Object detection on tensorflow using [TensorFlow Hub Object Detection Colab]

Selected best performing models from list of [Object detections models]

Model Name	Speed (ms)	COCO mAP
CenterNet HourGlass104 512x512	70	41.9
EfficientDet D3 896x896	95	45.4

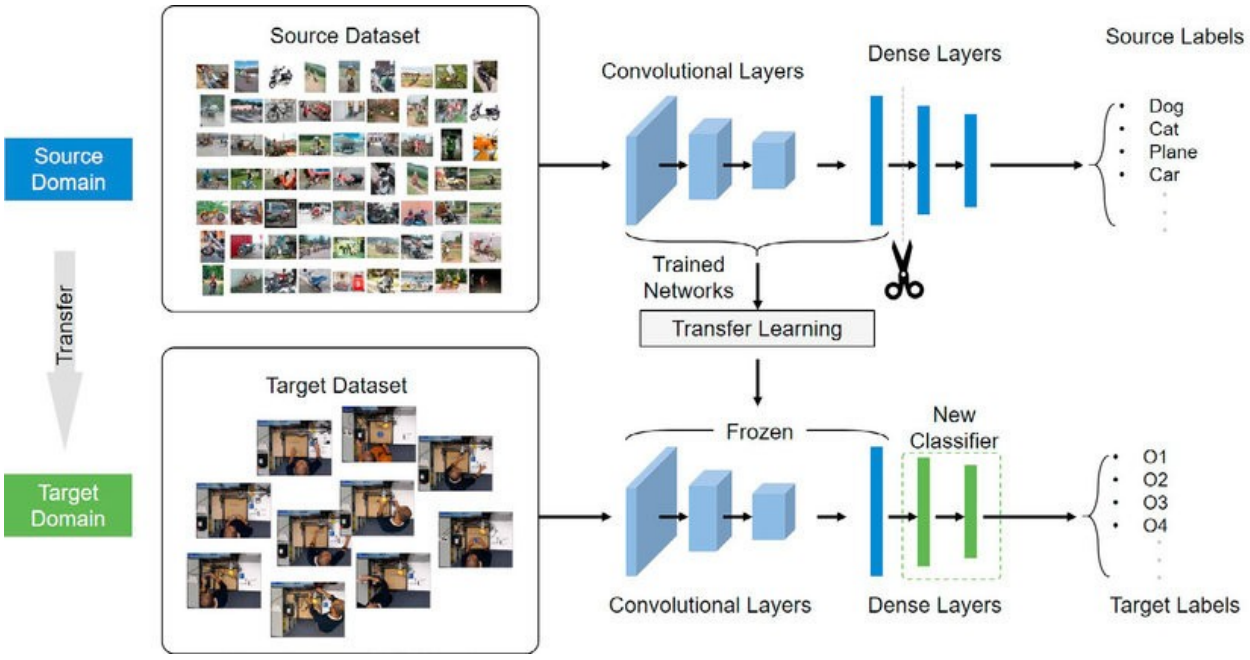
Testing both the pre trained models of tensorflow on specific images

	CenterNet HourGlass104 512x512	EfficientDet D3 896x896
femaleconnector-0.jpg		
femaleconnector-46.jpg		
femaleconnector-60.jpg		
femaleconnector-73.jpg		
femaleconnector-82.jpg		
femaleconnector-91.jpg		
femaleconnector-123.jpg		

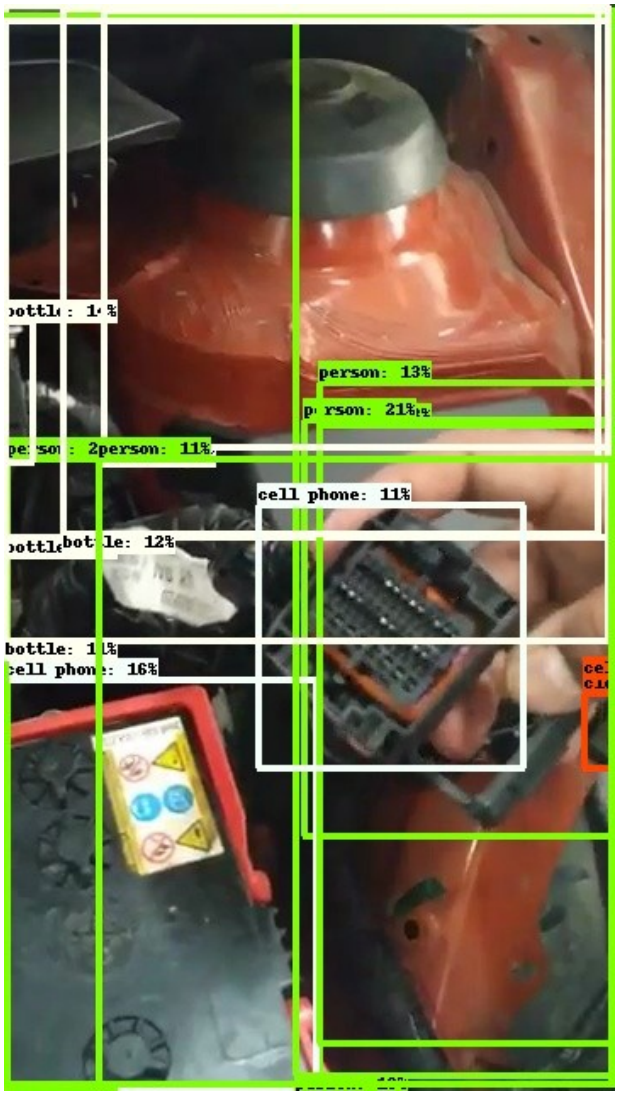
Result

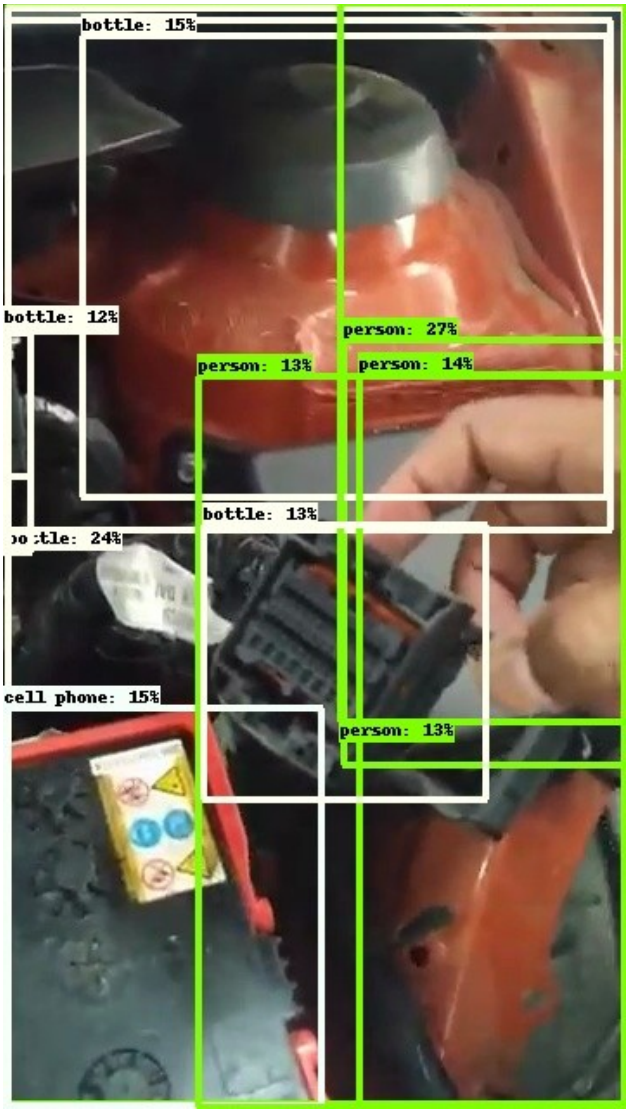
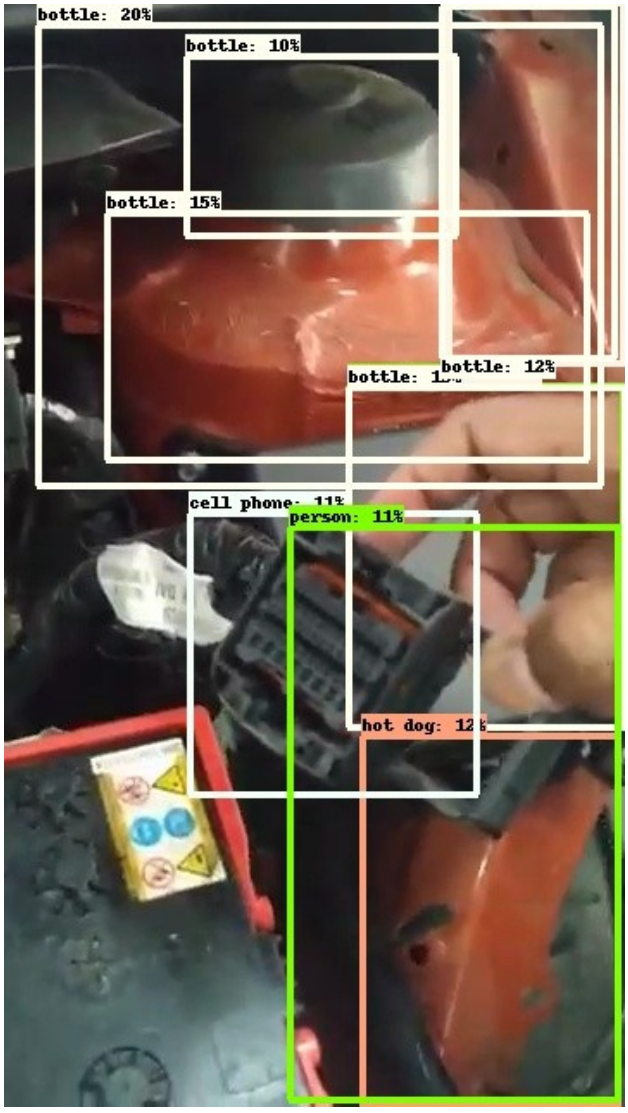
1. Detections directly implies the features learnt by the pre trained network for our custom task
2. More features learnt equals more detection equals good model for transfer learning

