CMPE 249
Report for HW1 – 2D Object Detection

Uttej Kumar Reddy Gade
016065543

Option: Training (1 model) & Inference (2 models)

Models for Training:
YOLOv5 (https://github.com/ultralytics/yolov5)

Models for Inference:
FastRCNN (from torchvision.models.detection import fasterrcnn_resnet50_fpn)
YOLOv5 (torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

Dataset: KITTI 2D Object Detection Dataset

Description:
Step 1: Reducing KITTI Dataset for quick training and inference

Extracted the first 1000 images and labels for training and the next 100 for testing from below links.
Download left color images of object data set (12 GB)
Download training labels of object data set (5 MB)

Organised the files in a manner suitable for YOLOv5.
Directory structure

```
dataset
  - images
      -train
      -test
  - labels-raw
      -train
      -test
```

TRAINING – YOLOv5

Step 2: Formatting dataset into suitable format for YOLOv5 training

YOLOv5 looks for labels in the format [CATEGORY] [BBOX_X] [BBOX_Y] [HEIGHT] [WIDTH] where BBOX_X and BBOX_Y are the center coordinates of the bounding box. However, the labels downloaded from the source has [CATEGORY] with 14 values encoding more attributes.

So, as the next step, converted the labels to required format using the below link as reference:
https://github.com/packyan/Kitti2Coco/blob/master/kitti2coco-label-trans.py

Code:
https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/kitti-labels-to-coco-format.ipynb

Summary:

Encode all categories in the label to a number and store it.

a. For each line in label file, extract the 5th-8th value and scale them to get required values.

   bbox_center_x = float((x1 + (x2 - x1) / 2.0) / img_width)
   bbox_center_y = float((y1 + (y2 - y1) / 2.0) / img_height)
   bbox_width = float((x2 - x1) / img_width)
   bbox_height = float((y2 - y1) / img_height)

b. Write them in new label files in the above mentioned format.

Raw Labels:
Pedestrian 0.00 0 -0.20 712.40 143.00 810.73 307.92 1.89 0.48 1.20 1.84 1.47 8.41 0.01

Processed Labels:
0 0.6221936274509804 0.6093513513513513 0.08033496732026148 0.4457297297297298

The processed labels are stored here.

dataset
  - labels

Step 3: Downloaded, configured and ran YOLOv5

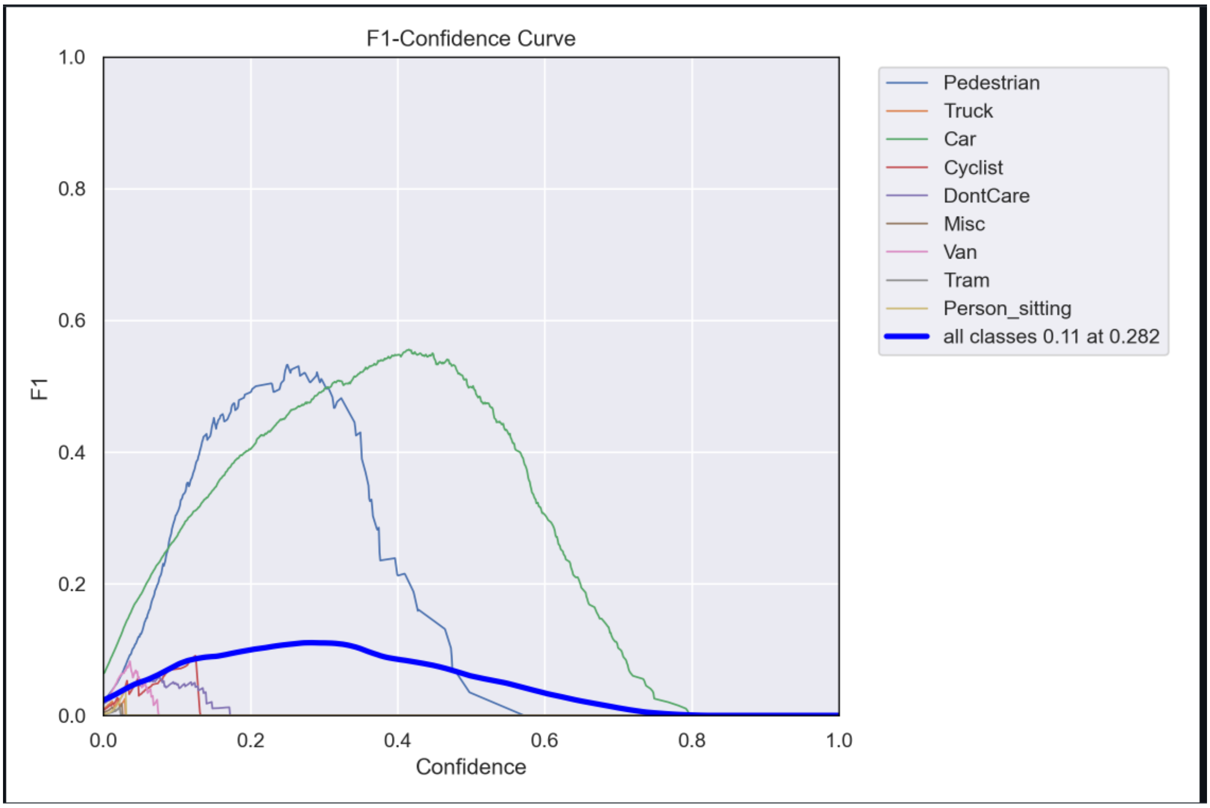
Downloaded YOLOv5 from Github. Configured the coco.yaml file in data folder as follows:

