

**Project Title:** Benchmarking Computer Vision Surveillance Algorithms for Practical Traffic Applications

**Professor Name:** Nicolas Saunier

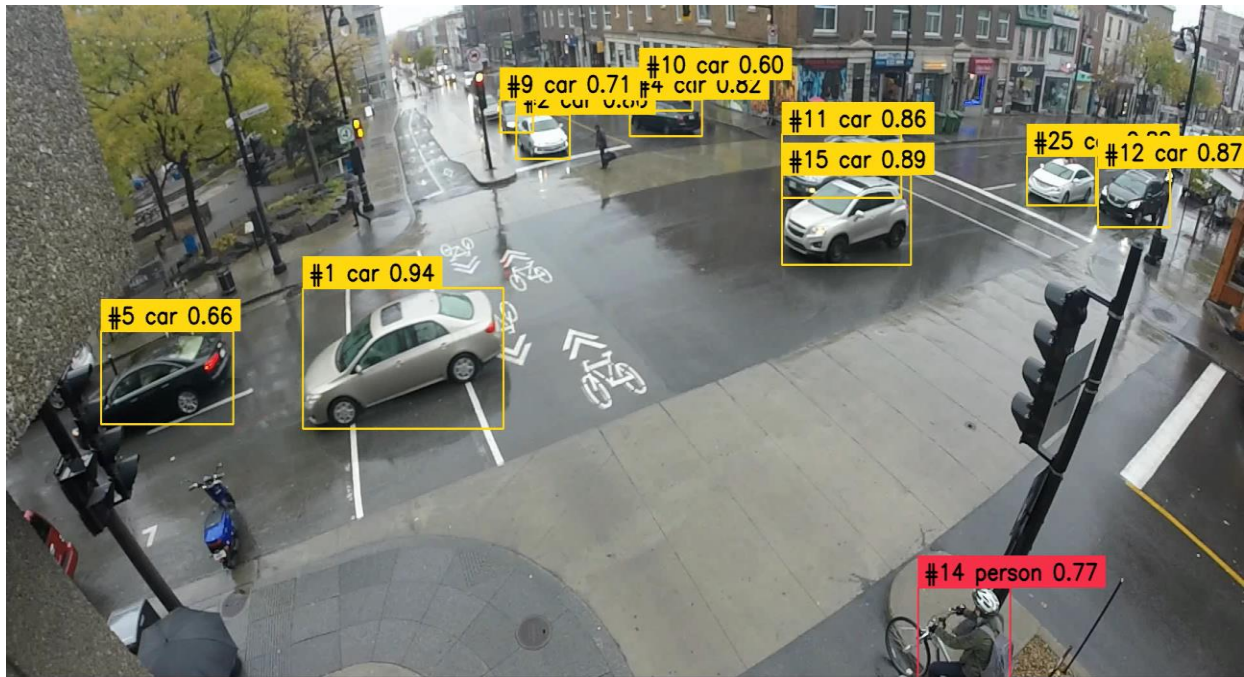
**Project Goals:** Currently, video analysis plays a significant role in numerous transportation applications, particularly in the collection of traffic data. The primary focus lies in extracting road user counts and their respective speeds. Despite the availability of various algorithms designed to automatically detect, track, and classify road users, their actual performance under real-world conditions (including varying weather, lighting, etc.) and in diverse traffic scenarios (such as dense urban traffic) remains uncertain.

To address this challenge, this project aims to establish a benchmark for state-of-the-art methods employed in detecting, tracking, and classifying road users using video data across diverse types of roads, streets, intersections, and public spaces. The benchmarking process will involve incorporating as much video data as possible to accurately represent practical traffic applications under these diverse conditions. Particular attention will be given to retrieving the actual positions of road users, enabling easy derivation of essential metrics like speed measures. It is also crucial that the code developed for this project is designed to be easily expandable to accommodate new datasets and algorithms in the future.

**Work done:** During my internship, I worked on several tasks related to object detection, object segmentation, object tracking, and speed estimation of vehicles on roads.

1. **Object Detection:** Object detection is a fundamental task in computer vision that involves locating and identifying objects of interest within an image or a video. To perform object detection, I used the Ultralytics library, a popular framework that simplifies working with deep learning models for object detection. Specifically, I utilized pretrained YOLOv8 models, which are part of the YOLO (You Only Look Once) family of object detection models.