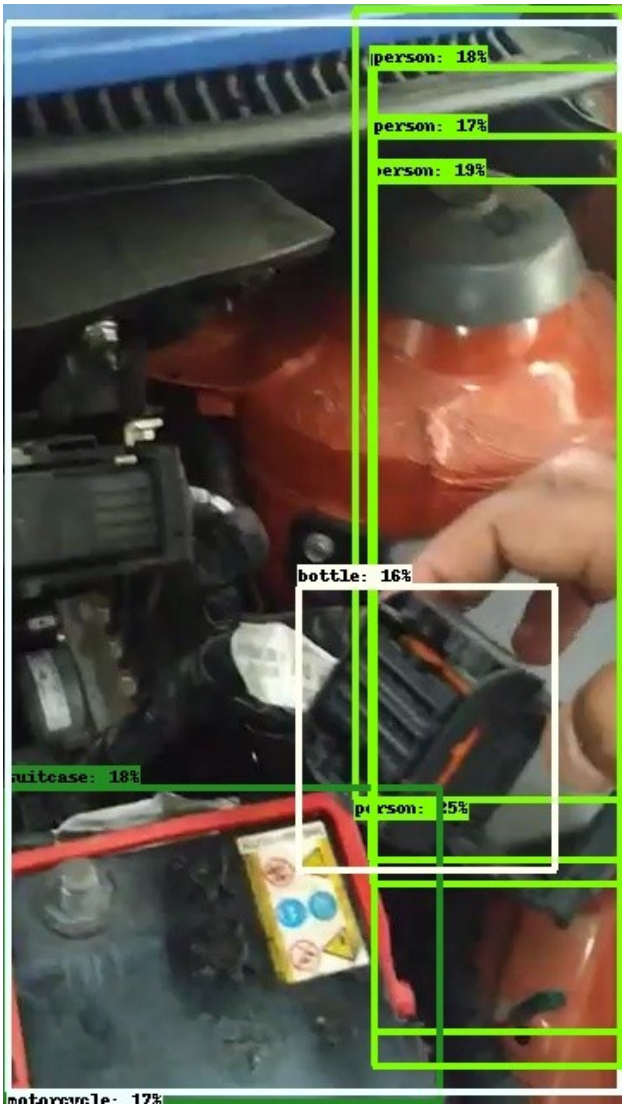
Transfer Learning

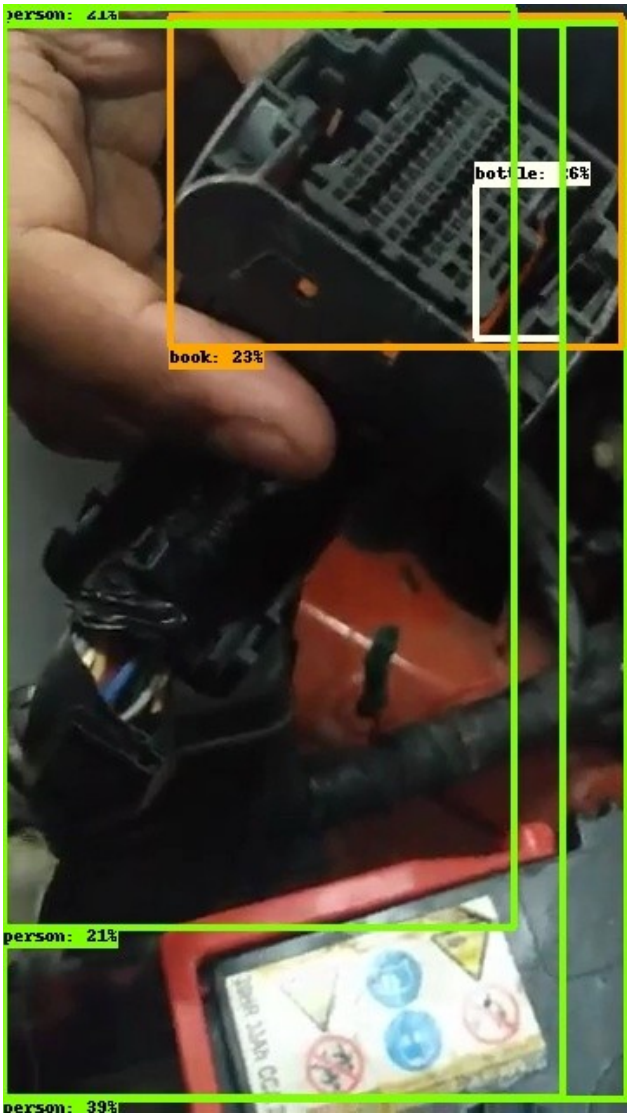
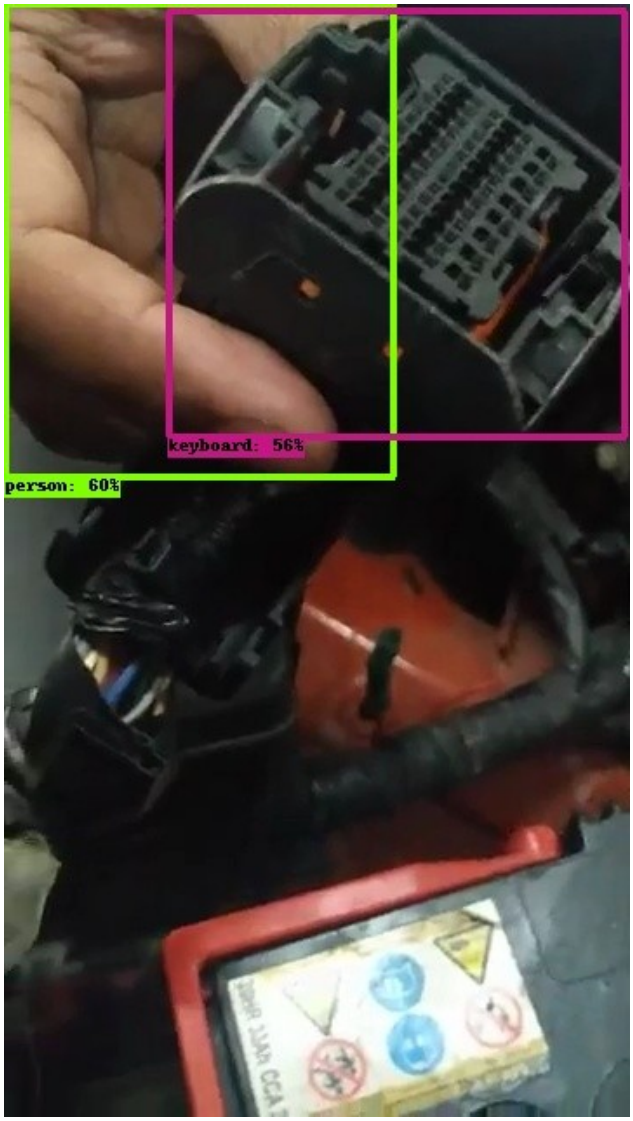


