |  RPN Mini-batch  |  Base
Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |     Accuracy     |       RMSE       |      Rate
|
|=========================================================================================================================================
====|
|       1 |           1 |       00:00:18 |       1.7563 |       43.84% |         0.17 |           56.25% |             0.80 |
0.0010 |
|       1 |          50 |       00:17:33 |       1.0675 |       98.91% |         0.15 |           80.47% |             0.92 |
0.0010 |
|       1 |         100 |       00:38:49 |       0.4331 |       99.46% |         0.17 |           96.09% |             0.61 |
0.0010 |
|       1 |         150 |       01:02:20 |       0.3787 |       99.45% |         0.11 |           95.31% |             0.63 |
0.0010 |
|       1 |         200 |       01:24:53 |       0.4050 |       99.40% |         0.14 |           92.19% |             0.63 |
0.0010 |
|       1 |         250 |       01:56:41 |       0.5314 |       99.49% |         0.18 |           97.66% |             0.81 |
0.0010 |
|       1 |         300 |       02:31:09 |       0.4447 |       97.16% |         0.12 |           96.88% |             0.63 |
0.0010 |
|       1 |         350 |       02:58:32 |       0.5876 |       99.83% |         0.13 |           85.94% |             0.92 |
0.0010 |
|       1 |         400 |       03:24:19 |       0.3720 |       98.99% |         0.15 |           96.09% |             0.33 |
```

