Real-Time Vehicle and its License Plate Detection

Sudhir Kumar Suman

IIT Bombay

Aim:

The aim of this project is to detect a vehicle(Motorcycle, Auto-rikshaw, Bus, Car, Truck) and its license plate present in the frame.

Abstract:

In the last couple of years, the number of vehicles running on the road has increased exponentially in India. With this drastic increase in the number of vehicles, it is becoming too difficult to keep track of each and every vehicle for the purpose of traffic management and law and order management. It is also becoming more important in big industries and colleges to keep track of each vehicles entering into their premises and the amount of time spent by the vehicles inside it. In order to automate this process, the system needs to easily identify the vehicle and detect its License Plate.

Solution Approach:

The YOLO implementations are amazing tools that can be used to start detecting common objects in images and videos. However, there are many cases when the object we want to detect is not part of the popular dataset. In such cases, we need to create our own training set and execute our own training. Vehicle and its License Plate detection are one such case. I have used tiny-yolov3 to detect the desired classes and for this purpose have collected around 1800+ images.

Dataset

The most time consuming and important step towards completing this project was the collection and annotation of data. I have collected the image of a vehicle of desired class from Google Image, Kaggle and by clicking some pictures of vehicles from the roads so that the license plate of vehicles is visible. I have collected around 1800+ images and manually annotated all the

desired classes in the image. Each class consists of at least 350 images in training dataset. Later converted the coordinates of the annotated object in YOLO format that is:

<object-class> <x_center> <y_center> <width> <height>

Where:

- **<object-class>** - integer object number from 0 to (classes-1)
- **<x_center> <y_center> <width> <height>** - float values relative to width and height of image, it can be equal from **(0.0 to 1.0]**
- for example: **<x> = <absolute_x> / <image_width>** or **<height> = <absolute_height> / <image_height>**
- atention: **<x_center> <y_center>** - are center of rectangle (are not top-left corner)

Object Class

Vehicle	Label Id
Car	0
Truck	1
Bus	2
Motorcycle	3
Auto	4
CarLP	5
TruckLP	6
BusLP	7
MotorcycleLP	8
AutoLP	9

For example for img1.jpg you will be created img1.txt containing:

```
1 0.716797 0.395833 0.216406 0.147222
0 0.687109 0.379167 0.255469 0.158333
1 0.420312 0.395833 0.140625 0.166667
```

Configuring Files:

To train Yolov3 on the custom dataset we need certain specific files which will tell Yolo how and what to train. These files are

1. obj.data
2. obj.names
3. obj.cfg

Obj.data

This basically says that we are training 10 classes, what the train and validation files are and which file contains the name of the object we want to detect. During training save the weight in a backup folder.

```
classes = 10
train = train.txt
valid = test.txt
names = obj.names
backup = backup
```

Obj.names

Every new category must be in a new line and its category number be the same what we have used at the time of annotating data.

```
Car
Truck
Bus
Motorcycle
Auto
CarLP
TruckLP
BusLP
MotorcycleLP
```

Obj.cfg

Just copied the tiny-yolov3.cfg files and made few changes in it.