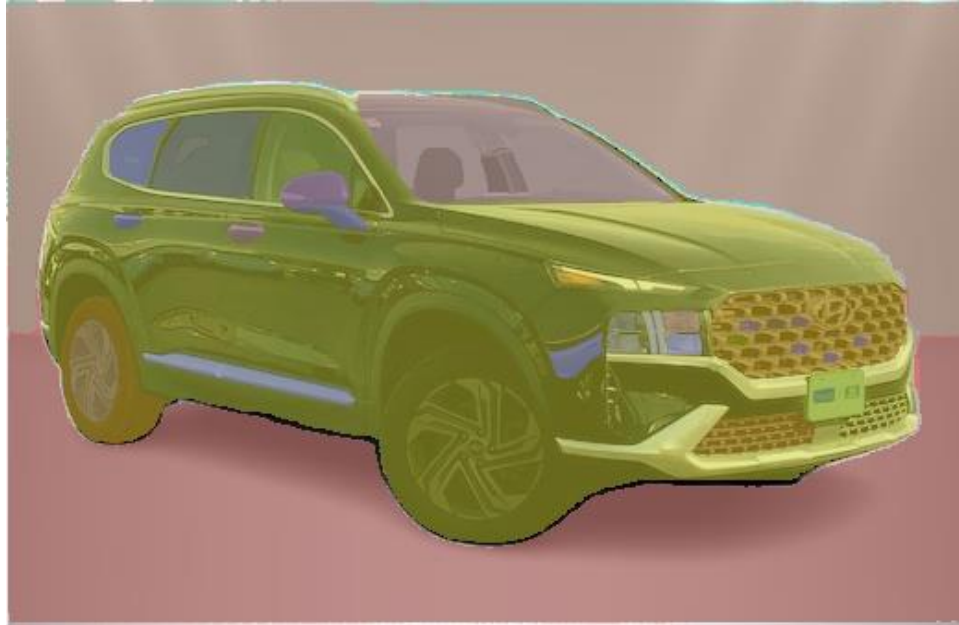
These YOLOv8 models were already trained on the COCO dataset, which contains a wide variety of objects from distinct categories. This allowed me to benefit from the prelearned knowledge of these models and apply them to detect objects in my own images or videos.



COCO dataset link - [COCO - Common Objects in Context \(cocodataset.org\)](https://cocodataset.org)

Ultralytics link - [Home - Ultralytics YOLOv8 Docs](https://ultralytics.com)

2. Object Segmentation: Object segmentation goes beyond object detection by not only identifying the objects but also separating them from the background. I employed the 'yolov8n-seg.pt' model, which is a variant of YOLOv8 specifically designed for object segmentation. This model allowed me to obtain pixel-level masks for each detected object, enabling more precise understanding of the objects' shapes and boundaries.