```
0.0010 |
|       2 |         900 |       07:51:50 |       0.3060 |       99.81% |         0.11 |           96.88% |             0.62 |
0.0010 |
|       2 |         950 |       08:17:08 |       0.6242 |       98.97% |         0.16 |           96.88% |             1.03 |
0.0010 |
|       2 |        1000 |       08:43:43 |       0.4227 |       99.68% |         0.11 |           98.44% |             0.76 |
0.0010 |
|       2 |        1050 |       09:10:21 |       0.3654 |       99.20% |         0.16 |           96.88% |             0.67 |
0.0010 |
|       2 |        1100 |       09:36:44 |       0.5332 |       99.84% |         0.08 |           97.66% |             0.78 |
0.0010 |
|       2 |        1150 |       10:03:04 |       0.7027 |       98.16% |         0.13 |           83.59% |             0.64 |
0.0010 |
|       2 |        1200 |       10:26:59 |       0.4356 |       99.44% |         0.12 |           92.97% |             0.66 |
0.0010 |
|       2 |        1250 |       10:51:22 |       0.5385 |       99.55% |         0.17 |           87.50% |             0.86 |
0.0010 |
|       2 |        1300 |       11:17:52 |       0.4901 |       97.53% |         0.11 |           91.41% |             0.78 |
0.0010 |
|       2 |        1350 |       11:42:46 |       0.8221 |       99.09% |         0.13 |           85.16% |             1.05 |
0.0010 |
|       2 |        1400 |       12:08:56 |       0.5075 |       99.70% |         0.18 |           93.75% |             0.67 |
0.0010 |
|       2 |        1404 |       12:10:51 |       0.5456 |       98.04% |         0.18 |           89.84% |             0.75 |
0.0010 |
|=========================================================================================================================================
====|
```

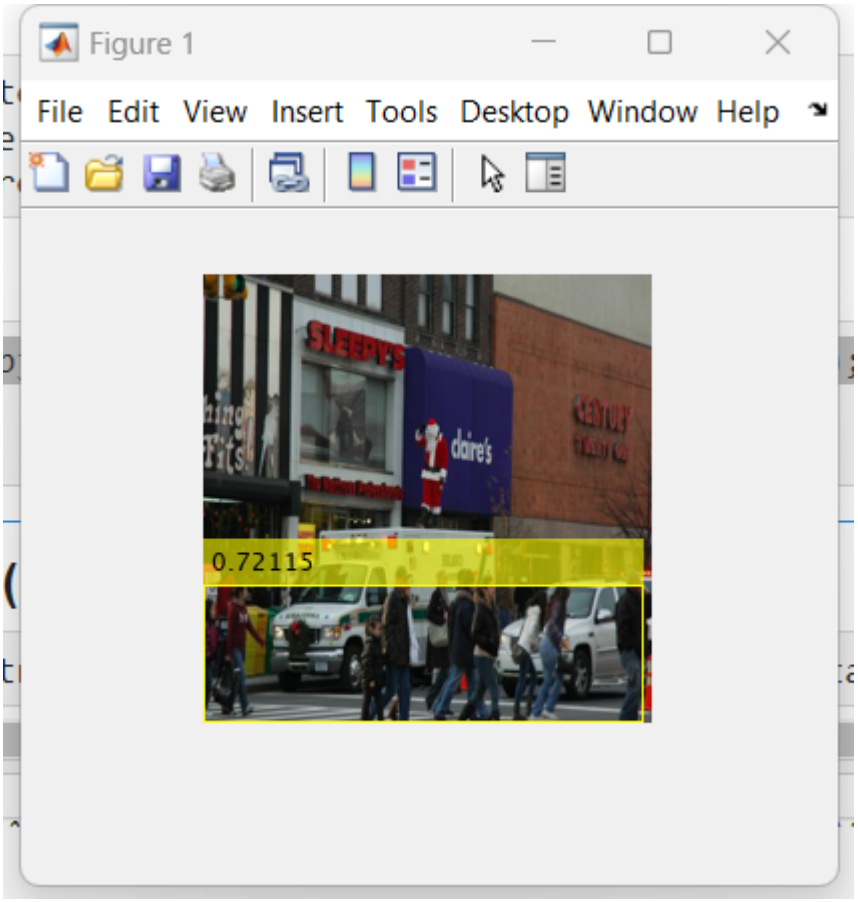
This time it was run only as a test, so only MaxEpochs was set to 2 and Mini BatchSize was set to 1, but only that, it took for 12 hours to run the training model.

