YOLOv5++ for multiple Object Detection from cluttered indoor shots, invariant to sensor, lighting, and affine transformation (TPA 3)

B.Laxman, ED18B006, and Sai Rohitth Chiluka, ED18B027

Abstract

The report below deals with the implementation of YOLOv5x on various image datasets and evaluates how the present YOLOv5x algorithm performs and checks whether further training of the model is required to cope with our classification and object detection needs.

INTRODUCTION

Perform/Enable localization of different types of objects in indoor scenes with the help of bounding boxes using Deep Learning techniques. The images may include challenges like occlusion, background clutter, camera shake, varied object size, affine transformation, and illumination conditions. Models based on recent state-of-the-art methods like YOLO v5, YOLO v3, RefineNet, Relation-Net [9], RFBNet [10], CornerNet [11], etc., or anything new showing substantial performance over the existing techniques is expected.

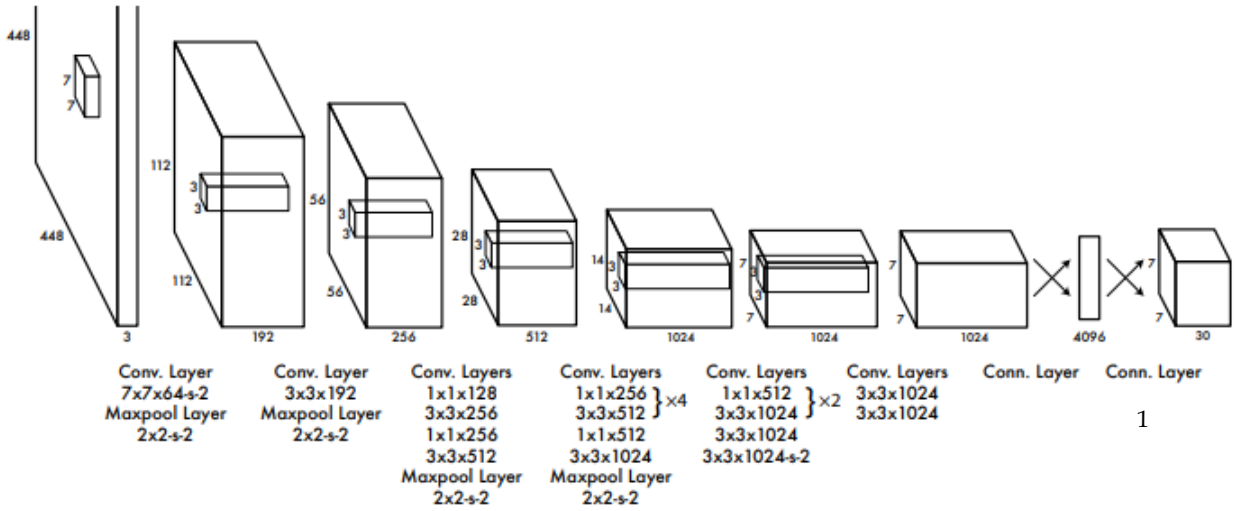
Outputs needed :

- Detect, identify and visualize the location of each of the objects in the input image using bounding boxes
- Obtain the class label of the detected objects.
- Calculate the Rank-1 recognition rate of each of the objects detected.
- Off-line performance on a set of test images (to be provided during evaluation) using mean Average Precision (mAP@IoU=0.5) as an evaluation metric.

ALGORITHMIC DESCRIPTION

The algorithm which we use “YOLO” refers to “You Only Look Once” is one of the most versatile and famous object detection models. YOLO algorithms divide all the given input images into the SxS grid system. Each grid is responsible for object detection. Now those Grid cells predict the boundary boxes for the detected object. For every box, we have five main attributes: x and y for coordinates, w and h for width and height of the object, and a confidence score for the probability that the box contains the object.YOLO uses a totally different approach. We apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by predicted probabilities. We’ll be using **Yolo v5x** among the different Yolo models available.

Figure 1: Architecture of the network used in YOLO



Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers. We train the convolutional layers on the ImageNet classification task at half the resolution (224 × 224 input image) and then double the resolution for detection.

Figure 2: Loss function of the system

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

OUTPUT

YOLOv5 Model Evaluation :

- To tackle the given problem statement, we first need to check how the default YOLO v5x algorithm performs. We will be using YOLOv5x from Ultralytics and import it into Collab via Pytorch, and check how the algorithm performs.
- For model evaluation, we used the ML developer tool **VOXEL 51,** with the given model and appropriate syntax, the evaluation is done easier and the dataset conversion necessary for model evaluation is also done by commands used in VOXEL 51.
- Since YOLOv5x is pre-trained on COCO image datasets, we will see how the algorithm performs on other COCO images.