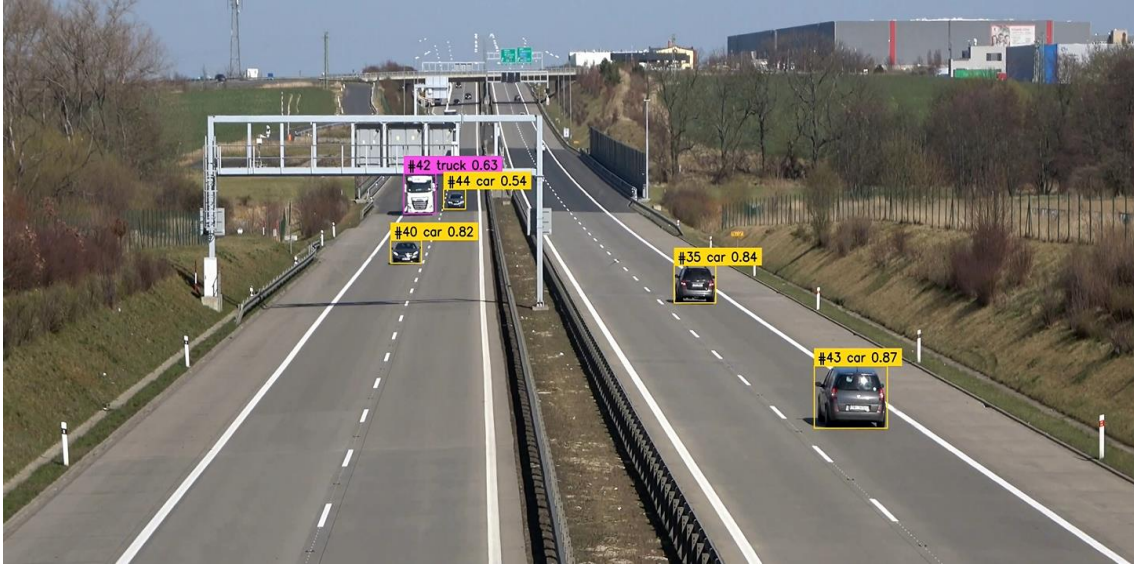


**YOLOv8 segmentation** - [Segment - Ultralytics YOLOv8 Docs](#)

**SAM (Segment Anything Mode) and TrackEval:** I attempted to use the SAM model by Meta for segmenting various objects. However, due to hardware limitations, I could not test it on long videos.

**SAM GitHub** - [GitHub - facebookresearch/segment-anything: The repository provides code for running inference with the SegmentAnything Model \(SAM\), links for downloading the trained model checkpoints, and example notebooks that show how to use the model.](#)

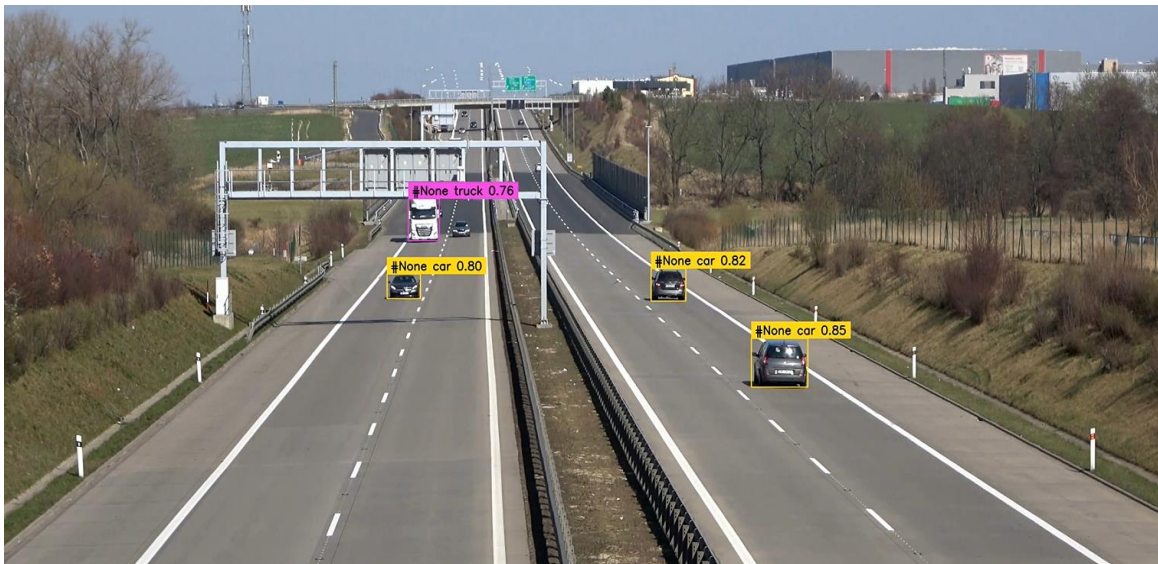
3. **Object Tracking:** Object tracking is the process of following objects across multiple frames in a video sequence. It is an essential task for applications like surveillance, autonomous vehicles, and augmented reality. During this phase, I explored two popular object tracking methods: ByteTrack and BotSort.
  - **ByteTrack:** ByteTrack is a sophisticated object tracking algorithm that considers all bounding boxes, even those with low detection scores. This robust approach often leads to better object tracking results compared to methods that might discard low-confidence detections.



(ByteTrack on video)

ByteTrack GitHub link - [GitHub - ifzhang/ByteTrack](https://github.com/ifzhang/ByteTrack): [ECCV 2022] ByteTrack: Multi-Object Tracking by Associating Every Detection Box

- BotSort: BotSort is another object tracking method, though its specific details are not mentioned. It is likely that I experimented with BotSort to compare its performance with ByteTrack in terms of object tracking accuracy and robustness.