Then I set the value of MaxEpochs to 10, and set MiniBatchSize back to 1 and try it. Here is a partial screenshot of the running result:

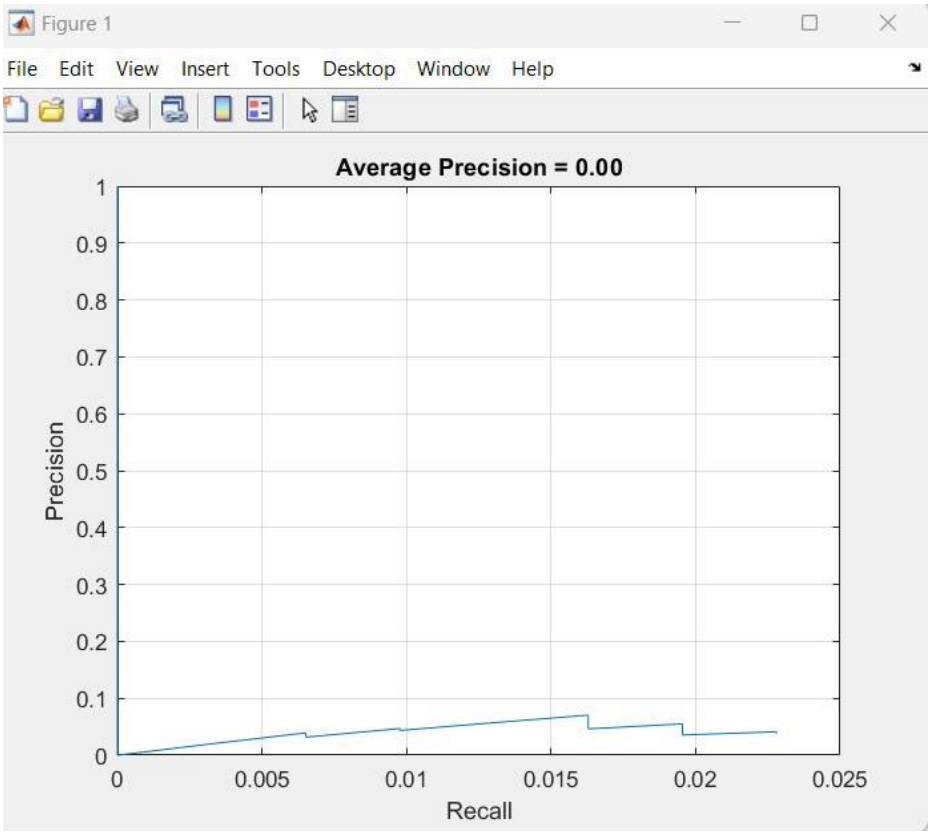
```
Training on single GPU.
Initializing input data normalization.
|=========================================================================================================================================
====|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Mini-batch  |  RPN Mini-batch  |  RPN Mini-batch  |  Base
Learning  |
|         |             |   (hh:mm:ss)   |     Loss     |   Accuracy   |     RMSE     |     Accuracy     |       RMSE       |      Rate
|
|=========================================================================================================================================
====|
|       1 |           1 |       00:00:02 |       2.2163 |       37.74% |         0.17 |           43.75% |             1.49 |
0.0010 |
|       1 |          50 |       00:01:09 |       1.0667 |       98.82% |         0.16 |           78.12% |             1.14 |
0.0010 |
|       1 |         100 |       00:02:35 |       0.9654 |       98.63% |         0.17 |           79.69% |             0.71 |
0.0010 |
|       1 |         150 |       00:04:07 |       0.5953 |       99.45% |         0.12 |           96.09% |             0.90 |
0.0010 |
|       1 |         200 |       00:05:35 |       0.6508 |       98.93% |         0.12 |           91.41% |             0.91 |
0.0010 |
|       1 |         250 |       00:06:50 |       1.6784 |       98.95% |         0.20 |           88.28% |             1.90 |
0.0010 |
|       1 |         300 |       00:08:22 |       0.9010 |      100.00% |         0.15 |           98.44% |             1.23 |
0.0010 |
|       1 |         350 |       00:09:54 |       0.6670 |       98.50% |         0.12 |           96.09% |             0.82 |
0.0010 |
```