A. Observation for COCO image-set :

Figure 3: Bad cases of YOLOv5x on COCO image-sets

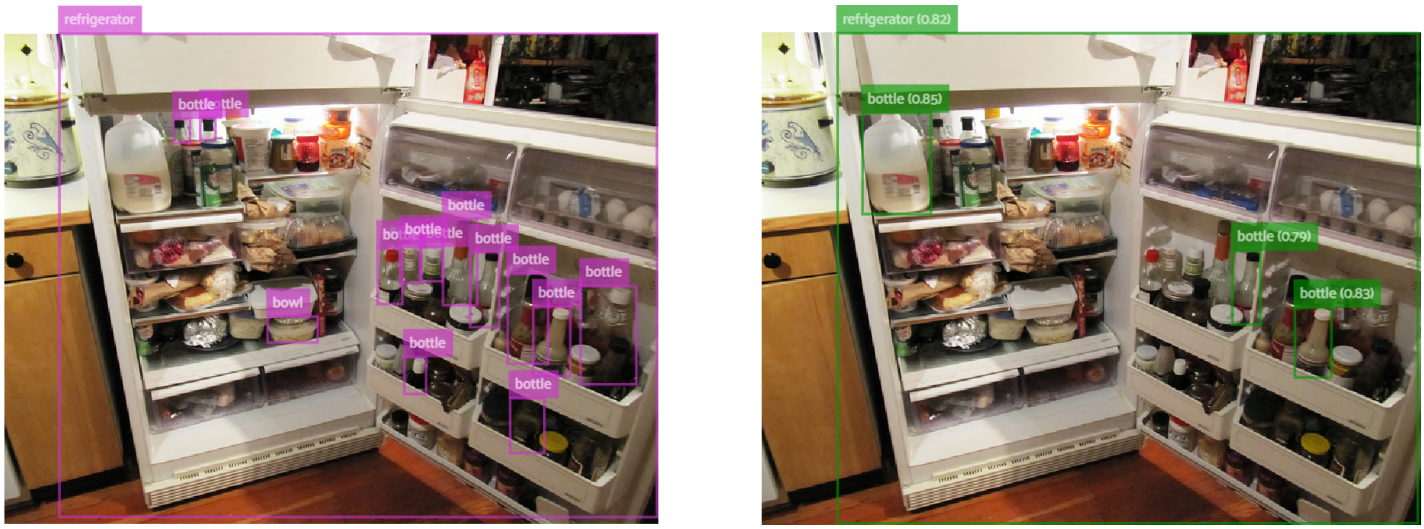


Figure 4: Good cases of YOLOv5x on COCO image-sets

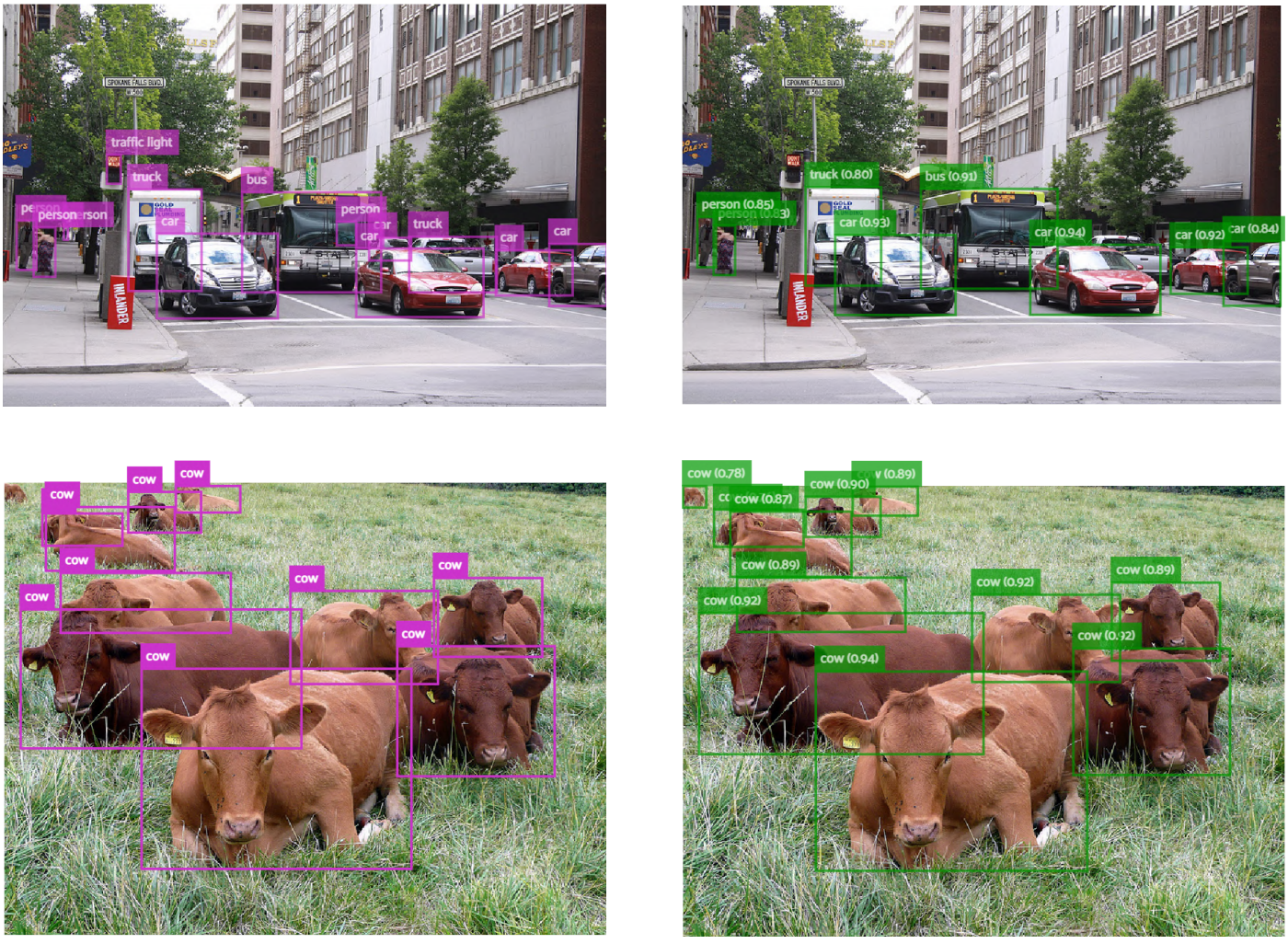


Table 1: Evaluation of YOLOv5x on COCO image-sets

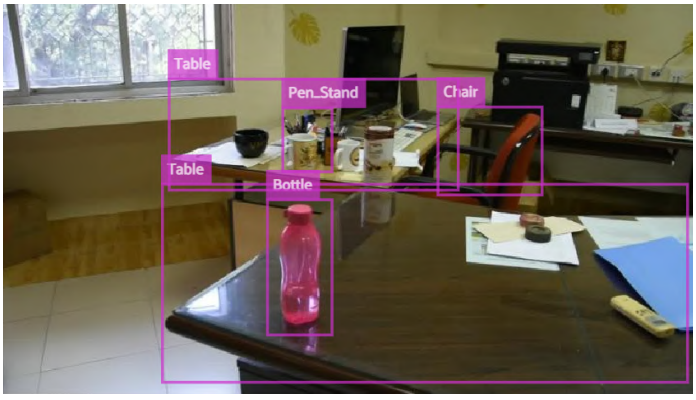
Class Labels	Precision	Recall	F1-score	Support
person	0.97	0.53	0.68	213
book	0.00	0.00	0.00	50
chair	1.00	0.36	0.53	39
cup	0.89	0.55	0.68	29
bottle	0.83	0.19	0.30	27
car	0.89	0.64	0.74	25
dining table	0.67	0.10	0.17	21
bowl	1.00	0.18	0.30	17
giraffe	0.92	0.75	0.83	16
kite	1.00	0.07	0.12	15