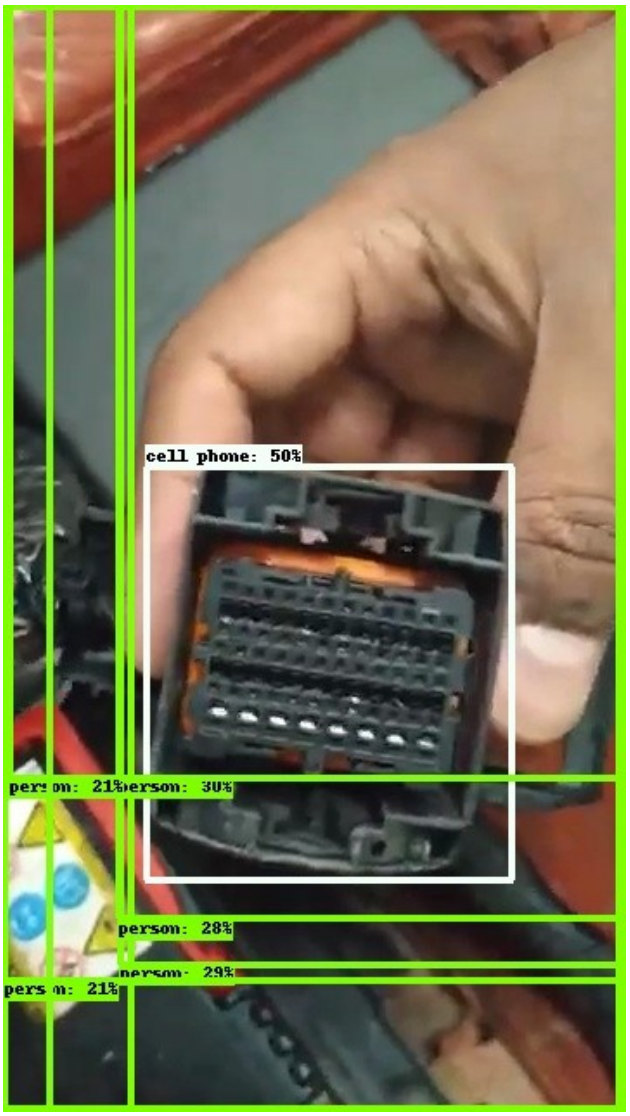
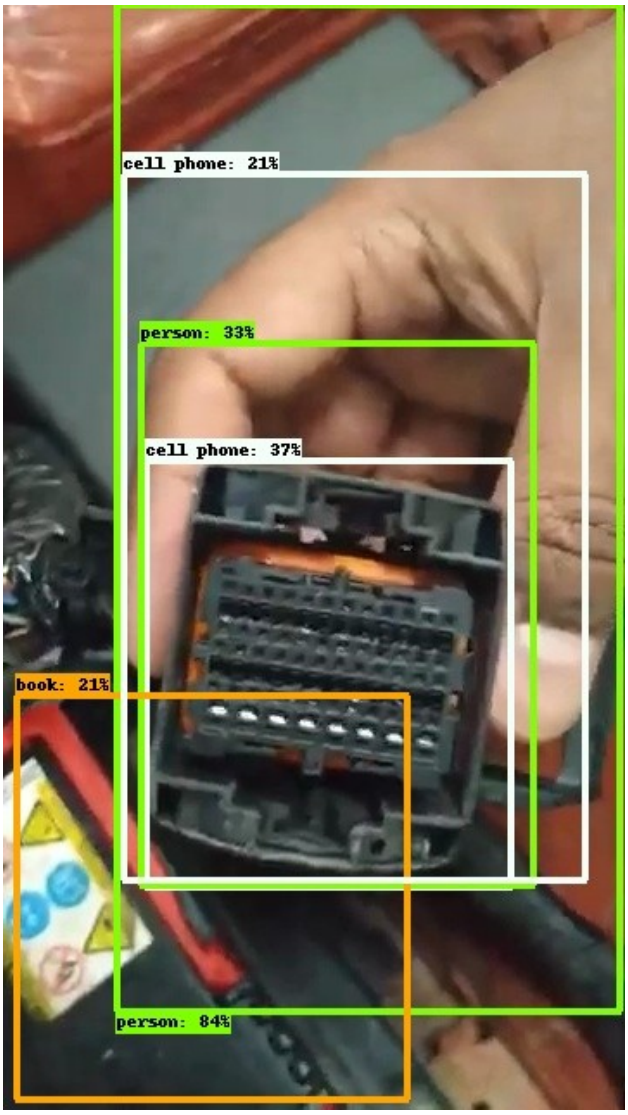
- Freeze/do not train the Convolution layers/feature extractors of the network
- Unfreeze/train the Dense layers/Prediction layer of the network

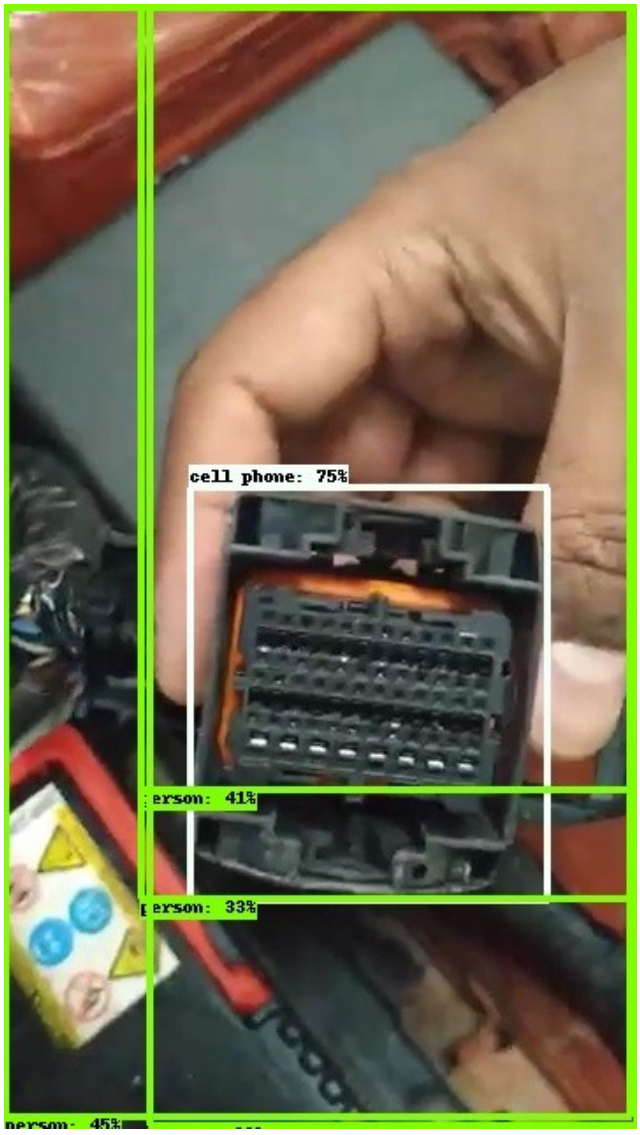
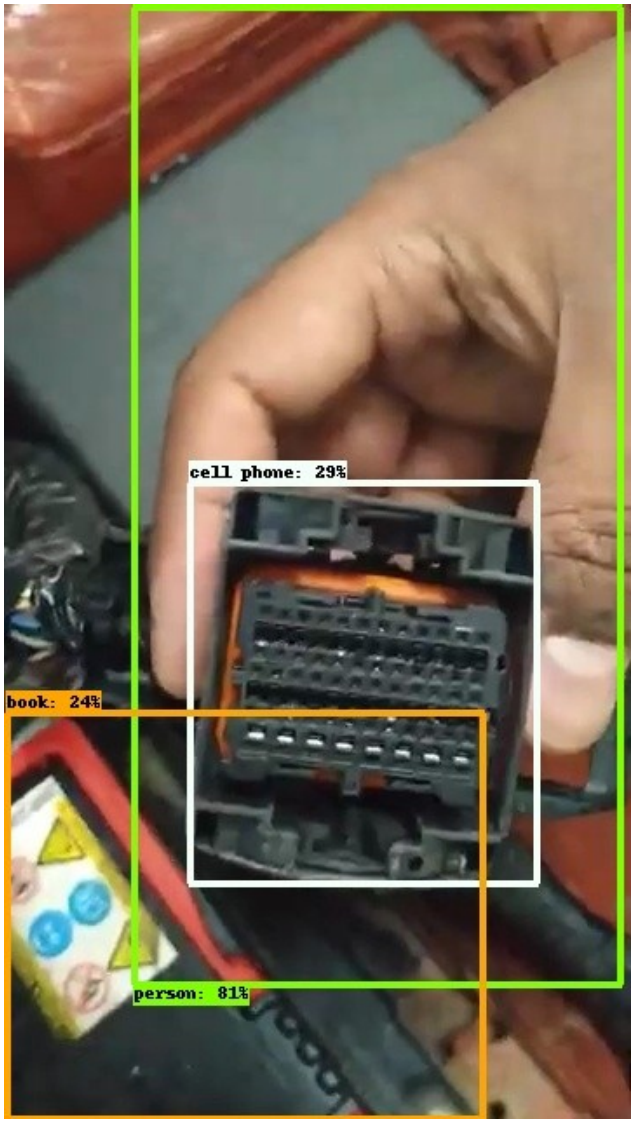
CenterNet HourGlass104 512x512	EfficientDet D3 896x896
femaleconnector-0.jpg	
	

CenterNet HourGlass104 512x512	EfficientDet D3 896x896
femaleconnector-46.jpg	
	

CenterNet HourGlass104 512x512	EfficientDet D3 896x896
femaleconnector-60.jpg	
	

CenterNet HourGlass104 512x512	EfficientDet D3 896x896
femaleconnector-73.jpg	
	

CenterNet HourGlass104 512x512	EfficientDet D3 896x896
femaleconnector-82.jpg	
	

CenterNet HourGlass104 512x512	EfficientDet D3 896x896
femaleconnector-91.jpg	
	

CenterNet HourGlass104 512x512	EfficientDet D3 896x896
femaleconnector-123.jpg	
	

Transfer learning with TensorFlow Hub has feature extractor layer trainable section is for classification not for object detection.

There isn't any feature extraction layer for object detection on tensorflow hub.