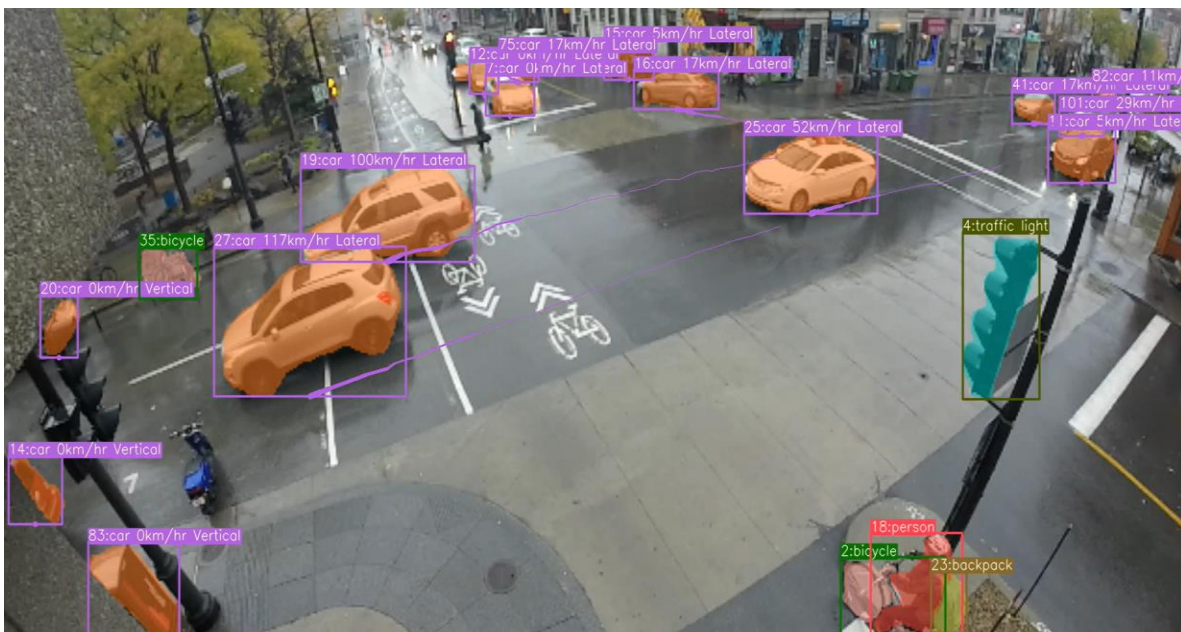


(BoTSORT on video)

BoTSORT GitHub link - [GitHub - NirAharon/BoT-SORT](https://github.com/NirAharon/BoT-SORT): BoT-SORT: Robust Associations Multi-Pedestrian Tracking



4. **Vehicle Speed Estimation:** The next task involved tracking and estimating the speed of vehicles on roads. I utilized DeepSort in combination with YOLOv8 for this purpose. To estimate the speed of a vehicle, I used the ratio of the bounding box width and height to determine its movement (lateral or vertical). A 2D array was employed to store the last locations of different vehicle IDs. By calculating the distance moved in pixels using the distance formula and converting it to meters based on the screen's PPI (pixels per inch), I determined the speed of the vehicles.



**Google Colab link** - [Yash Using DeepSort.ipynb - Colaboratory \(google.com\)](#)

**Code File** - [Speed Estimation and Segmentation - Google Drive](#) (This needs to be replaced with /content/YOLOv8\_Segmentation\_DeepSORT\_Object\_Tracking/ultralytics/yolo/v8/segment/ predict.py file)

**# Vehicle Speed estimation process -**

**#Euclidean Distance Formula**

$$d\_pixel = \text{math.sqrt}(\text{math.pow}(\text{Location2}[0] - \text{Location1}[0], 2) + \text{math.pow}(\text{Location2}[1] - \text{Location1}[1], 2))$$