MAP = 0.38275402904033085

When we performed the model using COCO image datasets, we can see from figure 3 that YOLOv5x performs well, validating our intuitive notion that any model performs well on trained datasets and datasets which are not trained directly but directly used from COCO. The MAP obtained by YOLOv5x on COCO (Table 1) is better than that of VP Lab image sets (Table 2). We can see that Yolo v5x on COCO really shines on recognizing people and living creatures rather than inanimate objects even when they are cluttered. **Next, our task is to perform the evaluation on image datasets other than COCO like PASCAL_VOC obtained from the VP_Lab dataset and see how the default model fares. Conversion of the image from PASCAL_VOC to YOLO format is done using VOXEL 51.**

B. Observation for VP Lab (PASCAL_VOC) image-set :

Figure 5: Bad cases of YOLO v5x on VP LAB dataset



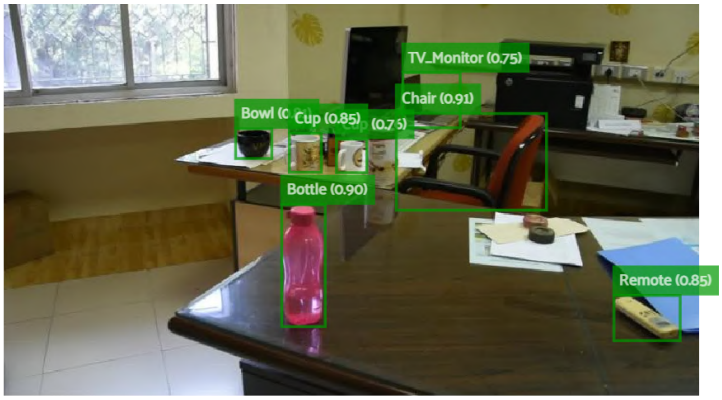
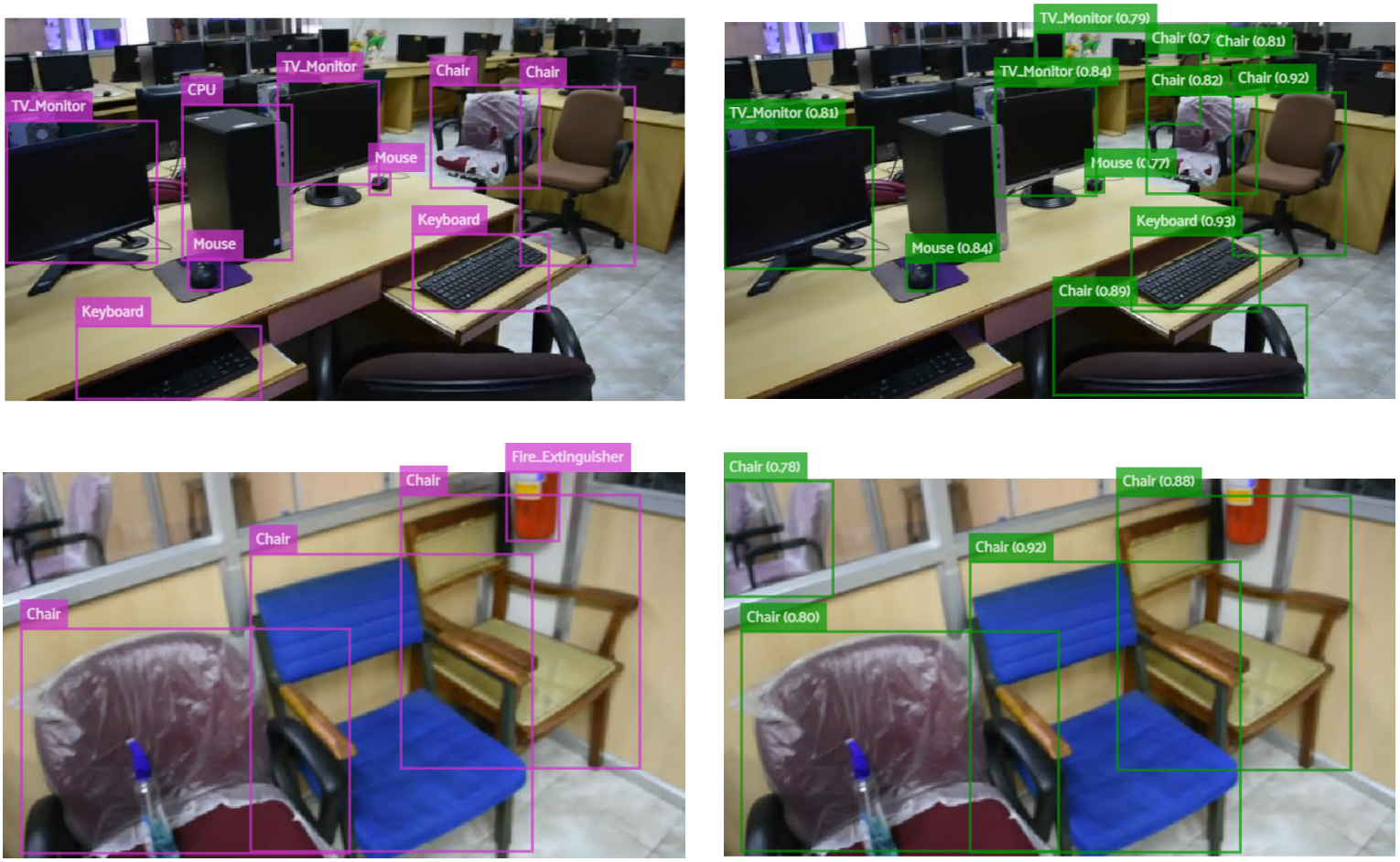


Figure 6: Good cases of YOLO v5x on VP LAB dataset



On observing the above output, we can see that the model performs poorly, and the confidence level in some of the bounding boxes in the dataset is not that high. We can say the model detects only people or any other common labels and that too the model doesn't do a great job. But what about the datasets that we need? Our main problem is that our model should be able to detect indoor objects that were found commonly in a computer lab, objects, in that case, were not detected properly. For the image datasets, to evaluate the current YOLOv5x model we used the datasets from VP Lab.

