**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](https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark)

**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)](https://s3.eu-central-1.amazonaws.com/avg-kitti/data_object_image_2.zip)  
[Download training labels of object data set (5 MB)](https://s3.eu-central-1.amazonaws.com/avg-kitti/data_object_label_2.zip)

Organised the files in a manner suitable for YOLOv5.   
[*Directory structure*](https://github.com/uttejkumarreddy/cmpe249-hw1/tree/master/dataset) *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](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/kitti-labels-to-coco-format.ipynb%20)

Summary:

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

1. 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)

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

[Raw Labels:](https://github.com/uttejkumarreddy/cmpe249-hw1/tree/master/dataset/labels-raw)  
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:](https://github.com/uttejkumarreddy/cmpe249-hw1/tree/master/dataset/labels-raw)  
0 0.6221936274509804 0.6093513513513513 0.08033496732026148 0.4457297297297298

The processed labels are stored [here.](https://github.com/uttejkumarreddy/cmpe249-hw1/tree/master/dataset/labels-raw)  
 *dataset  
 - labels*

Step 3: Downloaded, configured and ran YOLOv5

[Downloaded YOLOv5](https://github.com/uttejkumarreddy/cmpe249-hw1/tree/master/training-yolov5/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

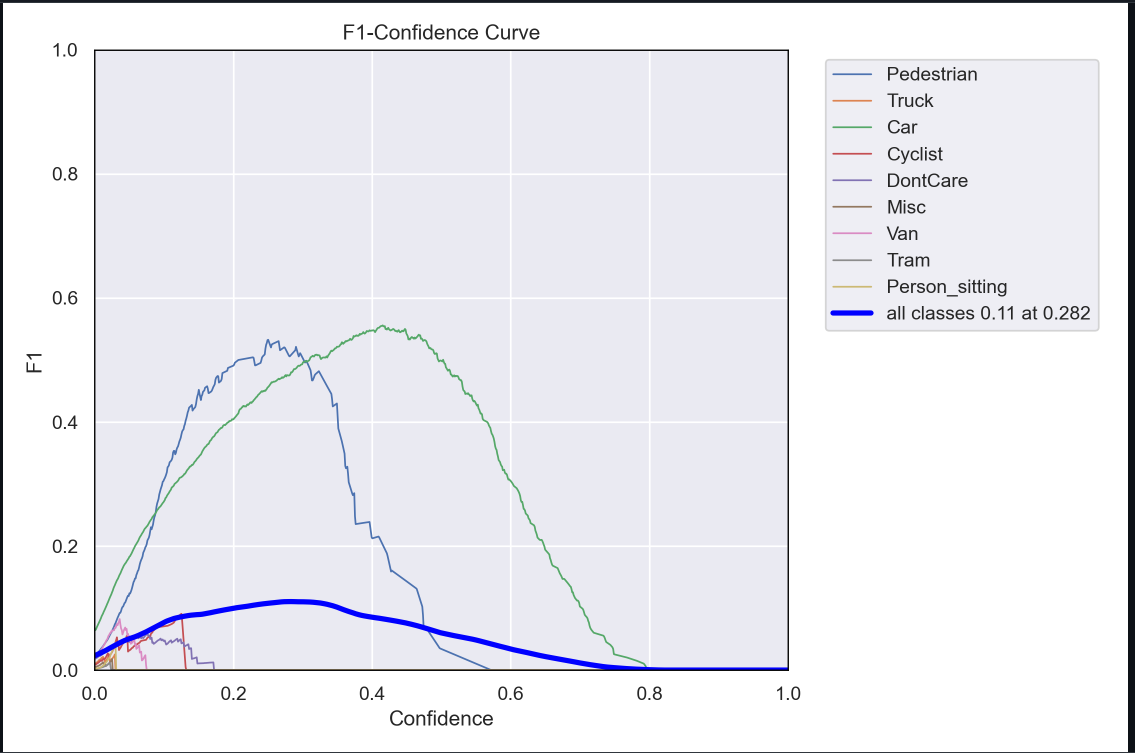
test:

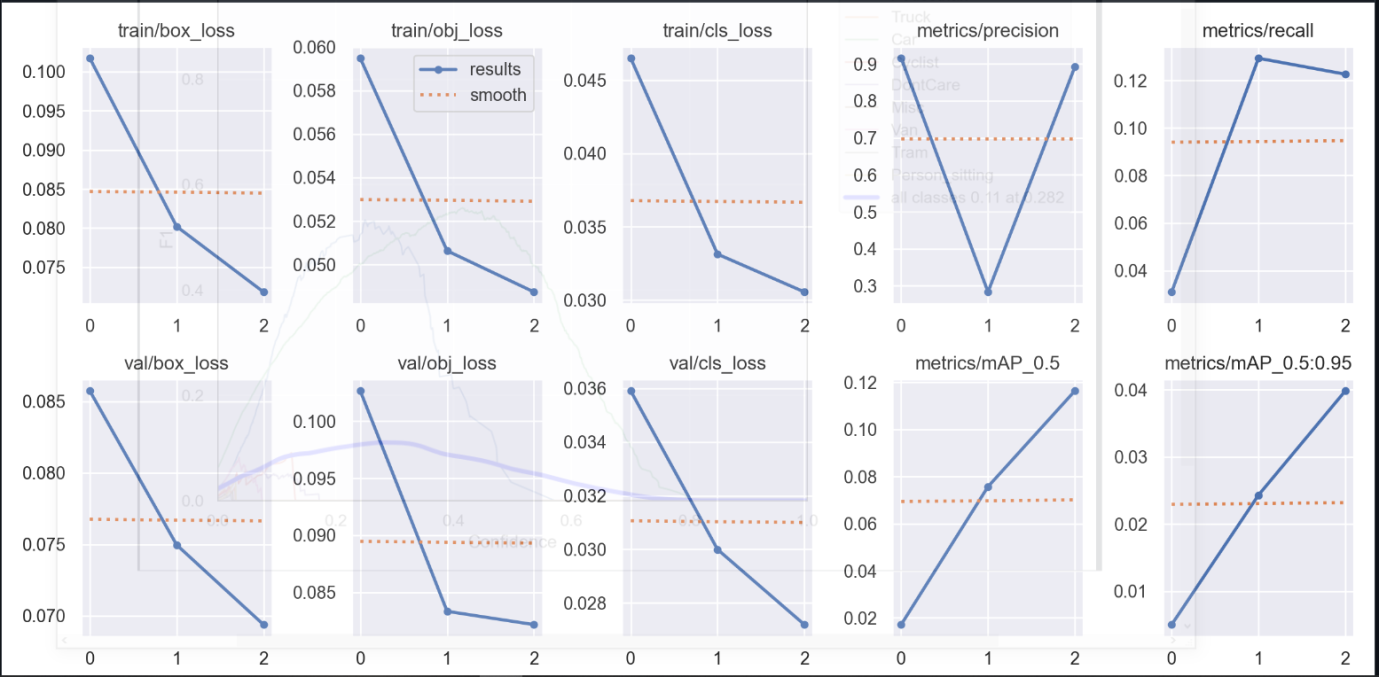
# Classes  
names:  
  0: Pedestrian  
  1: Truck  
  2: Car  
  3: Cyclist  
  4: DontCare  
  5: Misc  
  6: Van  
  7: Tram  
  8: Person\_sitting

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](https://github.com/uttejkumarreddy/cmpe249-hw1/tree/master/yolov5-results/exp).

For 3 epochs:





Step 4: Evaluate coco metrics

To evaluate coco metrics, run the [validation script, val.py](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/training-yolov5/yolov5/val.py) on the generated model, [best.pt](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/training-yolov5/results/exp/weights/best.pt)  
python val.py --weights best.pt --data coco.yaml

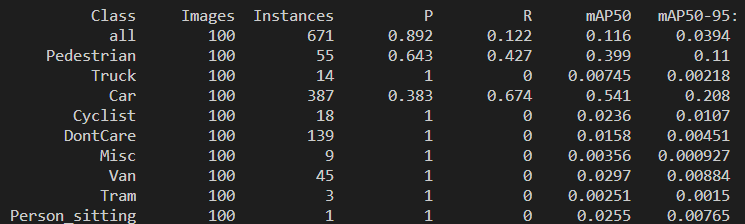
Changes made in val.py,

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:



***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](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-fastrcnn/00-kitti-labels-to-coco-annotations.ipynb). The following is the summary of the code.

