Machine Learning Engineer Nanodegree

Capstone Project

Chandana Neerukonda
October 17 2017

I. Definition

Project Overview:

Autonomous Vehicles are one of the most in vogue today.Autonomous Driving touches many challenges in different fields, especially computer vision and machine learning. One of the most talked is pedestrian detection.

Pedestrian detection systems may be possible to implement without the use of spinning light detection and ranging (LIDAR) devices, which can often cost more than $10,000. Instead, cheap video cameras may be used for the same purpose of recognizing and tracking nearby pedestrians.[1] Research Papers show impressive results using transfer learning but the most common errors for detectors are as follows:

- Detecting tree leaves or traffic lights in background as pedestrian
- Detecting the same person twice
- Not detecting small persons

Other approaches like [2] use a real-time algorithm for the detection and trackcinf of image regions that possibly contain pedestrians. It then uses a classification algorithm based on the typical motion patterns of a pedestrian's legs. These approaches assume the visibility of the legs and works for walking people only!

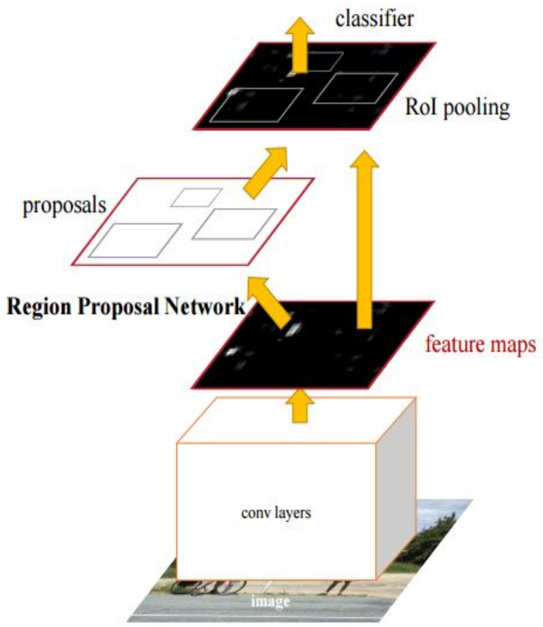
There has been a lot of work done on Caltech Dataset[3] which is considerably the largest Pedestrian dataset.

Problem statement:

The problem is Pedestrian Detection on Caltech data set. I am personally motivated to explore the implementation of mobile nets for pedestrian detection to have this feature on an Android mobile. I believe this problem is solvable because of the new advances including SSD, YOLO and Faster RCNN and the support from Tensor flow for object detection => TFDetect (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android)

Faster RCNN uses a Region Proposal network with anchor boxes to get an estimate of the possible bounding boxes and then regresses over those with the least error. It also uses the same

Faster RCNN architecture in the figure below.



.

feature maps for classification too

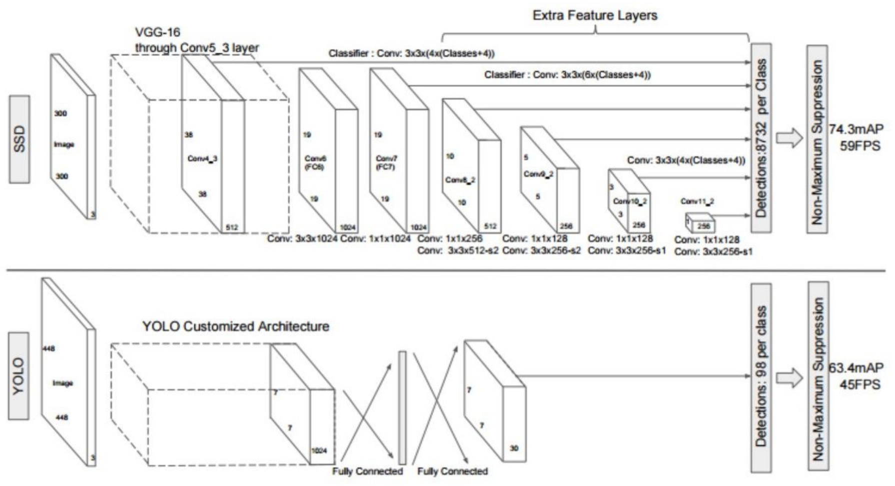


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300 × 300 input size significantly outperforms its 448 × 448 YOLO counterpart in accuracy on VOC2007 `test` while also improving the speed.

The above image from SSD paper explains how the single shot mechanisms of YOLO and SSD differ.

Evaluation Metrics:

The Evaluation metric I propose would be the mAP value at an intersection-over-union greater than 0.5.

For detection, a common way to determine if one object proposal was right is *Intersection over Union* (IoU, IU). This takes the set A of proposed object pixels and the set of true object pixels B and calculates:

$$IoU(A,B)=A\cap B\ /\ A\cup B$$

Commonly, IoU > 0.5 means that it was a hit, otherwise it was a fail. For each class, one can calculate the

- True Positive ($TP(c)$): a proposal was made for class c and there actually was an object of class c

- False Positive ($FP(c)$): a proposal was made for class c, but there is no object of class c
- Average Precision for class c:#TP(c)/(#TP(c) + #FP(c))
- The mAP (mean average precision) = $\frac{1}{|classes|} \sum_{c \varepsilon classes} \frac{\#TP(c)}{\#TP(c) + \#FP(c)}$

If one wants better proposals, one does increase the IoU from 0.5 to a higher value (up to 1.0 which would be perfect). One can denote this with mAP@p, where $p \in (0,1)$p∈(0,1) is the IoU.

IoU is considered a reasonable evaluation metric in the scenario of object detection.

II. Analysis

Data Exploration:

Calltech Dataset (http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/)

The Caltech Pedestrian Dataset consists of approximately 10 hours of 640x480 30Hz video taken from a vehicle driving through regular traffic in an urban environment. About 250,000 frames (in 137 approximately minute long segments) with a total of 350,000 bounding boxes and 2300 unique pedestrians were annotated. The annotation includes temporal correspondence between bounding boxes and detailed occlusion labels.

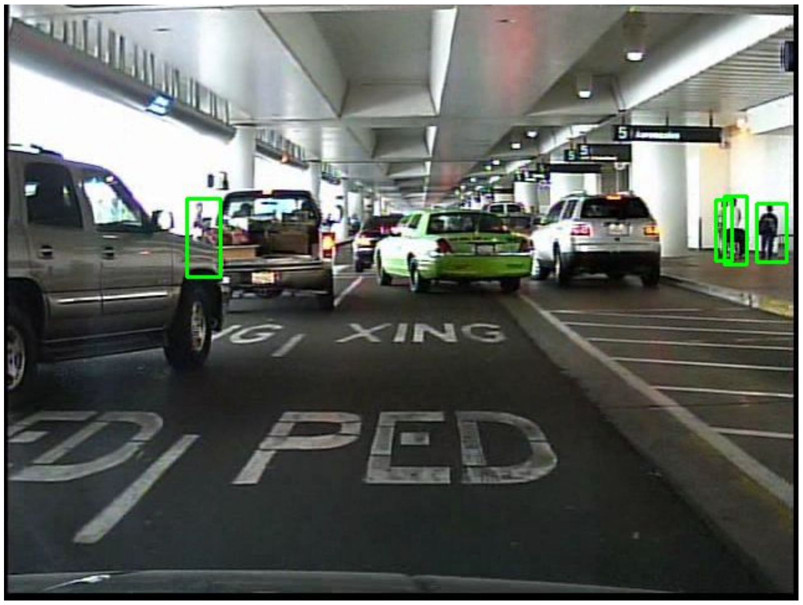
The training data (set00-set05) consists of six training sets (~1GB each) (six different video files with annotations), each with 6-13 one-minute long seq files, along with all annotation information . The testing data (set06-set10) consists of five sets (five different video files with annotations), again ~1GB each. Annotations for the entire dataset are also provided.