- At first, when used with datasets of VP Lab, we see that some objects are not detected in the first place, leading to poor MAP and Precision scores.
- Additionally, it does not recognize objects as intended, it is recognizing "Water_can" as "Bottle", and not recognizing the "Water_Can" in the second image of figure 5, which is a major concern on why their performance is poor.
- It also recognizes things that are not annotated or labeled in the ground positives of the image which can be seen from the first image in figure 5 (it recognized "Cup" and "Remote" which are not annotated).
- Moreover, there were also labeling inconsistencies in the YOLOv5x model and with the labels in the annotations of the VP Lab dataset for the same object, telling us that the model performs poorly on the given dataset, **we need to perform custom training on the algorithm to gain new weights so that we evaluate the new model to get better results.**

Table 2: MAP, Precision, Recall Scores for default Yolo v5x on PASCAL image-sets

Class Labels	Precision	Recall	F1-score	Support
Mouse	0.96	0.50	0.66	44
Book	0.86	0.12	0.21	50
TV_Monitor	0.68	0.50	0.58	34
CPU	0.00	0.00	0.00	22
Board	0.00	0.00	0.00	16
Chair	0.46	0.90	0.61	20
Box	0.00	0.00	0.00	7
Speakers	0.00	0.00	0.00	3
Potted_Plant	0.82	1.00	0.90	9
Water_Can	0.00	0.00	0.00	12

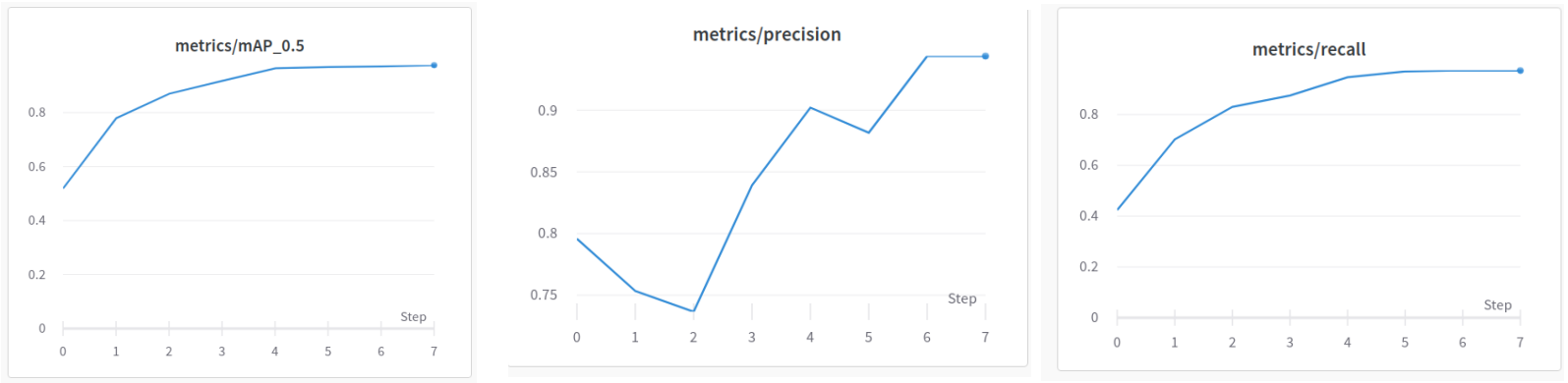
MAP = 0.18969731278045782

Custom-training a new model and evaluating its performance :

When we custom-train our model using the datasets from VP LAB, we split the dataset randomly for training, validation, and testing, and the evaluation is performed on the testing dataset to get a fair result on the model's performance. Out of 5980 samples, we randomly split 4000 for training and used 1000 samples for validation and 980 samples for testing; and we performed training using 4 batches of gradient descent and cycled it through 7 epochs, and collected its weights. The model improvement on the training dataset over each epoch is

shown in **figure 7.** We collected the weights from the 7th epoch and used the model for evaluation on the testing dataset, which doesn't comprise any images used in Training. Let's see how our new model performs.

Figure 7: MAP, Precision and Recall scores over each epoch



From figure 7, we can see that the respective scores (MAP, Precision, and Recall scores) were improving on each epoch while training.