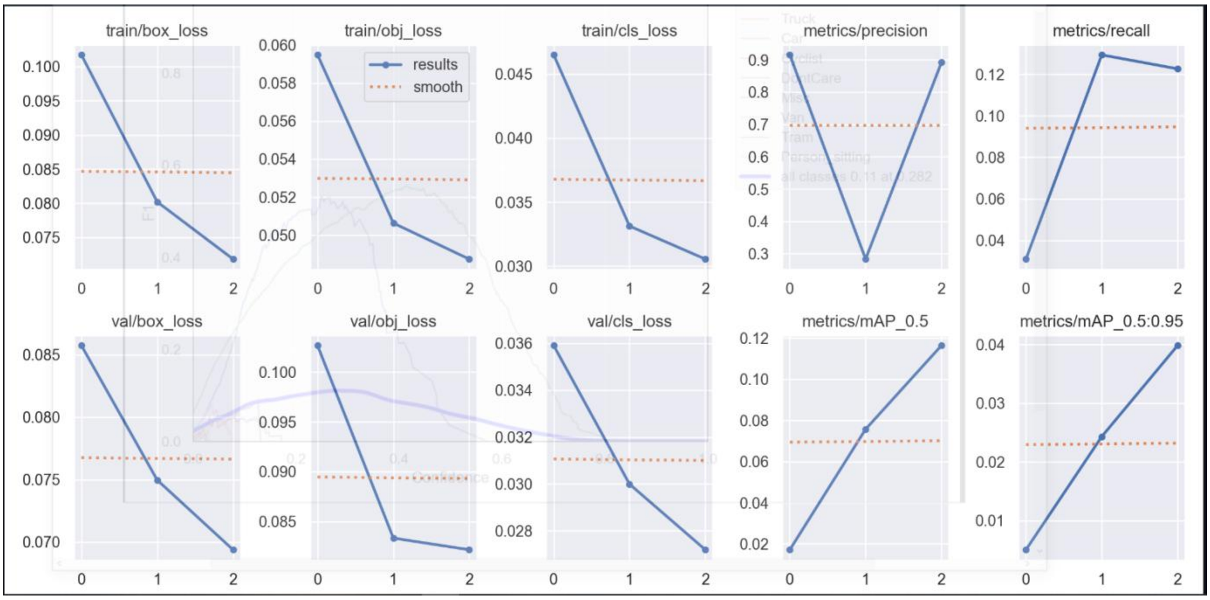
```
path: D:\present\cmpe249-hw1\dataset
train: D:\present\cmpe249-hw1\dataset\images\train
val: D:\present\cmpe249-hw1\dataset\images\test
test:

# Classes
names:
  0: Pedestrian
  1: Truck
  2: Car
  3: Cyclist
  4: DontCare
  5: Misc
  6: Van
  7: Tram
  8: Person_sitting
```

As the training took a long time for each epoch, ran the model for only 3 epochs with the command:
python train.py --epochs 3 --data coco.yaml --weights yolov5s.pt

The results can be seen here.

For 3 epochs:





Step 4: Evaluate coco metrics

To evaluate coco metrics, run the [validation script, val.py] on the generated model, [best.pt]
python val.py --weights best.pt --data coco.yaml

Changes made in val.py,

save_json = True and

anno_json = '../../dataset/coco-annotations/annotations_test.json'

Note: annotations_test.json is created as described in Step 1 of Inference – Fast RCNN section.
mAP50 values:

Class	Images	Instances	P	R	mAP50	mAP50-95:
all	100	671	0.892	0.122	0.116	0.0394
Pedestrian	100	55	0.643	0.427	0.399	0.11
Truck	100	14	1	0	0.00745	0.00218
Car	100	387	0.383	0.674	0.541	0.208
Cyclist	100	18	1	0	0.0236	0.0107
DontCare	100	139	1	0	0.0158	0.00451
Misc	100	9	1	0	0.00356	0.000927
Van	100	45	1	0	0.0297	0.00884
Tram	100	3	1	0	0.00251	0.0015
Person_sitting	100	1	1	0	0.0255	0.00765

INFERENCE – FASTERRCNN

Step 1: Convert the KITTI dataset to COCO format

Fast RCNN requires COCO annotations for inference. Following these references, https://medium.com/codable/convert-any-dataset-to-coco-object-detection-format-with-sahi-95349e1fe2b7 and https://pypi.org/project/sahi/
converted the KITTI dataset to COCO format. The following is the summary of the code.

a. Encode the categories from the labels files.
b. For every training label, create a COCOImage and for every annotation in that label, create a COCOAnnotation object with the bounding boxes information (mid point and height and width of the box) and the category.
c. Similarly, perform the same for test labels.
d. Save both in .json format.