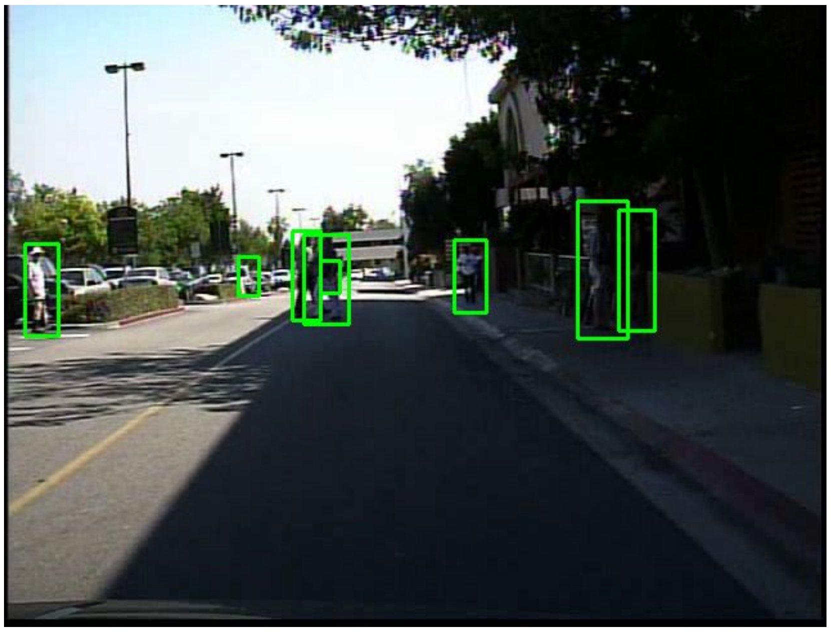
An seq file is a series of concatenated image frames with a fixed size header. Matlab routines for reading/writing/manipulating seq files can be found in Piotr's Matlab Toolbox (version 3.20 or later required). These routines can also be used to extract an seq file to a directory of images.

A bounding box is a set of coordinates describing a rectangular box around the object of interest. Look at the figures in the **Exploratory Visualization** section to get an idea of the boundary box**.**

Exploratory Visualization:

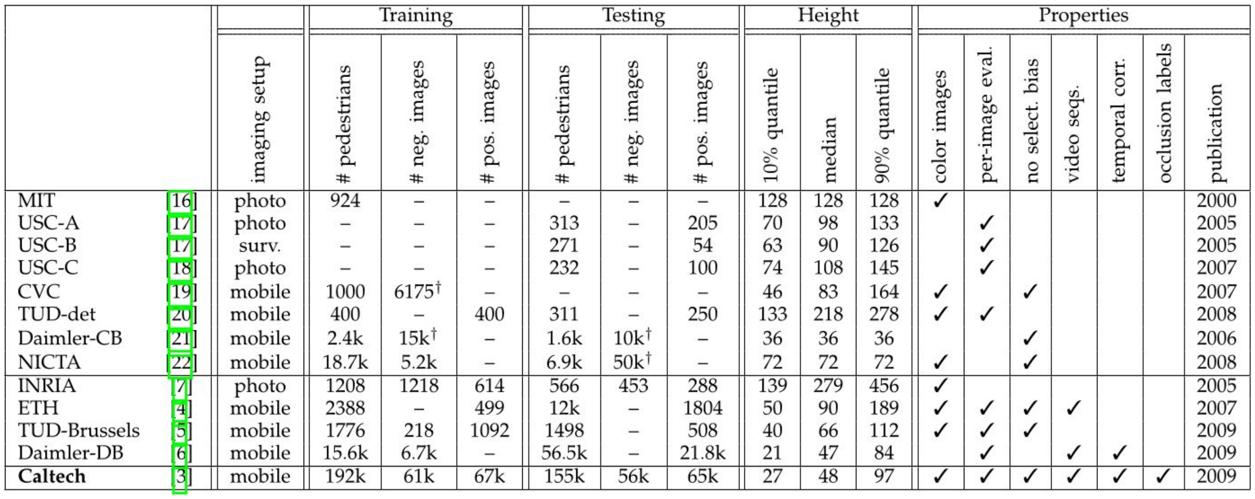
I plotted the ground truth bounding boxes on the images to do a sanity-check on the bounding box parameters.





I see the existence of multiple boxes because I mixed the labels for both person and people classes.

The following table is taken from Piotr Dollar et. al. ‘s paper on the state of art pedestrian detectors which indicates that the CalTech data set is the largest and the most diverse pedestrian dataset available by the time the paper got published.



			Training			Testing			Height			Properties						
		imaging setup	# pedestrians	# neg. images	# pos. images	# pedestrians	# neg. images	# pos. images	10% quantile	median	90% quantile	color images	per-image eval.	no select. bias	video seqs.	temporal corr.	occlusion labels	publication
MIT	[16]	photo	924	–	–	–	–	–	128	128	128	✓						2000
USC-A	[17]	photo	–	–	–	313	–	205	70	98	133		✓					2005
USC-B	[17]	surv.	–	–	–	271	–	54	63	90	126		✓					2005
USC-C	[18]	photo	–	–	–	232	–	100	74	108	145		✓					2007
CVC	[19]	mobile	1000	6175†	–	–	–	–	46	83	164	✓		✓				2007
TUD-det	[20]	mobile	400	–	400	311	–	250	133	218	278	✓	✓					2008
Daimler-CB	[21]	mobile	2.4k	15k†	–	1.6k	10k†	–	36	36	36			✓				2006
NICTA	[22]	mobile	18.7k	5.2k	–	6.9k	50k†	–	72	72	72	✓		✓				2008
INRIA	[7]	photo	1208	1218	614	566	453	288	139	279	456	✓						2005
ETH	[4]	mobile	2388	–	499	12k	–	1804	50	90	189	✓	✓	✓	✓			2007
TUD-Brussels	[5]	mobile	1776	218	1092	1498	–	508	40	66	112	✓	✓	✓				2009
Daimler-DB	[6]	mobile	15.6k	6.7k	–	56.5k	–	21.8k	21	47	84		✓		✓	✓		2009
Caltech	[3]	mobile	192k	61k	67k	155k	56k	65k	27	48	97	✓	✓	✓	✓	✓	✓	2009