```
0.0010 |
|      10 |        6550 |       05:40:38 |       0.0288 |      100.00% |         0.07 |           99.22% |             0.18 |
0.0010 |
|      10 |        6600 |       05:43:43 |       0.0260 |      100.00% |         0.09 |          100.00% |             0.19 |
0.0010 |
|      10 |        6650 |       05:46:50 |       0.0942 |       99.73% |         0.07 |           98.44% |             0.23 |
0.0010 |
|      10 |        6700 |       05:49:16 |       0.0438 |      100.00% |         0.06 |          100.00% |             0.25 |
0.0010 |
|      10 |        6750 |       05:52:13 |       0.0670 |       99.87% |         0.12 |          100.00% |             0.30 |
0.0010 |
|      10 |        6800 |       05:55:10 |       0.0527 |       99.74% |         0.14 |          100.00% |             0.23 |
0.0010 |
|      10 |        6850 |       05:58:35 |       0.0674 |       99.68% |         0.15 |          100.00% |             0.27 |
0.0010 |
|      10 |        6900 |       06:02:19 |       0.1711 |      100.00% |         0.11 |          100.00% |             0.57 |
0.0010 |
|      10 |        6950 |       06:05:23 |       0.0217 |      100.00% |         0.09 |          100.00% |             0.17 |
0.0010 |
|      10 |        7000 |       06:08:13 |       0.0690 |      100.00% |         0.16 |          100.00% |             0.29 |
0.0010 |
|      10 |        7020 |       06:09:47 |       0.1587 |       99.66% |         0.13 |           99.22% |             0.48 |
0.0010 |
|=========================================================================================================================================
====|
Training finished: Max epochs completed.
Detector training complete.
```

Then I use code to show the picture:



The value looks good however when we use the results of this training to evaluate and draw the curve, and the results are as follows:



As the result shows, the curve floats between 0 and 0.1 of the precision value. Obviously, the effect of this result is very poor and takes a lot of time, so after discussing with the group, we finally gave up using the Faster Rcnn model and focused on the Yolov5 model.

4. Discussion and Challenges

Achievement of this project and Future direction

Through this project, we could understand the structure of object detection models, training methodologies and how to optimise the model during the training process. Moreover, we acknowledge the way to evaluate the training process and test our model using key criteria such as mAP, IoU, and loss values. These insights have been used for refining and enhancing the model performance.

Although we struggled with the Faster-RCNN model, we successfully obtained a YOLOv5 model which can detect and classify multiple vehicles on the road. This model is expected to train with larger dataset to enhance the performance and detect wider spectrum of vehicles, which can be applied in road surveillance cameras to detect ineligible vehicles in real-time for safety and improve traffic flow in future.

In addition, the team's primary takeaways from this assignment on "Vehicle Object Detection of Road Traffic" included not just technical elements but also efficient teamwork and flexibility. When faced with resource limitations, the team showed adaptability by switching our focus from Faster R-CNN to YOLOv5 using the Google Collab, demonstrating their ability to work as a cohesive unit to adapt to changing project needs. Together, we developed critical abilities such as effective resource management and the understanding of how to balance resource consumption and data augmentation.

Regular team meetings and the usage of visual aids allowed the team to obtain important insights into model selection, further demonstrating adeptness at project management. Collaboration was further emphasised via practical experience in assessing model performance, comparing models, and developing visualisation and communication skills. The team will surely be more equipped for future object detection and computer vision endeavours with a deeper understanding of the nuances of model training and the importance of benchmarking and performance metrics in assessing model quality, highlighting the critical role that teamwork plays in project success.

Challenges

- **Resource restriction (GPU)**

In order to overcome the challenge of resource constraints, we appropriately recognized the constraints associated with GPU availability and wisely chose to switch from Faster R-CNN to lighter YOLOv5 small model. This choice greatly improved the efficiency of resource utilisation, resulting in quicker model training and more efficient resource allocation, and offered a workable solution to this constraint.

- **Small dataset**

We freely recognized the constraints of having a tiny dataset for our project when tackling the difficulty of a small dataset. We showed meticulous data management by dividing the dataset into

training, validation, and test subsets in order to handle this difficulty. We chose to forgo data augmentation due to practical considerations regarding resource limitations, and the outcomes accurately represent the performance obtained with the original dataset.

- **Low experience**

We acknowledged the limitations of our team's deep learning experience especially for object detection when we set out to address the difficulty of poor experience, viewing it as a significant obstacle from the beginning of the project. What's amazing, though, is that our group's commitment and resolve were evident even in the face of this first obstacle. We showed incredible flexibility and a strong dedication to learning throughout the project by effectively implementing and training deep learning models. This practical experience has shown to be extremely beneficial, providing a strong basis for our future work in the sector and highlighting our team's potential for advancement in the machine learning and deep learning fields.

- **Data Augmentation**

In light of the data augmentation challenge, it is important to note that, although data augmentation is a frequently used method to overcome the restrictions of tiny datasets, we consciously and rationally chose not to apply it because of the resource limitations we found. Our team's capacity to adapt and be resourceful in overcoming the obstacles we encountered is evidenced by our conscious decision to work with the original images, which also shows how realistic and practical our approach was to solving the problem.

- **Challenge for Faster Rcnn**

I tried many times when running the Faster Rcnn model, and only showed the first and last test results in the evaluation section. The second time, we changed MaxEpochs value to 50 and MiniBatchSize and training again, however, When it took 30 hours to run to the 12th epochs, the code reported an error and stopped running because the computer performance had reached its limit. So we changed the value of MaxEpochs to 12 and tried to run it again. We waited for 12 hours and got the same error.