Looking into pipeline.config of the object detection of **CenterNet HourGlass104 512x512** doesn't contain any freeze_variable or trainable layers as seen below

```
model {
  center_net {
    num_classes: 1
    feature_extractor {
      type: "hourglass_104"
      channel_means: 104.01361846923828
      channel_means: 114.03422546386719
      channel_means: 119.91659545898438
      channel_stds: 73.60276794433594
      channel_stds: 69.89082336425781
      channel_stds: 70.91507720947266
      bgr_ordering: true
    }
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 512
        max_dimension: 512
        pad_to_max_dimension: true
      }
    }
    object_detection_task {
      task_loss_weight: 1.0
      offset_loss_weight: 1.0
      scale_loss_weight: 0.10000000149011612
      localization_loss {
        l1_localization_loss {
        }
      }
    }
    object_center_params {
      object_center_loss_weight: 1.0
      classification_loss {
        penalty_reduced_logistic_focal_loss {
          alpha: 2.0
          beta: 4.0
        }
      }
      min_box_overlap_iou: 0.699999988079071
      max_box_predictions: 100
    }
  }
}
train_config {
  batch_size: 128
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_aspect_ratio: 0.5
      max_aspect_ratio: 1.7000000476837158
      random_coef: 0.25
    }
  }
  data_augmentation_options {
    random_adjust_hue {
```

```
    }
  }
  data_augmentation_options {
    random_adjust_contrast {
    }
  }
  data_augmentation_options {
    random_adjust_saturation {
    }
  }
  data_augmentation_options {
    random_adjust_brightness {
    }
  }
  data_augmentation_options {
    random_absolute_pad_image {
      max_height_padding: 200
      max_width_padding: 200
      pad_color: 0.0
      pad_color: 0.0
      pad_color: 0.0
    }
  }
  optimizer {
    adam_optimizer {
      learning_rate {
        manual_step_learning_rate {
          initial_learning_rate: 0.0010000000474974513
          schedule {
            step: 90000
            learning_rate: 9.999999747378752e-05
          }
          schedule {
            step: 120000
            learning_rate: 9.999999747378752e-06
          }
        }
      }
      epsilon: 1.0000000116860974e-07
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "/content/drive/MyDrive/Tensorflow2/training_demo/pre-trained/centernet_hg104_512x512_coco17_tpu-
8/checkpoint/ckpt-0"
  num_steps: 140000
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "/content/drive/MyDrive/Tensorflow/dataset/labelmap.pbtxt"
  tf_record_input_reader {
    input_path: "/content/drive/MyDrive/Tensorflow/dataset/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
  batch_size: 1
}
eval_input_reader {
  label_map_path: "/content/drive/MyDrive/Tensorflow/dataset/labelmap.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "/content/drive/MyDrive/Tensorflow/dataset/test.record"
  }
}
```

There seems to be no support from the tensorflow group from a long time with regards to how to freeze the layers for object detection task from these links

1. Tensorflow HUB - transfer learning for object detection
2. Object detection have no freeze variables
3. Freeze variable not being accessed
4. Possible solution Freeze weight
    1. Solution 1 not working
    2. Solution 2 no reponse from official team

Training the pre-trained object detection using the custom data section shown in the TensorFlow 2 Object detection API doesn't seem to yield good results as the entire network gets trained and the pre-trained models accuracy also decreases according to this review

I have raised an issue myself on github tensorflow models repository Transfer learning and fine tuning of hidden layers and the final layer #10602 by looking at tensorflow github response to this topic I doubt it will ever be answered.

Tensorflow doesn't seem to be a good choice for such a task

Pytorch

Object Detection Models

Model Name	COCO mAP
Faster R-CNN ResNet-50 FPN	37
FCOS ResNet-50 FPN	39.2
Mask R-CNN ResNet-50 FPN	37.9

File: **PytorchObjectDetectionForCustomDataInferenceAndTraining.ipynb**

Testing both the pre trained models of pytorch on specific images

	Faster R-CNN ResNet-50 FPN	Mask R-CNN ResNet-50 FPN
femaleconnector-0.jpg		
femaleconnector-46.jpg		
femaleconnector-60.jpg		
femaleconnector-73.jpg		
femaleconnector-82.jpg		
femaleconnector-91.jpg		
femaleconnector-123.jpg		

Mask R-CNN - requires mask too, object detection can't be trained alone as see from error I got the colab page I tried to train

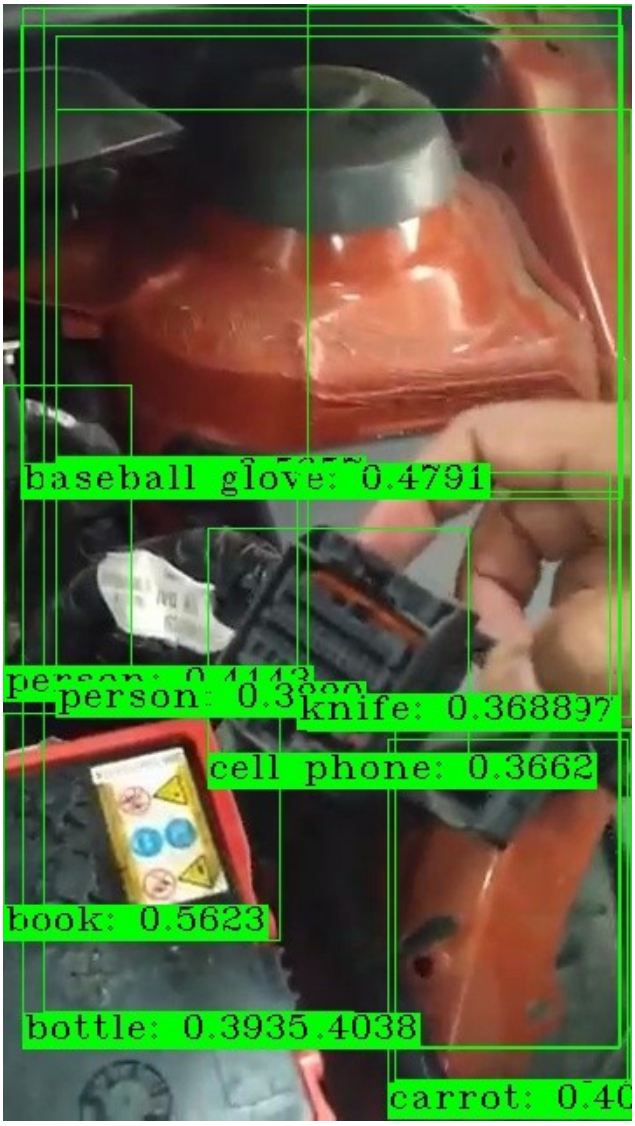
	Faster R-CNN ResNet-50 FPN	FCOS ResNet-50 FPN
femaleconnector-0.jpg		
femaleconnector-46.jpg		
femaleconnector-60.jpg		
femaleconnector-73.jpg		
femaleconnector-82.jpg		
femaleconnector-91.jpg		
femaleconnector-123.jpg		

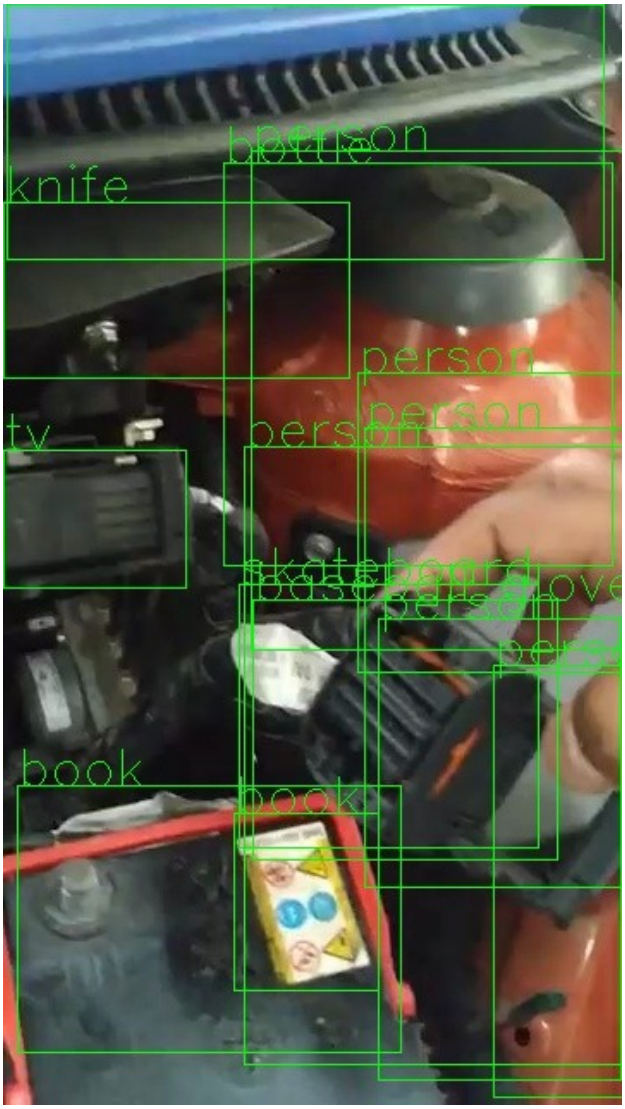
FCOS ResNet-50 FPN - 91 COCO classes have been hardcoded into the pytorch official model, I have raised the issue on the official github page of pytorch Transfer learning using pre trained objective detection model FCOS: Fully Convolutional One-Stage Object Detection architecture #5932