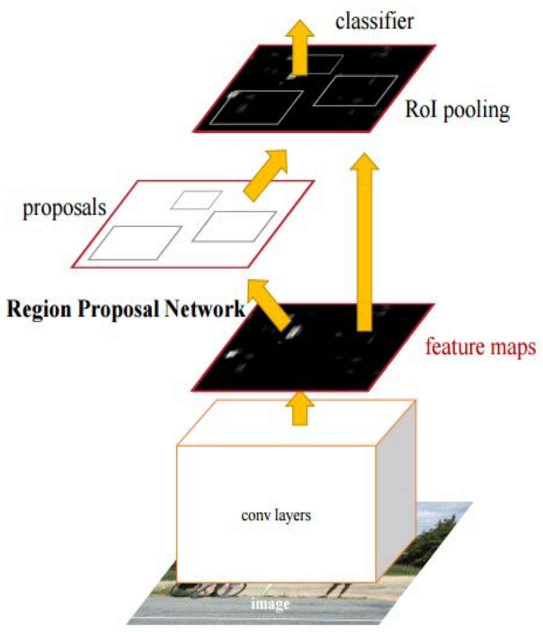
Like I mentioned in the Solution_statement, I am planning to extract the images from the .seq files and the corresponding annotations from the .vbb files provided. I am planning to build tf_record files for both training and testing and use them to train my model using the pre-trained object_detection module on the coco_dataset.

Algorithms and Techniques

I aim to extract the images from the .seq files and the corresponding annotations from the .vbb files provided. I am planning to build tf_record files for both training and testing and use them to train my model using the pre-trained object_detection module on the coco_dataset.

Like I explained in the **Problem statement** section,
Faster RCNN uses a Region Proposal network with anchor boxes to get an estimate of the possible bounding boxes and then regresses over those with the least error. It also uses the same

Faster RCNN architecture in the figure below.



.

feature maps for classification too

In Single shot detectors, instead of having two networks Region Proposals Network + Classifier Network In Single-shot architectures, bounding boxes and confidences for multiple categories are predicted directly with a single network. Similar to the anchors of Faster R-CNN, with the difference that SSD applies them on several feature maps of different resolutions

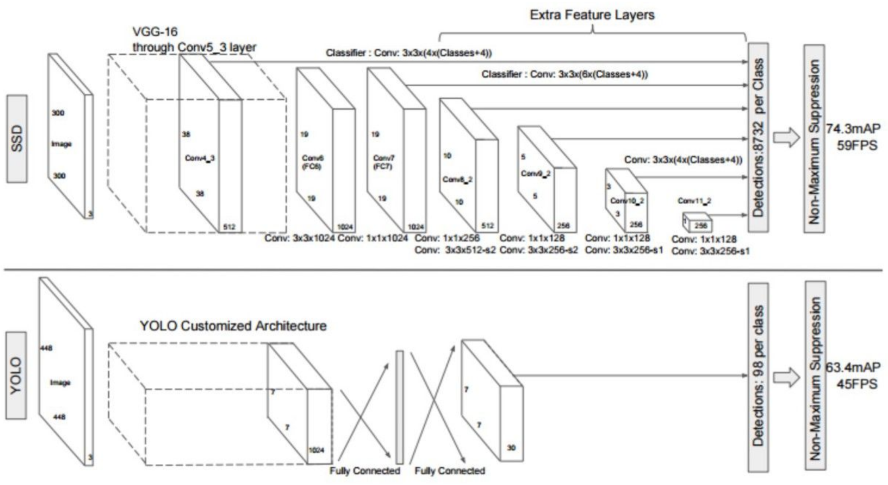
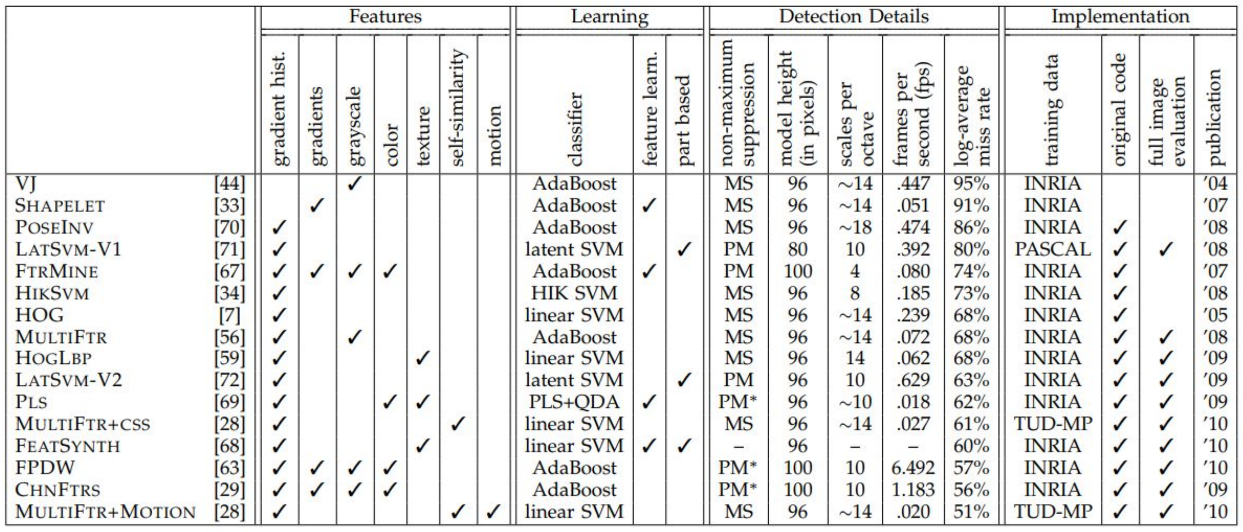


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300 × 300 input size significantly outperforms its 448 × 448 YOLO counterpart in accuracy on VOC2007 `test` while also improving the speed.

The above image from SSD paper explains how the single shot mechanisms of YOLO and SSD differ.

Benchmark Model

http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/files/PAMI12pedestrians.pdf
This paper does benchmarking of several algorithms (look at the table below from the paper) on the Caltech dataset.



		Features							Learning			Detection Details					Implementation			
		gradient hist.	gradients	grayscale	color	texture	self-similarity	motion	classifier	feature learn.	part based	non-maximum suppression	model height (in pixels)	scales per octave	frames per second (fps)	log-average miss rate	training data	original code	full image evaluation	publication
VJ	[44]			✓					AdaBoost			MS	96	∼14	.447	95%	INRIA			’04
Shapelet	[33]		✓						AdaBoost	✓		MS	96	∼14	.051	91%	INRIA			’07
PoseInv	[70]	✓							AdaBoost			MS	96	∼18	.474	86%	INRIA	✓		’08
LatSvm-V1	[71]	✓							latent SVM		✓	PM	80	10	.392	80%	PASCAL	✓	✓	’08
FtrMine	[67]	✓	✓	✓	✓				AdaBoost	✓		PM	100	4	.080	74%	INRIA	✓		’07
HikSvm	[34]	✓							HIK SVM			MS	96	8	.185	73%	INRIA	✓		’08
HOG	[7]	✓							linear SVM			MS	96	∼14	.239	68%	INRIA	✓		’05
MultiFtr	[56]	✓		✓					AdaBoost			MS	96	∼14	.072	68%	INRIA	✓	✓	’08
HogLbp	[59]	✓				✓			linear SVM			MS	96	14	.062	68%	INRIA	✓	✓	’09
LatSvm-V2	[72]	✓							latent SVM		✓	PM	96	10	.629	63%	INRIA	✓	✓	’09
Pls	[69]	✓			✓	✓			PLS+QDA	✓		PM*	96	∼10	.018	62%	INRIA	✓	✓	’09
MultiFtr+CSS	[28]	✓					✓		linear SVM			MS	96	∼14	.027	61%	TUD-MP	✓	✓	’10
FeatSynth	[68]	✓				✓			linear SVM	✓	✓	–	96	–	–	60%	INRIA	✓	✓	’10
FPDW	[63]	✓	✓	✓	✓				AdaBoost			PM*	100	10	6.492	57%	INRIA	✓	✓	’10
ChnFtrs	[29]	✓	✓	✓	✓				AdaBoost			PM*	100	10	1.183	56%	INRIA	✓	✓	’09
MultiFtr+Motion	[28]	✓					✓	✓	linear SVM			MS	96	∼14	.020	51%	TUD-MP	✓	✓	’10

TABLE 2
Comparison of Evaluated Pedestrian Detectors (see §4.2 for details)

But my approach would be entirely different from the bench_mark model because I am planning to use SSD_MobileNet / Faster_RCNN for training.

III. Methodology

Data Pre-Processing :

The ground_truth annotations have different classes like Person, People, person(?) for the detection with lesser confidence and also occlusion_infomation. I am planning to not worry about the occluded detections but to consider them as true_positives. I am also planning to eliminate the person(?) class annotations and to merge the person and people detections. So, basically I will have just one class person. I might have false_negatives in the ground_truth data. But I would like to go with it just to simplify the process.

Techniques:

An seq file is a series of concatenated image frames with a fixed size header. Matlab routines for reading/writing/manipulating seq files can be found in Piotr's Matlab Toolbox (version 3.20 or later required). These routines can also be used to extract an seq file to a directory of images.

I have extracted all the images from all the sets (set00-set10) using the matlab toolbox and the annotations from the .vbb files. But ,the annotations obtained are the text files for each image and the format of the annotation is as follows:

% bbGt version=3people 54 149 68 95 0 0 0 0 0 0 0person 259 160 22 66 0 0 0 0 0 0 0person 18 177 11 31 0 0 0 0 0 0 0person 321 112 235 369 0 0 0 0 0 0 0

People and person being the classes and the the 4 numbers after the class are the (l,t,w,h) left, top, width and height respectively.

The other digits indicated presence of an occlusion and if yes, give the (l,t,w,h) values of the visible part of the person/people. The primary difference of this video bounding box annotation from the static annotation is that each object can exist for multiple frames i.e. the vbb annotation not only provides the locations of objects but also tracking information.

Since, I am not considering occlusion information, I wrote a python script to make a csv file by reading each text annotation file for each image.

The CSV file has the annotations in the following format :

filename	width	height	class	xmin	ymin	xmax	ymax
set04_V005_I00025.jp	640	480	person	112	196	131	243

g

Now, 267994 detections are selected randomly for training and 66977 detections for testing. Also notice that since the images are randomly selected, the images are no longer temporally related to each other. Now, I go through these CSV files and read through the "filename" column to grab the corresponding images from the pool of images and place them in the corresponding train_images and test_images folders.

Complications during this step: I have tried several other open-sourced resources to extract the images and annotations from the .seq and .vbb files but could not find a smarter way of bringing the data to the shape I wanted. I finally used the matlab toolbox to extract the data and to parse the annotation files. I believe this is still a rot system.

Algorithms:

SSD Object detection

Creating TFRecords - Tensorflow Object Detection API:

From [4], I create TFRecords files using the generate_tfrecord.py available from the raccoon detection example [5]. The only modification that I had to to make here was in the class_text_to_int function. I changed this to the class "person".

Implementation:

Next, I set up the object detection API by cloning the tensorflow object detection API (models) on my computer. (https://github.com/tensorflow/models.git). I installed the object detection library and generate the tf_records for both training data and test data.

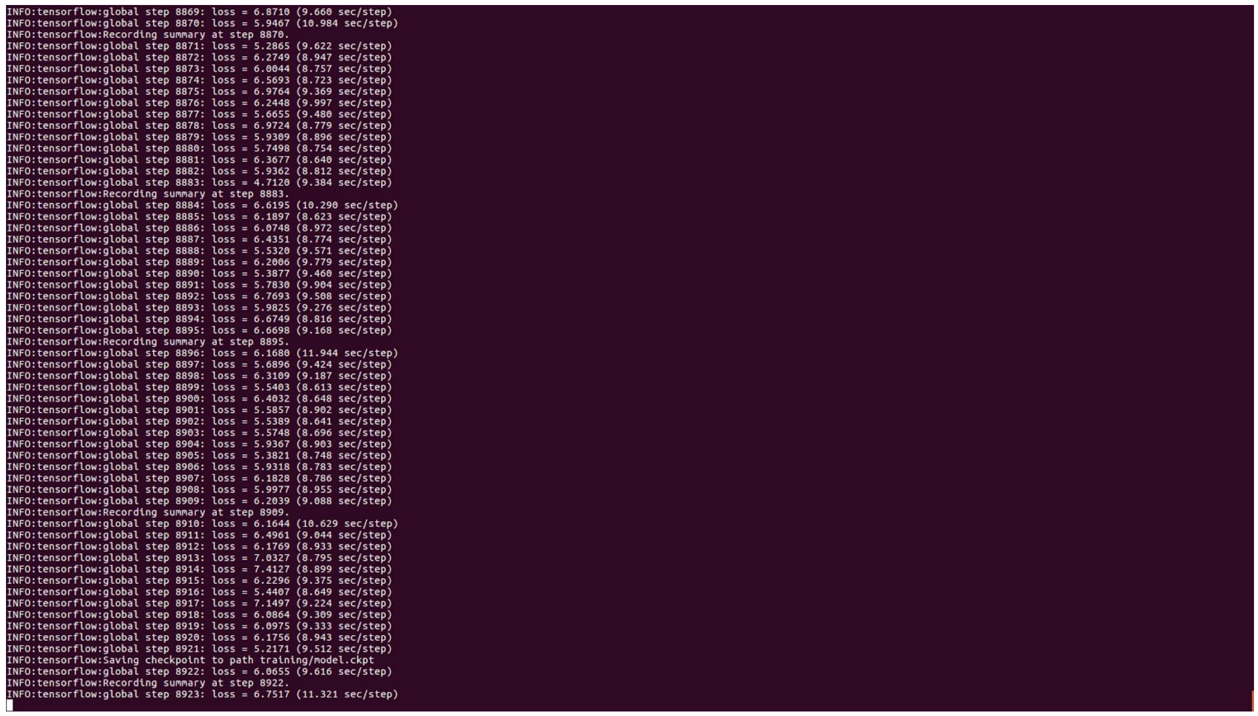
Instead of training a model from scratch, I have decided to use transfer learning by using a pre-trained model on coco-object detection dataset.
TensorFlow has quite a few pre-trained models with checkpoint files available, along with configuration files. The object API also provides some sample configurations to choose from. I chose ssd_mobilenet_v1_pets.config file, and have changed the following paramters:

```
num_classes: 1
initial_learning_rate: 0.0004
```

And gave the corresponding input path for the train.record file. I also had to define the object_detection label text file as follows:

```
item {
  id: 1
  name: 'person'
}
```

And now I start executing the train.py file that comes with the package.



```
INFO:tensorflow:global step 8869: loss = 6.8710 (9.660 sec/step)
INFO:tensorflow:global step 8870: loss = 5.9467 (10.984 sec/step)
INFO:tensorflow:Recording summary at step 8870.
INFO:tensorflow:global step 8871: loss = 5.2865 (9.622 sec/step)
INFO:tensorflow:global step 8872: loss = 6.2749 (8.947 sec/step)
INFO:tensorflow:global step 8873: loss = 6.0044 (8.757 sec/step)
INFO:tensorflow:global step 8874: loss = 6.5693 (8.723 sec/step)
INFO:tensorflow:global step 8875: loss = 6.9764 (9.369 sec/step)
INFO:tensorflow:global step 8876: loss = 6.2448 (9.997 sec/step)
INFO:tensorflow:global step 8877: loss = 5.6655 (9.480 sec/step)
INFO:tensorflow:global step 8878: loss = 6.9724 (8.779 sec/step)
INFO:tensorflow:global step 8879: loss = 5.9309 (8.896 sec/step)
INFO:tensorflow:global step 8880: loss = 5.7498 (8.754 sec/step)
INFO:tensorflow:global step 8881: loss = 6.3677 (8.640 sec/step)
INFO:tensorflow:global step 8882: loss = 5.9362 (8.812 sec/step)
INFO:tensorflow:global step 8883: loss = 4.7120 (9.384 sec/step)
INFO:tensorflow:Recording summary at step 8883.
INFO:tensorflow:global step 8884: loss = 6.6195 (10.290 sec/step)
INFO:tensorflow:global step 8885: loss = 6.1897 (8.623 sec/step)
INFO:tensorflow:global step 8886: loss = 6.0748 (8.972 sec/step)
INFO:tensorflow:global step 8887: loss = 6.4351 (8.774 sec/step)
INFO:tensorflow:global step 8888: loss = 5.5320 (9.571 sec/step)
INFO:tensorflow:global step 8889: loss = 6.2006 (9.779 sec/step)
INFO:tensorflow:global step 8890: loss = 5.3877 (9.460 sec/step)
INFO:tensorflow:global step 8891: loss = 5.7830 (9.904 sec/step)
INFO:tensorflow:global step 8892: loss = 6.7693 (9.508 sec/step)
INFO:tensorflow:global step 8893: loss = 5.9825 (9.276 sec/step)
INFO:tensorflow:global step 8894: loss = 6.6749 (8.816 sec/step)
INFO:tensorflow:global step 8895: loss = 6.6698 (9.168 sec/step)
INFO:tensorflow:Recording summary at step 8895.
INFO:tensorflow:global step 8896: loss = 6.1680 (11.944 sec/step)
INFO:tensorflow:global step 8897: loss = 5.6896 (9.424 sec/step)
INFO:tensorflow:global step 8898: loss = 6.3109 (9.187 sec/step)
INFO:tensorflow:global step 8899: loss = 5.5403 (8.613 sec/step)
INFO:tensorflow:global step 8900: loss = 6.4032 (8.648 sec/step)
INFO:tensorflow:global step 8901: loss = 5.5857 (8.902 sec/step)
INFO:tensorflow:global step 8902: loss = 5.5389 (8.641 sec/step)
INFO:tensorflow:global step 8903: loss = 5.5748 (8.696 sec/step)
INFO:tensorflow:global step 8904: loss = 5.9367 (8.903 sec/step)
INFO:tensorflow:global step 8905: loss = 5.3821 (8.748 sec/step)
INFO:tensorflow:global step 8906: loss = 5.9318 (8.783 sec/step)
INFO:tensorflow:global step 8907: loss = 6.1828 (8.786 sec/step)
INFO:tensorflow:global step 8908: loss = 5.9977 (8.955 sec/step)
INFO:tensorflow:global step 8909: loss = 6.2039 (9.088 sec/step)
INFO:tensorflow:Recording summary at step 8909.
INFO:tensorflow:global step 8910: loss = 6.1644 (10.629 sec/step)
INFO:tensorflow:global step 8911: loss = 6.4961 (9.044 sec/step)
INFO:tensorflow:global step 8912: loss = 6.1769 (8.933 sec/step)
INFO:tensorflow:global step 8913: loss = 7.0327 (8.795 sec/step)
INFO:tensorflow:global step 8914: loss = 7.4127 (8.899 sec/step)
INFO:tensorflow:global step 8915: loss = 6.2296 (9.375 sec/step)
INFO:tensorflow:global step 8916: loss = 5.4407 (8.649 sec/step)
INFO:tensorflow:global step 8917: loss = 7.1497 (9.224 sec/step)
INFO:tensorflow:global step 8918: loss = 6.0864 (9.309 sec/step)
INFO:tensorflow:global step 8919: loss = 6.0975 (9.333 sec/step)
INFO:tensorflow:global step 8920: loss = 6.1756 (8.943 sec/step)
INFO:tensorflow:global step 8921: loss = 5.2171 (9.512 sec/step)
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:global step 8922: loss = 6.0655 (9.616 sec/step)
INFO:tensorflow:Recording summary at step 8922.
INFO:tensorflow:global step 8923: loss = 6.7517 (11.321 sec/step)
```

I left the model to train overnight and I see that the total loss almost plateaued at around 5.
I think ideally, the total loss reaching under 2 or even 1 would be great.

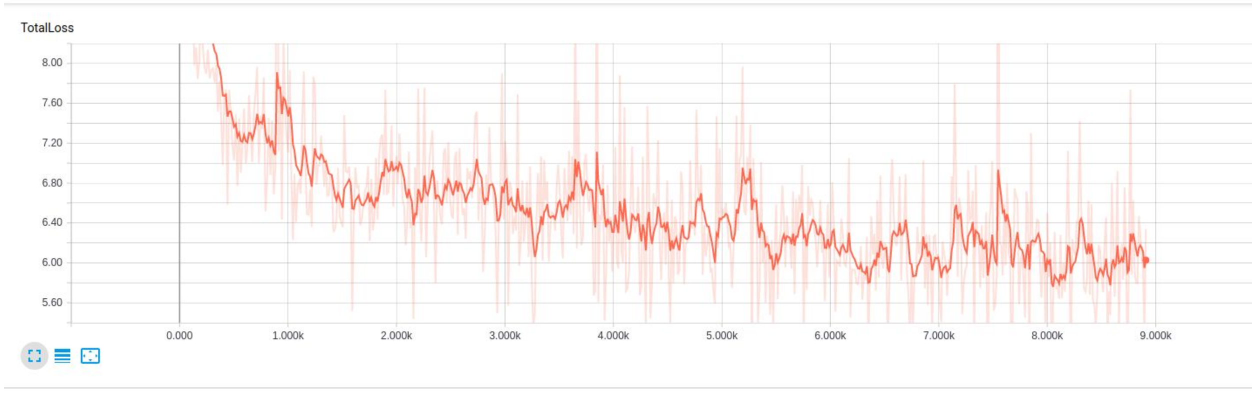
Refinement:

Since I am using the predefined configuration from the tensorlfow_library, One refinement I have done is to change the learning rate from 0.04 to 0.004 and retraining the model. But unfortunately, there is not much improvement in the performance of the model. I have discussed the results in the following section. Hence I am not showing the plots with higher learning rate.

IV. Results:

Model Evaluation and Validation:

TensorBoard analysis on training :

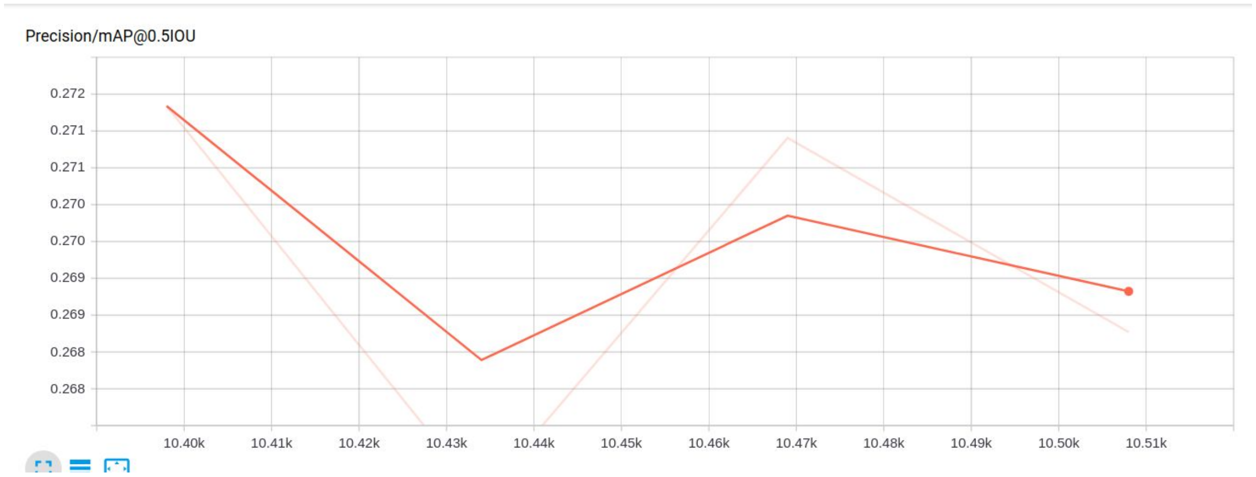


You see that the Total_loss in the training is around 5 to 6. Ideally having this loss below 2 would be better. I can think of the following reasons for this behavior of the data.

1) Data is not large enough or diverse enough.
2) Changing the anchor boxes would have improved the accuracy.

The eval.py script provided by tensor flow object detection package helps evaluate the model and also provides the evaluation checkpoints. I used the tensorboard at a different port to see the results.

The evaluation of the model on test data is as follows:



The model reaches an mAP@0.5IOU score of around 27% . I don't think this is the best possible model. But certainly gives a good intuition of what it is to have a pedestrian detection model.

Justification:

In the Benchmarks section, the table uses log-average missrate as the criteria. But unfortunately, I have only mAP@0.5IOU criteria as an evaluation of the model, I cannot compare these two.

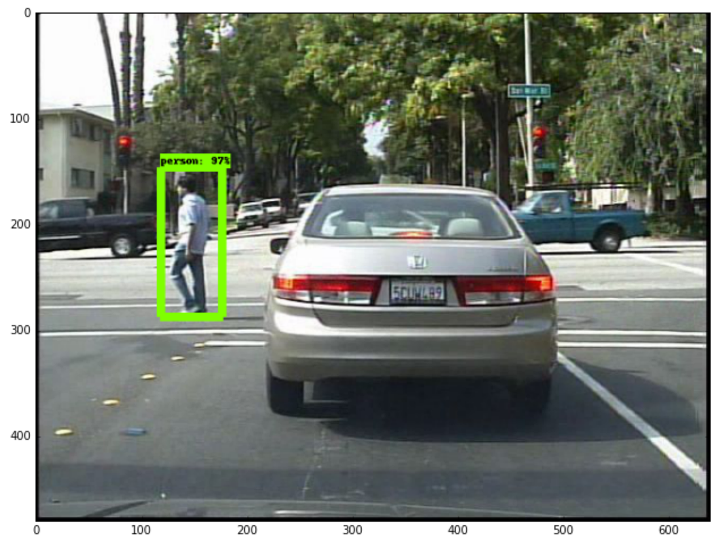
But,
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

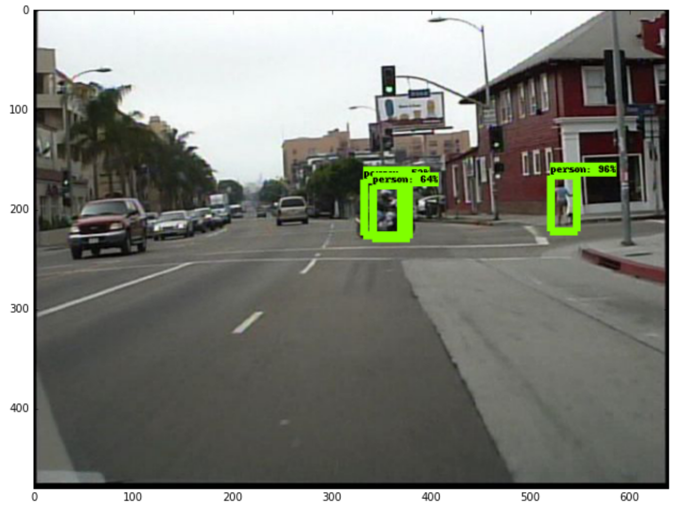
The Tensorflow_model_zoo states that the ssd_mobilenet trained on coco_dataset has a mAP value of about 21, which makes my model with mAP score of 27 to have almost comparable object_detection capabilities (which it should have, because I used the same config). But I do agree that this is not entirely valid because the models are trained on different datasets (although coco dataset has person class)

V. Conclusion

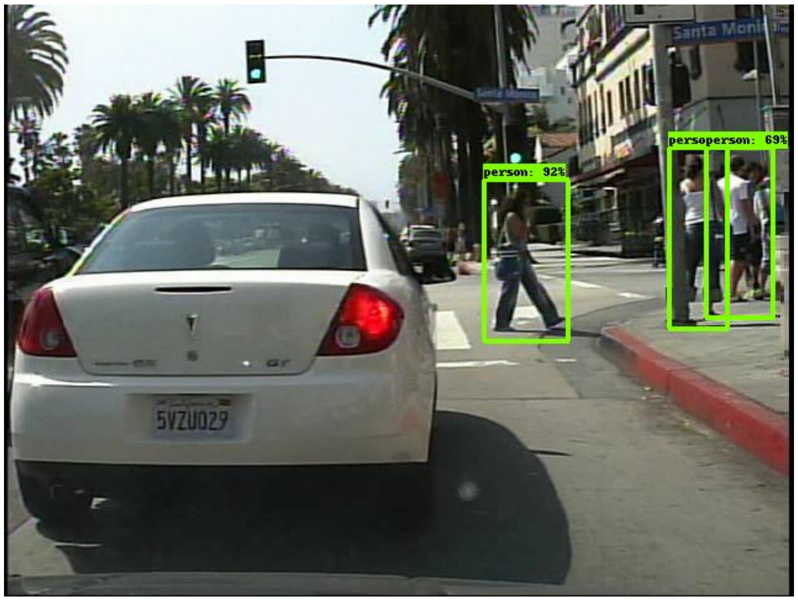
So, I have generated a frozen inference graph and followed the demo object_detection jupyter notebook to load the frozen tensor flow model into memory and have loaded the label map.

I have tested on the couple of images and the detection results are as follows:



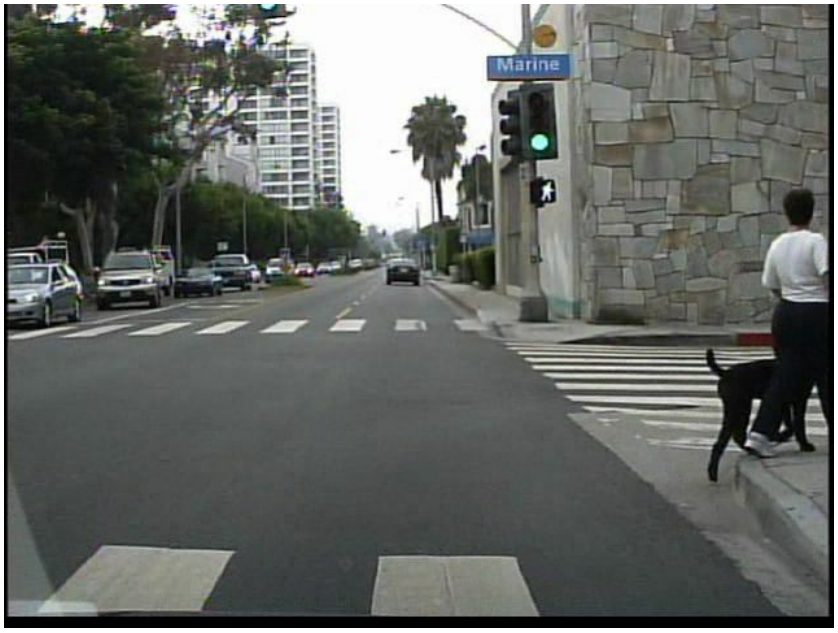


In the above , the first two images had really good detection results. In the second image, the biker was detected twice.



Negative example:

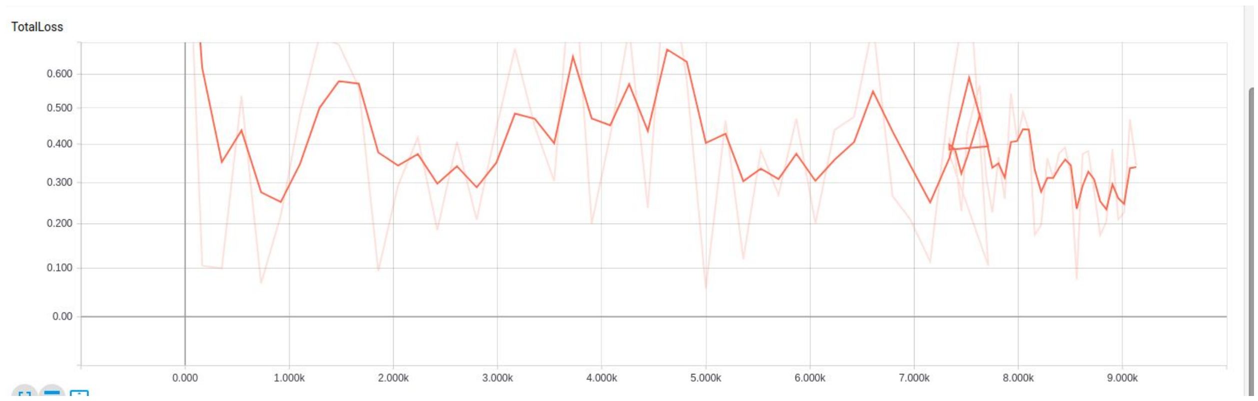
The following pedestrian with the dog was not detected by the model.



Trying Different Model:

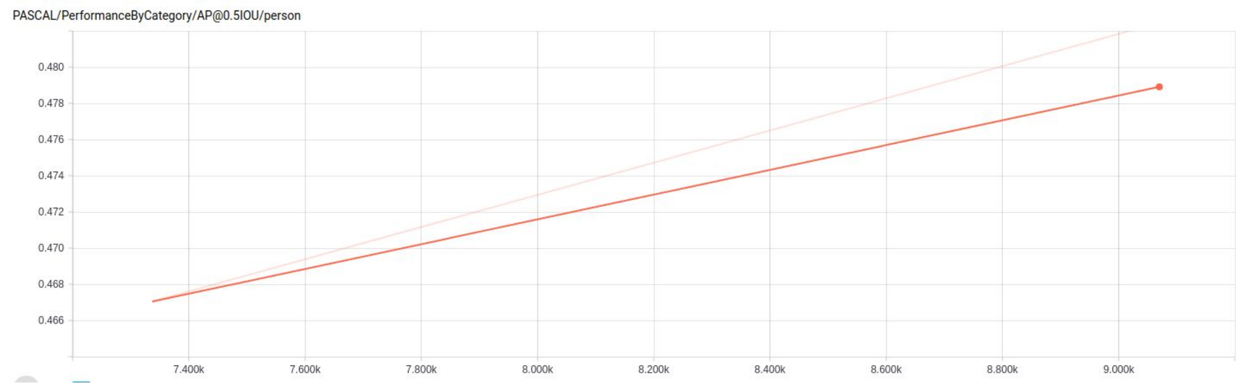
I have trained the data with another architecture called **Faster_rcnn_resnet101_kitti** pretrained on Kitti dataset and the training_loss, evaluation and some examples are as follows:

Total_loss during training:

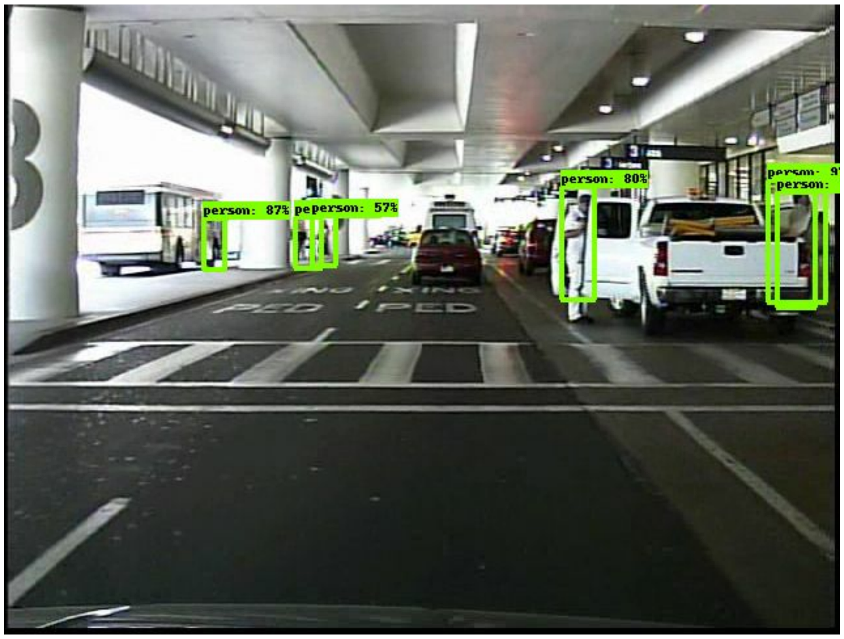


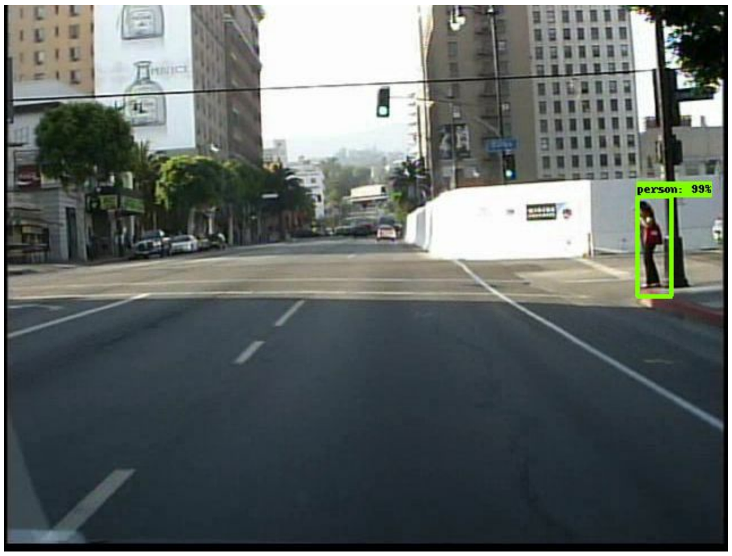
It is under 2.

Evaluation of the model:

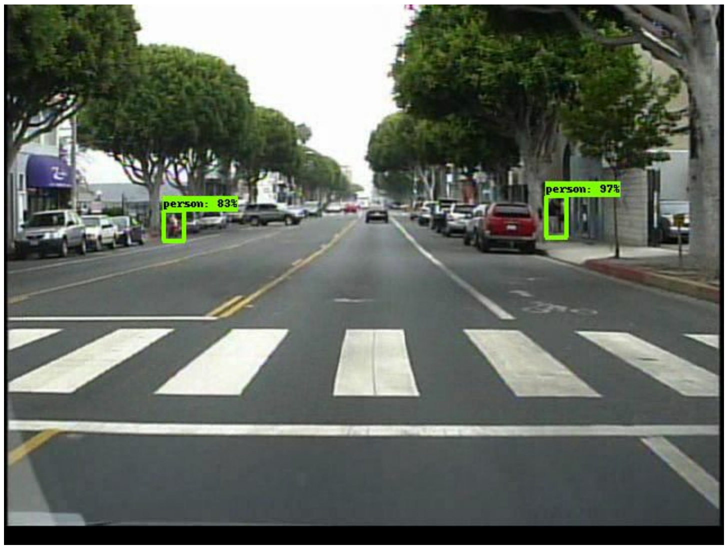


The mAP@0.5IOU is around 0.47 which is quite an improvement from the SSD_Mobilenet model.

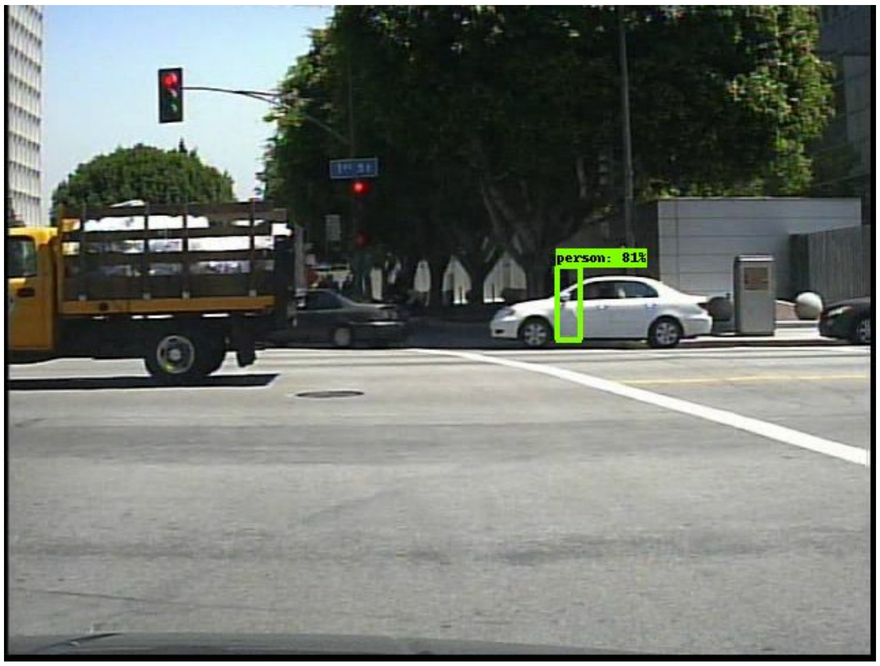




The model is able to recognize even the smaller images of pedestrians with more confidence.



One negative example:



Clearly, there is no person in the image. I don’t think the bounding box is around the driver of the white car too.

Reflection:

SO, basically I took a large data of pedestrians and used an existing framework provided by Tensorflow and a pre-trained SSD_MobileNet model trained on coco_dataset to train on my pedestrian data for pedestrian detection.

Getting the data into usable and useful form for the framework was challenging to me and was very interesting. I have tried to tune/refine the model to get better performance in vain.

Now I have used the Faster_rcnn_resnet101_kitti model pretrained on Kitti dataset in the same framework

Improvement:

I can think of the following reasons for this behavior of the data.

- Data is not large enough or diverse enough.
- Changing the anchor boxes would have improved the accuracy.

Also, like I explained in the above section, I have changed the underlying model architecture from SSD_mobilnet to Faster_RCNN_resnet and the performance improved (mAP value). I believe having better data will definitely make the model more general.

References:

[1].http://www.govtech.com/fs/Googles-New-Pedestrian-Recognition-Tech-Could-Quicken-Self-Driving-Cars-to-Market.html

[2] https://pdfs.semanticscholar.org/b406/6e7b3c227bcf8549af551edb163cedc1b931.pdf
[3] http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/files/algorithms.pdf
[4]
https://pythonprogramming.net/creating-tfrecord-files-tensorflow-object-detection-api-tutorial/?completed=/custom-objects-tracking-tensorflow-object-detection-api-tutorial/
[5]
https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9