```
|       2 |        1250 |       00:38:21 |       1.0062 |       93.24% |         0.14 |           79.69% |
Out of memory.

Error in nnet.internal.cnn.layer.strategy.GlobalAveragePoolingStrategy/backward (line 48)
            dX = 1/prod(szPool) * ones(szX) .* dZ;

Error in nnet.internal.cnn.layer.GlobalAveragePooling2D/backward (line 81)
            dX = this.ExecutionStrategy.backward(dZ, X);
```

In addition, there is another reason to give up using Fatser Rcnn. The original intention of the design code is to detect the size of the vehicle and set it into two categories, but the data set we found on Roboflot shows five categories, and the correction needs to be marked with a picture in MATLAB, which requires a lot of working time, and the follow-up training model also requires a lot of working time, so. This is also one of the reasons to give up Faster Rcnn finally.

If we had more time, we would try to use colab to train the Faster Rcnn model. In colab, we can use GPU to get faster training time, and I can solve the problem of multi-classification through the following code:

```
from torchvision.models.detection.faster_rcnn import FastRCNNPredicto
num_classes = 6

model = fasterrcnn_resnet50_fpn(pretrained=False)

in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

model.load_state_dict(torch.load(model_path))

```

5. Project Management and TeamWork

5.1 Project Management Methodology

To manage our project, we developed a Swimlane diagram to understand the entire scope of the project. We divided the project into 5 phases: project preparation, data pre-processing, Model implementation, Model Evaluation and Summarisation.

In the initial phase, we identified the main problem to be solved for our project and researched object detection using computer vision. Then we selected Faster-RCNN and YOLO as our models.

Our next step is research about the vehicle image dataset for re-training and fine-tuning the model. We aimed to find the large open-source image dataset included the annotation files. During the data pre-processing, we attempted various data augmentation techniques, changed the format to fit into the specific models, and splitted the dataset. However as data augmentation makes model training longer and decreased performance, we decided to proceed with the original images.

Then we divided our team into two sub-groups for the Faster-RCNN and YOLOv5 model training. We kept trying to adjust the parameters to find the best model.

Following the training, we conducted a thorough analysis of the training result and model performance using the evaluation criteria mAP and IoU, and tested the model with sepearte test dataset. Lastly we summarised our findings, discussed our challenges and future improvement and planned for writing the report and preparing presentation

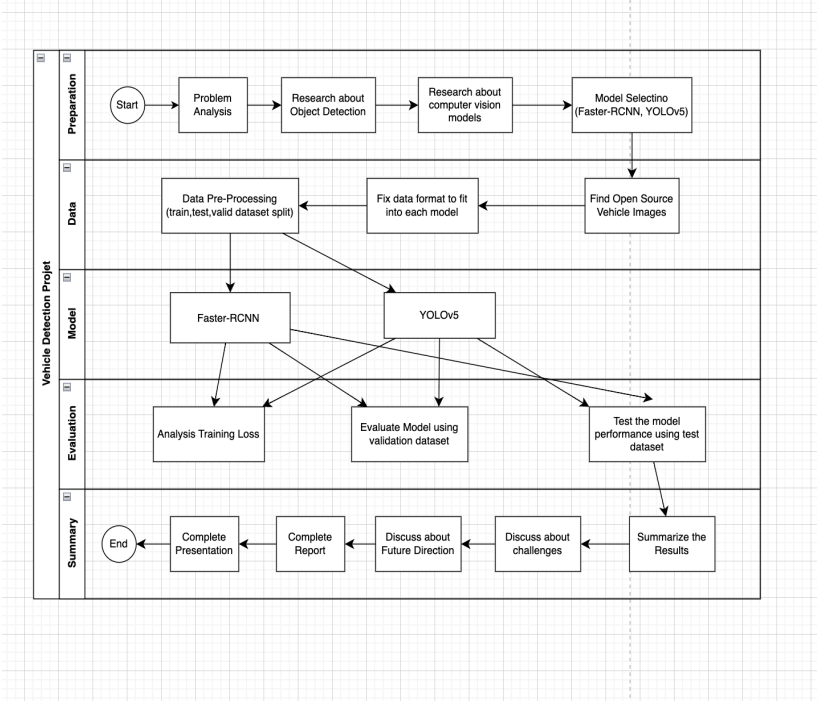


Figure: Swimlane Diagram

6. References

Roboflow dataset
https://public.roboflow.com/object-detection/vehicles-openimages

YOLOv5 model
https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a

Ultralytics
https://docs.ultralytics.com/yolov5/quickstart_tutorial/

Faster-RCNN
https://medium.com/mlearning-ai/object-detection-explained-faster-r-cnn-23e7ab57991d

7. Appendix

Links of the Colab notebook for YOLOv5
https://drive.google.com/file/d/14j__j3zS3pByAgS7mrSJRDsPHileSB_c/view?usp=sharing

Link of Swimlane Diagram for Project Management
https://app.diagrams.net/#G11TBs1BJAyvwx8W3G5Le2Je3MIBU1RJhd

Screenshot of Yolov5

```
1 from google.colab import drive
2 drive.mount('/content/drive')
3
```

```
Mounted at /content/drive
```

```
1 !pip install -q roboflow
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 kB 364.9 kB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 155.3/155.3 kB 3.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.7/178.7 kB 9.8 MB/s eta 0:00:00
```

```
1 %cd /content
2 !curl -L "https://public.roboflow.com/ds/toVJdWBuRp?key=I1KM5dFkuo" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

```
/content
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   893  100   893    0     0   1377      0 --:--:-- --:--:-- --:--:--  1375
100 38.4M  100 38.4M    0     0  30.3M      0  0:00:01  0:00:01 --:--:--  225M
Archive:  roboflow.zip
 extracting: README.dataset.txt
 extracting: README.roboflow.txt
 extracting: data.yaml
   creating: test/
```

```
1 %cd /content
2 ! git clone https://github.com/ultralytics/yolov5.git
```

```
/content
Cloning into 'yolov5'...
remote: Enumerating objects: 16031, done.
remote: Counting objects: 100% (64/64), done.
remote: Compressing objects: 100% (39/39), done.
```

```
%cd /content/yolov5/
!pip install -r requirements.txt
```

```
/content/yolov5
Collecting gitpython>=3.1.30 (from -r requirements.txt (line 5))
  Downloading GitPython-3.1.40-py3-none-any.whl (190 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 190.6/190.6 kB 761.4 kB/s eta 0:00:00
Requirement already satisfied: matplotlib>=3.3 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 6)) (3.7.1)
Requirement already satisfied: numpy>=1.22.2 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 7)) (1.23.5)
Requirement already satisfied: opencv-python>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from -r requirements.txt (line 8)) (4.8.0.76)
```