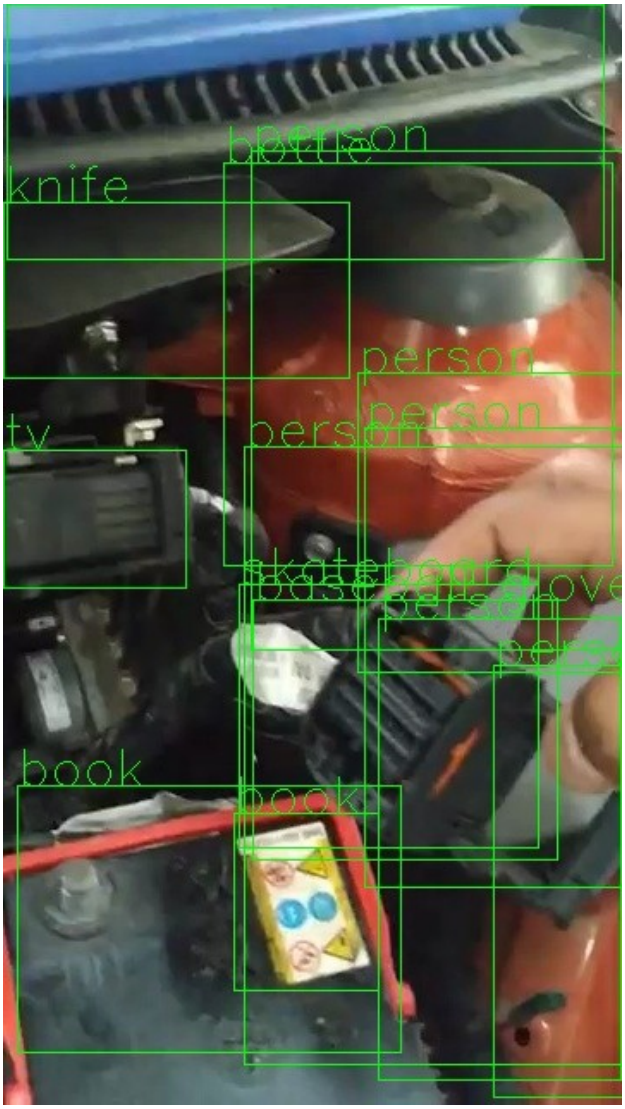
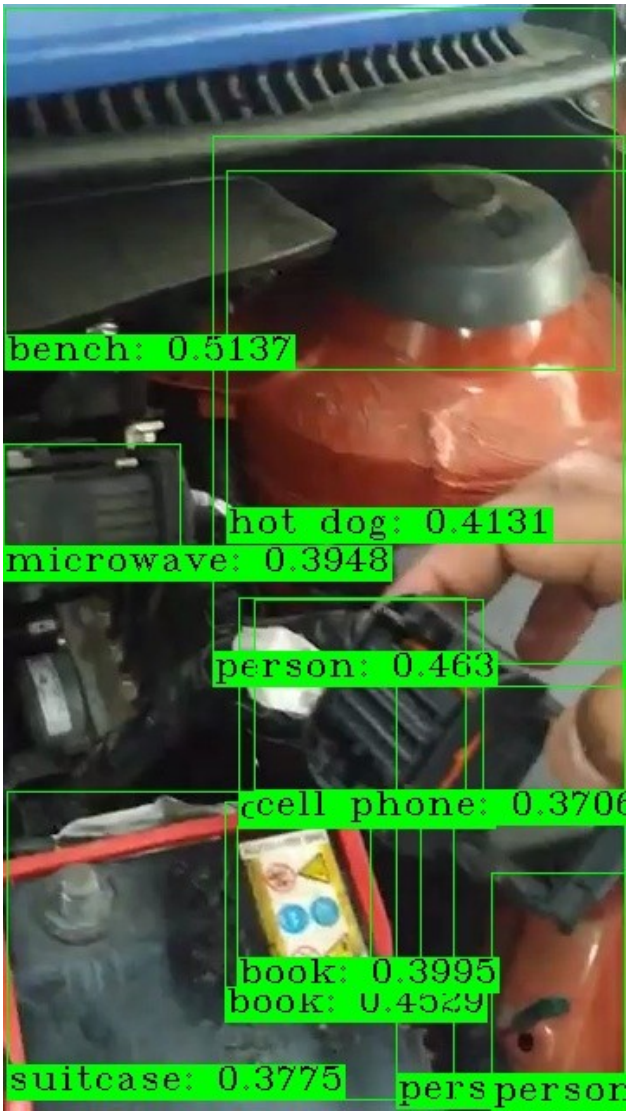
Faster R-CNN ResNet-50 FPN	Mask R-CNN ResNet-50 FPN
femaleconnector-0.jpg	
	

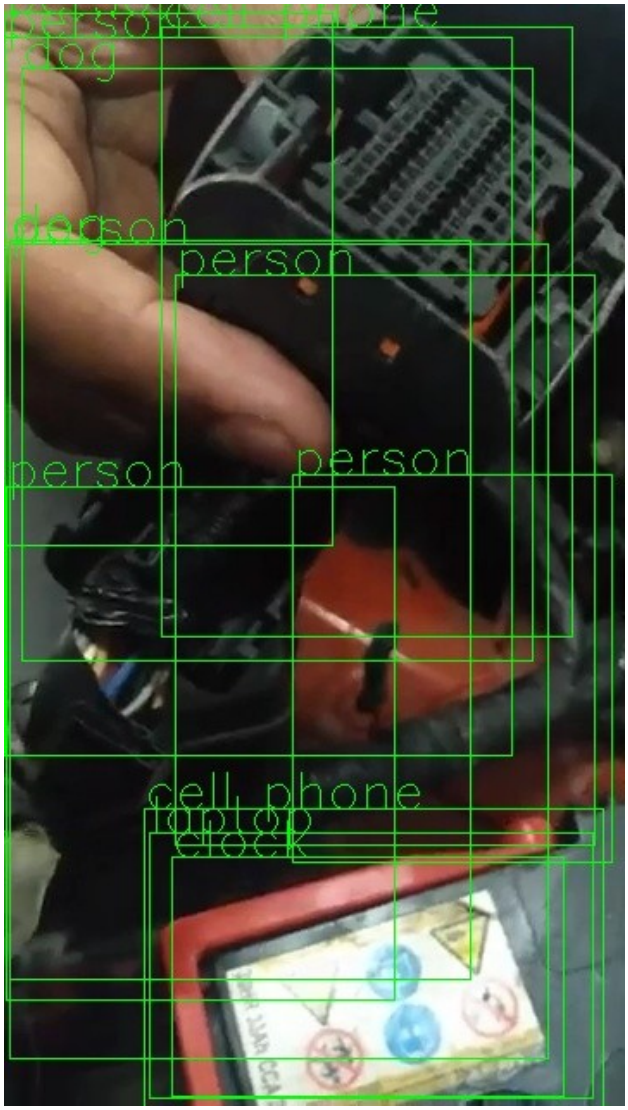
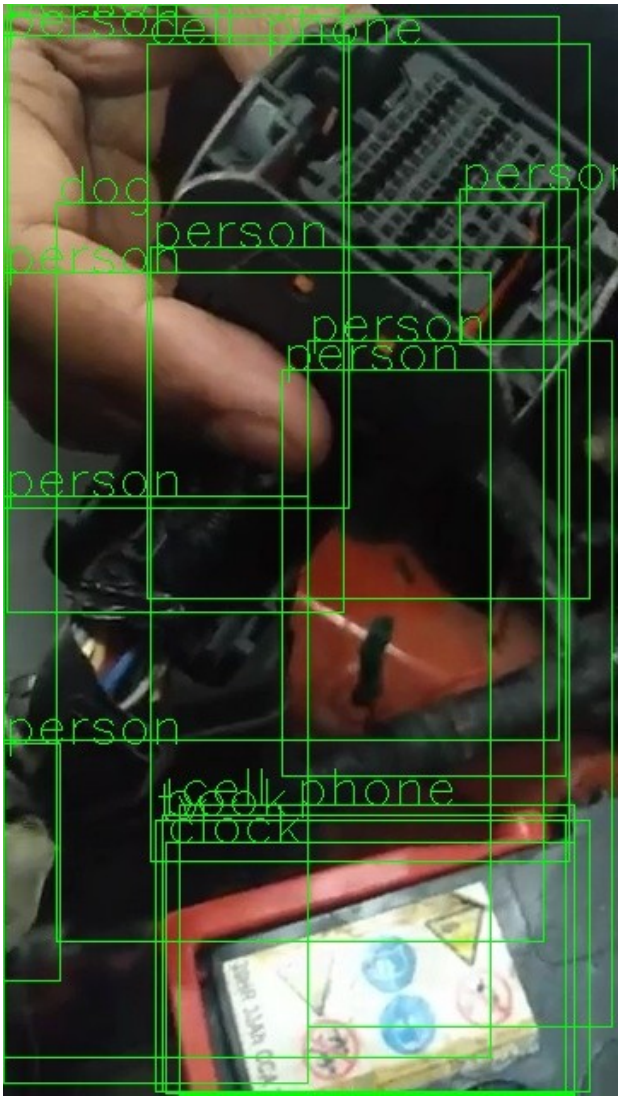
Faster R-CNN ResNet-50 FPN	FCOS ResNet-50 FPN
femaleconnector-0.jpg	
	