# defining thr pixels per meter

p2m = 0.2772540756349118 # (1/142)/0.0254 (1 inch = 0.0254)

d\_meters = d\_pixel\*p2m

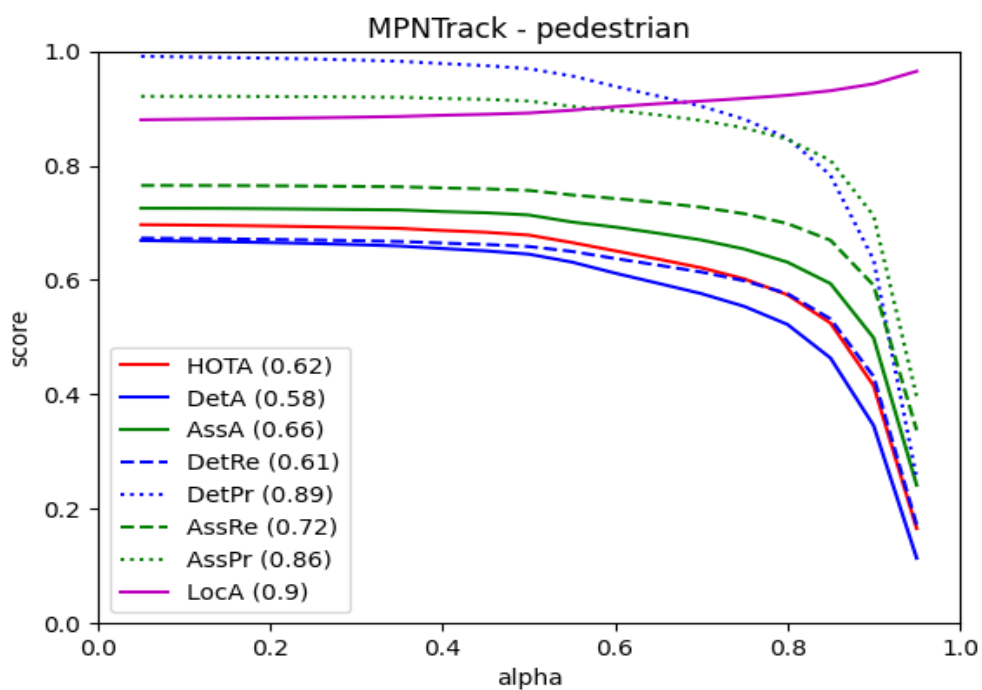
time\_constant = 1/5.9 # 5.9 is FPS

#speed = distance/time

speed = d\_meters / time\_constant

5. Additionally, I used the TrackEval evaluation method to assess the performance of the MPNTrack (multi-object detection) model. I calculated HOTA (Higher Order Tracking Accuracy) metrics and plotted different results to decide the model's accuracy and overall performance.

TrackEval link - [JonathonLuiten/TrackEval: HOTA \(and other\) evaluation metrics for Multi-Object Tracking \(MOT\). \(github.com\)](https://github.com/JonathonLuiten/TrackEval)



(MPNTrack different accuracy parameters on MOT17 dataset)

**Note - Here alpha is the spatial distance between the centers of ground-truth(gt) and predicted bounding box by the tracker**

6. MOT17 Benchmark: I took on the task of training ByteTrack on the MOT17 dataset, which is a benchmark dataset widely used for evaluating multi-object tracking algorithms. I evaluated the performance of the model by calculating various metrics such as MOTA (Multiple Object Tracking Accuracy), MOTP (Multiple Object Tracking Precision), Accuracy, Recall, and others.

Google Colan link for training and testing of ByteTrack on MOT17 Benchmark - [Copy of "bytetrack \(1\).ipynb - Colaboratory \(google.com\)](#)

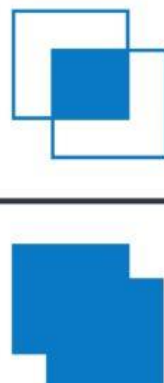
	Rc1l	Prcn	GT	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP	num_objects
MOT17-02-FRCNN	85.3%	96.1%	62	69.4%	22.6%	8.1%	3.4%	14.7%	0.6%	1.2%	81.3%	0.148	18581
MOT17-04-FRCNN	98.8%	99.5%	83	100.0%	0.0%	0.0%	0.5%	1.2%	0.0%	0.1%	98.3%	0.081	47557
MOT17-05-FRCNN	88.1%	94.6%	133	61.7%	30.1%	8.3%	5.0%	11.9%	1.2%	1.4%	81.9%	0.153	6917
MOT17-09-FRCNN	90.2%	97.8%	26	84.6%	15.4%	0.0%	2.0%	9.8%	0.6%	1.1%	87.6%	0.138	5325
MOT17-10-FRCNN	83.6%	95.9%	57	70.2%	29.8%	0.0%	3.6%	16.4%	1.1%	1.5%	79.0%	0.189	12839
MOT17-11-FRCNN	93.9%	97.0%	75	76.0%	14.7%	9.3%	2.9%	6.1%	0.3%	0.6%	90.7%	0.124	9436
MOT17-13-FRCNN	90.4%	97.3%	110	78.2%	16.4%	5.5%	2.5%	9.6%	0.3%	0.7%	87.6%	0.169	11642
OVERALL	92.5%	97.8%	546	75.6%	19.0%	5.3%	2.1%	7.5%	0.4%	0.7%	90.0%	0.122	112297