First training

```
%cd /content/yolov5/
!python train.py --img 416 \
                 --batch 16 \
                 --epochs 10 \
                 --data /content/data.yaml \
                 --cfg /content/yolov5/models/yolov5s.yaml \
                 --weights yolov5s.pt \
                 --name vehicle_yolov5s_results #我需要详细介绍这部分
```

```
/content/yolov5
train: weights=yolov5s.pt, cfg=/content/yolov5/models/yolov5s.yaml, data=/content/data.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=10, batch_size=16, imgs
github: up to date with https://github.com/ultralytics/yolov5 ✅
YOLOv5 🚀 v7.0-231-gc2f131a Python-3.10.12 torch-2.1.0+cu118 CPU

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5 🚀 runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
Downloading https://ultralytics.com/assets/Arial.ttf to /root/.config/Ultralytics/Arial.ttf...
```

First TensorBoard

```
%load_ext tensorboard
%tensorboard --logdir /content/yolov5/runs/
```


```
from IPython.display import Image
```

```
# we can also output some older school graphs if the tensor board isn't working for whatever reason...
from utils.plots import plot_results  # plot results.txt as results.png
Image(filename='/content/yolov5/runs/train/vehicle_yolov5s_results/results.png', width=1000)  # view results.png
```

```
import os
SAVE_PATH = "/content/drive/MyDrive/Colab_Models"
os.makedirs(SAVE_PATH, exist_ok=True)
```

```
import shutil
```

```
destination_path = '/content/drive/MyDrive/Colab_Models'
```

```
shutil.copy('/content/yolov5/runs/train/vehicle_yolov5s_results/weights/last.pt', destination_path)
shutil.copy('/content/yolov5/runs/train/vehicle_yolov5s_results/weights/best.pt', destination_path)
```

```
'/content/drive/MyDrive/Colab_Models/best.pt'
```

Second Training

```
%cd /content/yolov5/

!python train.py --img 416 --batch 16 --epochs 10 --data /content/data.yaml --cfg /content/yolov5/models/yolov5s.yaml --weights /content/drive/MyDrive/Colab_
#解释下面几个hyperparameters，Comet，TensorBoard，optimizer
```

```
/content/yolov5
train: weights=/content/drive/MyDrive/Colab_Models/last.pt, cfg=/content/yolov5/models/yolov5s.yaml, data=/content/data.yaml, hyp=data/hyps/hyp.scratch-low.yam
github: up to date with https://github.com/ultralytics/yolov5 ✅
YOLOv5 🚀 v7.0-228-g4d687c8 Python-3.10.12 torch-2.1.0+cu118 CPU
```

Second TensorBoard

```
%load_ext tensorboard
%tensorboard --logdir /content/yolov5/runs/
```

```
from utils.plots import plot_results  # plot results.txt as results.png
Image(filename='/content/yolov5/runs/train/vehicle_yolov5s_results_ continued/results.png', width=1000)  # view results.png
```

Last Training

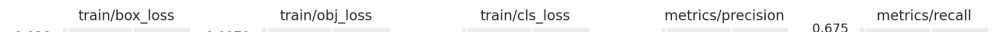
```
%cd /content/yolov5/

!python train.py --img 416 --batch 16 --epochs 10 --data /content/data.yaml --cfg /content/yolov5/models/yolov5s.yaml --weights /content/drive/MyDrive/Colab_
```

Last TensorBoard

```
%load_ext tensorboard
%tensorboard --logdir /content/yolov5/runs/
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 30097), started 2:07:21 ago. (Use '!kill 30097' to kill it.)
```


```
from utils.plots import plot_results  # plot results.txt as results.png
Image(filename='/content/yolov5/runs/train/vehicle_yolov5s_results_continued2/results.png', width=1000)  # view results.png
```



