Machine Learning Engineer Nanodegree

Capstone

Mahavir Dwivedi

June 6th 2018

Domain Background

After learning a wide spectrum of ML algorithms and tools through the Udacity Machine Learning NanoDegree, I wanted to get my hands dirty on some problem statement where I can implement the concepts of Machine Learning majorly Deep Learning. As I was deciding on my problem statement for my capstone, I came across a lot of advancements going on in the field of Self-Driving Car. This inspired me to pick up a topic which is related to this theme.

There are a lot of aspects involved in making of a successful Self Driven Car, one of the most important aspects is that the car should be able to differentiate between different vehicles running over the road. A self-driven car should be able to recognize the vehicle around itself. As the recent progress in the field of computer vision majorly object-detection, using these techniques I will try to build a deep learning model which can be used for vehicle detection .

As I don't have a Self-driving Car environment I will be using our vehicle detection model in another scenario.I will be setting up a smart Vehicle Detection system which will be installed at an entry/exit gate barrier, where it will be used to maintain the database and to monitor the different type of vehicles entering the premises.

Problem Statement :

Following is the block diagram for the problem statement:

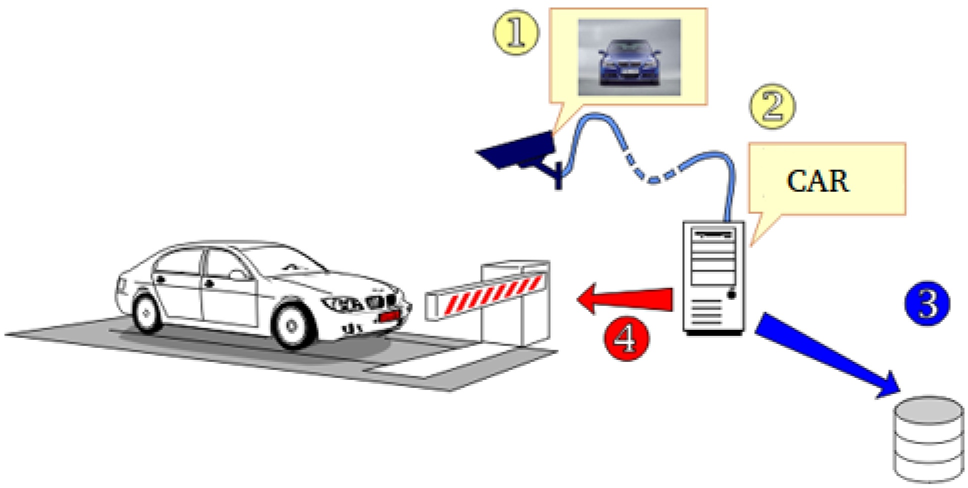


Fig-1 Block Diagram of problem statement

1. The Internet Protocol Camera at the Entry barrier takes the picture
2. The Picture is send to the CPU.
3. The Picture is being processed at the GPU
4. The decision is made to open the gate by the CPU.

Here I am proposing a deep learning model which can detect vehicles in a video stream.Following is the list of vehicles which I propose to detect:

1. Car
2. Bus
3. Motor Bike
4. Bicycle

The recognition will be performed on a video stream. I aim to achieve a processing speed where no vehicle is missed.

Datasets Exploration :

As I am building a system for detecting vehicles, the training will be performed on following publicly available dataset:

VOC Dataset (http://host.robots.ox.ac.uk/pascal/VOC/voc2007/)

Dataset Attributes:

In order to evaluate the classification and detection challenges, the image annotation includes the following attributes for every object in the target set of object classes:–

- *Class:* aeroplane, bird, bicycle, boat, bottle, bus,car, cat, chair, cow, dining table, dog, horse, motorbike,person, potted plant, sheep, sofa, train, tv/monitor.
- *Bounding box:* an axis-aligned bounding box surrounding the extent of the object visible in the image.

Out of the 20 classes:

- *Person:* person
- *Animal:* bird, cat, cow, dog, horse, sheep
- *Vehicle:* aeroplane, bicycle, boat, bus, car, motorbike, train
- *Indoor:* bottle, chair, dining table, potted plant, sofa, tv/monitor

I will be using the model over following 4 vehicle classes:

Vehicle: Bicycle, Bus, Car, Motorbike

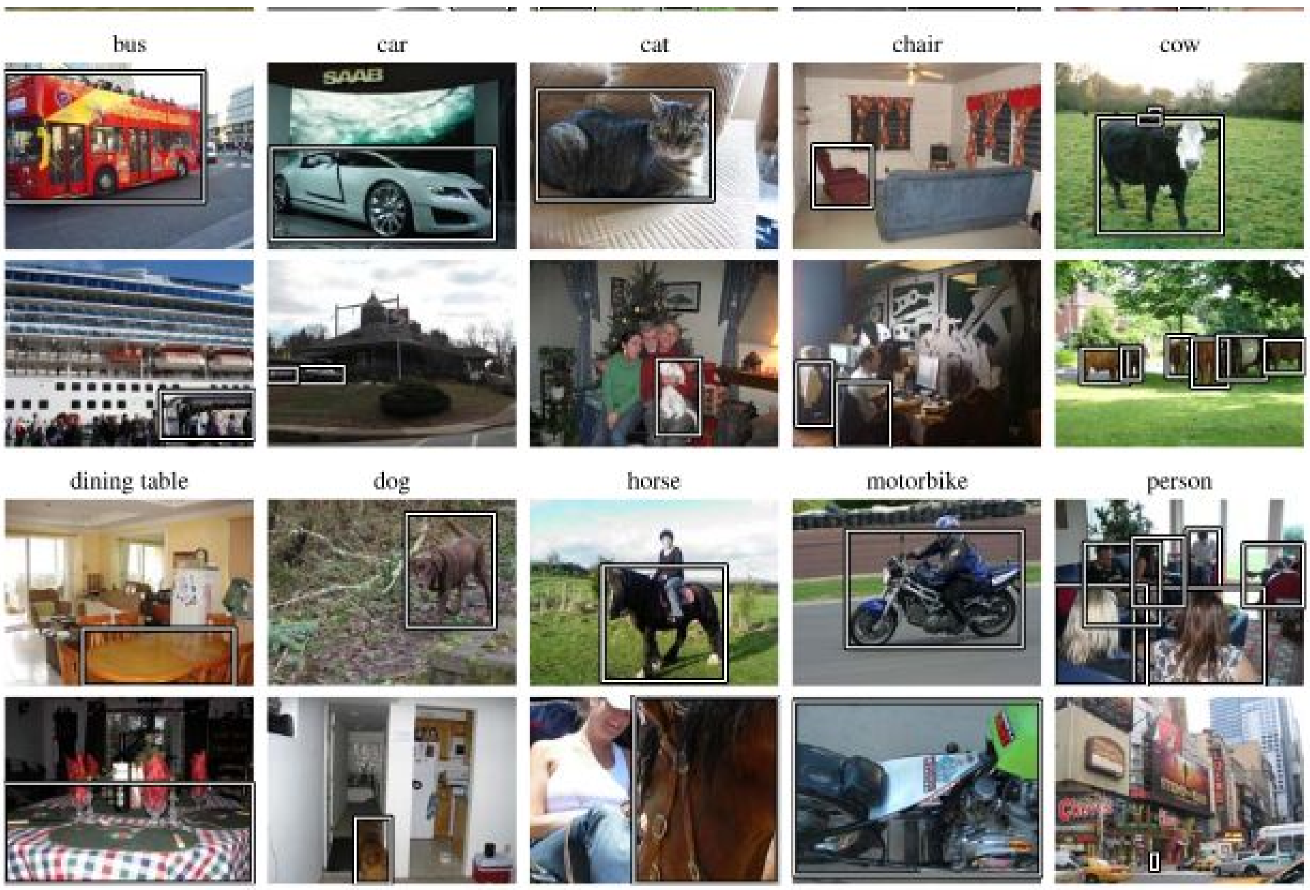


Fig- 2 Example images from the VOC 2007 dataset. These are few of the 20 classes annotated, for each of the classes above in the image two examples are shown. Note the wide range of pose, scale, clutter, occlusion and imaging conditions.

VOC datasets contain significant variability in terms of object size, orientation, pose, illumination, position and occlusion. It is also important to notice that the datasets do not exhibit systematic bias,for example, favouring images with centred objects or good illumination. Similarly, to ensure accurate training and evaluation, image annotations are consistent, accurate and exhaustive for the specified classes.

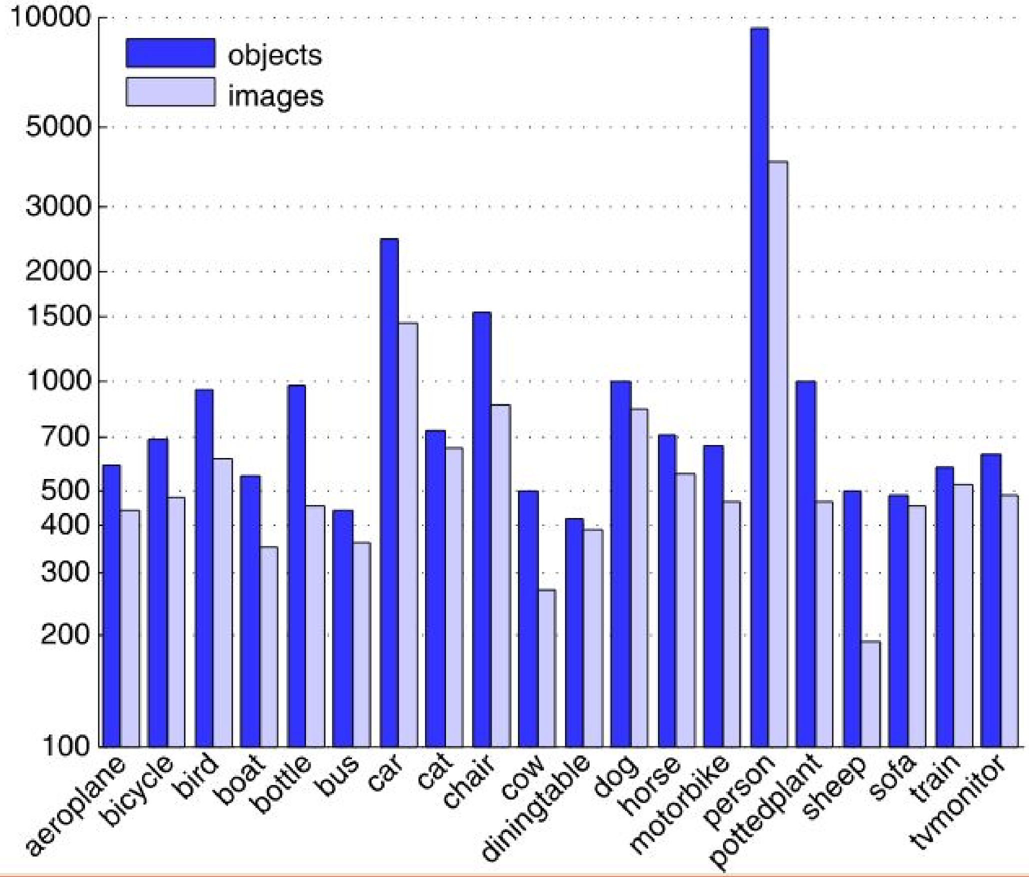


Fig-3 Summary of the entire VOC2007 dataset. Histogram by class of the number of objects and images containing at least one object of the corresponding class.

Evaluation Metrics :

The evaluation matrix I will be using here is mean average precision or "mAP score". It has become the accepted way to evaluate object detection competitions, such as for the PASCAL VOC, ImageNet, and COCO challenges.

In object detection, evaluation is non trivial, because there are two distinct tasks to measure:

1. Determining whether an object exists in the image (classification)
2. Determining the location of the object (localization, a regression task).

Furthermore, in a typical data set there will be many classes and their distribution is non-uniform. So a simple accuracy-based metric will introduce biases. It is also important to assess the risk of misclassifications. Thus, there is the need to associate a “confidence score” or **model score** with each bounding box detected and to assess the model at various level of confidence.

In order to address these needs, the Average Precision (AP) was introduced. To understand the AP, it is necessary to understand the precision and recall of a classifier. Briefly, in this context, *precision measures the “false positive rate” or the ratio of true object detections to the total number of objects that the classifier predicted*. If you have a precision score of close to 1.0 then there is a high likelihood that whatever the classifier predicts as a positive detection is in fact a correct prediction. *Recall measures the “false negative rate” or the ratio of true object detections to the total number of objects in the data set*. If you have a recall score close to 1.0 then almost all objects that are in your dataset will be positively detected by the model

To calculate the AP, for a specific class (say a “person”) the precision-recall curve is computed from the model’s detection output, by varying the model score threshold that determines what is counted as a model-predicted positive detection of the class.

The final step to calculating the AP score is to take the average value of the precision across all recall values . This becomes the single value summarizing the shape of the precision-recall curve. To do this unambiguously, the AP score is defined as the mean precision at the set of 11 equally spaced recall values, *Recall_i* = [0, 0.1, 0.2, …, 1.0]. Thus,

$$AP = \frac{1}{11} \sum_{\text{Recall}_i} \text{Precision}(\text{Recall}_i)$$

The precision at recall *i* is taken to be the maximum precision measured at a recall exceeding *Recall_i*.

Up until now, I have been discussing only the classification task. For the localization component (was the object's **location** correctly predicted?) I must consider the amount of overlap between the part of the image segmented as true by the model vs. that part of the image where the object is actually located.

Localization and Intersection over Union

In order to evaluate the model on the task of object localization, we must first determine how well the model predicted the location of the object. Usually, this is done by drawing a bounding box around the object of interest, but in some cases it is an N-sided polygon or even pixel by pixel segmentation. For all of these cases, the localization task is typically evaluated on the Intersection over Union threshold (IoU). Many good explanations of IoU exist but the basic idea is that *it summarizes how well the ground truth object overlaps the object boundary predicted by the model*.

Putting it all together :

Now that I have defined Average Precision (AP) and seen how the IoU threshold affects it, the mean Average Precision or mAP score is calculated by taking the mean AP over all classes and/or over all IoU thresholds, depending on the competition. For example:

- PASCAL VOC2007 challenge only 1 IoU threshold was considered: 0.5 so the mAP was averaged over all 20 object classes.
- For the COCO 2017 challenge, the mAP was averaged over all 80 object categories and all 10 IoU thresholds.

Averaging over the 10 IoU thresholds rather than only considering one generous threshold of IoU ≥ 0.5 tends to reward models that are better at precise localization.

Model object detections are determined to be true or false depending upon the IoU threshold. This IoU threshold(s) for each competition vary, but in the COCO challenge, for example, 10 different IoU thresholds are considered, from 0.5 to 0.95 in steps of 0.05. For a specific object (say, ‘person’) this is what the precision-recall curves may look like when calculated at the different IoU thresholds of the COCO challenge:

Algorithms And Techniques:

Given the problem statement of vehicle detection in a video stream, I went through the various Computer vision models to arrive at the solution.

As the task is to figure out all the vehicles present in a given frame at a given time, I didn’t use a classification model, rather I built my pipeline based on an object detection model.

A classification model gives us the probability of all the classes being present in that picture.And then we output that the class with highest probability in that picture as our target class.

Whereas in an object detection model the model will give us all the objects with their respective classes and their positions respectively.

We used a LPR camera to capture the video stream, once the image is captured it is resized to a particular size.The preprocessed image is then passed to the object detection model where the vehicles of the above mentioned classes are detected.Once vehicles are detected, a bounding box is created on the display screen as well as the same is printed on the screen. Based on the model output further decisions can be taken by the computer.

BenchMark:

I have aimed to achieve couple of benchmarks :

1. The model which I train over VOC dataset should achieve a map of 0.699.
2. As we are building a system for vehicle detection, we shouldn’t miss any passing by vehicle. As we know at any of the boom barrier the vehicle stops, hence we can operate at low fps. Given the above specifications we aim to achieve a processing speed of 1 fps.

Methodology:

- **Data preprocessing:**

1. **Data Normalization:** We have removed the mean of the input color components and set their variance to 1.This has improved the results by 0.8%.
2. **Transform:** Given that,we are using pytorch library, we transform the image into a pytorch tensor.
3. As we are using a RPN based model, We need to preprocess an image for feature extraction.

- The length of the shorter edge is scaled to a min size set by the user
- After the scaling, if the length of the longer edge is longer than the set scaled size
- the image is scaled to fit the longer edge to max image size
- After resizing the image, the image is subtracted by a mean image value

Implementation :

Model Architecture

Faster-rcnn: It all starts with an image, from which we want to obtain:

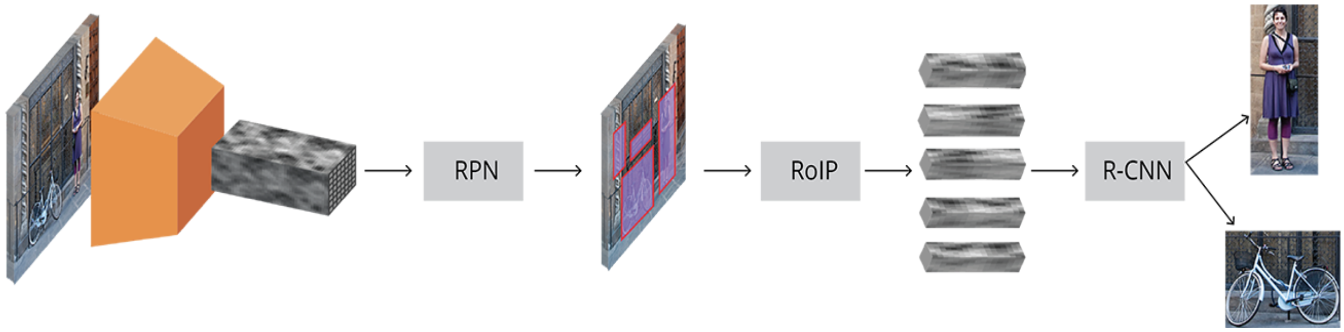


Fig-4 : Faster-RCNN Architecture

- a list of bounding boxes.
- a label assigned to each bounding box.
- a probability for each label and bounding box.

The input images are represented as Height×Width×Depth tensors (multidimensional arrays), which are passed through a pre-trained CNN up until an intermediate layer, ending up with a convolutional feature map. We use this as a feature extractor for the next part.

This technique is very commonly used in the context of Transfer Learning, especially for training a classifier on a small dataset using the weights of a network trained on a bigger dataset.

Next, we have what is called a Region Proposal Network (RPN, for short). Using the features that the CNN computed, it is used to find up to a predefined number of regions (bounding boxes), which may contain objects.

Probably the hardest issue with using Deep Learning (DL) for object detection is generating a variable-length list of bounding boxes. When modeling deep neural networks, the last block is usually a fixed sized tensor output (except when using

Recurrent Neural Networks). For example, in image classification, the output is a (N,) shaped tensor, with N being the number of classes, where each scalar in location i contains the probability of that image being labeli.

The variable-length problem is solved in the RPN by using anchors: fixed sized reference bounding boxes which are placed uniformly throughout the original image. Instead of having to detect where objects are, we model the problem into two parts. For every anchor, we ask:

Does this anchor contain a relevant object?

After having a list of possible relevant objects and their locations in the original image, it becomes a more straightforward problem to solve. Using the features extracted by the CNN and the bounding boxes with relevant objects, we apply Region of Interest (RoI) Pooling and extract those features which would correspond to the relevant objects into a new tensor.

Finally, comes the R-CNN module, which uses that information to:

- Classify the content in the bounding box (or discard it, using “background” as a label).
- Adjust the bounding box coordinates (so it better fits the object).

Obviously, some major bits of information are missing, but that’s basically the general idea of how Faster R-CNN works.

Base network :

As we mentioned earlier, the first step is using a CNN pre trained for the task of classification (e.g. using ImageNet) and using the output of an intermediate layer. This may sound really simple for people with a deep learning background, but it’s important to understand how and why it works, as well as visualize what the intermediate layer output looks like.

There is no real consensus on which network architecture is best. The original Faster R-CNN used ZF and VGG pretrained on ImageNet but since then there have been lots of different networks with a varying number of weights. For example, MobileNet, a smaller and efficient network architecture optimized for speed, has approximately 3.3M parameters, while ResNet-152 (yes, 152 layers), once the state of the art in the ImageNet classification competition, has around 60M. Most recently, new architectures like DenseNet are both improving results while lowering the number of parameters.

VGG :

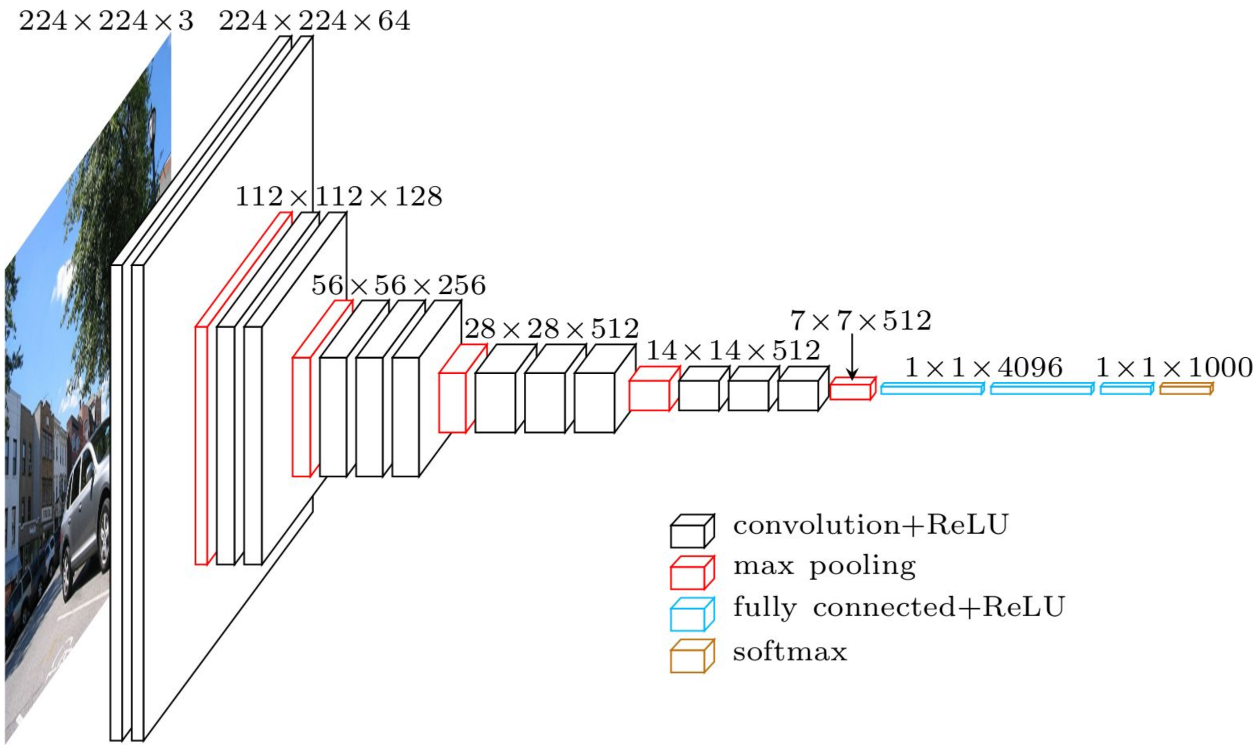


Fig - 6 VGG is being used as the base network

VGG, whose name comes from the team which used it in the ImageNet ILSVRC 2014 competition, was published in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition” by Karen Simonyan and Andrew Zisserman. By today’s standards it would not be considered very deep, but at the time it more than doubled the number of layers commonly used and kickstarted the “deeper → more capacity → better” wave (when training is possible).

When using VGG for classification, the input is a 224×224×3 tensor (that means a 224x224 pixel RGB image). This has to remain fixed for classification because the final block of the network uses fully-connected (FC) layers (instead of convolutional), which require a fixed length input. This is usually done by flattening the output of the last convolutional layer, getting a rank 1 tensor, before using the FC layers.

Since we are going to use the output of an intermediate convolutional layer, the size of the input is not our problem. At least, it is not the problem of this module since only convolutional layers are used. Let's get a bit more into low-level details and define which convolutional layer we are going to use. The paper does not specify which layer to use; but in the official implementation you can see they use the output of conv5/conv5_1 layer.

Each convolutional layer creates abstractions based on the previous information. The first layers usually learn edges, the second finds patterns in edges in order to activate for more complex shapes and so forth. Eventually we end up with a convolutional feature map which has spatial dimensions much smaller than the original image, but greater depth. The width and height of the feature map decrease because of the pooling applied between convolutional layers and the depth increases based on the number of filters the convolutional layer learns.

In its depth, the convolutional feature map has encoded all the information for the image while maintaining the location of the "things" it has encoded relative to the original image. For example, if there was a red square on the top left of the image and the convolutional layers activate for it, then the information for that red square would still be on the top left of the convolutional feature map.

Train test graphs :

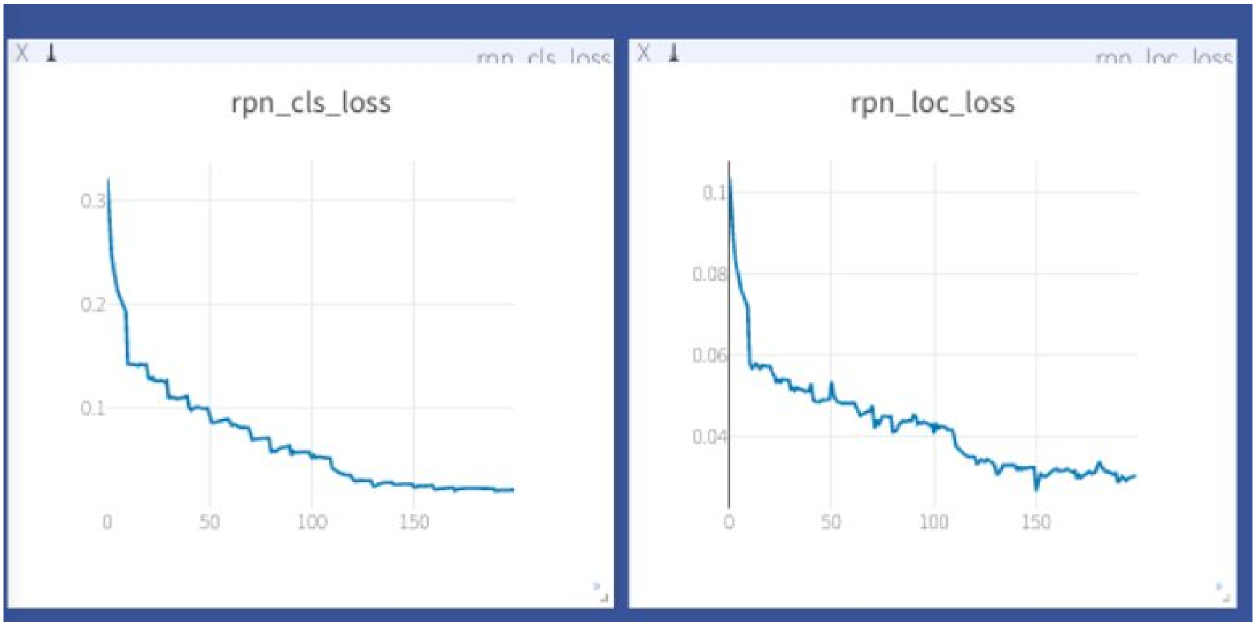


Fig 7: Rpn Losses plotted in above figure.