Figure 8: Trend of the loss function over training data

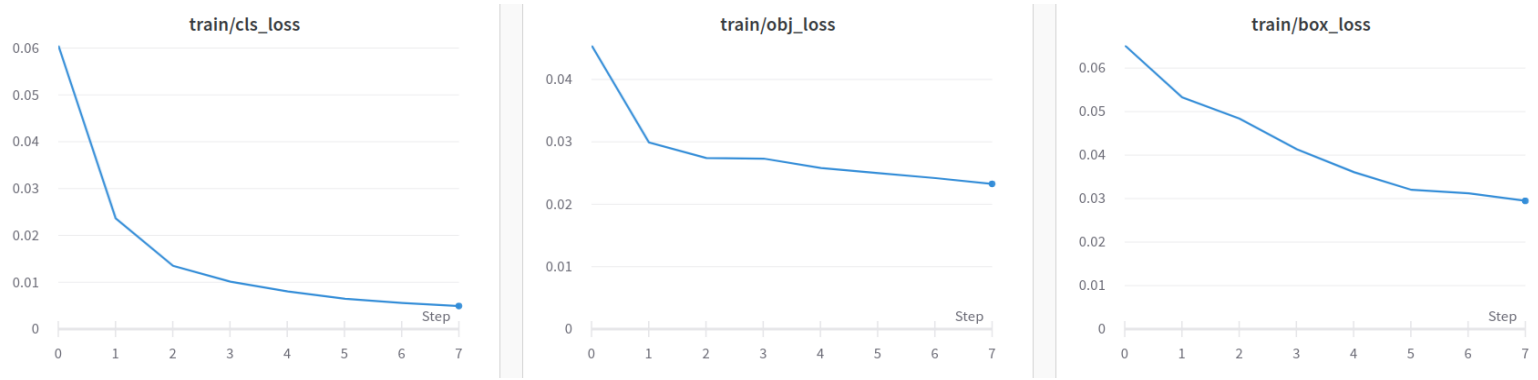


Figure 9: Trend of the loss function over validation data

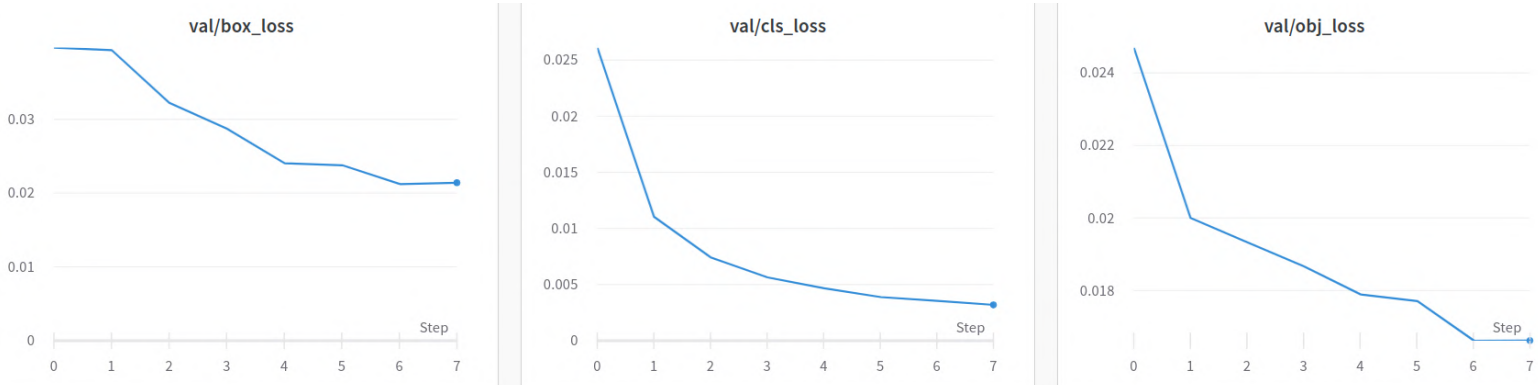


Figure 10: GPU utilization during training :

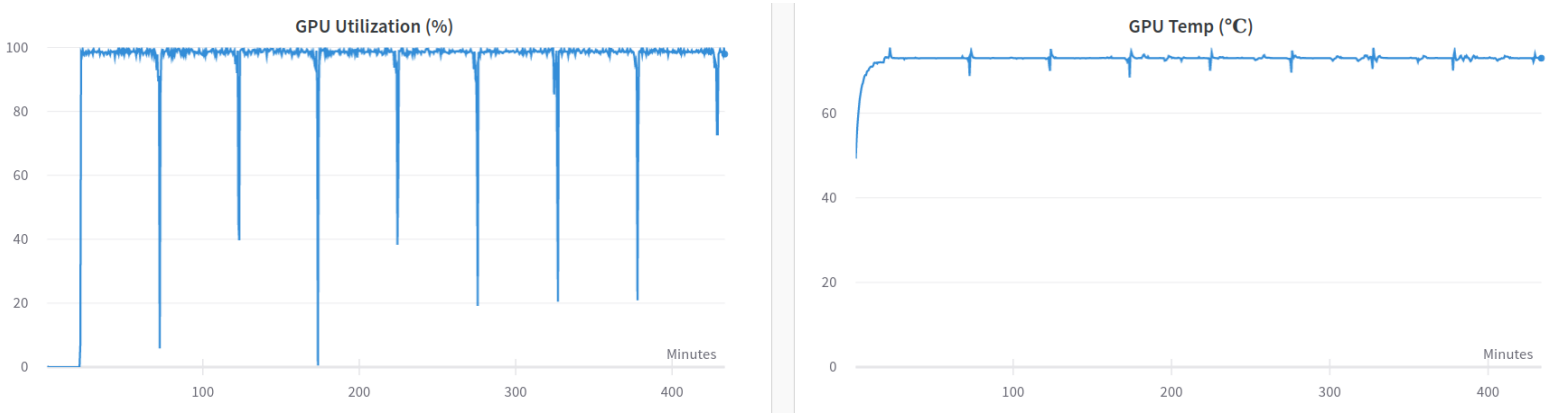
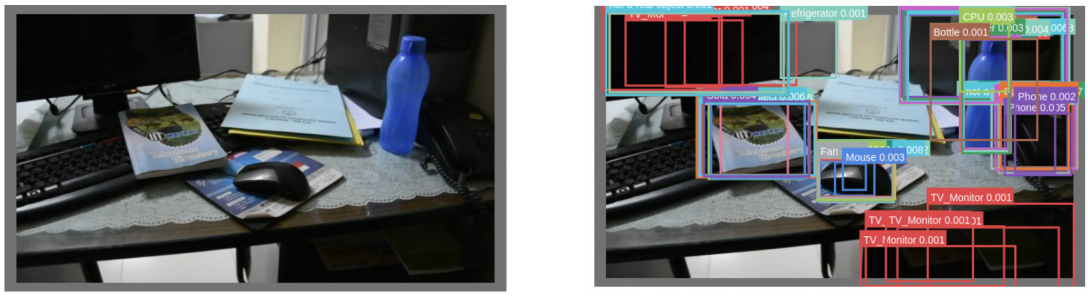


Figure 10 tells us that whenever there is a dip in the graph, one epoch is completed