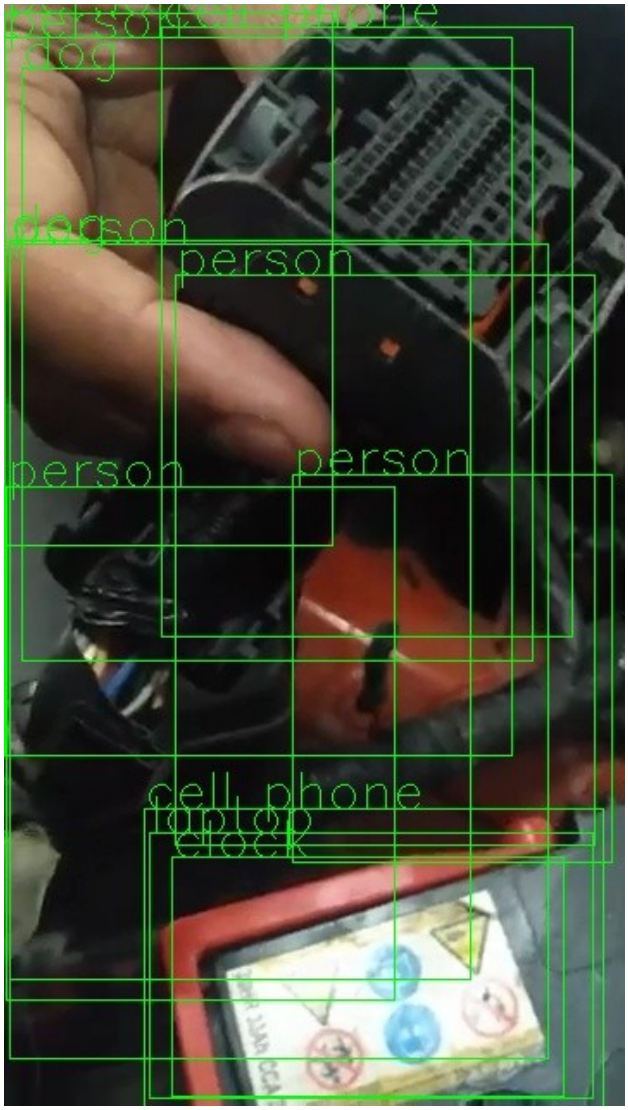
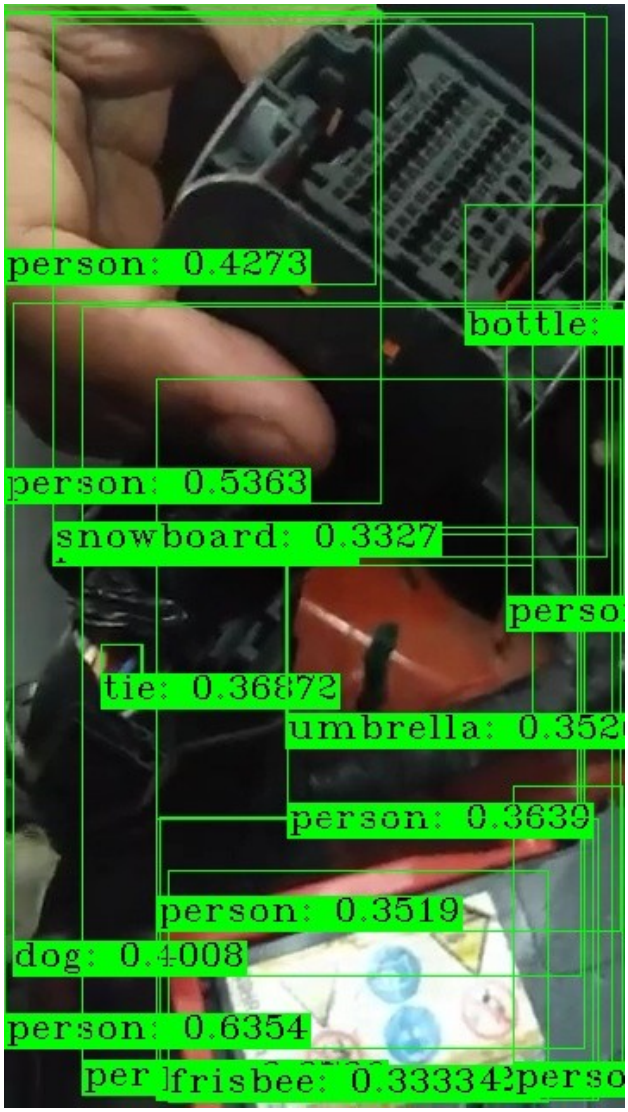
Faster R-CNN ResNet-50 FPN	Mask R-CNN ResNet-50 FPN
femaleconnector-46.jpg	
	

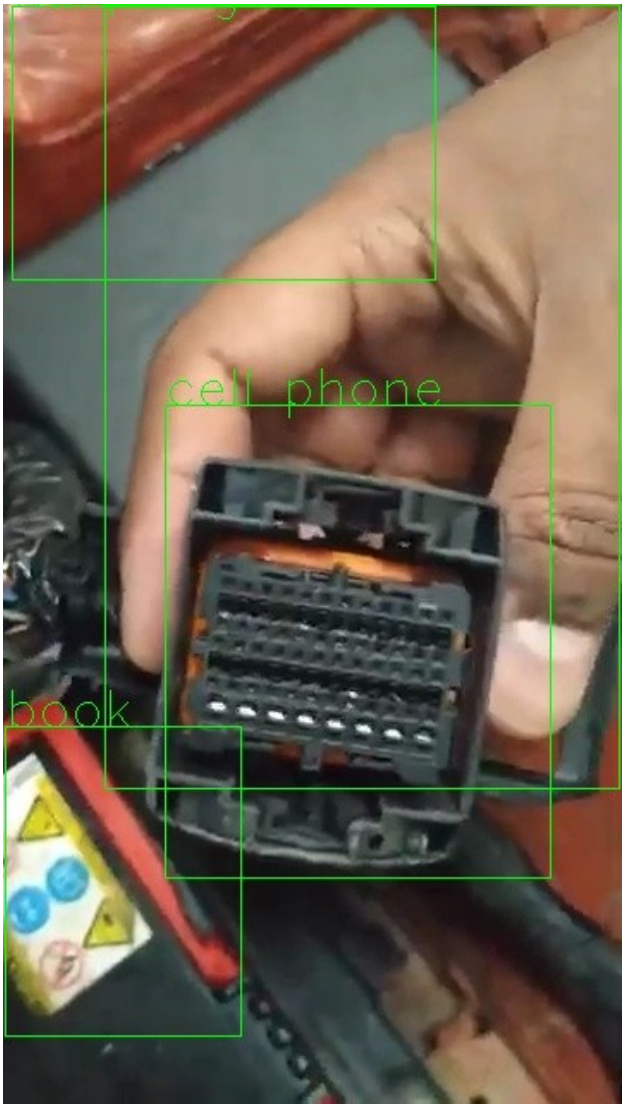
Faster R-CNN ResNet-50 FPN	FCOS ResNet-50 FPN
femaleconnector-46.jpg	
	

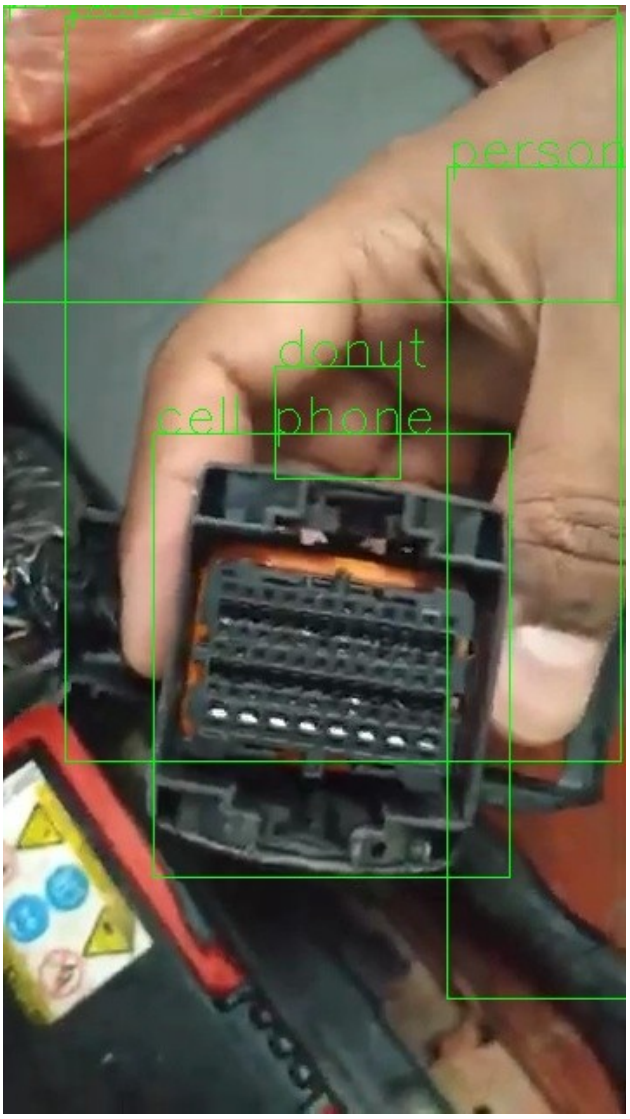
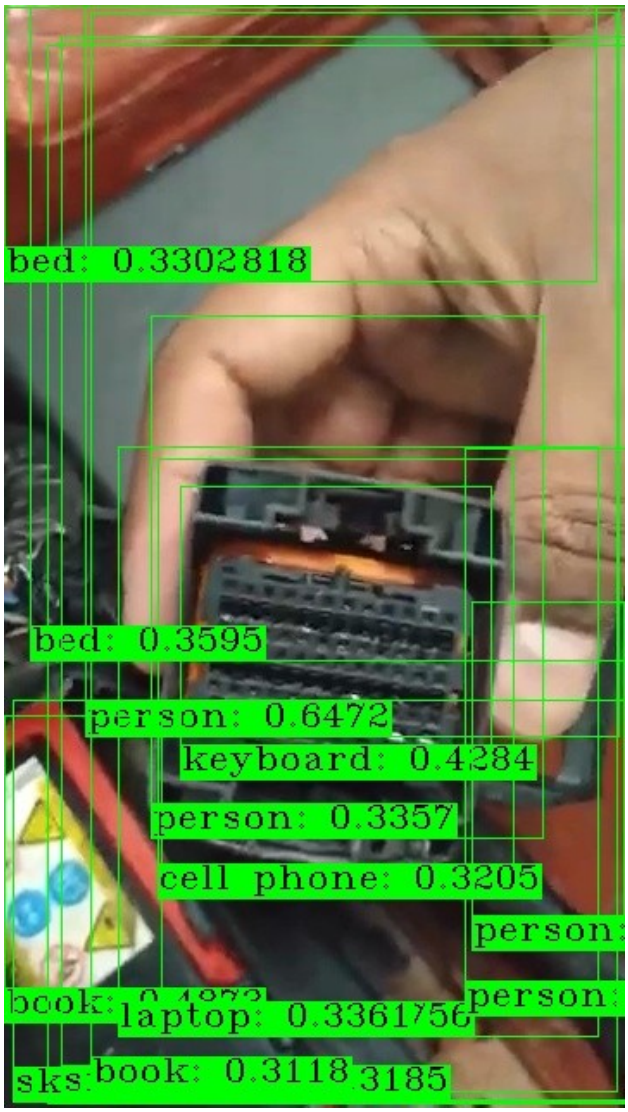
Faster R-CNN ResNet-50 FPN	Mask R-CNN ResNet-50 FPN
femaleconnector-60.jpg	
	

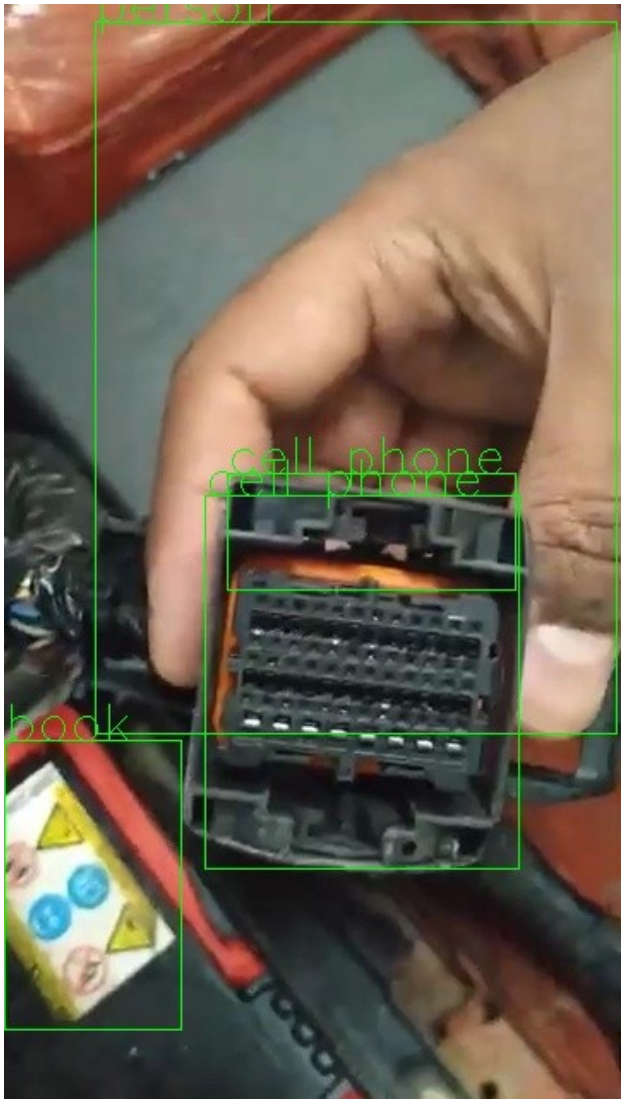
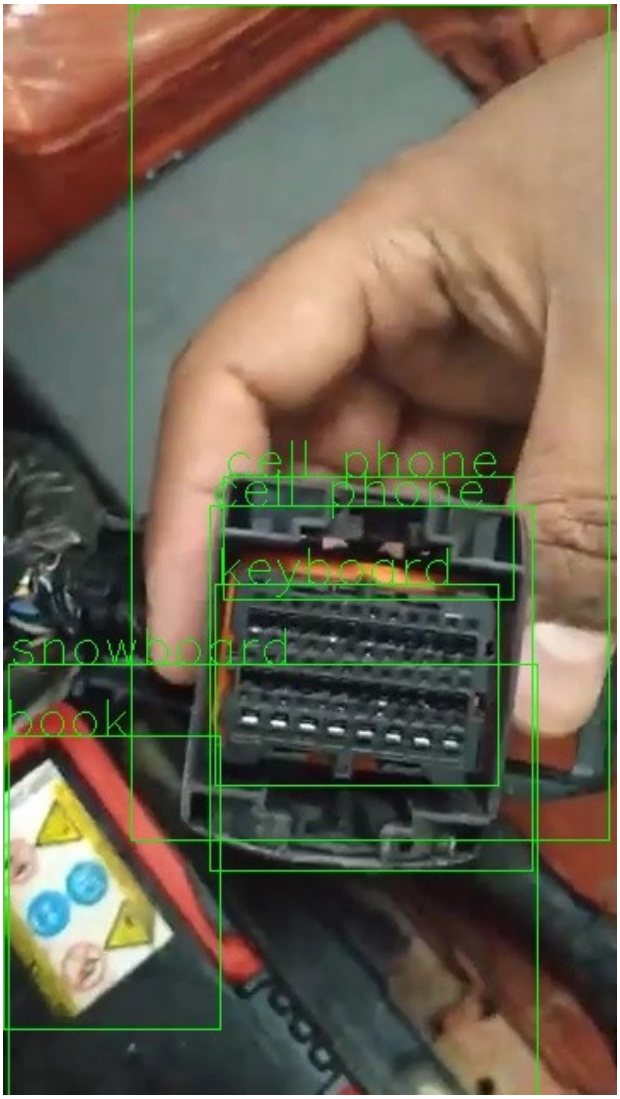
Faster R-CNN ResNet-50 FPN	FCOS ResNet-50 FPN
femaleconnector-60.jpg	
	

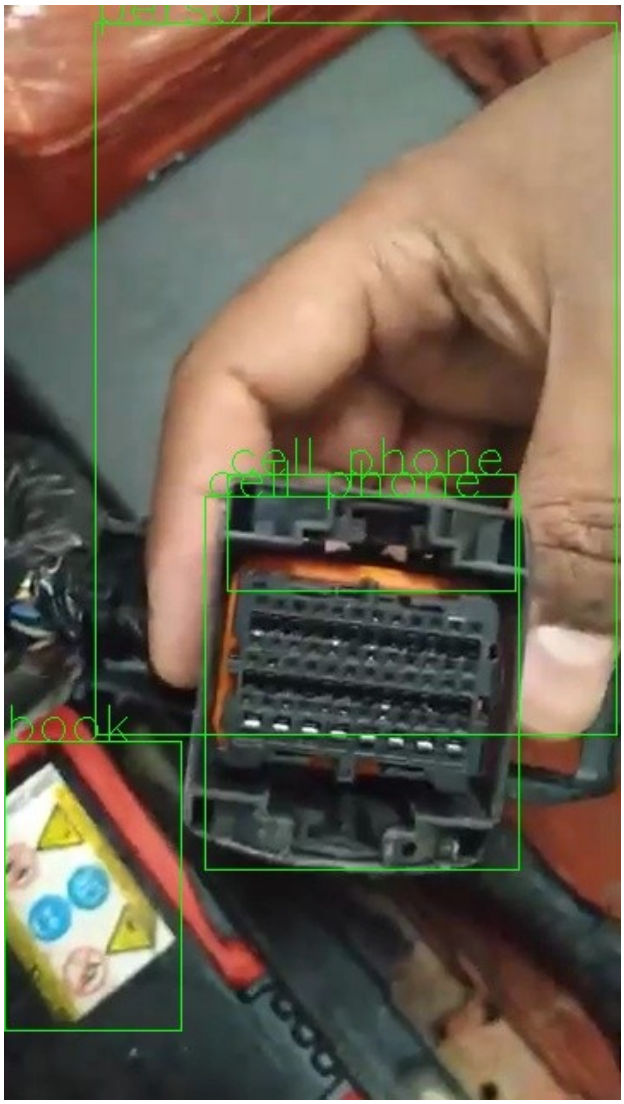
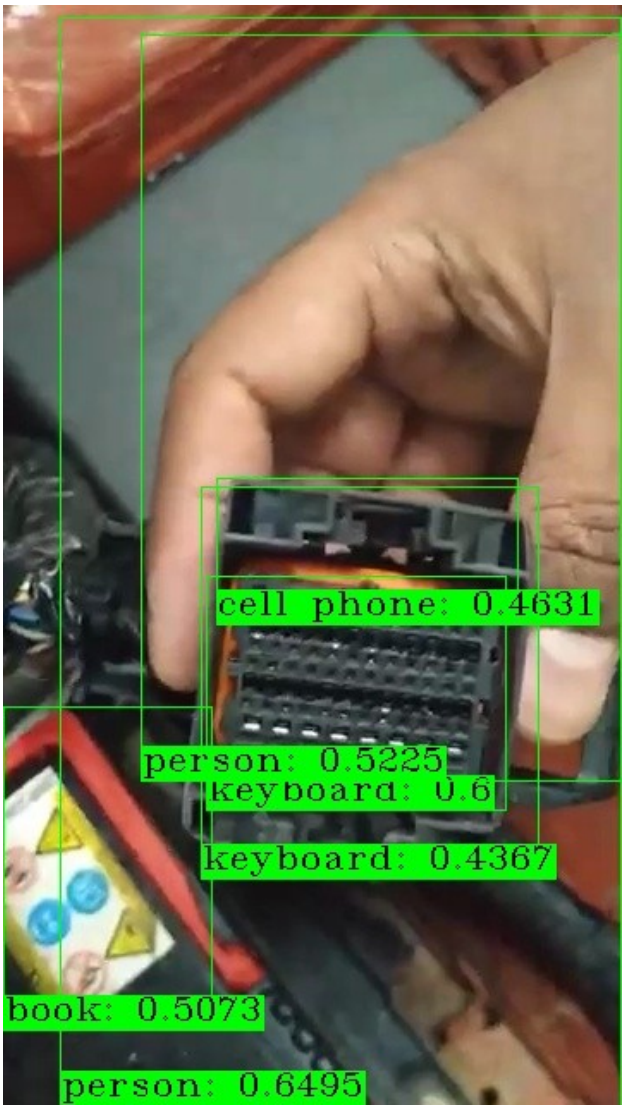
Faster R-CNN ResNet-50 FPN	Mask R-CNN ResNet-50 FPN
femaleconnector-73.jpg	
	

Faster R-CNN ResNet-50 FPN	FCOS ResNet-50 FPN
femaleconnector-73.jpg	
	

Faster R-CNN ResNet-50 FPN	Mask R-CNN ResNet-50 FPN
femaleconnector-82.jpg	
	

Faster R-CNN ResNet-50 FPN	FCOS ResNet-50 FPN
femaleconnector-82.jpg	
	

Faster R-CNN ResNet-50 FPN	Mask R-CNN ResNet-50 FPN
femaleconnector-91.jpg	
	

Faster R-CNN ResNet-50 FPN	FCOS ResNet-50 FPN
femaleconnector-91.jpg	
	

Faster R-CNN ResNet-50 FPN	Mask R-CNN ResNet-50 FPN
femaleconnector-123.jpg	
	

Faster R-CNN ResNet-50 FPN	FCOS ResNet-50 FPN
femaleconnector-123.jpg	
	

Training Using **Faster R-CNN ResNet-50 FPN in pytorch**

- Looking at the prediction layer which contains **0 background** + **90 classes**

```
roi_heads.box_predictor.cls_score.weight
 torch.Size([91, 1024])
roi_heads.box_predictor.cls_score.bias
 torch.Size([91])
```

- Selecting label **77 Cell Phone** tensor in the final layer

```
Replacing roi_heads.box_predictor.cls_score.weight layer containing 90 class score weight , with 77 th class score weight
####################################
Original layer size(0:Background + 90 classes):  torch.Size([91, 1024])
Alteres layer size(0:Background + 77 th class):  torch.Size([2, 1024])
####################################
Finished enabling requires gradient for roi_heads.box_predictor.cls_score.weight layer.....

Replacing roi_heads.box_predictor.cls_score.bias layer containing 90 class score bias , with 77 th class score bias
####################################
Original layer size(0:Background + 90 classes):  torch.Size([91])
Alteres layer size(0:Background + 77 th class):  torch.Size([2])
####################################
Finished enabling requires gradient for roi_heads.box_predictor.cls_score.weight layer.....
```

- Splitting **1890** images into **train:test** = **80:20**

```
We have: 1890 examples, 1512 are training and 378 testing
```

- Training
  - lr: learning rate
  - loss: training losses
  - loss_class: loss_classifier
  - loss_box: loss_rpn_box_reg
  - loss_object: loss_objectness
  - loss_r_b_g: loss_rpn_box_reg
  - **Adam + LamdbLR**

```
optimizer = torch.optim.Adam(params,lr=0.005,betas=(0.9,0.999),eps=1e-08,weight_decay=0.0005,amsgrad=False)
lambda1 = lambda epoch: 0.65 ** epoch
lr_scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda=lambda1)
```

  - **SGD + StepLR**

```
optimizer = torch.optim.SGD(params, lr=0.005,momentum=0.9, weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,step_size=3,gamma=0.1)
```

- Training loss chart

Epoch	Adam + LambdLR						SGD + StepLR					
	Lr	Loss	loss_class	loss_box	loss_object	loss_r_b_g	Lr	Loss	loss_class	loss_box	loss_object	loss_r_b_g
0	0.005	0.1684	0.0370	0.1215	0.006	0.0054	0.005	0.1822	0.0509	0.1215	0.0063	0.0054
1	0.0032	0.1665	0.0327	0.1244	0.0067	0.0047	0.005	0.1809	0.0421	0.1244	0.0066	0.0047
2	0.0021	0.1724	0.0333	0.1212	0.0065	0.0054	0.005	0.1827	0.0415	0.1212	0.0079	0.0054
3	0.0013	0.1601	0.0314	0.1156	0.0052	0.0052	0.0005	0.1736	0.0423	0.1156	0.0053	0.0052
4	0.000893	0.1527	0.0286	0.1091	0.0054	0.0051	0.0005	0.1579	0.0364	0.1091	0.0052	0.0051
5	0.000580	0.1585	0.0271	0.1103	0.0061	0.0059	0.0005	0.1687	0.0368	0.1103	0.0048	0.0059
6	0.000377	0.1621	0.0308	0.1170	0.0061	0.0052	0.00005	0.1714	0.0423	0.1170	0.0046	0.0052
7	0.000245	0.1600	0.0294	0.1154	0.0064	0.0050	0.00005	0.1639	0.0378	0.1154	0.0048	0.0050
8	0.000159	0.1602	0.0283	0.1206	0.0051	0.0047	0.00005	0.1732	0.0375	0.1206	0.0055	0.0047
9	0.000104	0.1595	0.0278	0.1183	0.0053	0.0051	.000005	0.1699	0.0380	0.1183	0.0057	0.0051

- **Adam seems to be better when looked at the learning rate to losses**

IoU Metric of bounding box

- Epoch 0

	Adam + LambdLR	SGD + StepLR
Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] =	0.062	0.062
Average Precision (AP) @[IoU=0.50 | area= all | maxDets=100] =	0.166	0.171
Average Precision (AP) @[IoU=0.75 | area= all | maxDets=100] =	0.039	0.033
Average Precision (AP) @[IoU=0.50:0.95 | area= small | maxDets=100] =	-1	-1
Average Precision (AP) @[IoU=0.50:0.95 | area=medium | maxDets=100] =	-1	-1
Average Precision (AP) @[IoU=0.50:0.95 | area= large | maxDets=100] =	0.063	0.063
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 1] =	0.104	0.098
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 10] =	0.329	0.342
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] =	0.329	0.342
Average Recall (AR) @[IoU=0.50:0.95 | area= small | maxDets=100] =	-1	-1
Average Recall (AR) @[IoU=0.50:0.95 | area=medium | maxDets=100] =	-1	-1
Average Recall (AR) @[IoU=0.50:0.95 | area= large | maxDets=100] =	0.329	0.342

- Epoch 9

	Adam + LambdLR	SGD + StepLR
Average Precision (AP) @[IoU=0.50:0.95 | area= all | maxDets=100] =	0.062	0.061
Average Precision (AP) @[IoU=0.50 | area= all | maxDets=100] =	0.170	0.170
Average Precision (AP) @[IoU=0.75 | area= all | maxDets=100] =	0.035	0.033
Average Precision (AP) @[IoU=0.50:0.95 | area= small | maxDets=100] =	-1	-1
Average Precision (AP) @[IoU=0.50:0.95 | area=medium | maxDets=100] =	-1	-1
Average Precision (AP) @[IoU=0.50:0.95 | area= large | maxDets=100] =	0.063	0.062
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 1] =	0.102	0.106
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets= 10] =	0.351	0.343
Average Recall (AR) @[IoU=0.50:0.95 | area= all | maxDets=100] =	0.351	0.343
Average Recall (AR) @[IoU=0.50:0.95 | area= small | maxDets=100] =	-1	-1
Average Recall (AR) @[IoU=0.50:0.95 | area=medium | maxDets=100] =	-1	-1
Average Recall (AR) @[IoU=0.50:0.95 | area= large | maxDets=100] =	0.351	0.343

    - **Adam seems better even from IoU than SGD**
- Loading the weights in the final layer as shown the tutorial will not give any results as can be seen from the file **FasterRCNN10epochsAdamLamdlrFIRSTstandard.txt**
- Training data can be found
    - FasterRCNN10epochsAdamLamdlrFIRSTRUN.txt
    - FasterRCNN10epochsSGDStepLRSecondRUN.txt
    - as
- I have open a discussion on **Improving transfer learning training of object detection model**
    - StackExchange
    - Pytorch discussion forum

I looked into latest object detection models

https://paperswithcode.com/sota/object-detection-on-coco
(Only object detection)

Model	Box AP	FPS	Transfer Learning	Remarks
DINO(Swin-L, multi-scale)	63.3	-	Code in development	Less epochs better result
DINO(Swin-L, single-scale)	63.2	-	Code in development	Less epochs better result
YOLOR-D6	57.3	34	Not available	Weight and config
YOLOR-E6	56.4	45	Not available	
YOLOR-W6	55.5	66	Not available	
YOLOv4-CSP-P7	55.4	16	Available(GPU)	Source
PyCenterNet (Swin-L, multi-scale)	57.1	-	-	Pre Trained not available
EfficientDet-D3 (single-scale)	47.5(D2-42.1)		Available	Source

- **DINO - DETR with Improved deNoising anchOr boxes**
    - **Paper:** https://arxiv.org/pdf/2203.03605.pdf
    - **Github Code:** https://github.com/IDEACVR/DINO
    - Highlight
        - Improvement in just 30+ epochs according to the published paper
- **EfficientDet**
    - It is not clear to me which layer does the classification in EfficientDet
    - I have raised the same on the two github repository which have implemented this
        1. Which layer is used for classification of classes? #720
        2. Which layers in the Efficientdet are used for classification of class scores? #273