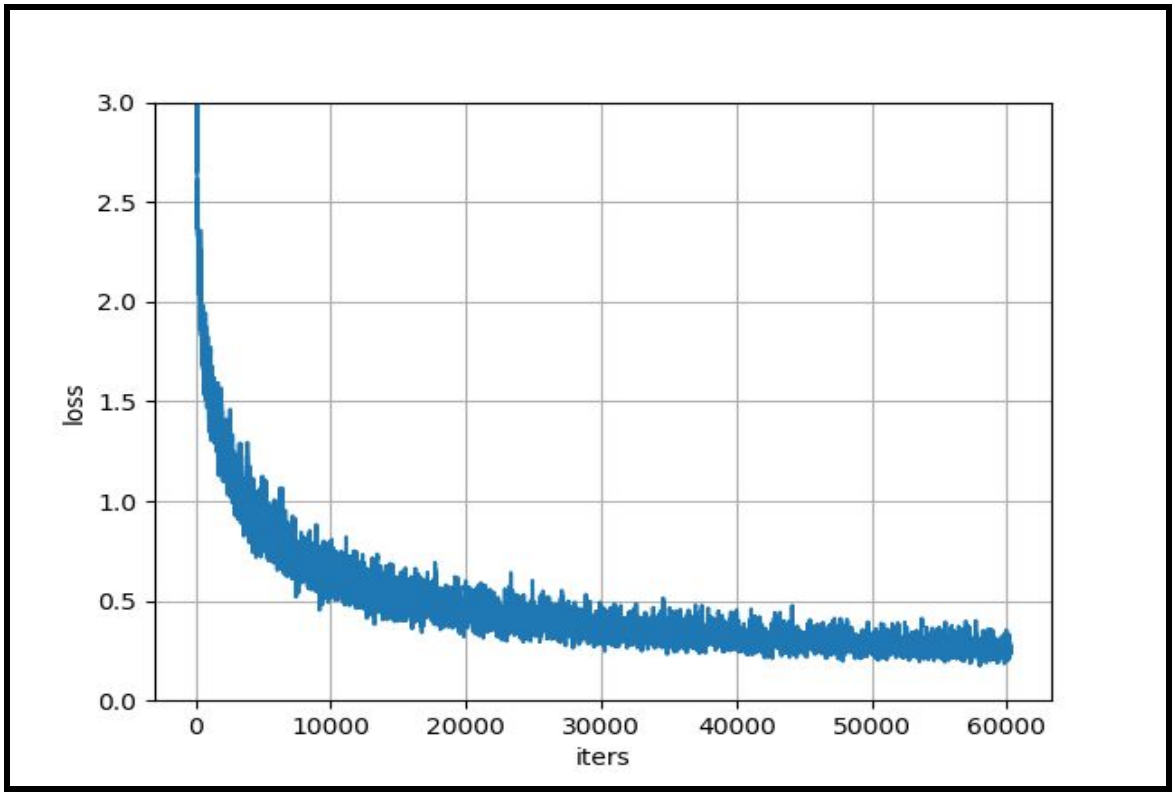
- In line 3, set `batch=24` to use 24 images for every training step.
- In line 4, set `subdivisions=8` to subdivide the batch by 8 to speed up the training process.
- In line 127, set `filters=(classes + 5)*3`, e.g `filter=45`.
- In line 135, set `classes=10`, number of custom classes.
- In line 171, set `filters=(classes + 5)*3`, e.g `filter=45`.
- In line 177, set `classes=10`, number of custom classes.

then, save the file

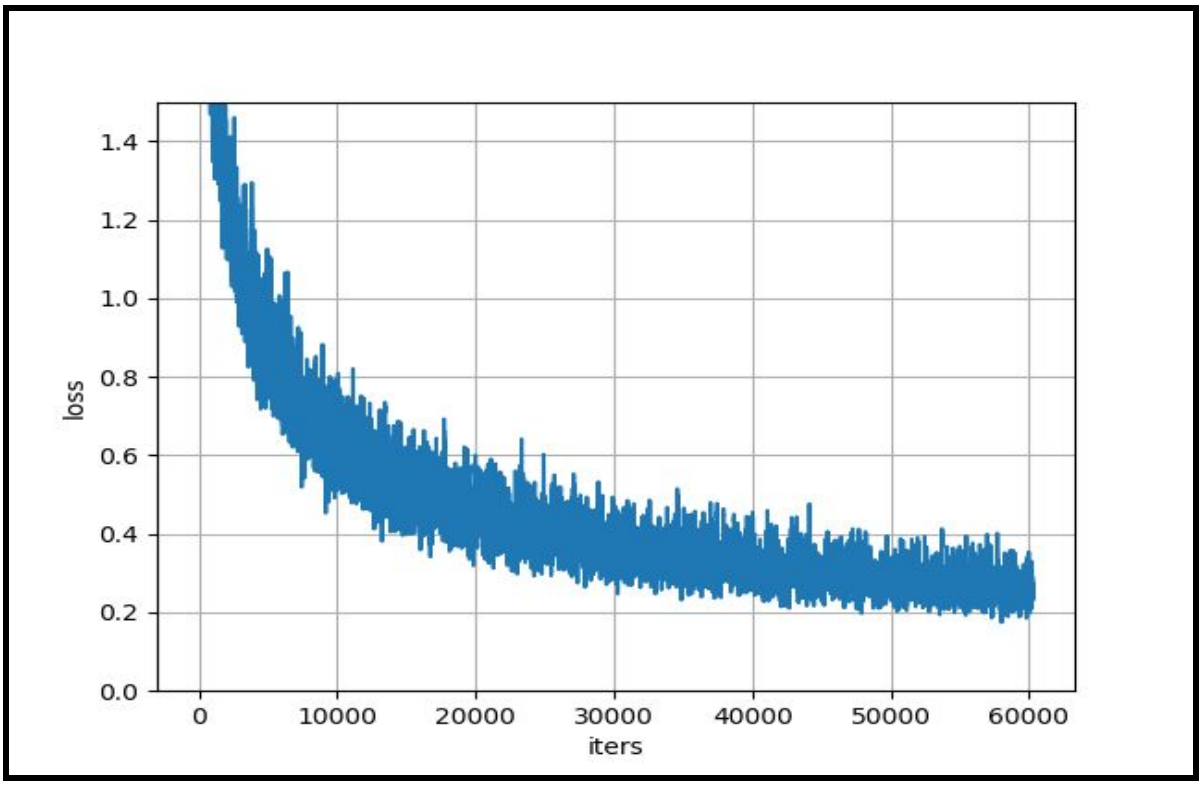
Training

I have trained tiny-yolov3 for about 60,000 iterations with the collected dataset and get the minimum loss of 0.2 with 0.001 learning rate, 0.9 Momentum and 0.0005 decay.

Here is the training loss curve:



Loss curve when loss is range is clamped between 0 and 1.5



For testing open the terminal and clone the repository with the command:

```
git clone https://github.com/SumanSudhir/Vehicle-and-Its-License-Plate-detection.git
cd Vehicle-and-Its-License-Plate-detection
make
```

Download weights from the link

http://storage.googleapis.com/sudhir_storage/Yolov3/obj_60000.weights in Vehicle-and-Its-License-Plate-detection directory. Move one of your images in the testing group to this directory and rename it as test.jpg.

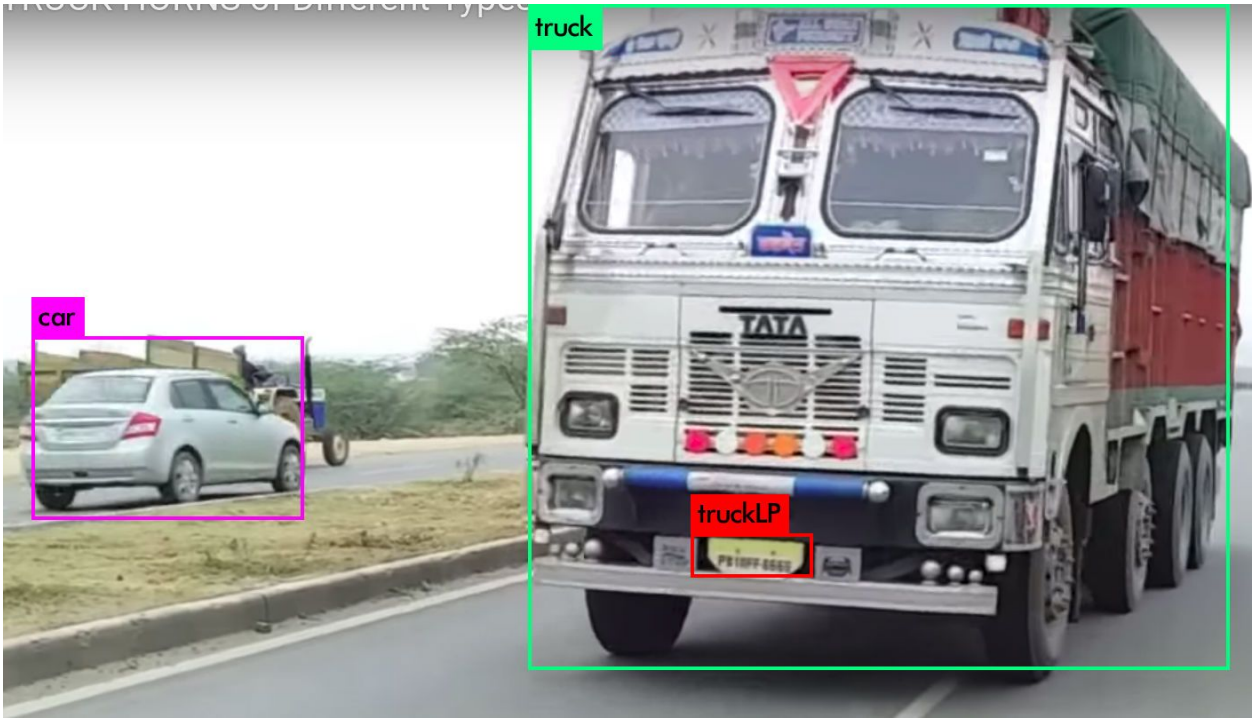
Next, open Terminal in Vehicle-and-Its-License-Plate-detection directory and run:

```
./darknet detector test obj.data cfg/obj.cfg obj_60000.weights test.jpeg
```

In the terminal, you will see the class of object detected and the resulting image will be stored in the directory with name prediction.jpg

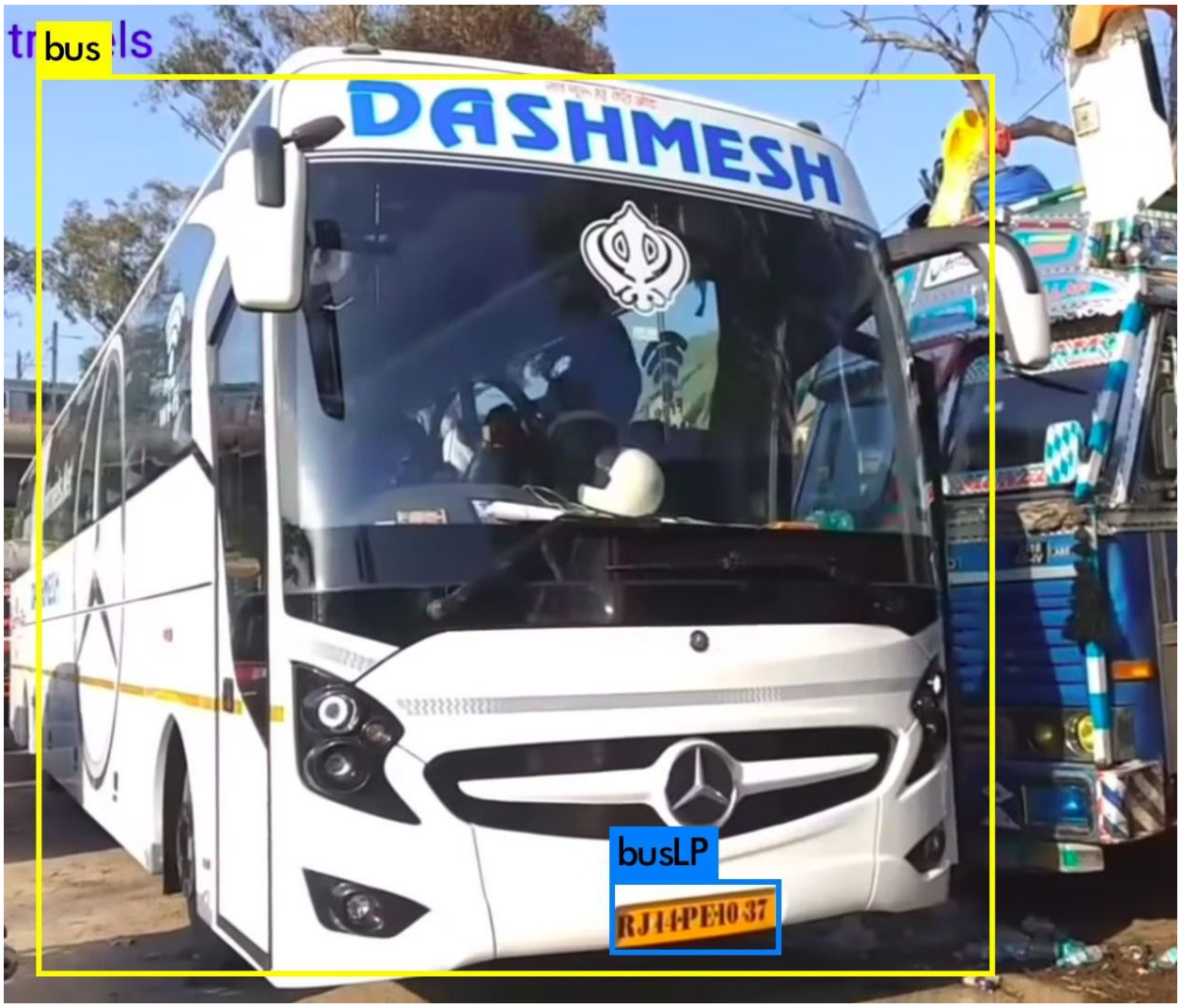
Some Results:

After training the model for 50,000 iterations below are some results:




motorcycle
motorcycleLP


car
carLP
S MB 5500


bus
DASHMESH
busLP
RJ44PE4037


bus
auto
autoLP
CNG
bus
auto
auto
auto
autoLP

Challenges:

- In India, the size and position of the License plate of the same vehicle category vary from one another which make the detection of License plate more challenging and leads to less accuracy while detecting it.
- In this one week, it was difficult to collect sufficient data and manually annotate all of them as annotation was taking too much of time. I have tried my best to collect 1800+ dataset but increasing the dataset will surely decrease the loss beyond 0.15 and increase the accuracy of the model.

References:

- For the purpose of understanding and training tiny-yolov3 on the custom dataset, I took help from this awesome repositoryhttps://github.com/AlexeyAB/darknet
- Which is the implementation of paper https://pjreddie.com/media/files/papers/YOLOv3.pdf
- For the purpose of manually annotating the data I used the BBox-Label-Tool