Figure 11: Weight updation over each epoch

```
Epoch   gpu_mem       box       obj       cls    labels  img_size
 0/29     4.69G   0.06523   0.04546   0.06057        12       640: 100% 998/998 [48:14<00:00,  2.90s/it]
         Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 125/125 [02:07<00:00,  1.02s/it]
           all        995       3152      0.796      0.424      0.519      0.285

Epoch   gpu_mem       box       obj       cls    labels  img_size
 1/29     4.86G   0.05328   0.02991   0.02363        12       640: 100% 998/998 [47:48<00:00,  2.87s/it]
         Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 125/125 [02:07<00:00,  1.02s/it]
           all        995       3152      0.753      0.702      0.779      0.446

Epoch   gpu_mem       box       obj       cls    labels  img_size
 2/29     4.86G   0.04839   0.02743   0.01349        25       640: 100% 998/998 [47:49<00:00,  2.87s/it]
         Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 125/125 [02:08<00:00,  1.03s/it]
           all        995       3152      0.737       0.83       0.87      0.546

Epoch   gpu_mem       box       obj       cls    labels  img_size
 3/29     4.86G   0.04135   0.02732   0.01013         7       640: 100% 998/998 [48:14<00:00,  2.90s/it]
         Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 125/125 [02:08<00:00,  1.03s/it]
           all        995       3152      0.839      0.875      0.918      0.611

Epoch   gpu_mem       box       obj       cls    labels  img_size
 4/29     4.86G   0.03606   0.02581  0.008017         4       640: 100% 998/998 [48:16<00:00,  2.90s/it]
         Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 125/125 [02:08<00:00,  1.03s/it]
           all        995       3152      0.902      0.947      0.964      0.704

Epoch   gpu_mem       box       obj       cls    labels  img_size
 5/29     4.86G     0.032     0.025  0.006488         7       640: 100% 998/998 [48:31<00:00,  2.92s/it]
         Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 125/125 [02:09<00:00,  1.04s/it]
           all        995       3152      0.882       0.97      0.969      0.708

Epoch   gpu_mem       box       obj       cls    labels  img_size
 6/29     4.86G   0.03121   0.02419  0.005574         6       640: 100% 998/998 [48:19<00:00,  2.91s/it]
         Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 125/125 [02:08<00:00,  1.03s/it]
           all        995       3152      0.944      0.974      0.971      0.728

Epoch   gpu_mem       box       obj       cls    labels  img_size
 7/29     4.86G   0.02947   0.02326  0.004914         5       640: 100% 998/998 [48:20<00:00,  2.91s/it]
         Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 125/125 [02:08<00:00,  1.03s/it]
           all        995       3152      0.944      0.974      0.976      0.724

Epoch   gpu_mem       box       obj       cls    labels  img_size
 8/29     4.86G   0.02967   0.02408   0.00458        38       640:  11% 109/998 [05:17<43:32,  2.94s/it]
```

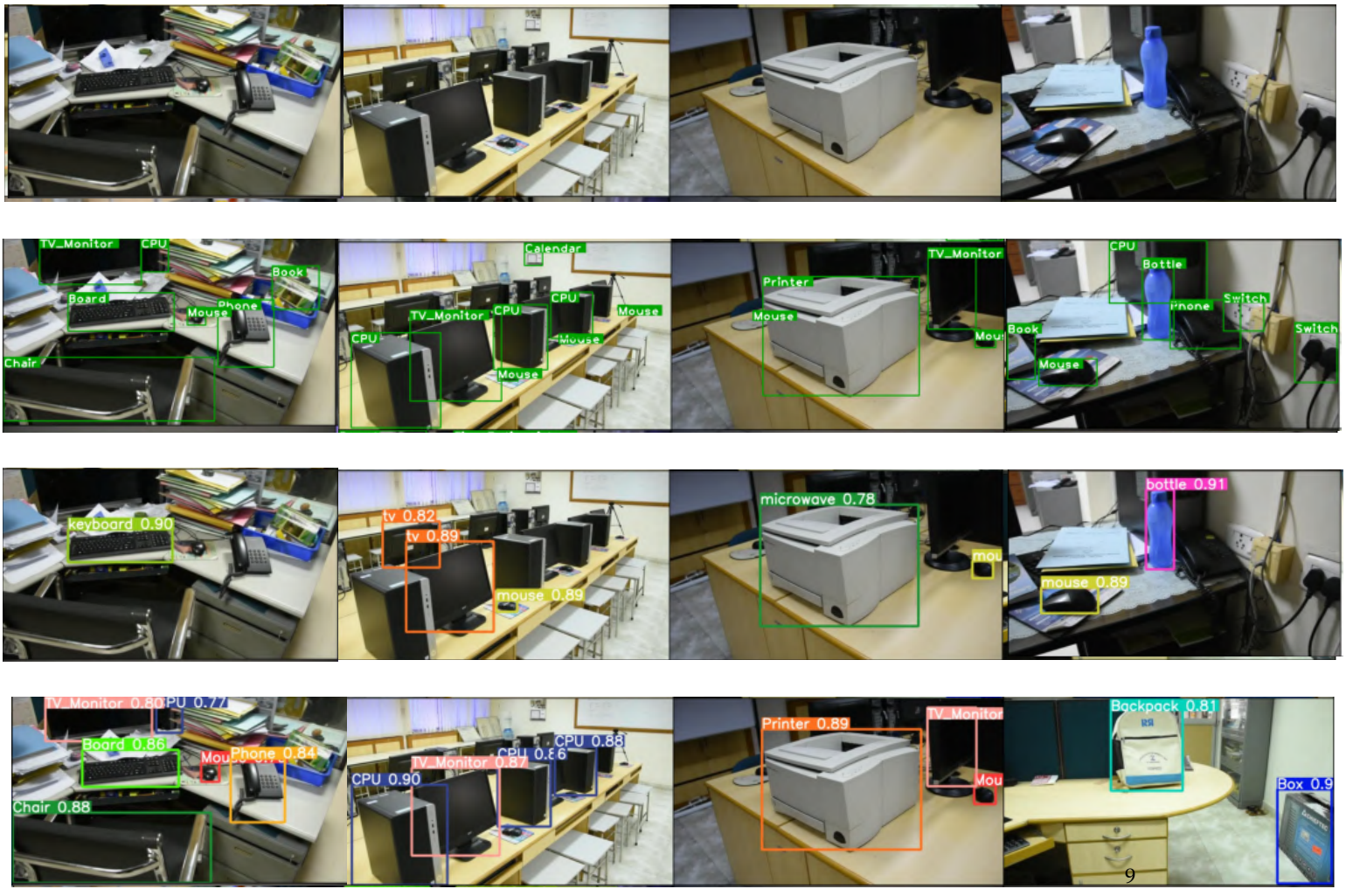
Figure 12: Non-maximal separation in Yolo v5x:



At first, what YOLO does is, consider all the bounding boxes while detecting objects in an image, but YOLO in itself does Non-maximal separation which rejects all the redundant bounding boxes with low confidence levels and gives us only the best bounding boxes upon thresholding the IOU to be 0.5.

C. Observation on evaluation of our custom-trained model :

Figure 13: Comparison of pre-trained and custom-trained images on Yolo v5x (First row: Input, **Second row:** Ground truth, **Third row**: Pretrained Yolo v5x, **Fourth row**: Custom-trained Yolo v5x)



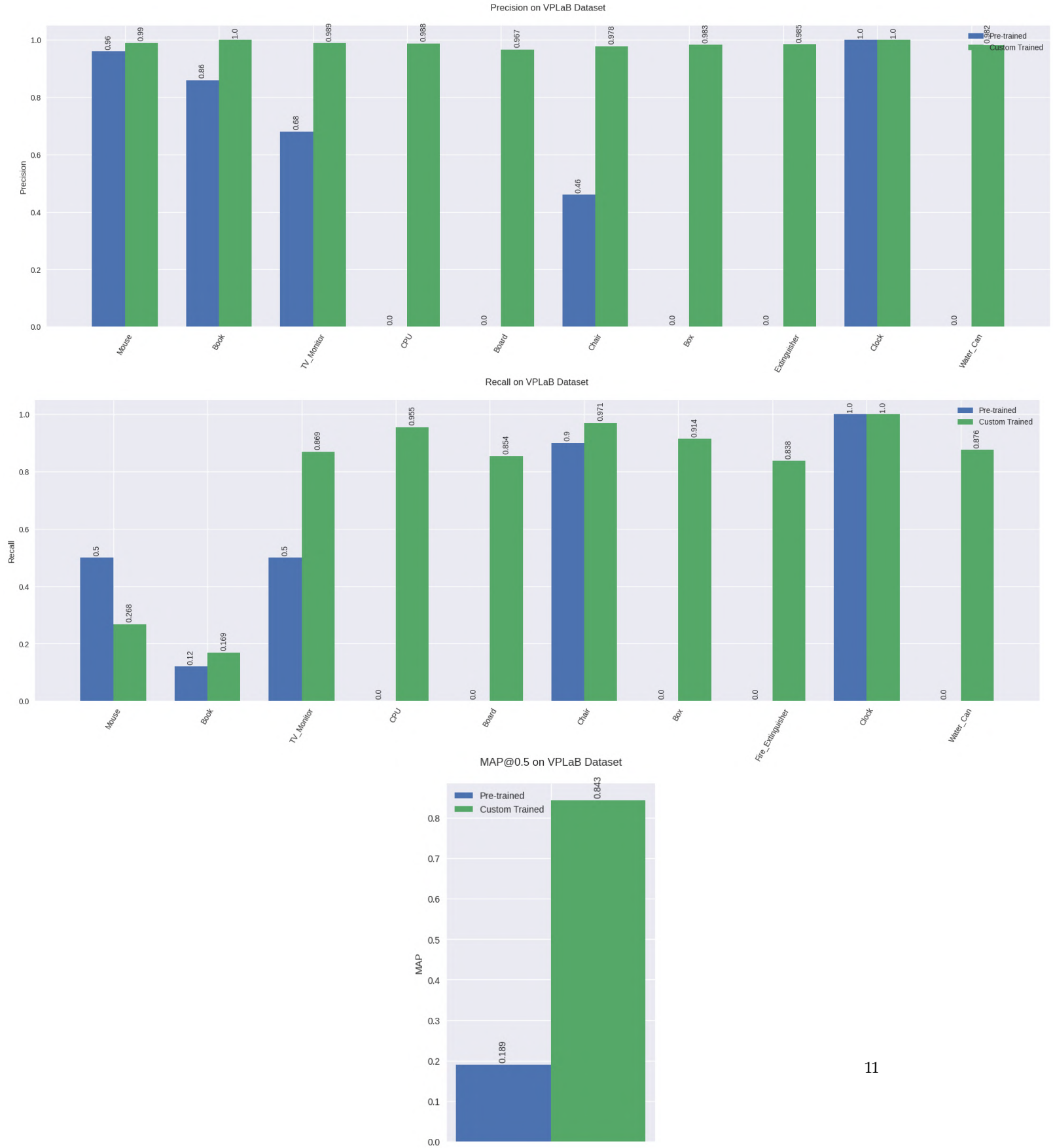
- From figure 13, we see that default YOLOv5x performs a very poor job in this sample image, that it couldn't find some of the objects in the image, whereas when we used our custom-trained model on the same testing sample, we better results, and the confidence levels in the bounding box is also high.
- Additionally in figure 13, we also see that even if some objects are recognized incorrectly, eg. Pre-trained Yolo v5x recognizes "Printer" as "Microwave" in the third column of the third image leading to poor MAP scores.
- The results for our custom-trained model, we see that it is much better than before, previously, we got a MAP = 0.189, but now we got MAP = 0.843 which is shown in Table 3, which is way better on the different samples of images that we used.

We can get an even better score on MAP but there are cases where the model detects objects even there is no annotation corresponding to it, so if in an image there are 5 monitors and our custom-trained model detects the 5 of them but if there were annotations for only any 2 monitors, the rest of the monitors even though they were detected good, the MAP score corresponding to the model will be a tad low corresponding to the True Positives present in the annotations of the dataset.