	IDF1	IDP	IDR	Rc1l	Prcn	GT	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP	IDt	Ida	IDm	num_objects
MOT17-02-FRCNN	69.6%	74.0%	65.6%	85.3%	96.1%	62	43	14	5	635	2728	103	226	81.3%	0.148	79	27	11	18581
MOT17-04-FRCNN	95.8%	96.2%	95.5%	98.8%	99.5%	83	83	0	0	232	572	12	60	98.3%	0.081	8	4	1	47557
MOT17-05-FRCNN	78.9%	81.8%	76.2%	88.1%	94.6%	133	82	40	11	346	820	83	100	81.9%	0.153	76	31	28	6917
MOT17-09-FRCNN	68.3%	71.2%	65.7%	90.2%	97.8%	26	22	4	0	108	520	31	58	87.6%	0.138	29	7	6	5325
MOT17-10-FRCNN	63.3%	68.0%	59.2%	83.6%	95.9%	57	40	17	0	458	2106	138	199	79.0%	0.189	101	41	8	12839
MOT17-11-FRCNN	84.5%	85.9%	83.1%	93.9%	97.0%	75	57	11	7	275	573	29	57	90.7%	0.124	18	11	4	9436
MOT17-13-FRCNN	80.8%	83.8%	77.9%	90.4%	97.3%	110	86	18	6	295	1116	33	83	87.6%	0.169	45	12	26	11642
OVERALL	83.2%	85.6%	80.9%	92.5%	97.8%	546	413	104	29	2349	8435	429	783	90.0%	0.122	356	133	84	112297

2023-07-27 19:55:48 | INFO | \_\_main\_\_:271 - Completed

(ByteTrack overall MOTA and MOTP results on training dataset of MOT17)

# For MOTA calculation IOU (Intersection Over Union is required) and is calculated as follows -

$$IoU = \frac{\text{Area of overlap}}{\text{Area of Union}}$$


$$MOTA = 1 - \frac{\sum_t FN_t + FP_t + IDS_t}{\sum_t GT_t}$$

Here  $FN_t$  – If the Ground Truth object has NO MATCH in detection output, then the count of MISS( $FN_t$ ) will be incremented by 1

$FP_t$  – If the object in detection output has NO MATCH in the real world (Ground Truth), it will be considered false detection. In that case, the count of FALSE POSITIVES( $FP_t$ ) will be incremented by 1

$IDS_t$  – Consider in Frame 1, and the correspondence is made between Ground Truth person( $P1$ ) with detection model output person ( $PD3$ ) the pair will be saved as (1, 3). In a case in the next frame ground truth person ( $P1$ ) is paired with a different person who is in detection output ( $PD4$ ), then the MISMATCH ERROR( $IDS_t$ ) will be incremented by 1(+1). The pair (1,3) will be deleted, and the pair (1,4) will be considered in future frames.

# For MOTP calculation Euclidean pixel distance is required and is calculated as follows -

$$d = (x1 - x2)^2 + (y1 - y2)^2$$

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}$$

$dt$  – Distance between the localization of objects in the ground truth and the detection output ( $dt = 1 - IOU$ )

$ct$  – total matches made between ground truth and the detection output

Note:– For our trained ByteTrack the MOTA and MOTP on MOT17 training dataset are 90% and 0.122 respectively.

**7. 3D Bounding Box Object Detection:** In the seventh week of my internship, I explored the task of 3D bounding box object detection using the Mediapipe Objectron library. This library is



designed to detect and estimate 3D bounding boxes for specific objects. However, the Objectron dataset used by the library had only four classes: Cup, Shoes, Camera, and Chair. Despite the limited dataset, I experimented with this task to gain hands-on experience in 3D object detection and to understand the challenges and applications of such techniques.

**Mediapipe Objectron link** - [Objectron \(3D Object Detection\) - mediapipe \(chuoling.github.io\)](#)

**Code link** - [Mediapipe Objectron - Google Drive](#)