Model Testing

```
!python detect.py --weights /content/drive/MyDrive/Colab_Models/best.pt \
                  --img-size 416 \
                  --conf-thres 0.25 \
                  --iou-thres 0.45 \
                  --source /content/test/images \
                  --project /content/drive/MyDrive/YOLOv5_Results \
                  --name exp \
                  --exist-ok
```

Visualise Test Result

```
from IPython.display import Image, display
import os

results_dir = "/content/drive/MyDrive/YOLOv5_Results/exp"
for img_file in os.listdir(results_dir):
    if img_file.endswith(".jpg") or img_file.endswith(".png"):
        display(Image(filename=os.path.join(results_dir, img_file)))

```

```
import yaml

# YAML 파일을 열고 데이터를 로드합니다.
with open('/content/data.yaml', 'r') as f:
    data = yaml.load(f, Loader=yaml.FullLoader)

# 테스트 데이터 세트의 경로를 추가합니다.
data['test'] = '../test/images'


# 데이터를 YAML 파일에 다시 저장합니다.
with open('/content/data.yaml', 'w') as f:
    yaml.dump(data, f)

```

Evaluate the model with valid dataset

```
%cd /content/yolov5
!python val.py --weights /content/drive/MyDrive/Colab_Models/best.pt --data /content/data.yaml --img-size 416 --conf-thres 0.25 --iou-thres 0.45 --task val

```

Evaluate the model with test dataset

```
%cd /content/yolov5
!python val.py --weights /content/drive/MyDrive/Colab_Models/best.pt --data /content/data.yaml --img-size 416 --conf-thres 0.25 --iou-thres 0.45 --task test

```

Screenshot of Faster-RCNN Model

```
data = load('matlab.mat');
Dataset = data.T;
```

```
rng(0)
shuffledIdx = randperm(height(Dataset));
idx = floor(0.8 * height(Dataset));
trainingDataTbl = Dataset(shuffledIdx(1:idx),:);
testDataTbl = Dataset(shuffledIdx(idx+1:end),:);
```

```
imdsTrain = imageDatastore(trainingDataTbl{:,'Path'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:,2:end));

imdsTest = imageDatastore(testDataTbl{:,'Path'});
bldsTest = boxLabelDatastore(testDataTbl(:,2:end));
```

```
trainingData = combine(imdsTrain,bldsTrain);

testData = combine(imdsTest,bldsTest);
```

```
data = read(trainingData);
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,'Rectangle',bbox);
annotatedImage = imresize(annotatedImage,2);
figure;
imshow(annotatedImage);
```

```
inputSize = [224 224 3];
numClasses = width(Dataset)-1;
```

```
preprocessedTrainingData = transform(trainingData, @(data)preprocessData(data,inputSize));
numAnchors = 4;
anchorBoxes = estimateAnchorBoxes(preprocessedTrainingData,numAnchors)
```

```
featureExtractionNetwork = resnet50;
```

```
featureLayer = 'activation_40_relu';
```

```
lgraph = fasterRCNNLayers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer);
```

```
options = trainingOptions('sgdm',...
    'MaxEpochs',10,...
    'MiniBatchSize',1,...
    'InitialLearnRate',1e-3,...
    'CheckpointPath',tempdir);
```

```
doTrainingAndEval = true;
if doTrainingAndEval
    [detector, info] = trainFasterRCNNObjectDetector(trainingData,lgraph,options, ...
        'NegativeOverlapRange',[0 0.3], ...
        'PositiveOverlapRange',[0.6 1]);
    save('net.mat','detector')
    save('netInfo.mat','info')
else
    pretrained = load('net.mat');
    detector = pretrained.detector;
end
```

```
I = imread(testDataTbl.imageFilename{10});
I = imresize(I,inputSize(1:2));
[bboxes,scores] = detect(detector,I);
```

```
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)
```

```
testData = transform(testData,@(data)preprocessData(data,inputSize));
```

```
doTrainingAndEval = true
if doTrainingAndEval
    detectionResults = detect(detector,testData,'MinibatchSize',4);
else
    pretrained = load('net.mat');
    detectionResults = pretrained.detectionResults;
end
```

```
classID = 1;
metrics = evaluateObjectDetection(detectionResults,bldsTest);
precision = metrics.ClassMetrics.Precision{classID};
recall = metrics.ClassMetrics.Recall{classID};
[ap, recall, precision] = evaluateDetectionPrecision(detectionResults,testData);
```

```
figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))
```