# Machine Learning Engineer Nanodegree

Capstone Project

Chandana Neerukonda
October 17 2017

**Domain Background:**

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)

**Datasets and inputs:**

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

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

**Solution Statement**

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.

**Benchmark Model**

http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/files/PAMI12pedestrians.pdf
This paper does benchmarking of several algorithms on the Caltech dataset.

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

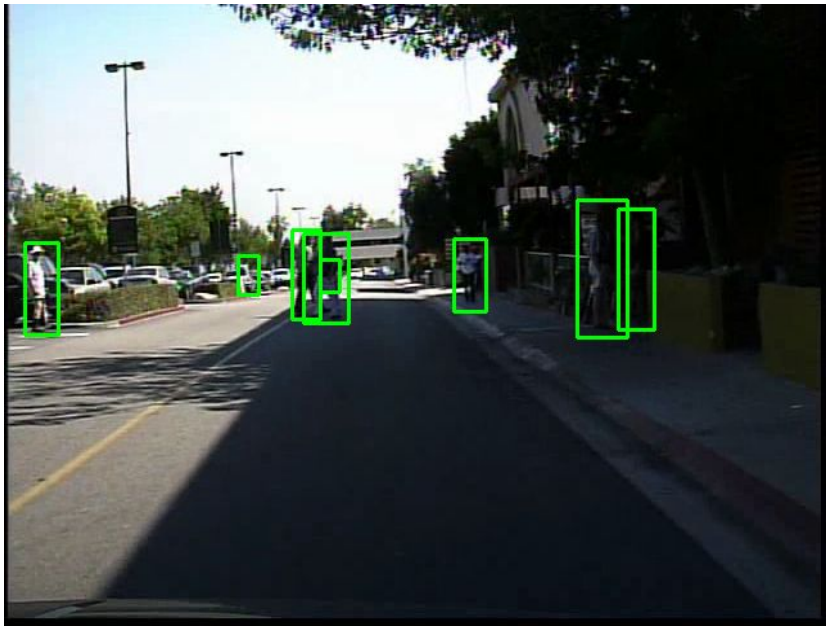
## Evaluation Metrics:

The Evaluation metric I would propose would be the mAP value ( detection and classification accuracy).

But the reference paper gave the evaluation metrics interms of missrate. I do know how exactly to compare these two metrics.

**Data Exploration:**

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.

**Data Pre-Processing and Implementation:**

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.jpg | 640 | 480 | person | 112 | 196 | 131 | 243 |

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.

**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.

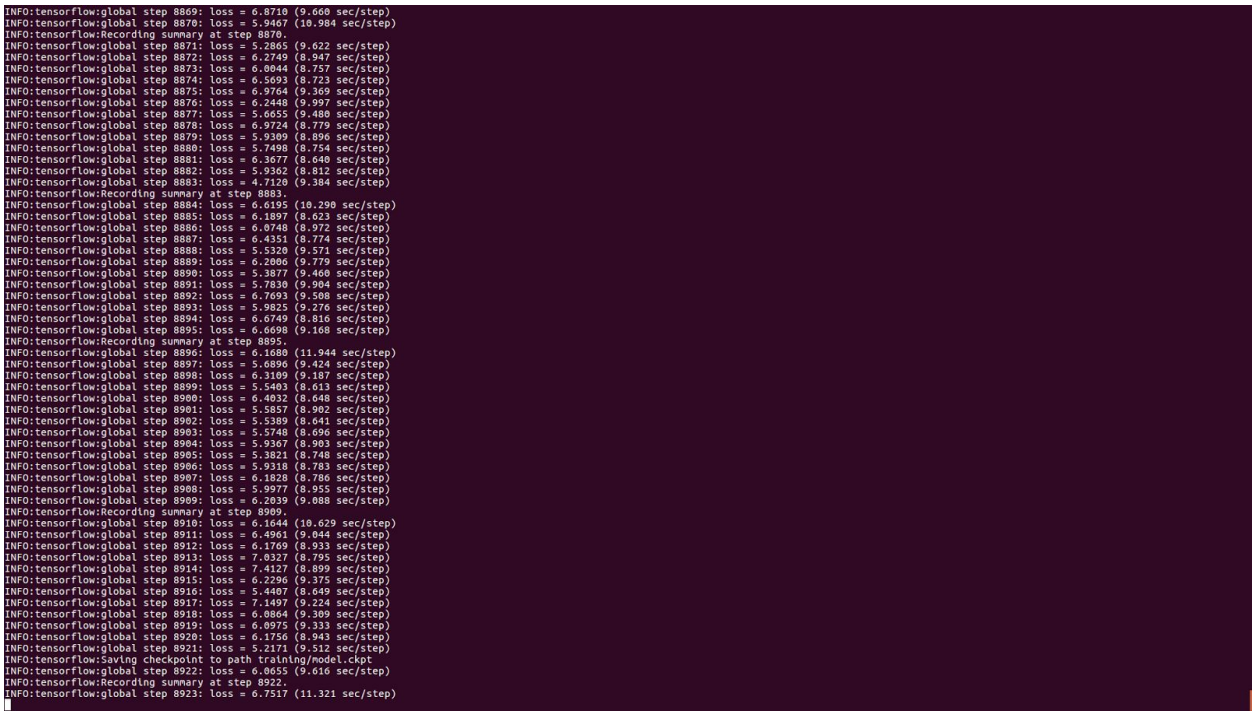
Instead of training a model from scratch, I have decided to use transfer learning by using a pre-trained model on coco-object detection dataset.
TensorFlow has quite a few pre-trained models with checkpoint files available, along with configuration files. The object API also provides some sample configurations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

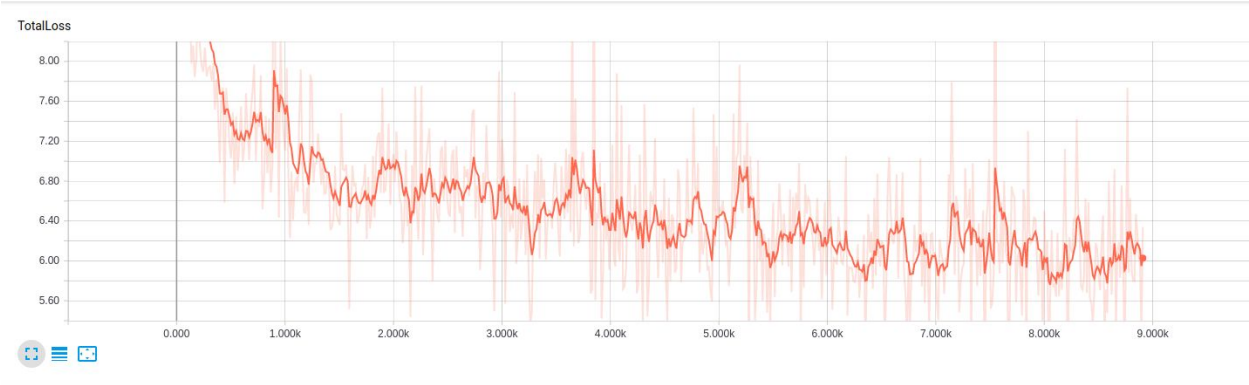
```
item {
  id: 1
  name: 'person'
}
```

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



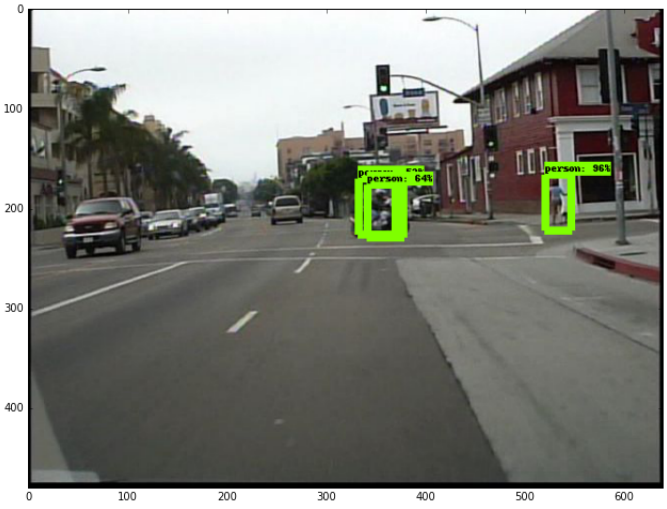

I left the model to train over night and I see that the total loss almost plateaued at around 5.
I think ideally, the total loss reaching under 2 or even 1 would be great.

TensorBoard analysis on training :



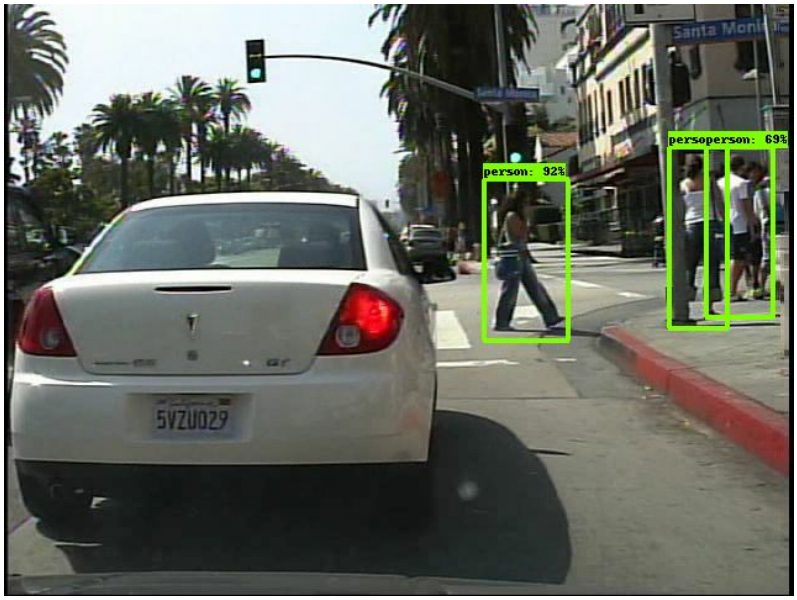

So, I have generated a frozen inference graph and followed the demo object_detection jupyter notebook to load the frozen tensor flow model into memory and have loaded the label map.

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

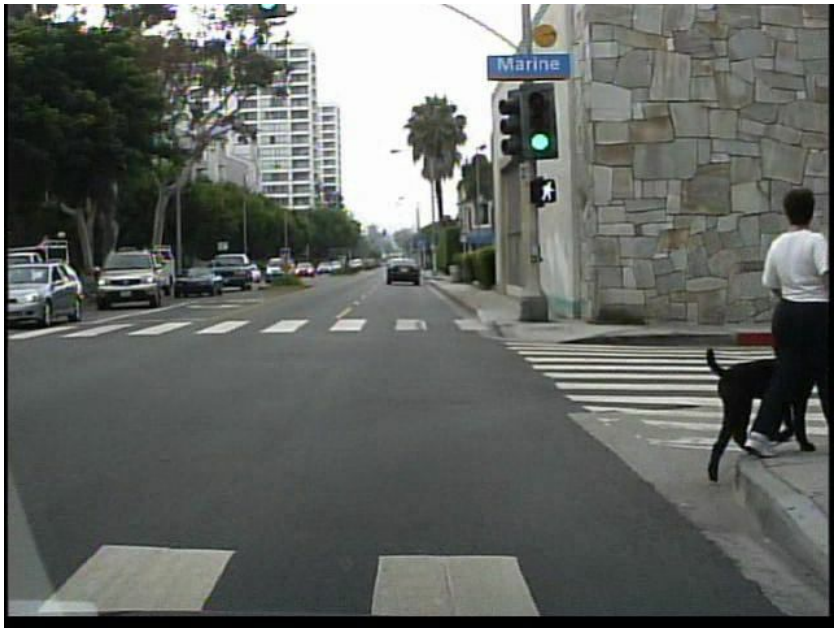

In the above , the first two images had really good detection results. In the second image, the biker was detected twice.

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.

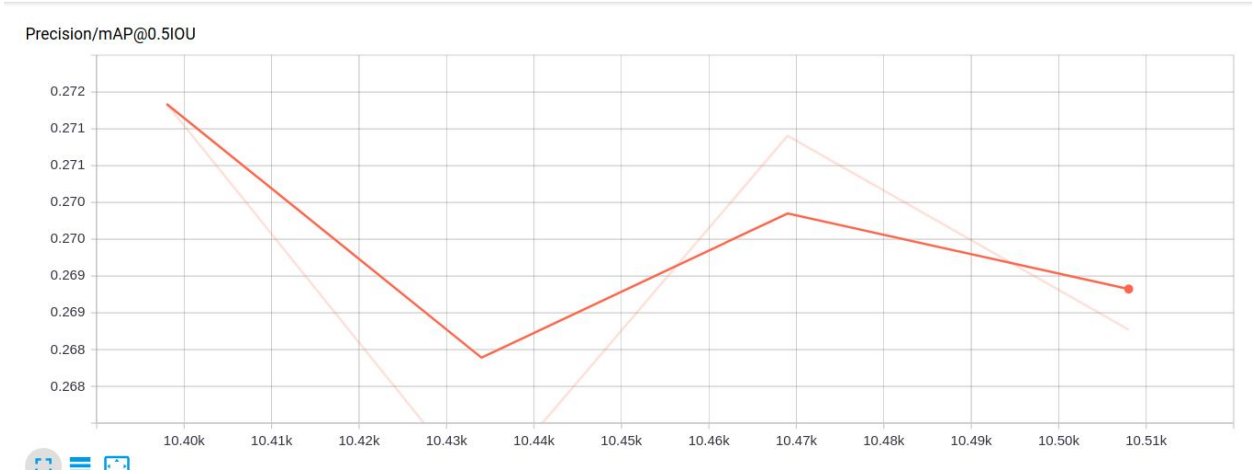



Negative example:

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




The mAP@0.5IOU Precision graph is as follows: The average value is around 0.27




**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