They can be found in the dataset directory here.

dataset
-coco-annotations
- annotations_test.json
- annotations_train.json

Step 2: Perform Inference (Code)
Next imported a pre-trained Fast RCNN model from torchvision.models.detection and performed inference and generated a predictions.json in COCO format.
Note: The following transformations need to be done on predictions to stay consistent with the process in Step 1.

```
x1 = float(box[0])
y1 = float(box[1])
x2 = float(box[2])
y2 = float(box[3])

intx1 = int(x1)
inty1 = int(y1)
intx2 = int(x2)
inty2 = int(y2)

bbox_center_x = float( (x1 + (x2 - x1) / 2.0) / img_width)
```

bbox_center_y = float((y1 + (y2 - y1) / 2.0) / img_height)
bbox_width = float((x2 - x1) / img_width)
bbox_height = float((y2 - y1) / img_height)

Step 3: COCO Metrics Evaluation
Used pycocotools for calculating COCO Evaluation metrics as seen from this reference:
https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocoEvalDemo.ipynb

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.026
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.074
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.026
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.028
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.029
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.029
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.029
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
```

INFERENCE – YOLOv5

YOLOv5 follows a similar pattern to FasterRCNN inference.

a. Generate the predictions json file on the test images using pretrained Yolov5 model.
   Code and results
   Reference: https://pytorch.org/hub/ultralytics_yolov5/
b. Calculate COCO metrics.

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.023
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.065
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.008
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.023
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.019
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.035
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.035
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.035
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
```

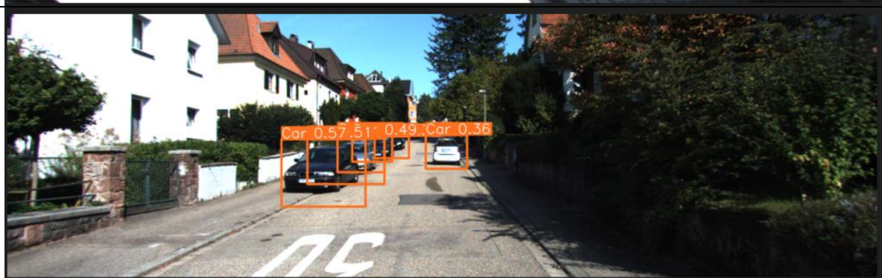
INFERENCE PIPELINE

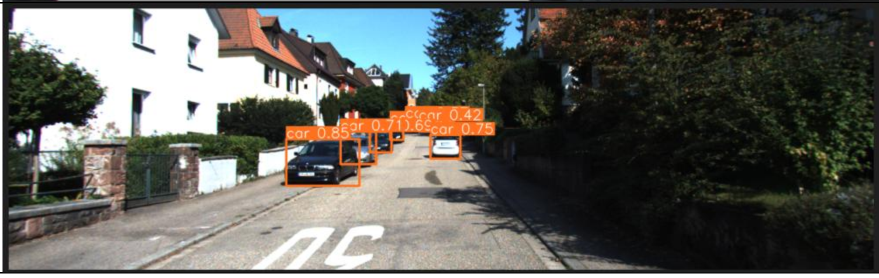
Code: https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-pipeline/infer.py

Summary:

a. Created a class InferencePipeline which initializes with model type (yolo-pretrained, yolo-custom, rcnn) and image path. The inputs are passed as args from command line.
b. The inferencepipeline follows the steps:
    a. Prechecks: Check if model and image inputs are valid
    b. Load models: Load the yolo-pretrained, yolo-custom or rcnn model if they are valid.
    c. Transform image: RCNN requires the image to be transformed to a tensor for prediction. If the model type if YOLO, this function does nothing.
    d. Predict: Model is run on the transformed image.
    e. Visualize: The images with bounding boxes, labels and scores are generated.

Running the inference pipeline for 2 models, pretrained-yolov5, custom-trained-yolov5 and fasterrcnn.

Model	**Image**	**Output Latency (s)**	**Result**
Custom Yolov5	001000.png	6.98	
Custom Yolov5	001001.png	6.098	
Custom Yolov5	001002.png	6.02	
Pretrained Yolov5	001000.png	6.08	

Pretrained Yolov5	001001.png	6.28	
Pretrained Yolov5	001002.png	5.93	
FasterRCNN	001000.png	2.21	
FasterRCNN	001001.png	2.18	
FasterRCNN	001002.png	2.36	

The approximate average output latencies for the 3 models on the constructed inference pipeline and from the chosen three images is:

a. Pretrained YOLO : 6.1 sec
b. Custom YOLO : 6.37 sec
c. FasterRCNN : 2.25 sec

References:

[1] https://medium.com/codable/convert-any-dataset-to-coco-object-detection-format-with-sahi-95349e1fe2b7
[2] https://pypi.org/project/sahi/
[3] https://github.com/packyan/Kitti2Coco/blob/master/kitti2coco-label-trans.py
[4] https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocoEvalDemo.ipynb
[5] https://pytorch.org/hub/ultralytics_yolov5/