1. Encode the categories from the labels files.
2. 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.
3. Similarly, perform the same for test labels.
4. Save both in .json format.

They can be found in the dataset directory [here](https://github.com/uttejkumarreddy/cmpe249-hw1/tree/master/dataset/coco-annotations).

*dataset*

*-coco-annotations*

*- annotations\_test.json*

*- annotations\_train.json*

Step 2: [Perform Inference (Code)](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-fastrcnn/01-perform-inference.ipynb)

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)

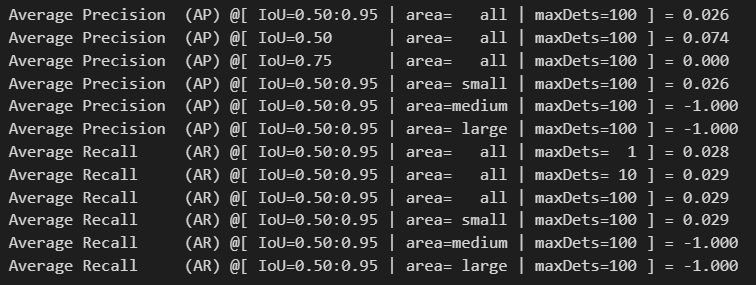
        bbox\_center\_y = float( (y1 + (y2 - y1) / 2.0) / img\_height)

        bbox\_width = float((x2 - x1) / img\_width)

        bbox\_height = float((y2 - y1) / img\_height)

Step 3: [COCO Metrics Evaluation](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-fastrcnn/02-coco-eval-metrics.ipynb)

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



***INFERENCE – YOLOv5***

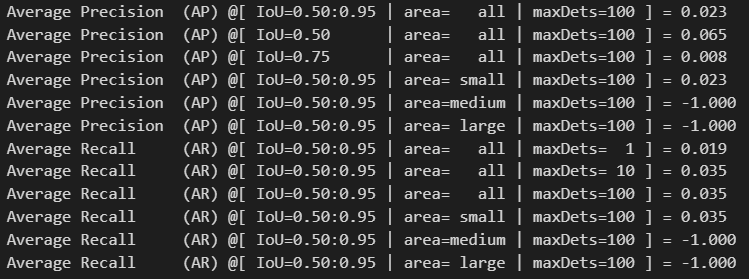
YOLOv5 follows a similar pattern to FasterRCNN inference.

1. Generate the predictions json file on the test images using pretrained Yolov5 model.

[Code](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-yolov5/00-yolov5-perform-inference.ipynb) and [results](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-yolov5/results/predictions.json)

Reference: <https://pytorch.org/hub/ultralytics_yolov5/>

1. [Calculate COCO metrics](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-yolov5/01-yolov5-coco-eval-metrics.ipynb).



***INFERENCE PIPELINE***

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

Summary:

1. 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.
2. The inferencepipeline follows the steps:
   1. Prechecks: Check if model and image inputs are valid
   2. Load models: Load the yolo-pretrained, yolo-custom or rcnn model if they are valid.
   3. Transform image: RCNN requires the image to be transformed to a tensor for prediction. If the model type if YOLO, this function does nothing.
   4. Predict: Model is run on the transformed image.
   5. Visualize: The images with bounding boxes, labels and scores are generated.

Running the inference pipeline for 2 models, [pretrained-yolov5](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-pipeline/models/pretrained-yolov5s.pt), [custom-trained-yolov5](https://github.com/uttejkumarreddy/cmpe249-hw1/blob/master/inference-pipeline/models/custom-trained-yolov5.pt) 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:

1. Pretrained YOLO : 6.1 sec
2. Custom YOLO : 6.37 sec
3. 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/>