Table 3: Comparison of MAP, Precision, Recall Scores for Pre-trained and Custom-trained Yolo v5x

Class Labels	Precision 1 (Pre-trained Yolo v5x)	Precision 2 (Custom-trained Yolo v5x)	Recall 1 (Pre-trained Yolo v5x)	Recall 2 (Custom-trained Yolo v5x)
Mouse	0.96	0.99	0.5	0.268
Book	0.86	1	0.12	0.169
TV_Monitor	0.68	0.989	0.5	0.869
CPU	0	0.988	0	0.955
Board	0	0.967	0	0.854
Chair	0.46	0.978	0.9	0.971
Box	0	0.983	0	0.914
Fire_Extinguisher	0	0.985	0	0.838
Clock	1	1	1	1
Water_Can	0	0.982	0	0.876
For VPLab Dataset	MAP for Pre-trained Yolo v5x		MAP for Custom-trained Yolo v5x	
	0.189		0.843	

Figure 14: Bar graphs showing the improvement of Precision, Recall scores over different classes, and overall MAP improvement



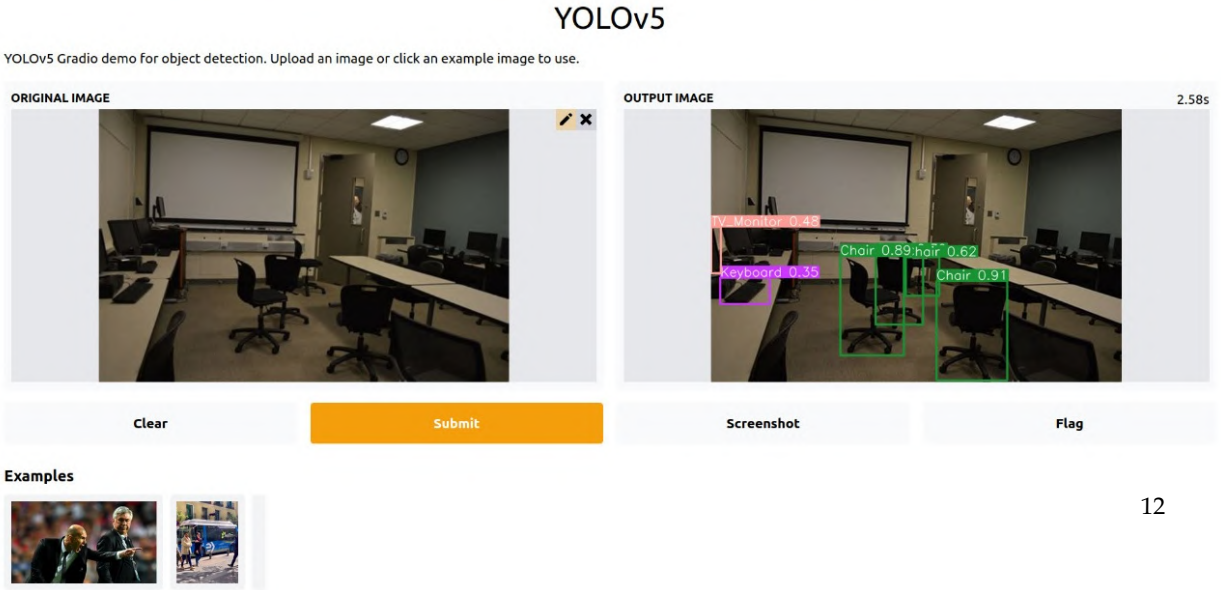
REAL-TIME EVALUATION OF IMAGES

During the TPA viva, we performed a real-time demonstration on the test images given, and let's check how our custom trained model performs for any general indoor lab images.

Figure 15: Comparison between default YOLOv5x and custom trained YOLOv5x on general demo images (First-Input image**, Second-**Pretrained Yolo v5x**, Third-**Custom trained Yolo v5x**)**

From figure 15, we can see that pre-trained Yolo v5x performs slightly better because the former is trained on a variety of datasets, but the custom-trained Yolov5x is trained on a dataset that is very less in comparison to the standard COCO dataset, but we can also see that our custom-trained Yolo v5x was able to detect objects like a dustbin and switch while former fails to do so, and another major difference is that our model (custom-trained Yolo v5x) is not performing well in comparison to that of of the former in case of cluttered images. This can be improved when we can customize the ground truth labels in the training dataset so that our model too can perform in cluttered environments.

Figure 16: Realtime Custom Trained YOLOv5 prototype using Gradio



CONCLUSION

On performing this experiment, we get that the model performs well if we give the datasets and features according to it, which is the basis for ML/DL algorithms as we need datasets to train upon a model so that we can see whether the algorithm performs well with similar features. So, if illumination, or cluttered images, or different noises are accounted into the training of the model, on testing when we use datasets, even present in different environments, if the given environment includes similar features used in training, our present algorithm performs well and it can be seen from our outputs and observations.

REFERENCES

- **Pytorch YOLOv5 Hub:** https://pytorch.org/hub/ultralytics_yolov5
- **Yolov5 GitHub**: https://github.com/ultralytics/yolov5
- **VP Lab** and **COCO** image datasets
- **Gradio**: https://www.gradio.app/
- **Weights and Biases**: https://wandb.ai/site
- **Voxel FiftyOne**: https://voxel51.com/fiftyone/
- https://curiousily.com/posts/object-detection-on-custom-dataset-with-yolo-v5-using-pytorch-and-python
- https://www.kaggle.com/aruchomu/yolo-v3-object-detection-in-tensorflow/notebook
- https://www.kaggle.com/dikshabhati2002/yolov4-object-detection
- https://www.kaggle.com/sajinpgupta/object-detection-using-yolov3/

YOLO Research Papers:

- You Only Look Once: Unified, Real-Time Object Detection; Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi
- YOLO9000:Better, Faster, Stronger; Joseph Redmon, Ali Farhadi
- YOLOv3: An Incremental Improvement; Joseph Redmon, Ali Farhadi
- YOLOv4: Optimal Speed and Accuracy of Object Detection; Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao