Object detection in real traffic

Presented by,

Arkadip Basu, Kumar Gaurav

12th December 2022

Introduction

Self Driving cars is the already the next big thing in the automobile sector and it relies heavily on the deep learning computer vision for object detection and tracking.

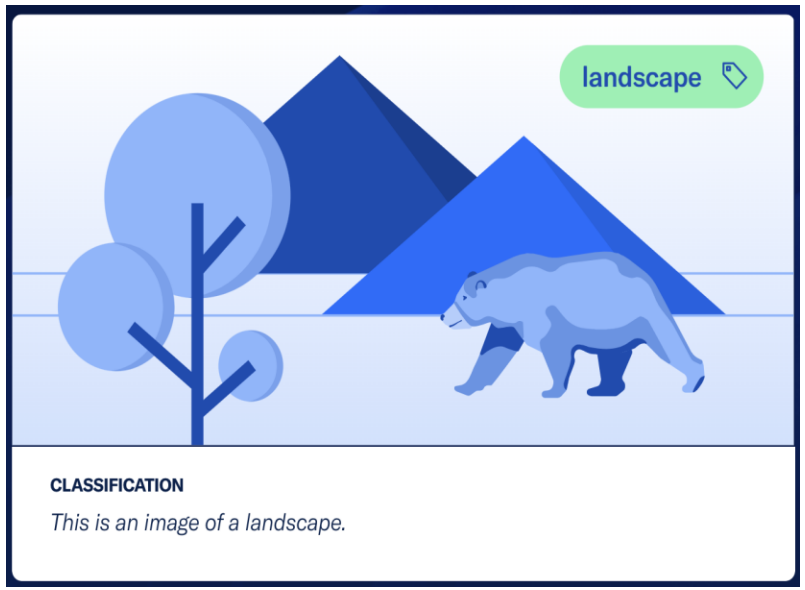
The model installed in the vehicle, is Eye + Brain, so it can detect various parameters as follows

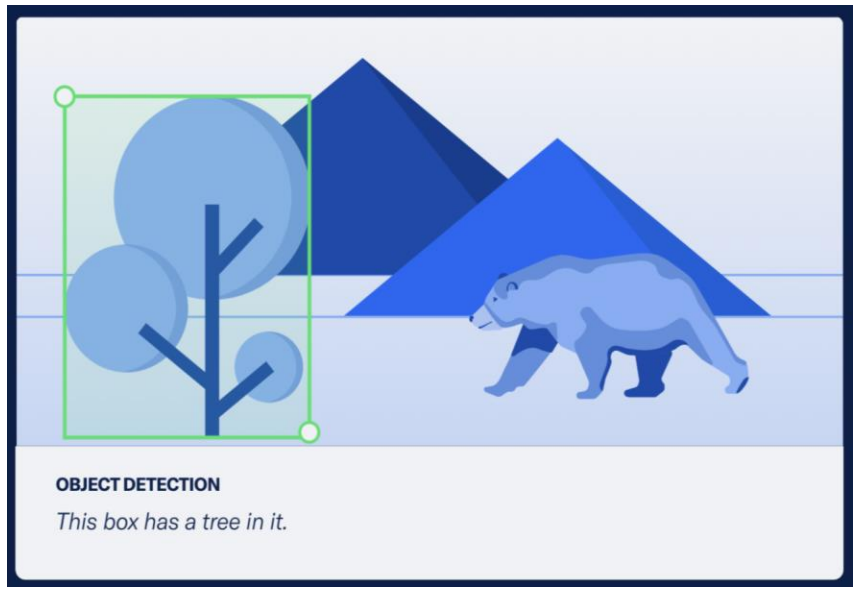
- Same / Opposite direction vehicle
- Speed of the vehicle and driving pattern
- Type of vehicle in front and back
- Things, those are object, but not obstacles

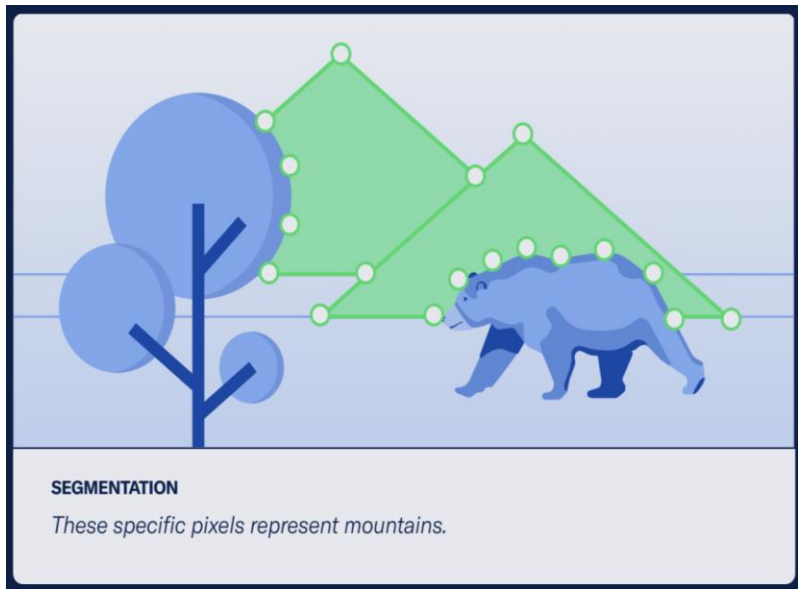
The challenges involved

- In practical sense , this model must work with ~100% accuracy
- It should also generalize better in a sense it works perfect with un-seen data
- Else it can cause casualties in case of in-accuracy in results.

Object Detection 101







Literature works & **Reference**

1. YOLO-v3 Paper : https://arxiv.org/abs/1804.02767
2. YOLO-v4 Paper : https://arxiv.org/abs/2004.10934v1
3. https://pjreddie.com/darknet/yolo/
4. Faster RCNN : https://arxiv.org/pdf/1506.01497.pdf
5. https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/
6. https://paperswithcode.com/
7. https://papertalk.org/

Shortcomings of literature works

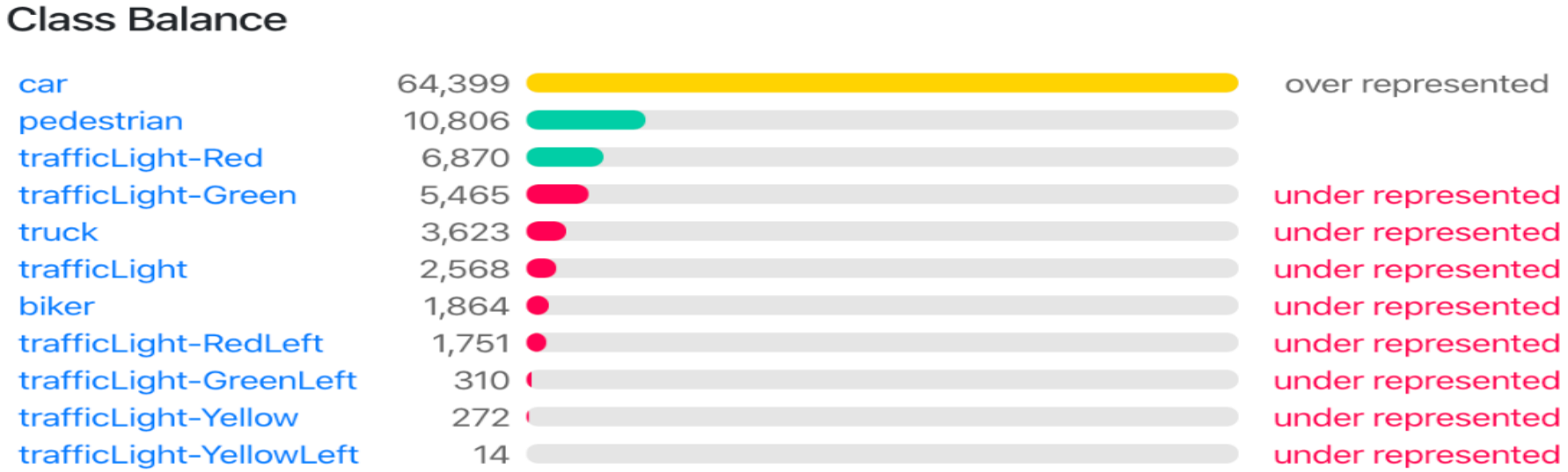
- As of now, there is no solution, which can correctly predict a stable object with confidence.
- If we add motion, there will be less FPS, detection with that is more challenging.
- The Model training is very resource and time intensive.

Notebook (Training & Testing)

https://drive.google.com/drive/folders/14QOIlJKCG9JxyS5O4NnY-amcd6z5_cI_?usp=sharing

Dataset details

- https://public.roboflow.com/object-detection/self-driving-car
- 15000 images , 97942 labels and 11 classes
- 1720 images with no labels
- All images are 512*512*3



Addressing Class imbalance

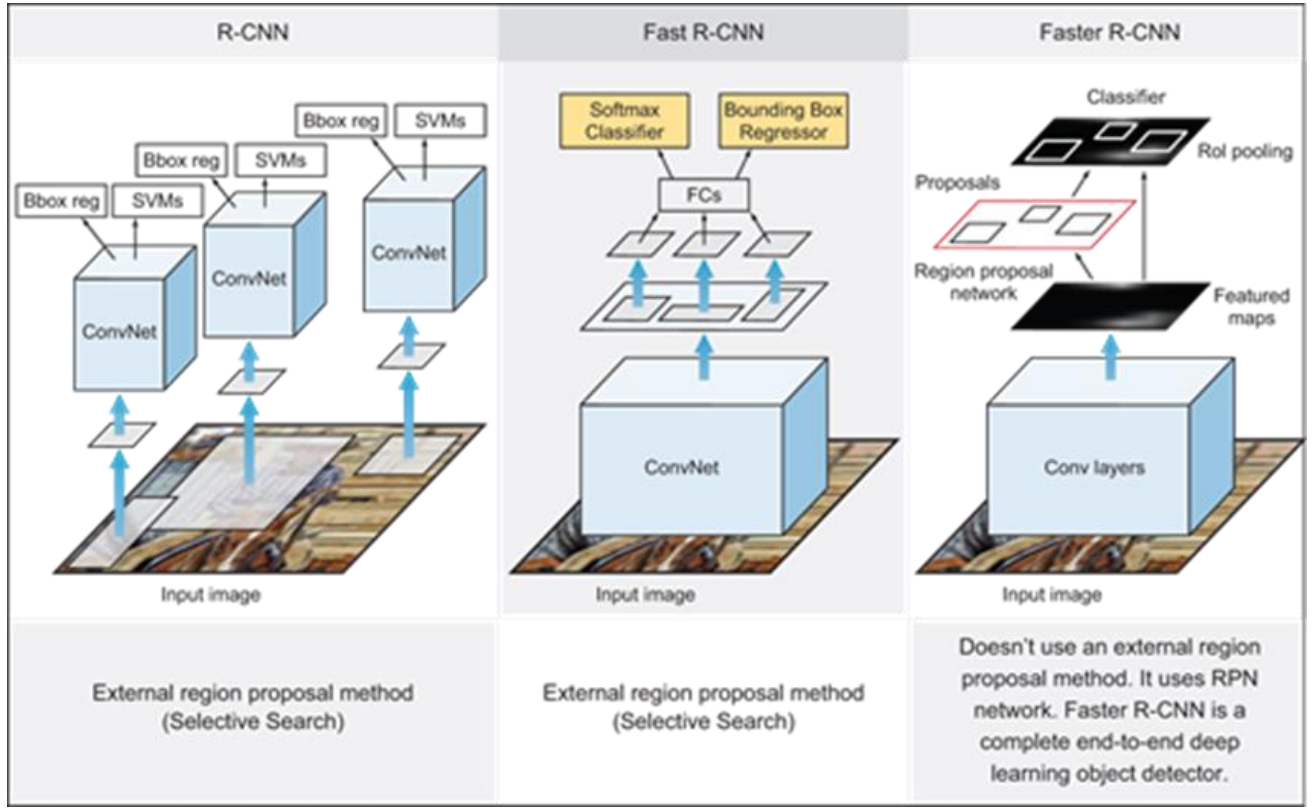
- A custom approach is taken

Removed those images with highly represented "car" class and with no under-represented class "traffic" .

```
{'biker': 2967,
 'car': 102361,
 'pedestrian': 17435,
 'trafficLight': 4121,
 'trafficLight-Green': 8720,
 'trafficLight-GreenLeft': 492,
 'trafficLight-Red': 10837,
 'trafficLight-RedLeft': 2770,
 'trafficLight-Yellow': 441,
 'trafficLight-YellowLeft': 27,
 'truck': 5773}
```

```
{'biker': 1889,
 'car': 40153,
 'pedestrian': 10733,
 'trafficLight': 4121,
 'trafficLight-Green': 8720,
 'trafficLight-GreenLeft': 492,
 'trafficLight-Red': 10837,
 'trafficLight-RedLeft': 2770,
 'trafficLight-Yellow': 441,
 'trafficLight-YellowLeft': 27,
 'truck': 2402}
```

Feasibility Study on RCNN, Fast RCNN, Faster RCNN



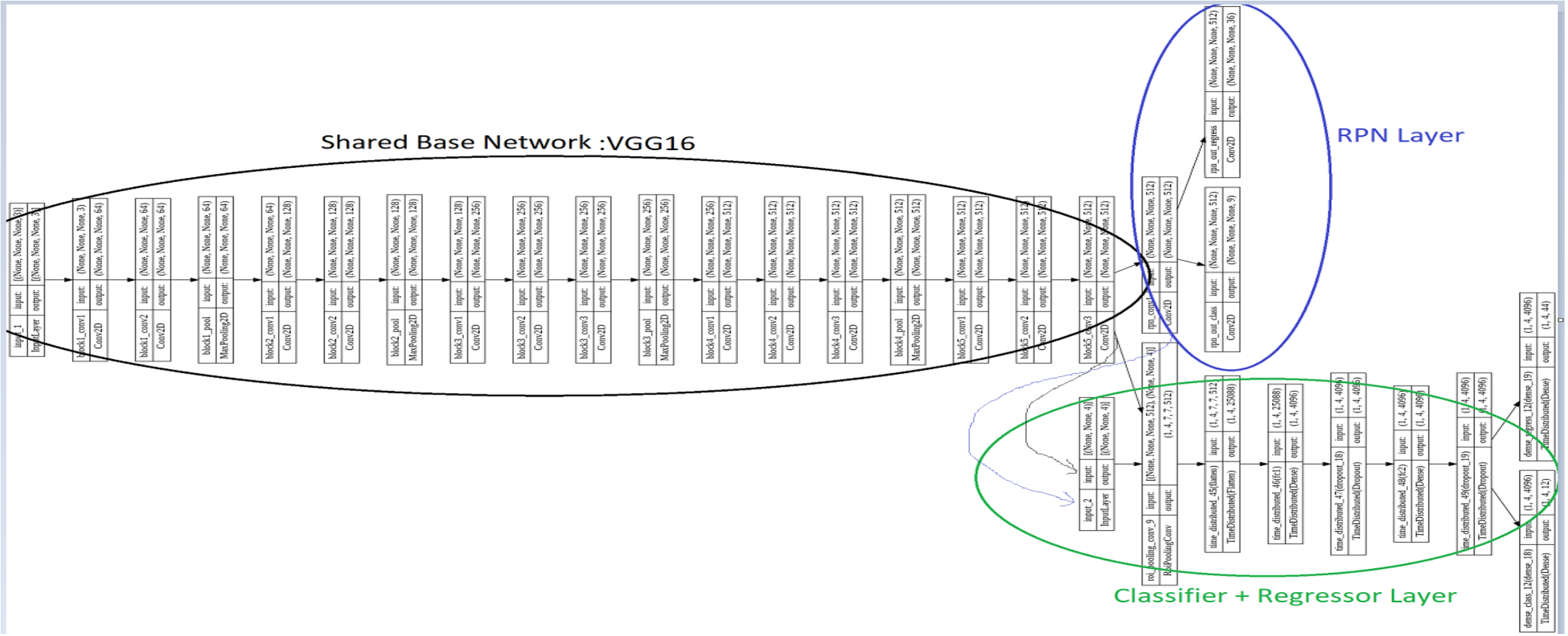
R-CNN :
Selective Search + SVMs

Fast R-CNN :
R-CNN + Regions of Interest (ROI) Pooling

Faster R-CNN :
Fast RCNN + **Regional Proposal Network (RPN)**

	R-CNN	Fast R-CNN	Faster R-CNN
Region proposals method	Selective search	Selective search	Region proposal network
Prediction timing	40-50 sec	2 seconds	0.2 seconds
Computation	High computation time	High computation time	Low computation time

Architecture



Model Parameters

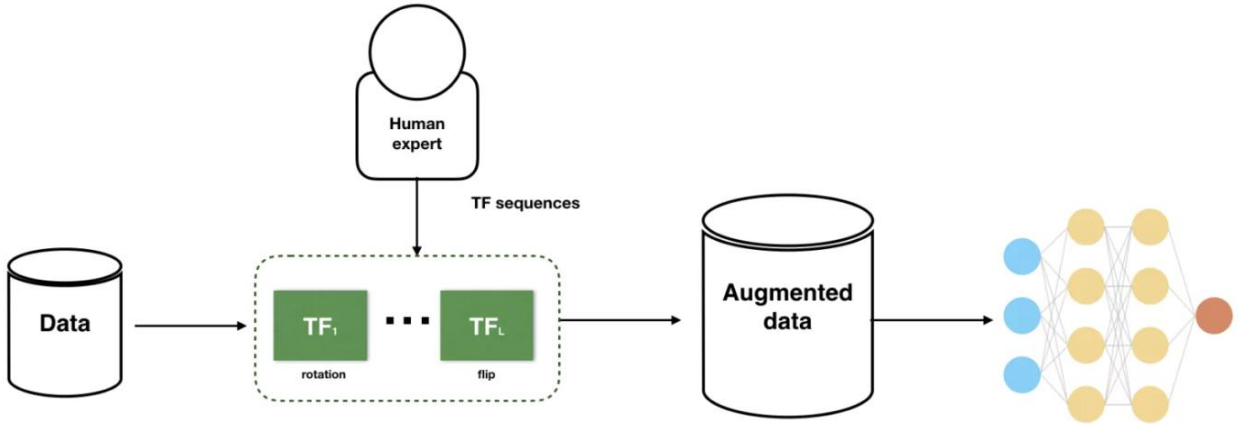
```
# overlaps for RPN
self.rpn_min_overlap = 0.3
self.rpn_max_overlap = 0.7

# overlaps for classifier ROIs
self.classifier_min_overlap = 0.1
self.classifier_max_overlap = 0.5
```

```
# number of ROIs at once
self.num_rois = 4

# stride at the RPN
self.rpn_stride = 16
```

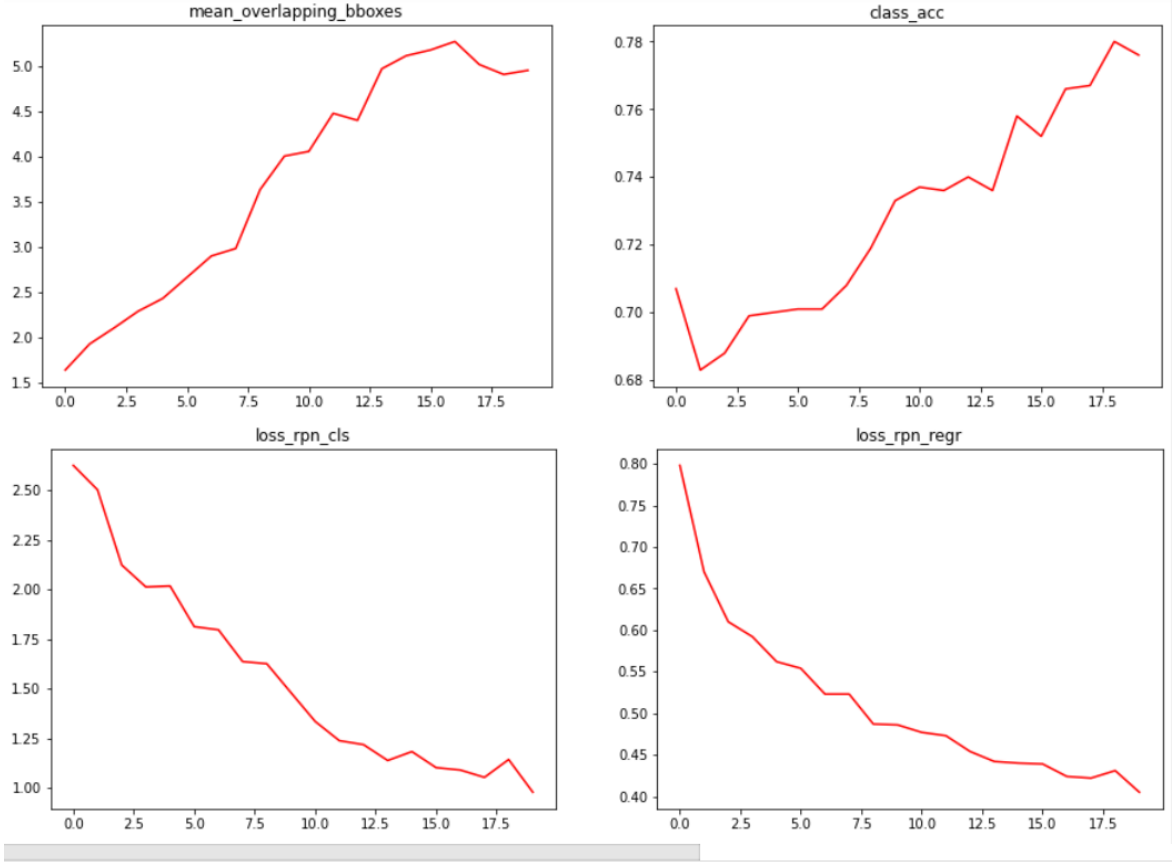
```
self.anchor_box_scales = [64, 128, 256]
```

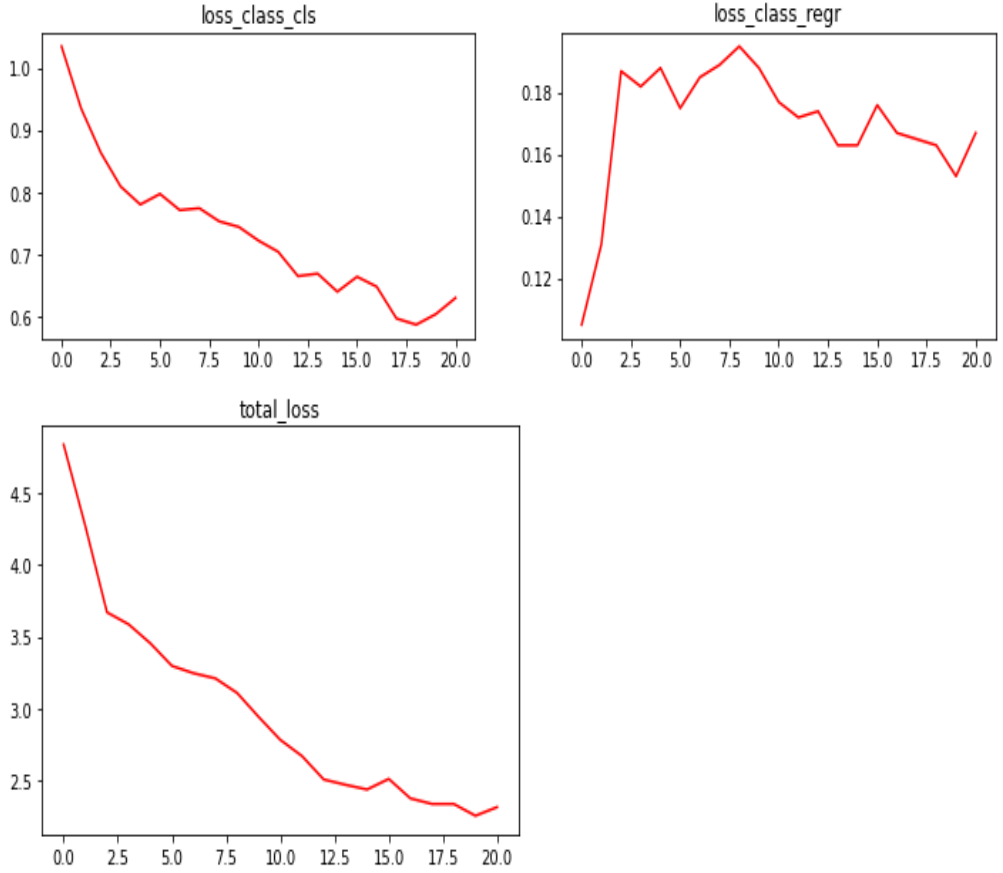
Data Augmentation

- Horizontal Flip
- Vertical Flip
- Image Rotation in the range 0 to 360 with 4 steps of 90 degree each
- Added Random noise to give blurred effect
- Used Augmented data for under-represented classes
- Varied brightness of the input image using Image Enhance.



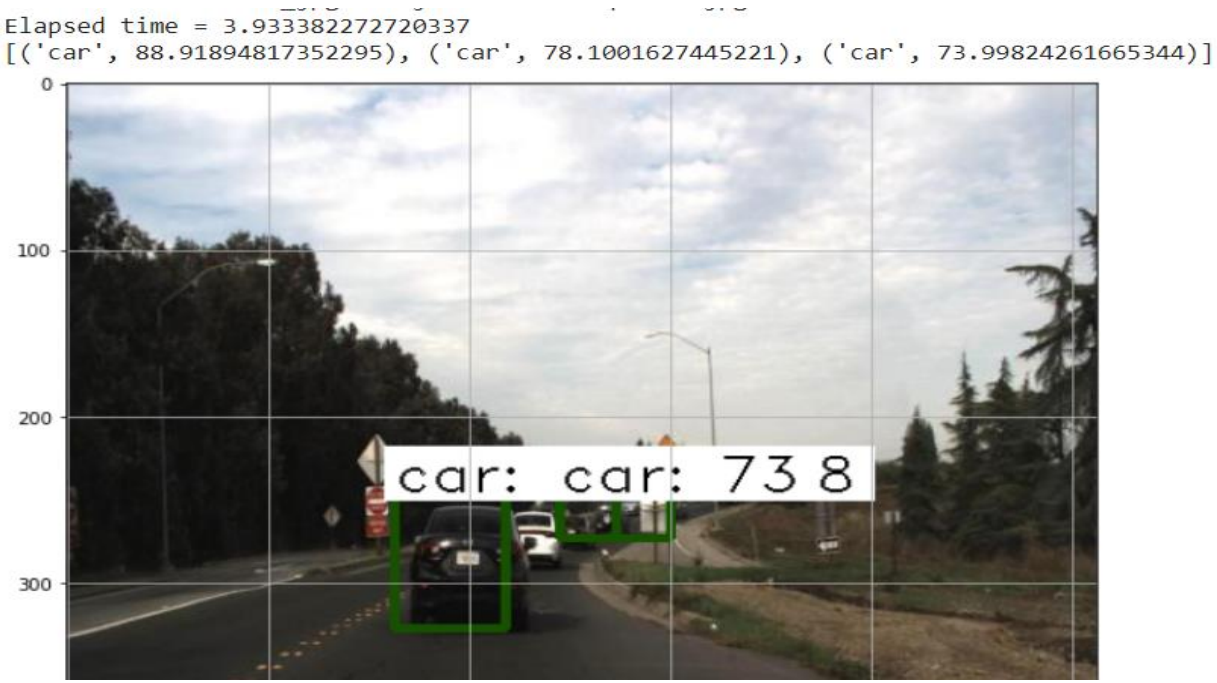
Training Results





Test Results

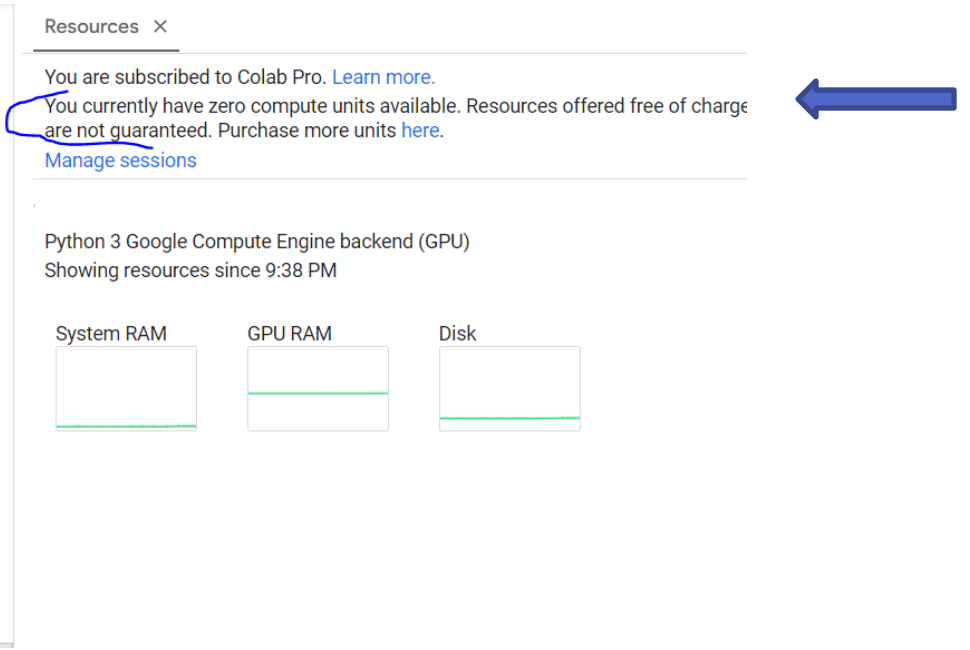
Time taken to predict was around 3-4 seconds



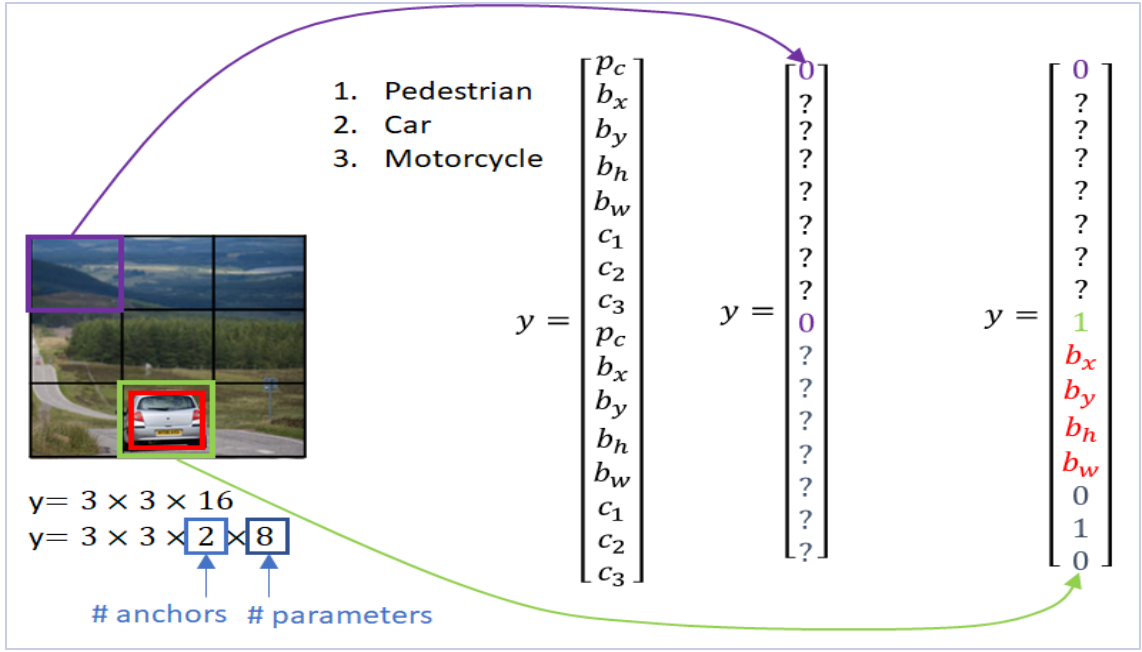
Challenges

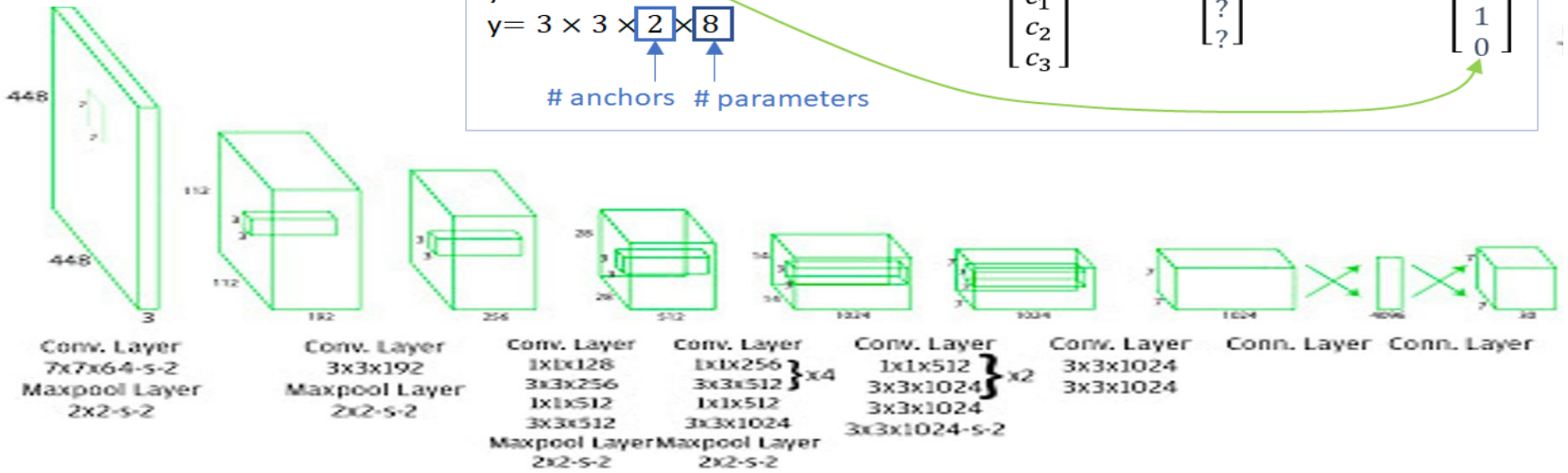
- Trained in batches of 1000 for 20 batches.
- One batch took around 4 mins on upgraded Colab compute
- Because of re source crunch, trained only 1 epoch of 20 batches

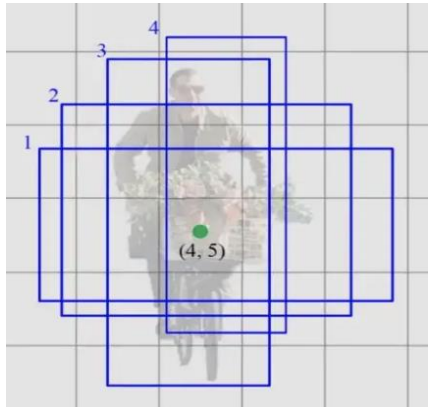
```
Loss RPN regression: 0.7976198073010892
Loss Detector classifier: 1.057278191247955
Loss Detector regression: 0.08225095948079979
Total loss: 4.562697342946934
Elapsed time: 166.37892198562622
Total loss decreased from inf to 4.562697342946934, saving weights
cp: cannot stat 'model_store/model_frcnn_vgg.hdf5': No such file or directory
cp: cannot stat 'model_store/record.csv': No such file or directory
Epoch 3/20
1000/1000 [==============================] - 176s 176ms/step - rpn_cls: 2.6135 - rpn_regr: 0.6887 - final_cls: 0
Mean number of bounding boxes from RPN overlapping ground truth boxes: 1.9248826291079812
Classifier accuracy for bounding boxes from RPN: 0.68325
Loss RPN classifier: 2.502833238922428
Loss RPN regression: 0.6702742880769074
Loss Detector classifier: 0.9543723192811012
Loss Detector regression: 0.1310768296423048
Total loss: 4.2585566759227405
Elapsed time: 177.43448448181152
Total loss decreased from 4.562697342946934 to 4.2585566759227405, saving weights
cp: cannot stat 'model_store/model_frcnn_vgg.hdf5': No such file or directory
cp: cannot stat 'model_store/record.csv': No such file or directory
Epoch 4/20
 773/1000 [======================>.......] - ETA: 37s - rpn_cls: 2.1296 - rpn_regr: 0.6462 - final_cls: 0.8427 -
```



YOLO (You only Look Once)







SSD (Multiple Boxes) +
NMS (Non Max Suppression)

Image Datasets

- PASCAL VOC 2007 (Pattern Analysis , Statistical Modelling and Computational Learning) 20 Classes, 11K training images,27K training objects
- PASCAL VOC 2012 : 20 categories
- ImageNet: 14 million images : 21,841 classes
- ImageNet ILSVRC 2013 : 200 Classes, 476K training images, 543 training objects
- Microsoft COCO 2015 (Common Objects in Context) : 80-90 classes and 3.30 lakh images out of 2lakh are labelled
- Google Images : 9 Million images , 600 classes
- KITTI

Training snapshots

```
import numpy as np

file = open("export/train_annotations.txt", 'r')

#Get dimensions of each of the bounding box in the format (width, height)
boxes = []
for line in file:
  dim_info = line.split(" ")
  length = len(dim_info)
  #ignore the image path. Start from the first bb
  for i in range(1, length):
    # Screen indexing starts at pixel 0,0 and ends at width-1,height-1.
    # The range of screen coordinates is inclusive:inclusive.
    # So we need to add 1

    #width = xmax- xmin + 1
    width = int(dim_info[i].split(",")[2]) - int(dim_info[i].split(",")[0]) + 1
    #height = ymax - ymin + 1
    height = int(dim_info[i].split(",")[3]) - int(dim_info[i].split(",")[1]) + 1
    boxes.append([width, height])

boxes = np.array(boxes)
file.close()

total_boxes = boxes.shape[0]
total_clusters = 9

def get_iou (boxes, center_boxes, total_clusters):
  n = boxes.shape[0] #total number of boxes
  k = total_clusters

  #calculate area of each box; width * height
  b_a = boxes[:, 0] * boxes[:, 1]
  # repeat k times to use with respect to each selected center box
  b_a = b_a.repeat(k)
  # Reshape to n x k matrix. (AB - Area of Box)
  # [ AB1 AB1 AB1 ...]
  # [ ...         ...]
  # [ ABn ABn ABn ...]
  b_a = np.reshape(b_a, (n, k))
```

```
def get_classes(classes_path):
    '''loads the classes'''
    with open(classes_path) as f:
        class_names = f.readlines()
    class_names = [c.strip() for c in class_names]
    return class_names

def get_anchors(anchors_path):
    '''loads the anchors from a file'''
    with open(anchors_path) as f:
        anchors = f.readline()
    anchors = [float(x) for x in anchors.split(',')]
    return np.array(anchors).reshape(-1, 2)
```

```
def box_iou(b1, b2):

    b1 = K.expand_dims(b1, -2)
    b1_xy = b1[..., :2]
    b1_wh = b1[..., 2:4]
    b1_wh_half = b1_wh/2.
    b1_mins = b1_xy - b1_wh_half
    b1_maxes = b1_xy + b1_wh_half

    b2 = K.expand_dims(b2, 0)
    b2_xy = b2[..., :2]
    b2_wh = b2[..., 2:4]
    b2_wh_half = b2_wh/2.
    b2_mins = b2_xy - b2_wh_half
    b2_maxes = b2_xy + b2_wh_half

    intersect_mins = K.maximum(b1_mins, b2_mins)
    intersect_maxes = K.minimum(b1_maxes, b2_maxes)
    intersect_wh = K.maximum(intersect_maxes - intersect_mins, 0.)
    intersect_area = intersect_wh[..., 0] * intersect_wh[..., 1]
    b1_area = b1_wh[..., 0] * b1_wh[..., 1]
    b2_area = b2_wh[..., 0] * b2_wh[..., 1]
    iou = intersect_area / (b1_area + b2_area - intersect_area)

    return iou
```

```
def yolo4_loss(args, anchors, num_classes, ignore_thresh=.5, print_loss=False, ):

    num_layers = len(anchors)//3 # default setting
    yolo_outputs = args[:num_layers]
    y_true = args[num_layers:]
    anchor_mask = [[6,7,8], [3,4,5], [0,1,2]] if num_layers==3 else [[3,4,5], [1,2,3]]
    input_shape = K.cast(K.shape(yolo_outputs[0])[1:3] * 32, K.dtype(y_true[0]))
    grid_shapes = [K.cast(K.shape(yolo_outputs[l])[1:3], K.dtype(y_true[0])) for l in
    loss = 0
    m = K.shape(yolo_outputs[0])[0] # batch size, tensor
```

Training snapshots

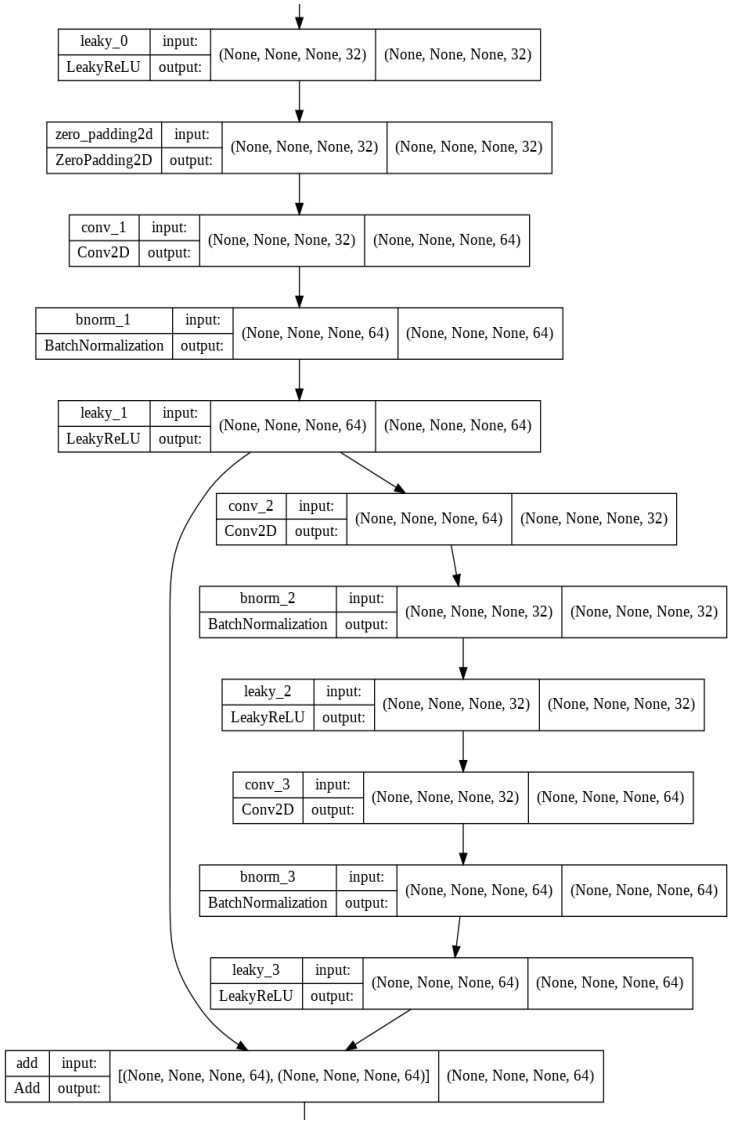
```
        image_data = np.array(image_data)
        box_data = np.array(box_data)
        y_true = preprocess_true_boxes(box_data, input_shape, anchors, num_classes)
        yield [image_data, *y_true], np.zeros(batch_size)
        #return image_data, y_true

def data_generator_wrapper(image_dir, annotation_lines, batch_size, input_shape, anchors, num_classes):
    n = len(annotation_lines)
    if n==0 or batch_size<=0: return None
    return data_generator(image_dir, annotation_lines, batch_size, input_shape, anchors, num_classes)
```

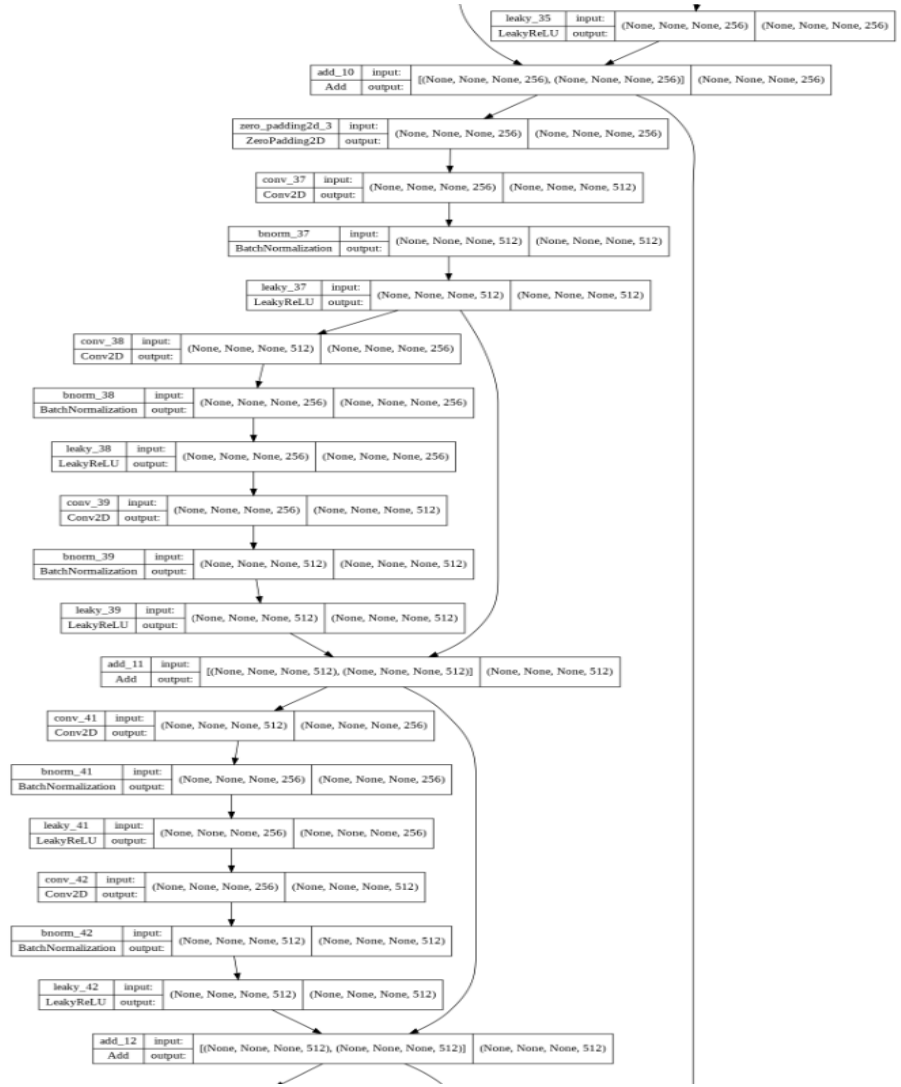
```
#annotation_path = 'export/_annotations.txt'  # path to Roboflow data annotations
classes_path = 'export/_classes.txt'         # path to Roboflow class names
anchors_path = 'anchors.txt'
class_names = get_classes(classes_path)
print("CLASS NAMES: " + str(class_names))
num_classes = len(class_names)
anchors = get_anchors(anchors_path)
print("ANCHORS: " + str(anchors))
print(anchors[0])
num_anchors = len(anchors)
```

```
CLASS NAMES: ['car', 'truck', 'bus', 'train', 'person', 'motorbike', 'boat', 'bic
ANCHORS: [[  9.  22.]
 [ 13.  39.]
 [ 18.  24.]
 [ 21.  68.]
 [ 27.  32.]
 [ 39.  41.]
 [ 52.  60.]
 [ 78.  97.]
 [144. 206.]]
[ 9. 22.]
```

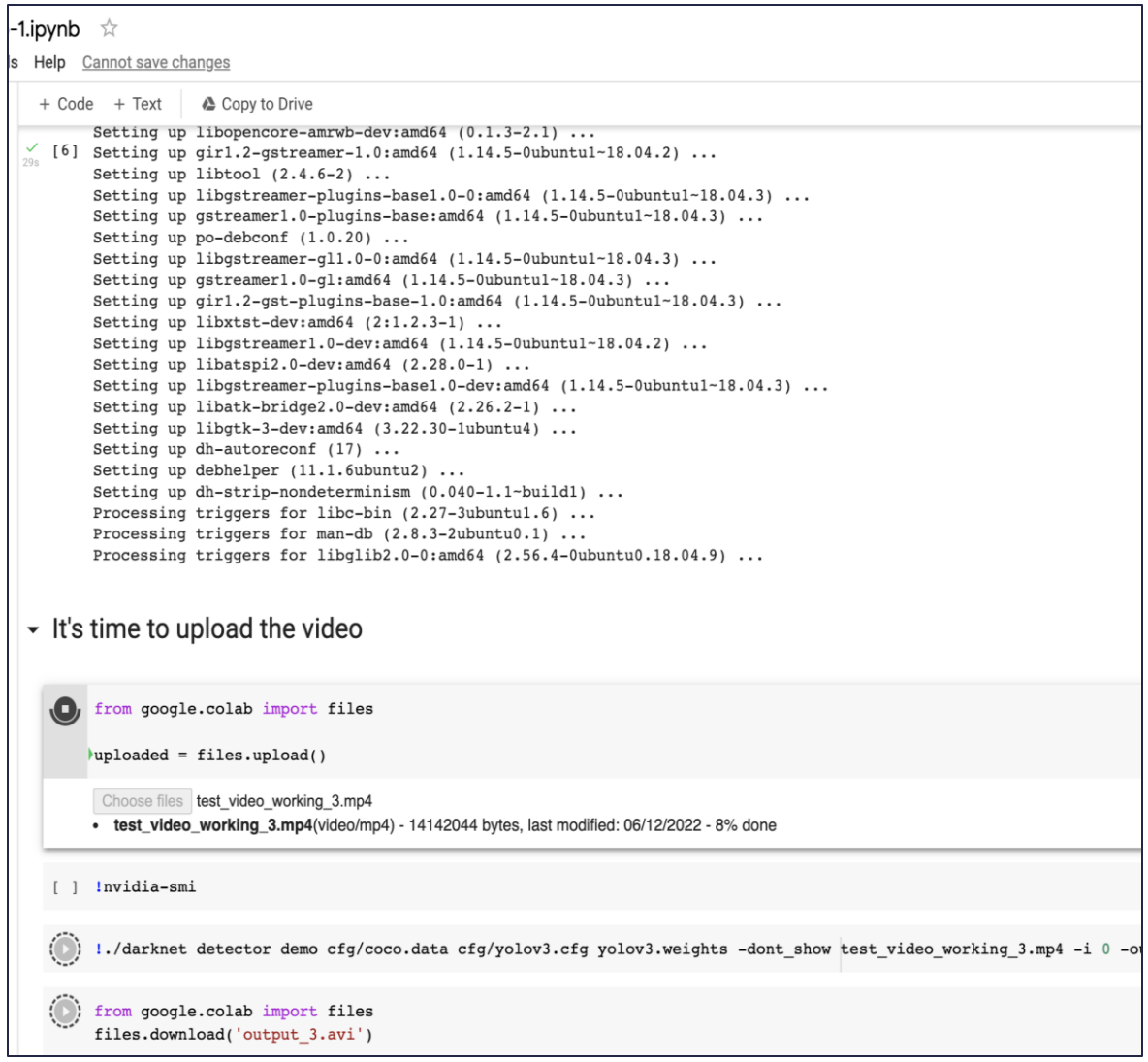
```
def make_yolov3_model(n_anchors, n_classes):

    input_image = Input(shape=(None, None, 3))
    #input_image = Input(shape=(512, 512, 3))

    # Layer  0 => 4
    x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': Tru
                                  {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': Tru
                                  {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': Tru
                                  {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': Tru
    # Layer  5 => 8
    x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky
                        {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky
    # Layer  9 => 11
    x = _conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky
    # Layer 12 => 15
    x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky
                        {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky
                        {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky
    # Layer 16 => 36
    for i in range(7):
        x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'l
                            {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'l
    skip_36 = x
    # Layer 37 => 40
    x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky
    # Layer 41 => 61
    for i in range(7):
        x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'l
                            {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'l
    skip_61 = x
    # Layer 62 => 65
    x = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leak
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leak
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leak
```

```
model_custom.compile(optimizer=Adam(learning_rate=1e-3), loss={'yolo3_loss': lambda y_true, y_pred: y_pred})
plot_model(model_custom, show_shapes=True)
```





Test & Validation



--batch 16 --epochs 50

train_split = 0.6
val_split = 0.2
test_split. = 0.2

```
from google.colab import files

uploaded = files.upload()
```

Choose files test_video_working_3.mp4

- **test_video_working_3.mp4**(video/mp4) - 14142044 bytes, last modified: 06/12/2022 - 74% done

```
[ ] !nvidia-smi
```

```
!./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights -dont_show te
```

```
from google.colab import files
files.download('output_3.avi')
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 608, 608, 3)	0	
conv2d_1 (Conv2D)	(None, 608, 608, 32)	864	input_1[0][0]
batch_normalization_1 (BatchNorm	(None, 608, 608, 32)	128	conv2d_1[0][0]
leaky_re_lu_1 (LeakyReLU)	(None, 608, 608, 32)	0	batch_normalization_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 304, 304, 32)	0	leaky_re_lu_1[0][0]
conv2d_2 (Conv2D)	(None, 304, 304, 64)	18432	max_pooling2d_1[0][0]
batch_normalization_2 (BatchNorm	(None, 304, 304, 64)	256	conv2d_2[0][0]
leaky_re_lu_2 (LeakyReLU)	(None, 304, 304, 64)	0	batch_normalization_2[0][0]

....

.....

......

leaky_re_lu_20 (LeakyReLU)	(None, 19, 19, 1024)	0	batch_normalization_20[0][0]
concatenate_1 (Concatenate)	(None, 19, 19, 1280)	0	space_to_depth_x2[0][0] leaky_re_lu_20[0][0]
conv2d_22 (Conv2D)	(None, 19, 19, 1024)	11796480	concatenate_1[0][0]
batch_normalization_22 (BatchNor	(None, 19, 19, 1024)	4096	conv2d_22[0][0]
leaky_re_lu_22 (LeakyReLU)	(None, 19, 19, 1024)	0	batch_normalization_22[0][0]
conv2d_23 (Conv2D)	(None, 19, 19, 425)	435625	leaky_re_lu_22[0][0]

Total params: 50,983,561
Trainable params: 50,962,889
Non-trainable params: 20,672

yolo_object_detection_video_bkp.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[11] 24s

```
50 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
51 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
52 Shortcut Layer: 49,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
53 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
54 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
55 Shortcut Layer: 52,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
56 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
57 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
58 Shortcut Layer: 55,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
59 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
60 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
61 Shortcut Layer: 58,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
62 conv   1024       3 x 3/ 2     26 x  26 x 512 ->   13 x  13 x1024 1.595 BF
63 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
64 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
65 Shortcut Layer: 62,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
66 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
67 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
68 Shortcut Layer: 65,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
69 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
70 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
71 Shortcut Layer: 68,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
72 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
73 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
74 Shortcut Layer: 71,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
75 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
76 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
77 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
```

0s

```
from google.colab import files
files.download('output_2.mp4')
```

Downloading "output_2.mp4":

```
+ Code + Text  Copy to Drive

person: 74%

FPS:21.0        AVG_FPS:19.5

 cvWriteFrame
Objects:


FPS:20.9        AVG_FPS:19.5

 cvWriteFrame
Objects:


FPS:21.0        AVG_FPS:19.5

 cvWriteFrame
Objects:

truck: 93%

FPS:20.9        AVG_FPS:19.5

 cvWriteFrame
Objects:

truck: 88%
person: 70%

FPS:20.9        AVG_FPS:19.5

 cvWriteFrame
Objects:

truck: 86%
person: 71%

FPS:20.9        AVG_FPS:19.5

 cvWriteFrame
Objects:
```

```
[9] from google.colab import files
    files.download('output_3.avi')
```

Downloading "output_3.avi":

```
options = { 'model': 'cfg/yolo.cfg', 'load': 'bin/yolov4.weights',
            'threshold': 0.5}
tfnet = TFNet(options)
cap = cv2.VideoCapture(test_video_working_2.mp4)
colors=[tuple(255 * np.random.rand(3)) for i in range(5)]
while(cap.isOpened()):
    stime= time.time()
    ret, frame = cap.read()
    results = tfnet.return_predict(frame)
    if ret:
        for color, result in zip(colors, results):
            tl = (result['topleft']['x'], result['topleft']['y'])
            br = (result['bottomright']['x'], result['bottomright']['y'])
            label = result['label']
            frame= cv2.rectangle(frame, tl, br, color, 7)
            frame= cv2.putText(frame, label, tl, cv2.FONT_HERSHEY_TRIPLEX,
        cv2.imshow('frame', frame)
        print('FPS {:1f}'.format(1/(time.time() -stime)))
        if cv2.waitKey(1)  & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
cv2.destroyAllWindows()
```

```
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   55C    P0    25W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

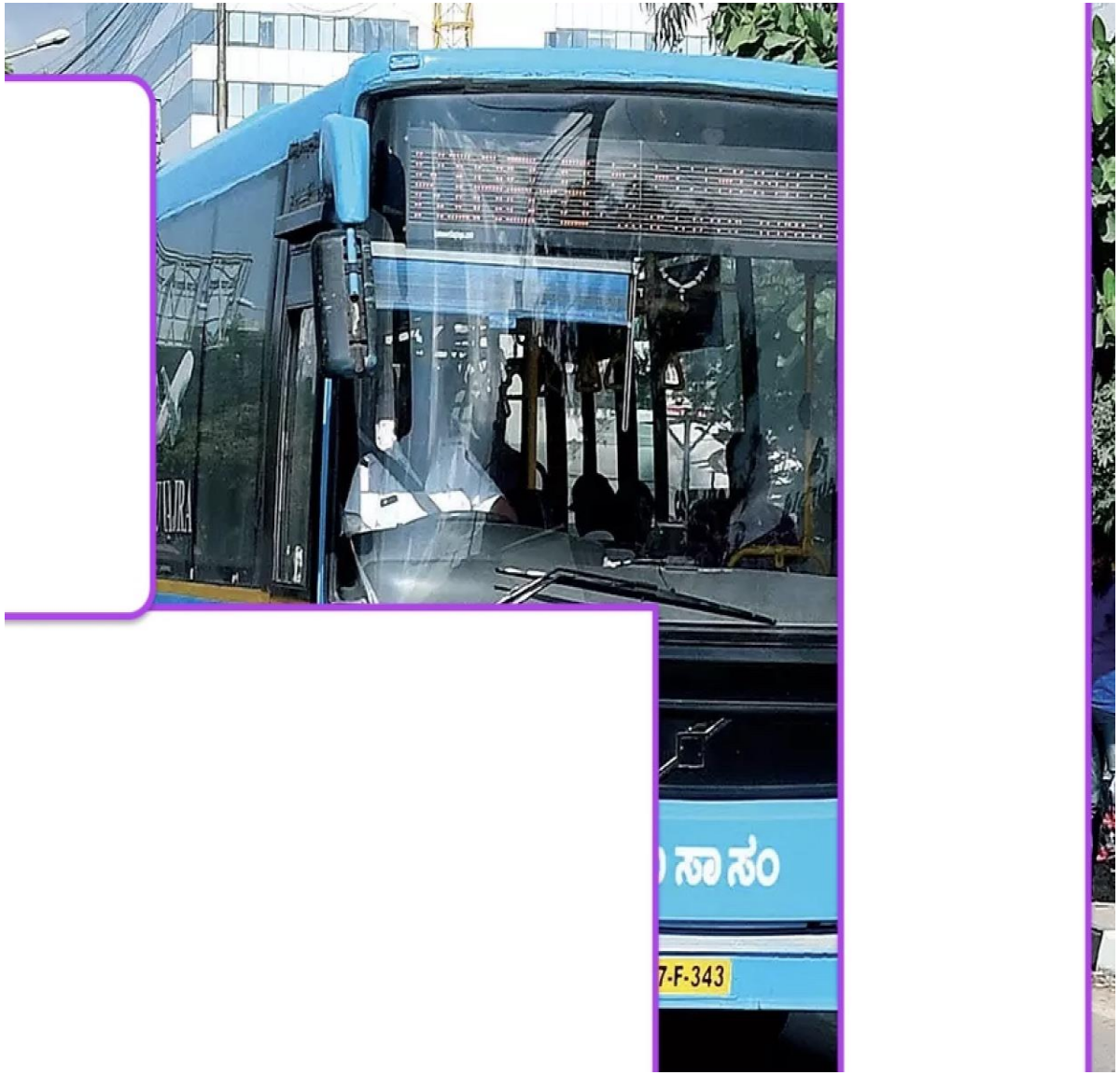
```
[8] !./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights -dont_show test_vid
```

```
bus: 100%
car: 99%
car: 96%
car: 90%
person: 83%
person: 81%

FPS:11.9        AVG_FPS:18.2

 cvWriteFrame
Objects:

bus: 100%
car: 99%
car: 95%
car: 86%
car: 70%
person: 87%
person: 82%
person: 74%

FPS:11.5        AVG_FPS:18.2

 cvWriteFrame
Objects:

bus: 100%
car: 99%
car: 90%
car: 83%
person: 94%

FPS:11.1        AVG_FPS:18.2

 cvWriteFrame
```

```
[9] from google.colab import files
    files.download('output_3.avi')
```

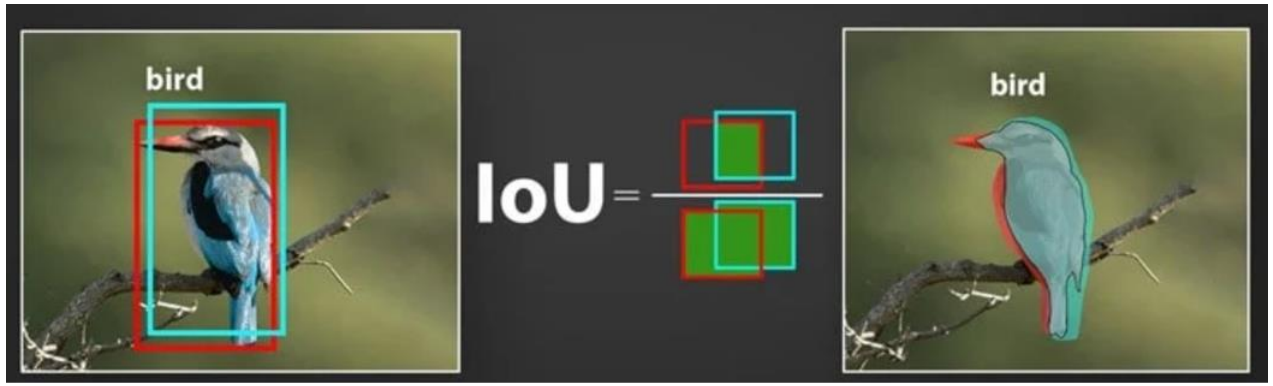
Downloading "output_3.avi":

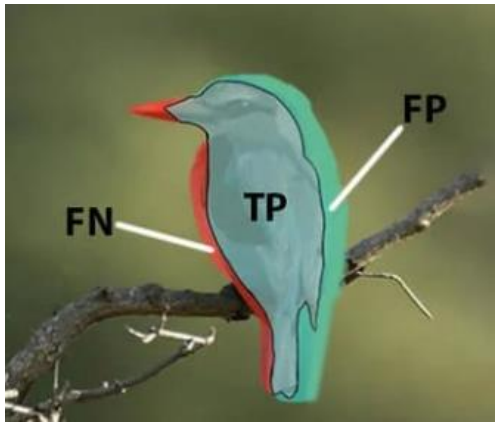

bus: [95.39]
BMTC
ಬೆಂಮಸಾಸಂ



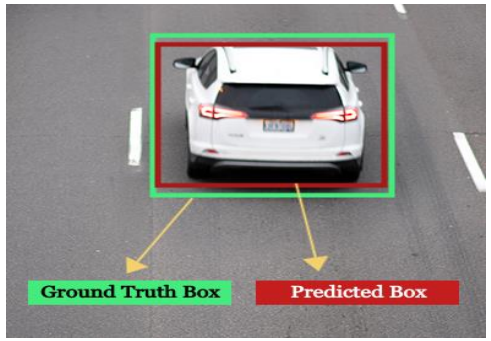
Evaluation criteria

The Accuracy metrics, of
- Creating bounding box against an object
- Identifying the class of the object
- Immune to external factors like brightness and rain/snow fall.
- How behave with rotated/croped objects
- Behave with unseen data





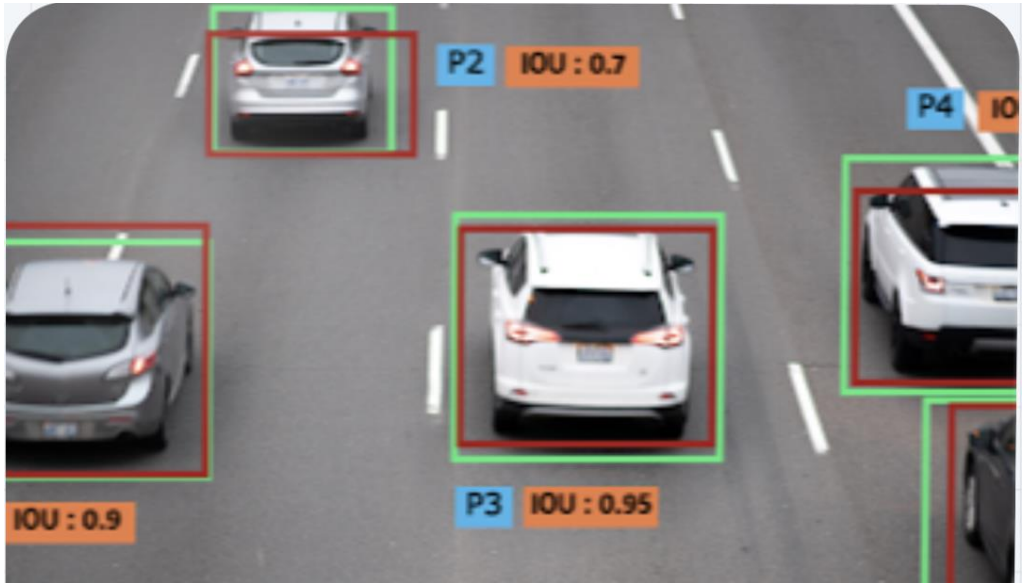
MAP (mean-average-precision of training & validation)

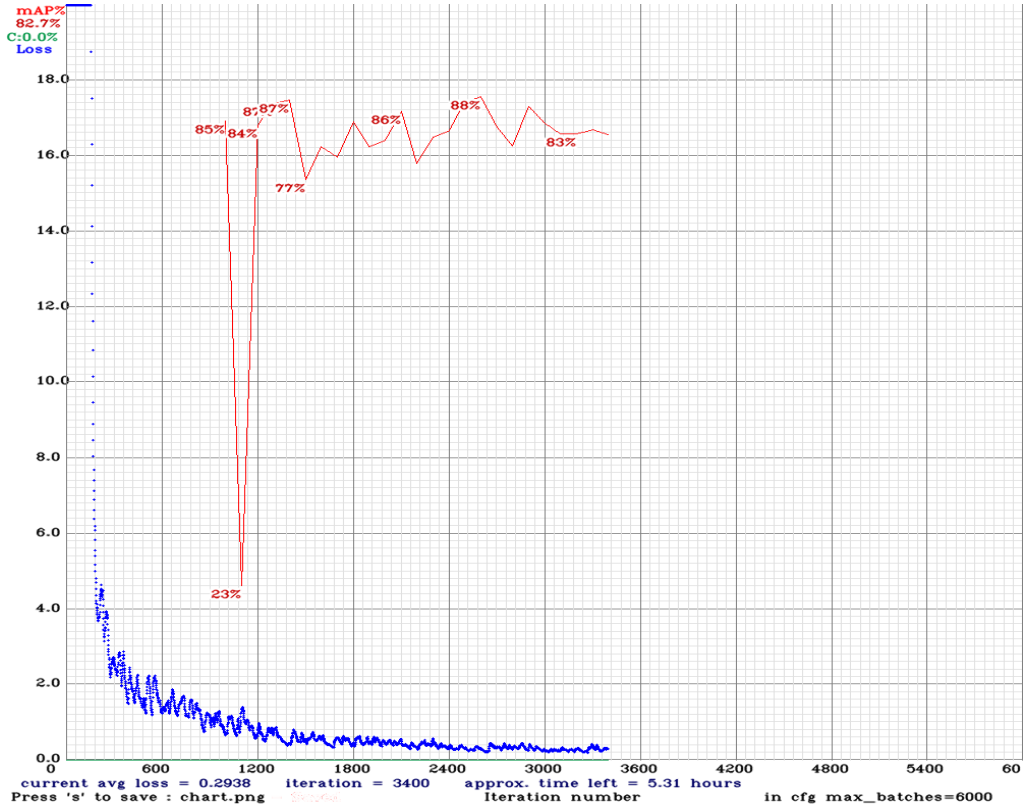


$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

$AP_k =$ *the AP of class k*

$n =$ *the number of classes*

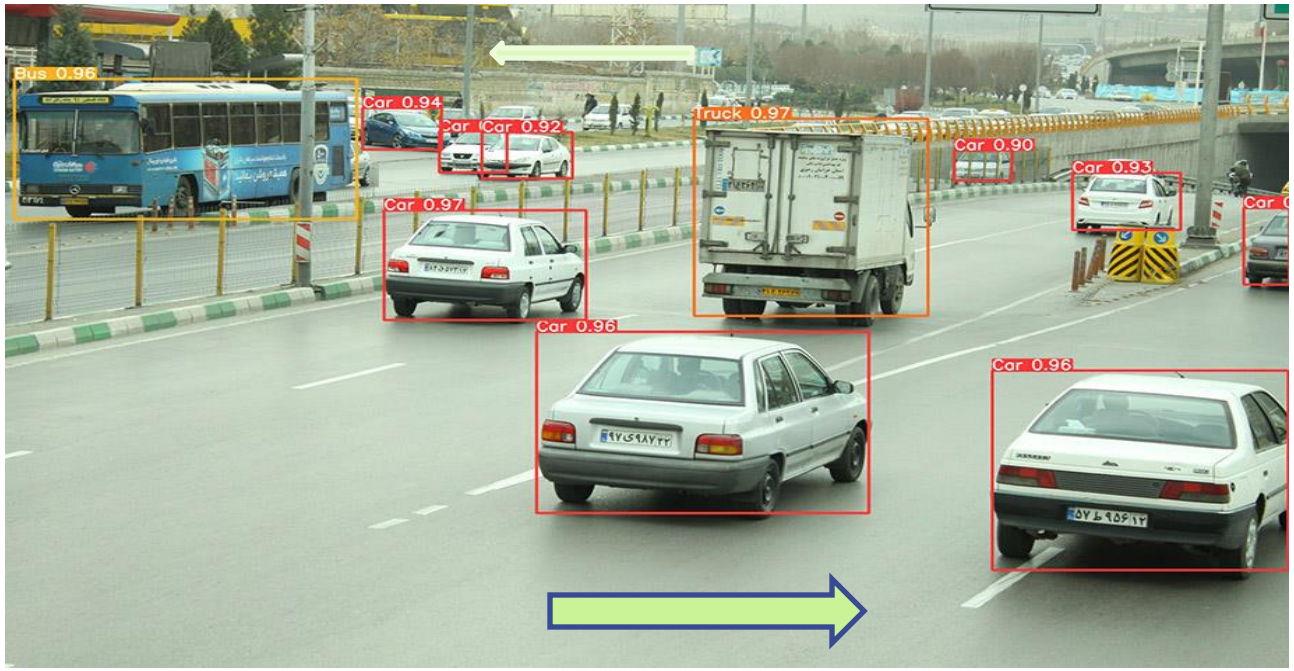


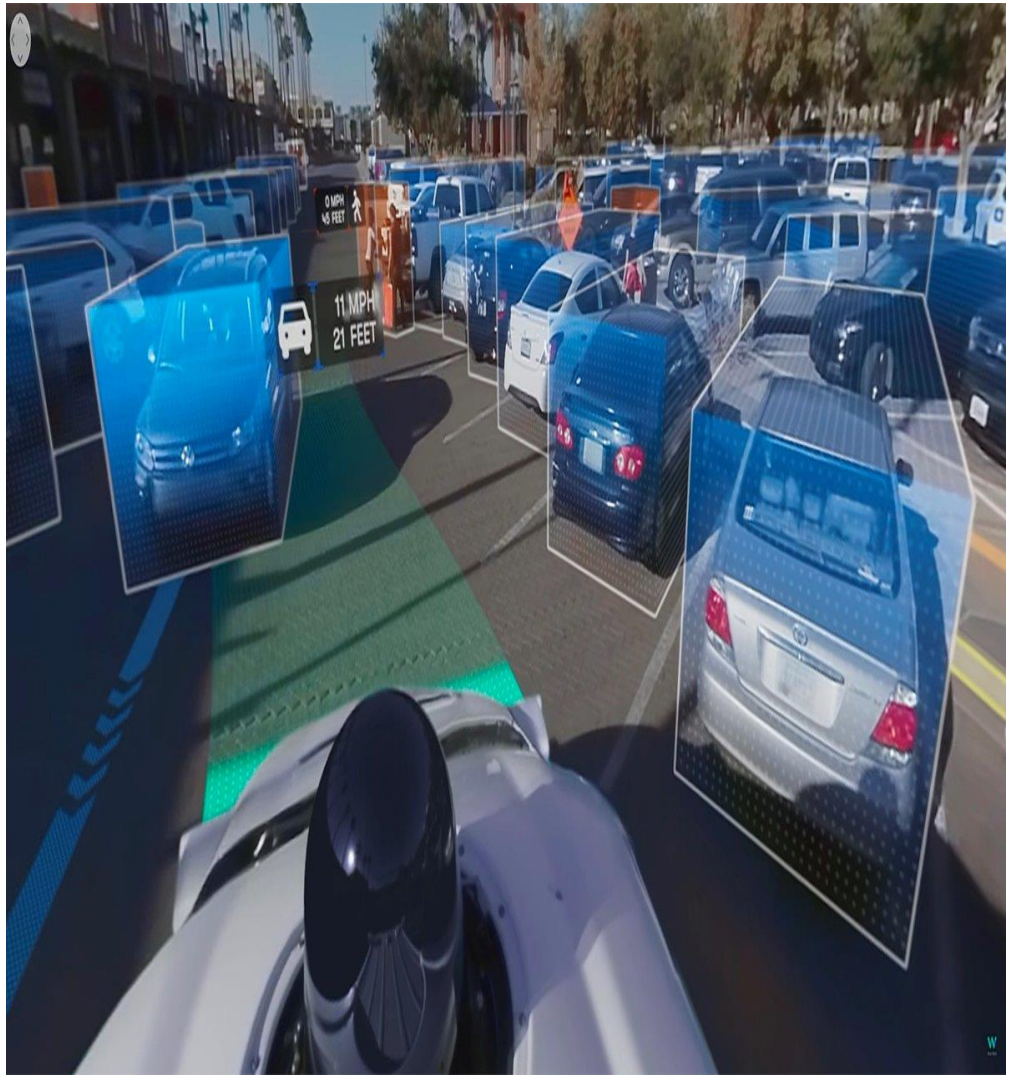


blue curve is the training loss or the error on the training dataset (Complete Intersection-Over-Union loss).
red line is the mean average precision at 50% Intersection-over-Union threshold (mAP@0.5), which checks if model it is generalizing well on a *never-before-seen dataset* or *validation set.*

Future work

- Develop the model
- Test with different scenarios
- Add deterrence to the model
- Invariant to external factors





Questions & Feedback

Thank You

arkadipbasu@iisc.ac.in
kumarg@iisc.ac.in