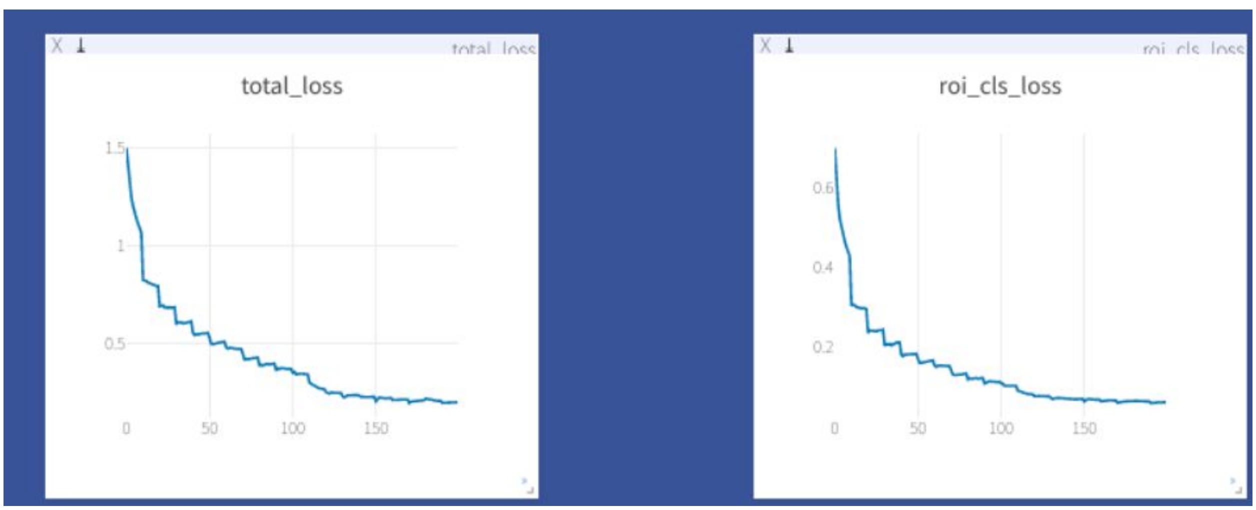


Fig-8 Total Loss being plotted using visdom server

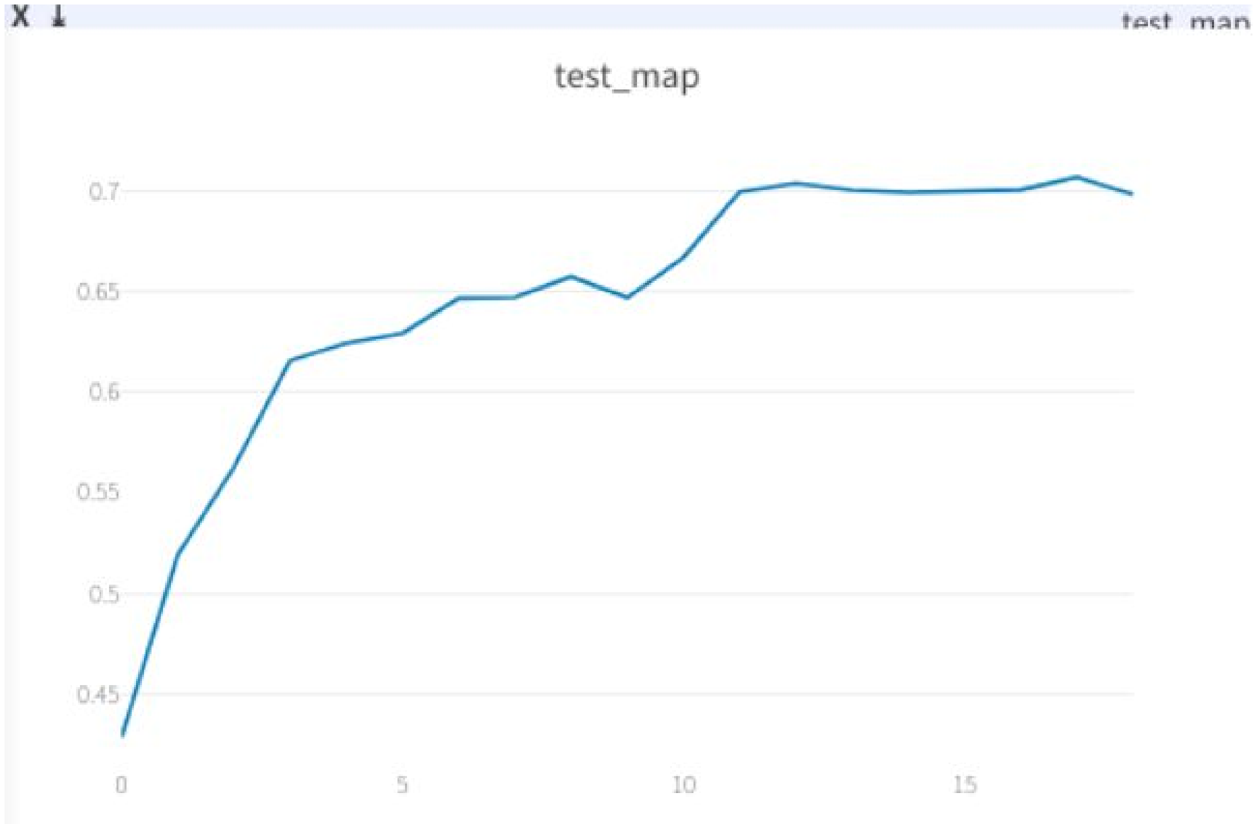


Fig -9 : plot for test_map

Train test speed :

While training and testing our Model on Tesla K80 , following was the train and test speed:

- Train Speed: 4 it/s
- Test Speed: 7 it/s

Results:

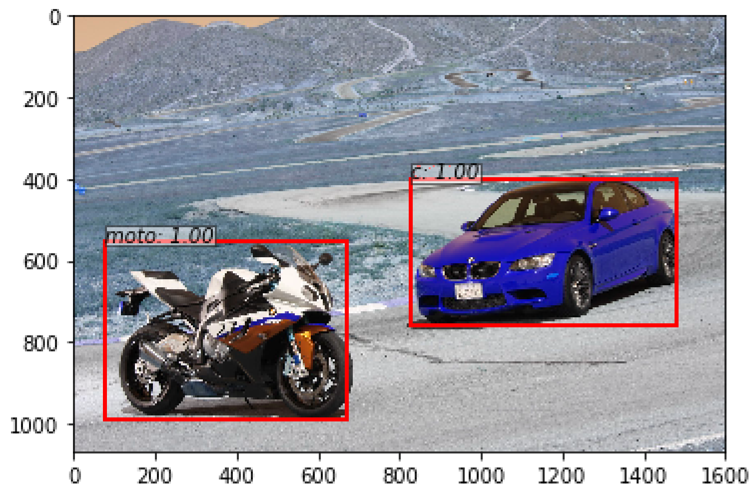
In the below figures you can see :

1. A bus has been detected
2. A car has been detected
3. A car and a motorbike is detected
4. A car has been detected

Note : In our results c stands for car and motto for motorbike.


bus: 0.91
Fernbus
Boom
alamy stock photo
0
200
400
600
800
0
200
400
600
800
1000
1200


car
1.00
0
25
50
75
100
125
150
175
0
50
100
150
200
250



Model evaluation and validation :

	Implementation	mAP
1	origin paper [1]	0.699
2	Our vehicle detection Model	0.7053

Implementation	GPU	Inference	Training
origin paper [1]	K40	5 fps	NA
vehicle-detection-model	TITAN Xp	15-17fps	6fps

Justification:

1. On a system which is meant for Vehicle Detection we couldn't have used a model which predicts other classes too along with vehicles.Hence we came up with a model which is meant just for vehicle detection.
2. On a low end GPU system such as (940 MX) we are able to process videos at a rate of 1 fps.
3. If we use higher end GPU such as Titan Xp we can process videos as 6fps.

Conclusion:

Through this capstone project we came up with a problem statement to detect vehicles. This resulted in a model which can detect vehicles with a state of art accuracy and a processing time where it doesn't miss any vehicle. As we already set up an environment(boom barriers) where the speed of incoming vehicles was slow.To implement the same on a barrier less traffic we will have to introduce high end GPUs. We also need to train our model for more class of vehicles.

References :

[1] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, https://arxiv.org/abs/1506.01497