Real Time Vehicle Recognition and Tracking

A Project Report

submitted by

ANUJA REDDY MANDLA

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

under the guidance of

Dr. KAUSHIK MITRA

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Real Time Vehicle Recognition and Tracking**, submitted by **Anuja Reddy Mandla**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by her under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Kaushik Mitra
Research Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 5th May 2019

ACKNOWLEDGEMENTS

I would like to thank my guide, **Dr. Kaushik Mitra** for his invaluable support and for steering me in the right direction whenever I needed help.

I would also like to acknowledge **Dr. Gita Krishnan Ramadurai** as the co-guide of this project, and I am grateful for his valuable insights into the project.

I would like to thank my project mates **Nithya Muralikrishnan** and **Manoj Bharadhwaj** for helping me out through the project. I have greatly benefited from my discussions with them.

Were it not for the infallible faith of my parents and sister, I would not be present here. They have been a constant support throughout my life and I am forever grateful to them for their blessing and encouragement.

Lastly, I would like to give a huge shout-out to every one of my friends who have made my experience at IIT Madras, a treasured one.

ABSTRACT

KEYWORDS: Neural Networks; YOLO; Faster-RCNN; SSD; Kalman filter

The rapid pace of developments in Artificial Intelligence (AI) is providing unprecedented opportunities to enhance the performance of different industries and businesses, including the transport sector. The innovations introduced by AI include highly advanced computational methods that mimic the way the human brain works. The successful application of AI requires a good understanding of the relationships between AI and data on one hand, and transportation system characteristics and variables on the other hand. Moreover, it is promising for transport authorities to determine the way to use these technologies to create a rapid improvement in relieving congestion, making travel time more reliable to their customers and improve the economics and productivity of their vital assets. The aim of this project is to build mechanisms which helps in studying traffic-flow in Indian Conditions in Real-Time. We first look at various deep learning techniques and architectures used worldwide to address transportation problems. Then we look at how these pre-trained architectures are working in Indian scenarios and compare them to the results of same architectures trained on Indian traffic. Then we talk about the working of kalman filter and how we used it to track and count vehicles in traffic.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

YOLO	You Only Look Once
RCNN	Region based Convolutional Neural Network
SSD	Single-Shot Multi-Box Detector
NP	Number Plate

CHAPTER 1

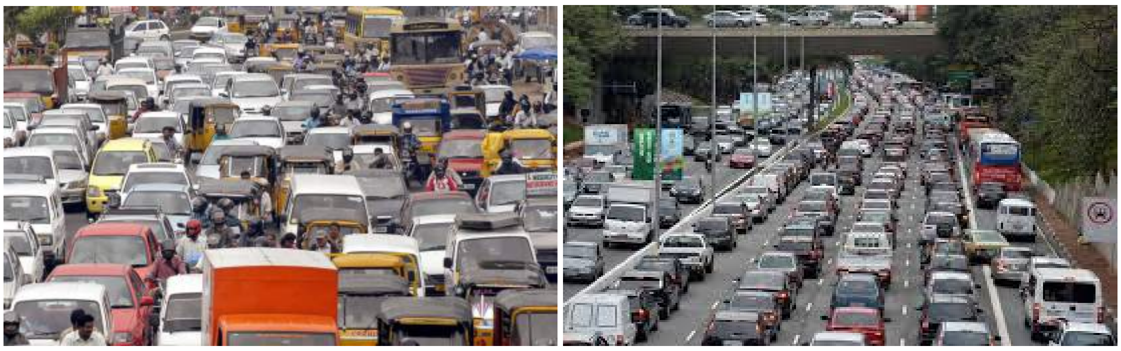
INTRODUCTION

There are several challenges that are persistent throughout the transportation industry and that have plagued this sector ever since its inception. Safety is arguably the most important consideration for those working within the travel or transportation industries. By monitoring traffic on the roads we can reduce accidents happening on the road by detecting vehicles moving with high velocity etc. Not only road accidents we can also catch frauds by building a model which can recognise and track them in traffic. We can solve many other issues by monitoring traffic on roads.

Application of AI in transportation industry is driving the evolution of next generation of Intelligent Transportation Systems. Artificial Intelligence and its branch Machine Learning are enabling transportation agencies, cities and soon private car owners to harness the power of modern compute and communication technologies and make mobility a much safer and greener activity.

Due to its processing, control and optimization capabilities, artificial intelligence could be applied to traffic management and decision-making systems in order to enhance and streamline traffic management and make our roads smarter.The predicative abilities of AI are also of huge benefit to traffic management systems as they are able to recognize the physical and environmental conditions that can lead to or be the result of heavier traffic flow and congestion. They can then in turn automatically suggest alternate routes to relieve any traffic that may have formed.

Since India is a country with high population ,its metrocities have high population. Currently there are 10 cities which have more than 10 million population and some of those even have population exceeding 20 and 25 million like Mumbai and Delhi having population as 26 million and 28 million respectively which also results in high no. of vehicles. Hence it becomes very difficult to control traffic in India. In developed countries, vehicles travel only in lanes i.e if there are three lanes on the raod, vehicles also travel in three lanes, but in India it's never that case. Fig 1.1 tells us the importance



(a) Indian Traffic (b) Foreign Country Traffic

Figure 1.1: Indian vs Foriegn Traffic

of improvements to be made in Indian Traffic Management System, in order to get traffic under control.

In this project we use **Deep Learning methods** to detect and recognize different vehicles in traffic in **real time** very specific to Indian Conditions. Using **Kalman filter** we also built a model which counts number of vehicles by continuously **tracking** them. We made our models robust to day and night conditions and also to front view and back-view of vehicles.We also made the model robust to different traffic conditions by collecting data from various streets having different traffic flow.

CHAPTER 2

BACKGROUND

Artificial Intelligence : The word Artificial Intelligence comprises of two words "Artificial" and "Intelligence". Artificial refers to something which is made by human or non natural thing and Intelligence means ability to understand or think. There is a misconception that Artificial Intelligence is a system, but it is not a system .AI is implemented in the system. There can be so many definition of AI, one definition can be "**It is the study of how to train the computers so that computers can do things which at present human can do better.**"Therefore It is a intelligence where we want to add all the capabilities to machine that human contain.

Machine Learning : A subset of **artificial intelligence (AI)**, **machine learning (ML)** is the area of computational science that focuses on analyzing and interpreting patterns and structures in data to enable learning, reasoning, and decision making outside of human interaction. Simply put, machine learning allows the user to feed a computer algorithm an immense amount of data and have the computer analyze and make data-driven recommendations and decisions based on only the input data. If any corrections are identified, the algorithm can incorporate that information to improve its future decision making.

Machine learning is made up of three parts :

- The computational algorithm at the core of making determinations
- Variables and features that make up the decision
- Base knowledge for which the answer is known that enables (trains) the system to learn

Initially, the model is fed parameter data for which the answer is known. The algorithm is then run, and adjustments are made until the algorithm's output (learning) agrees with the known answer.

Deep learning : Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain

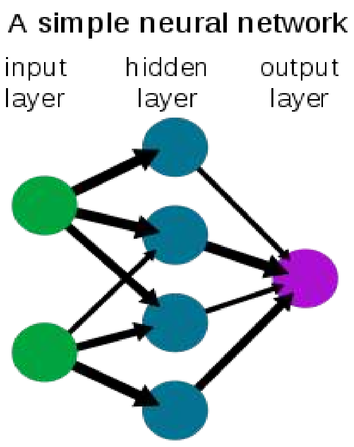


Figure 2.1: Basic Neural Network

so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now.

Deep learning surrounds us every day, and this will only increase with time. Whether you are are thinking about cars that drive autonomously or even have some new technology like parking assistance, traffic control, or face recognition technology at airports.

2.1 Neural Networks

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problems.Fig 2.1 is a basic neural network. The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

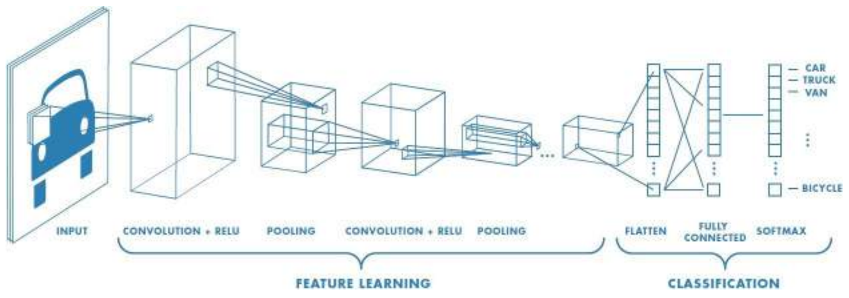


Figure 2.2: Example of Convolutional Neural Network
[14]

2.2 Convolutional Neural Networks

One of the most popular uses of this architecture is **Image Classification** as shown in Figure 2.2 . A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layers i.e. activation function, pooling layers, fully connected layers and normalization layers.

Receptive field : In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its receptive field.

Convolutional Layer : Each convolutional neuron processes data only for its receptive field. Convolution operation allows the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. In this way, it resolves the vanishing or exploding gradients problem in training traditional multi-layer neural networks with many layers by using backpropagation.[citation needed]

Pooling : Convolutional networks may include local or global pooling layers. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

Fully connected : Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Weights : Each neuron in a neural network computes an output value by applying some function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is specified by a vector of weights and a bias (typically real numbers). Learning in a neural network progresses by making incremental adjustments to the biases and weights.

A distinguishing feature of CNNs is that **many neurons share the same filter**. This **reduces memory** footprint because a single bias and a single vector of weights is used across all receptive fields sharing that filter, rather than each receptive field having its own bias and vector of weights.[16]

2.3 Popular Object Detectors

Detection is a more complex problem than classification, which can also recognize objects but doesn't tell you exactly where the object is located in the image and a classifier won't work for images that contain more than one object.
In our project we compared three architectures to detect vehicles in traffic - **Faster-RCNN, YOLO, SSD**.

2.3.1 Faster-RCNN

Faster-RCNN [1] is one of the most well known object detection neural networks.

The 3 networks of Faster-RCNN :

- Feature Network
- Region Proposal Network (RPN)
- Detection Network.

The **Feature Network** is usually a well known pre-trained image classification network such as VGG minus a few last/top layers. The function of this network is to

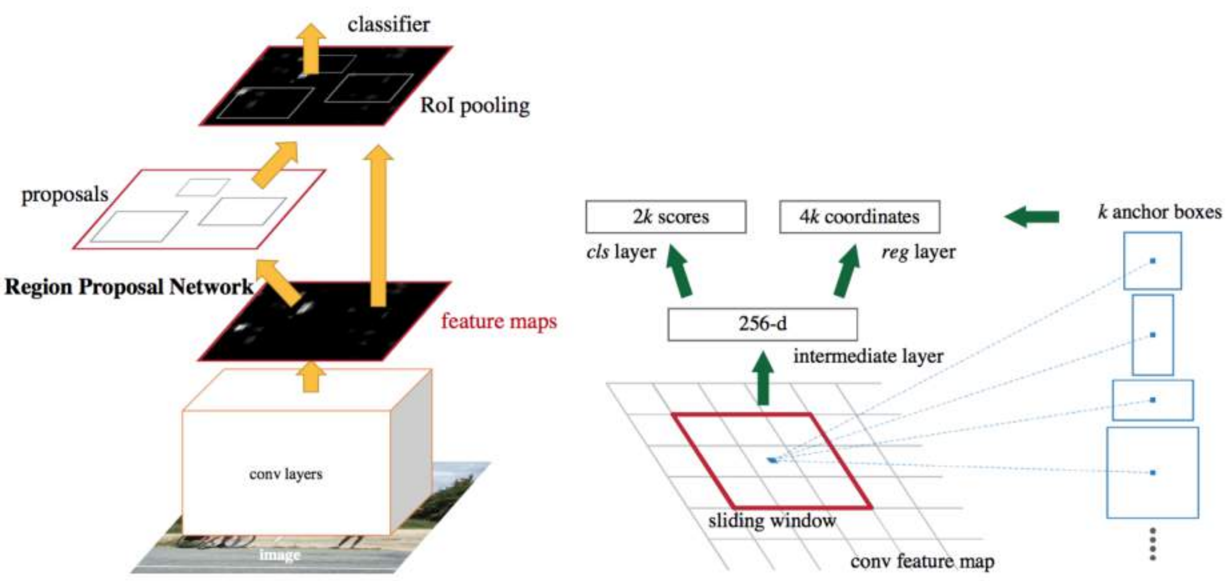


Figure 2.3: Working of Faster-RCNN [1]

generate good features from the images. The output of this network maintains the the shape and structure of the original image (i.e. still rectangular, pixels in the original image roughly gets mapped to corresponding feature "pixels", etc.)

The **RPN** is usually a simple network with a 3 convolutional layers. There is one common layer which feeds into a two layers-one for classification and the other for bounding box regression. The purpose of RPN is to generate a number of bounding boxes called Region of Interests (ROIs) that has high probability of containing any object. The output from this network is a number of bounding boxes identified by the pixel co-ordinates of two diagonal corners, and a value (1, 0, or -1, indicating whether an object is in the bounding box or not or the box can be ignored respectively).

The Detection Network (sometimes also called the RCNN network) takes input from both the Feature Network and RPN , and generates the final class and bounding box. It is normally composed of 4 Fully Connected or Dense layers. There are 2 stacked common layers shared by a classification layer and a bounding box regression layer. To help it classify only the inside of the bounding boxes, the features are cropped according to the bounding boxes.

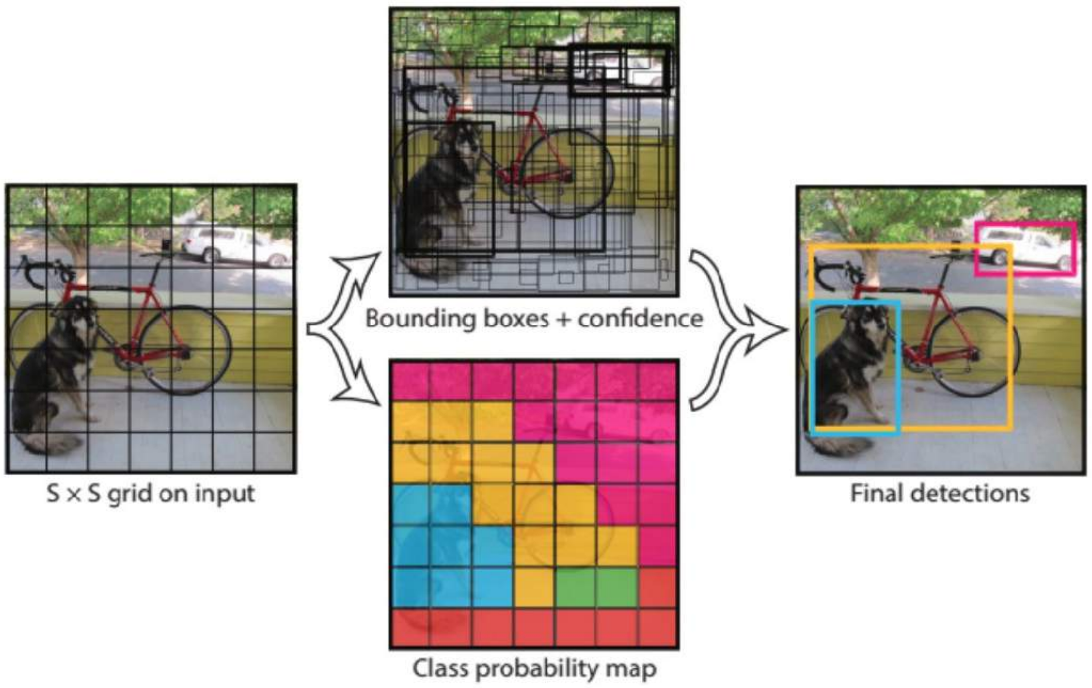


Figure 2.4: Working of YOLO [6]

2.3.2 YOLO

YOLO [6] is not a traditional classifier that is repurposed to be an object detector. YOLO actually looks at the image just once (hence its name: **You Only Look Once**) but in a clever way.

YOLO divides the input image into an S X S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.

Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $Pr(Object) * IOU$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the **intersection over union (IOU)** between the predicted box and the ground truth.

Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, Pr(Classi |Object). These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B. At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$Pr(Classi|Object) * Pr(Object) * IOU = Pr(Classi) * IOU \tag{2.1}$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object. As it divides the image into an SxS grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an SxSx(B*5+C) tensor.

2.3.3 SSD - Sing Shot Multibox Detector

The SSD [3] approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections (for bounding boxes with most overlap keep the one with highest score).

The early network layers are based on a standard architecture used for high quality image classification (truncated before any classification layers), which we will call the base network . We then add auxiliary structure to the network to produce detections with the following key features:

Multi-scale feature maps for detection: We add convolutional feature layers to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales. The convolutional model for predicting detections is different for each feature layer (cf Overfeat[4] and YOLO[5] that operate on a single scale feature map). **Convolutional predictors for detection:** Each added feature layer (or optionally an existing feature layer from the base network) can produce a fixed set of detection predictions using a set of convolutional filters. For a feature layer of size m X n with p channels, the basic element for predicting parameters of a potential detection is a 3 X 3 X p small kernel that produces either a score for a category, or a

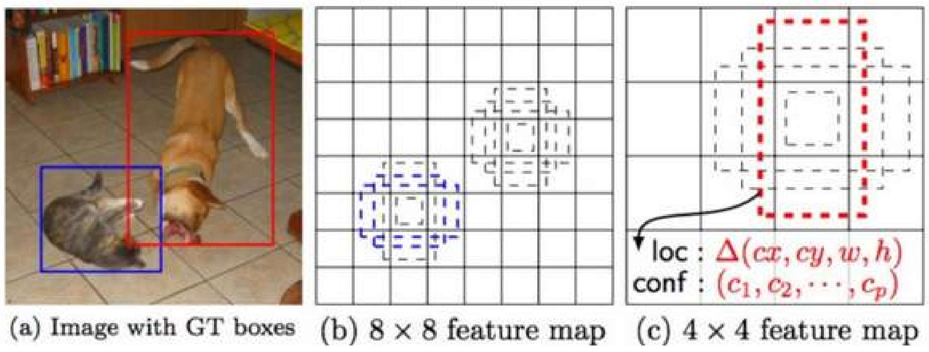
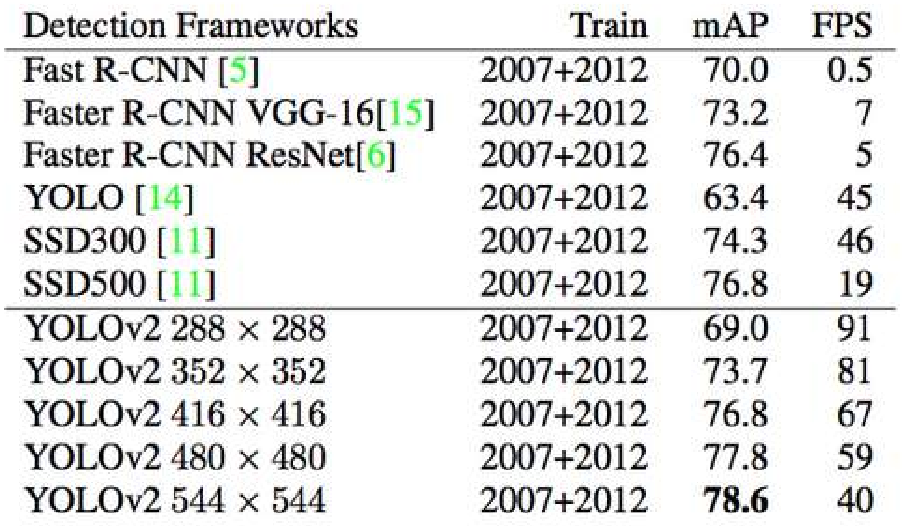


Figure 2.5: SSD Framework [3]

shape offset relative to the default box coordinates. At each of the m X n locations where the kernel is applied, it produces an output value. The bounding box offset output values are measured relative to a default box position relative to each feature map location (cf the architecture of YOLO that uses an intermediate fully connected layer instead of a convolutional filter for this step).

Default boxes and aspect ratios: We associate a set of default bounding boxes with each feature map cell, for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box relative to its corresponding cell is fixed. At each feature map cell, we predict the offsets relative to the default box shapes in the cell, as well as the per-class scores that indicate the presence of a class instance in each of those boxes. Specifically, for each box out of k at a given location, we compute c class scores and the 4 offsets relative to the original default box shape. This results in a total of (c + 4)k filters that are applied around each location in the feature map, yielding (c + 4)kmn outputs for a m X n feature map. For an illustration of default boxes, please refer to Figure 2.5. Our default boxes are similar to the anchor boxes used in Faster R-CNN, however we apply them to several feature maps of different resolutions. Allowing different default box shapes in several feature maps let us efficiently discretize the space of possible output box shapes.

Figure 2.6 shows comparision of performances of YOLO, SSD and Faster-RCNN on Pascal VOC. We could see that YOLO v2 has best mAP over Faster-RCNN and SSD. And for our project we used next better version of YOLO v2 which is YOLO v3 [2].



Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Figure 2.6: Comparision of Faster-RCNN , YOLO and SSD [5]

2.4 Tracking

Video tracking is the process of locating a moving object (or multiple objects) over time using a camera. It has a variety of uses, some of which are: human-computer interaction, security and surveillance, video communication and compression, augmented reality, traffic control, medical imaging and video editing.Video tracking can be a time consuming process due to the amount of data that is contained in video. Adding further to the complexity is the possible need to use object recognition techniques for tracking, a challenging problem in its own right. Two techniques that are very popular to track objects are :

Kalman Filter :

This algorithm can predict future positions based on current position. It can also estimate current position better than what the sensor is telling us. It will be used to have better association.

Hungarian Algorithm :

This algorithm can tell if an object in current frame is the same as the one in previous frame. It will be used for association and id attribution.

2.4.1 Kalman Filter

Kalman filtering [4], also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe. The filter is named after Rudolf E. Kalman, one of the primary developers of its theory. Kalman filter can be modeled by :

State : The state is a description of all the parameters we will need to describe the current system and perform the prediction. For example, we'll use two numbers: The current vertical position (y), and our best estimate of the current slope (let's call it m). Thus, the state is in general a vector, commonly denoted **x**, and we can include many more parameters to it if we wish to model more complex systems.

Model : The model describes how we think the system behaves. In an ordinary Kalman filter, the model is always a linear function of the state. In our simple case, our model is:

$$y(t) = y(t-1) + m(t-1) \tag{2.2}$$

$$m(t) = m(t-1) \tag{2.3}$$

Expressed as a matrix, this is:

$$x_t = \begin{pmatrix} y(t) \\ m(t) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y(t-1) \\ m(t-1) \end{pmatrix} = Fx_{t-1} \tag{2.4}$$

We add an additional term to the state - **the process noise,** v_t which is assumed to be normally distributed. Although we don't know the actual value of the noise, we assume we can estimate how "large" the noise is, as we shall presently see. All this gives us the state equation, which is simply:

$$x_t = Fx_{t-1} + v_{t-1} \tag{2.5}$$

Measurement : When we get new data, our parameters should change slightly to refine our current model, and the next predictions. What is important to understand is that one does not have to measure exactly the same parameters as the those in the state. For instance, a Kalman filter describing the motion of a car may want to predict the car's acceleration, velocity and position, but only measure say, the wheel angle and rotational velocity. In our example, we only "measure" the vertical position of the new points, not the slope. That is:

$$measurement = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} y(t) \\ m(t) \end{pmatrix} \tag{2.6}$$

In the more general case, we may have more than one measurement, so the measurement is a vector, denoted by **z**. Also, the measurements themselves are noisy, so the general measurement equation takes the form:

$$z_t = Hx_t + w_t \tag{2.7}$$

Where **w** is the **measurement noise**, and H is in general a matrix with width of the number of state variables, and height of the number of measurement variables.

CHAPTER 3

Detection and Recognition in Indian Traffic

We want to build a robust detector and classifier to classify vehicles especially for Indian Vehicles. Robust in the sense that classifer should be robust to day-time and night-time and also should be able to detect and classify vehicles from both front-side and back-side view of vehicles.
Hence we studied this project in 4 parts :

- Day-Time Front-View
- Day-Time Back-View
- Night-Time Front-View
- Night-Time Back-View

3.1 Datasets Used

In the whole project we used 4 different datasets - Pascal-VOC [10] , IITM-HeTra, Front-View and Back-View Datasets

3.1.1 IITM-HeTra Dataset

We used **IITM-HeTra Dataset** which is available on Kaggle[7] along with its annotations. We used images from dataset2 folder of IITM-HeTra which has only day-time front-view images. Sample images of this dataset are shown in Figure 3.1.
It has **1418** images. Each image in this dataset has **640x482** resolution. This Dataset has 4 classes - **Auto , Bus, Car, Person**. In the annotation files they labelled 2-wheelers as **Person**. In this project as we are focused to classify vehicles we changed the label **Person** to **Bike**.

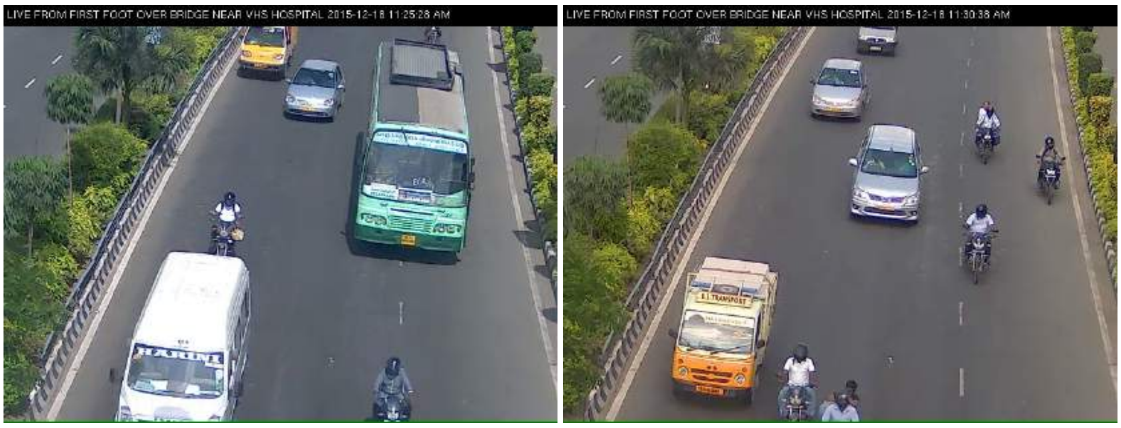
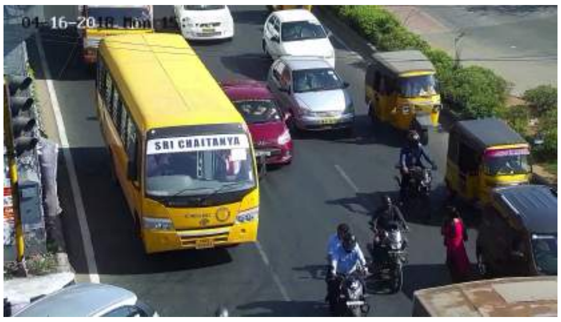

LIVE FROM FIRST FOOT OVER BRIDGE NEAR VHS HOSPITAL 2015-12-18 11:25:28 AM
LIVE FROM FIRST FOOT OVER BRIDGE NEAR VHS HOSPITAL 2015-12-18 11:30:38 AM

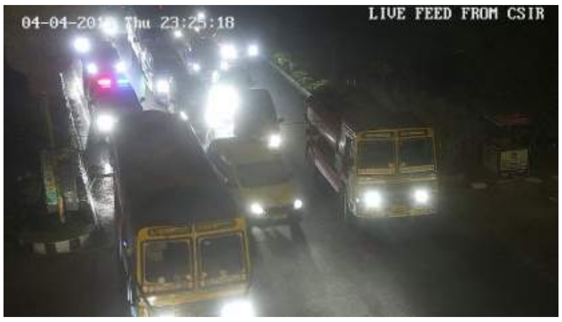
Figure 3.1: Sample Images from IITM-HeTra Dataset

3.1.2 Front-View and Back-View datasets

We also generated a new data-sets - **Front-View , Back-View** with images with very high resolution **1920x1080 pixels** to improve accuracy. Also this high resolution helps us in detecting and recognising number-plates of vehicles.The images in this dataset are taken from different streets in Chennai. Sample Images of this dataset are shown in Figures 3.2 and 3.3.

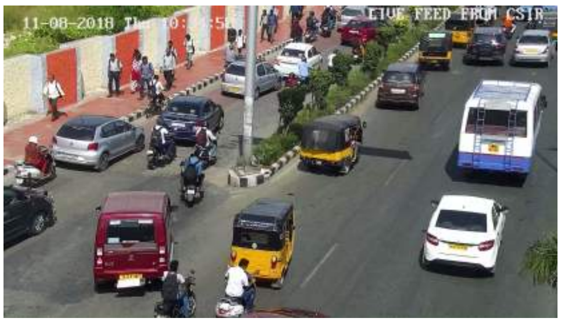


(a) Day Time Front View

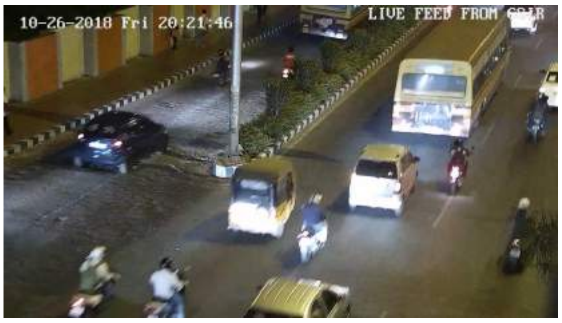


(b) Night Time Front View

Figure 3.2: Sample Images from Front-View Dataset



(a) Day Time Back View



(b) Night Time from Back-View

Figure 3.3: Sample Images from Back-View Dataset

The new dataset has 4 kinds of images.

- **Day-Time Front-View** (1503 images)
- **Night-Time Front-View** (650 images)
- **Day-Time Back-View** (125 images)
- **Night-Time Back-View** (570 images)

We annotated these datasets(Front-View and Back-View) with following classes - **Auto, Bike, Bus, Car, NP, Person**. We also annotated the truncated vehicles. The number of instances of each class are given in **Tabel 3.1**.

Label	IITM-HeTra	Front-View	Back-View	Total
Auto	598	1360	478	2436
Bike	3335	1248	247	4830
Bus	279	1419	458	2156
Car	2148	5716	1707	9571
Person	0	365	414	779
NP	0	4046	118	4164

Table 3.1: Count of instances of classes in each Dataset

3.2 Models, Training and Testing

Researchers have proposed a data efficient way for training neural network models by doing fine-tuning. In the process of fine-tuning, we have a base neural network which is trained on a large dataset (say source dataset). Given this trained model (or pre-trained model), we want to re-train the same model on a small dataset (say target dataset). We take the pre-trained model as a starting point for training the model and then modify its parameters according to the small target dataset. Since we are starting with an already trained model, we need fewer data points to do the fine-tuning. Thus with a small target dataset, we can train a large neural network model. Fine-tuning a pre-trained deep neural network is a standard practice in computer vision community.

YOLO : Many versions of YOLO exist, of we used YOLO v3 [2] for whole project. We modified the last layer in YOLO. We changed number of neurons from 80 to 6 in the last layer, as we are training only on 6 classes. This will also significantly reduce the size of weights saved as last layer in YOLO is fully connected. **yolov3.weights**[8] are pretrained weights of YOLO trained on Microsoft's COCO dataset. To fine-tune YOLO, we cannot use these pretrained weights directly as initial weights as the architecture is changed. So we removed weights in the last layer of yolov3.weights, and then fine-tuned these weights by training the whole architecture on IITM HeTra + Front-View + Back-View datasets. Pretrained YOLO can detect relevant classes like car, bike, bus, pedestrian. Hence we compared the outputs from new fine-tuned weights with the outputs of pretrained weights(yolov3.weights) to analyse the importance of training. PreTrained YOLO cannot detect Auto and Number Plate (NP).

PyFaster-RCNN : The paper **Training a deep learning architecture for vehicle detection using limited heterogeneous traffic data [9]** states that finetuning PyFaster-RCNN [1] with a smaller dataset performs poorly. And it suggested to train PyFaster-RCNN by augmenting PASCAL VOC[10] with smaller dataset to get good results. We followed the same procedure,i.e added Pascal VOC dataset to IITM-HeTra + Front-View + Back-View Dataset to make one large dataset. Then we trained PyFaster-RCNN on this big dataset. Pascal VOC dataset has around 10000 images of 20 different classes including cat, dog, train, boat along with relevant classes such as car, truck and bus. It takes longer time to train because there will be more number of computations because of large dataset.

Always tested on new images which are not part of training. Default parameters are used while training, fine-tuning and testing.

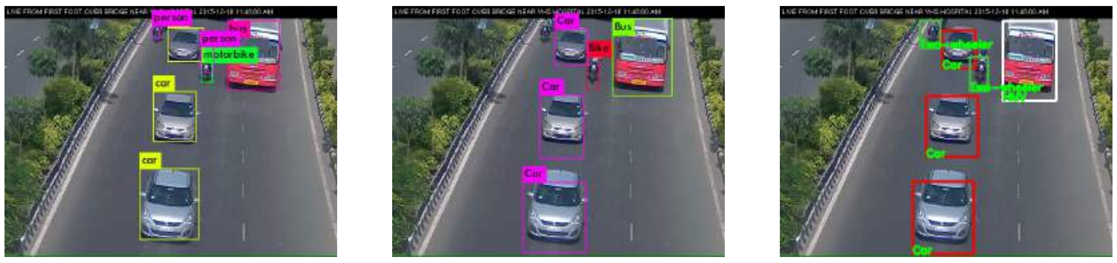
3.3 Results on Day-Time Images

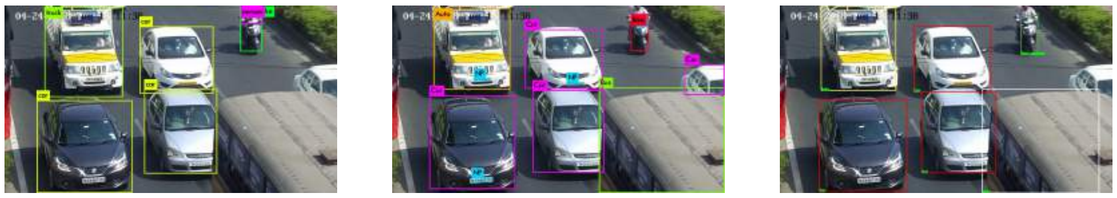
We tested pre-trained YOLO , finetuned YOLO and PyFasterRCNN on day time images and compared the results.

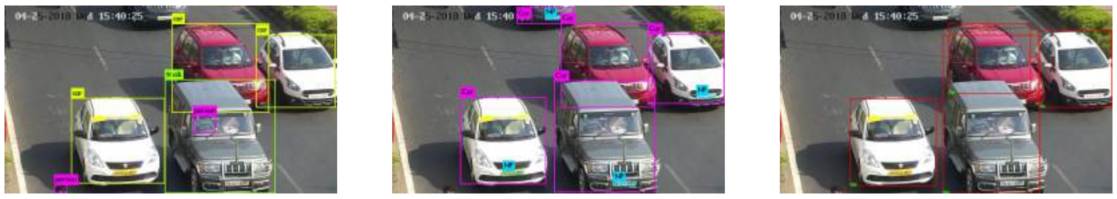
3.3.1 On Day-Time Front-View Images

From **Figure 3.4** we can infer that :

- Fine-Tuned YOLO and trained Faster-RCNN did better in detecting Truncated Vehicles, Autos. It is expected because our dataset included truncated vehicles ,


(a)


(b)


(c)

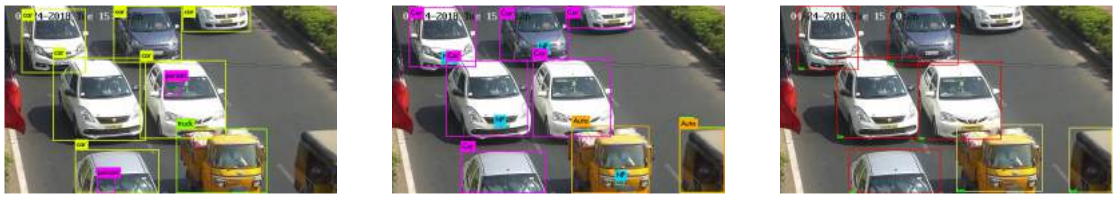

(d)

Figure 3.4: PreTrained-YOLO vs FineTuned YOLO vs PyFaster-RCNN

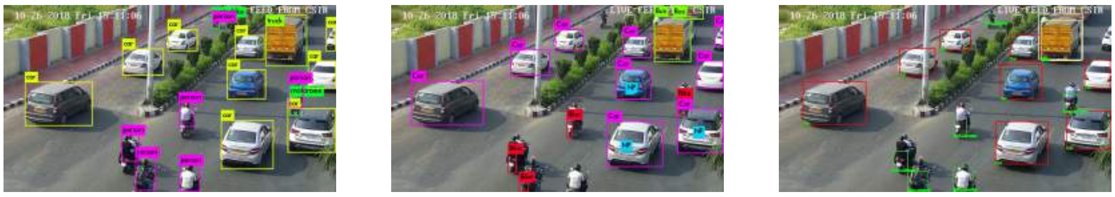
autos .

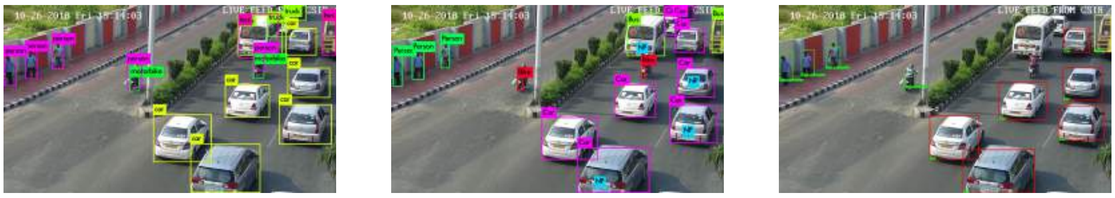
- We could observe that YOLO detected number plates better than Faster RCNN

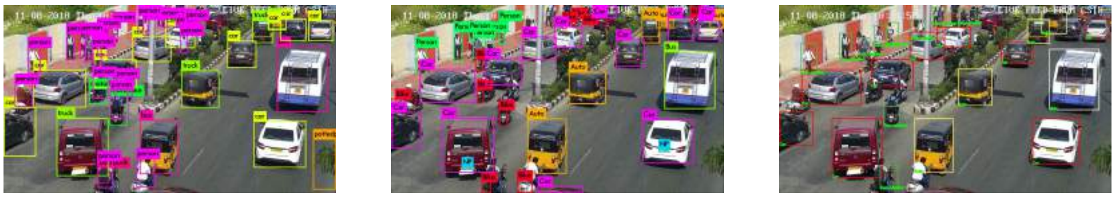
3.3.2 On Day-Time Back-View Images

From **Figure 3.5** we can infer that :

- Pre-Trained YOLO couldn't detect all bikes from back-view, which clearly says there is a need to train architectures with back-view of vehicle images too.
- Fine Tuned YOLO detected Bikes and Truncated Vehicles better than pre-trained YOLO
- Though most of the Number Plate dataset in the training data is from day-time front-view images, Fine Tuned YOLO decently enough recognised the number plates in back-view of vehicle.
- Py-Faster-RCNN performed poorly again in detecting number plates.


(a)


(b)


(c)

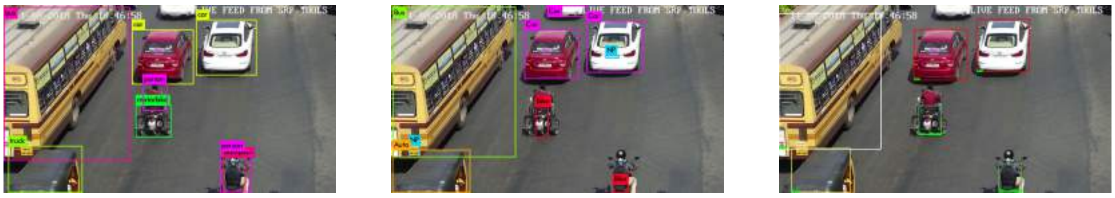

(d)

Figure 3.5: PreTrained-YOLO vs FineTuned YOLO vs PyFaster-RCNN

3.4 Results on Night-Time Images

During night time, head lights of vehicles will be ON and because of the glare from these head-lights , the vehicles will not be clearly visible and hence it becomes difficult for the detectors to detect and recognise them. Hence to improve the accuracy we need to reduce the glare from vehicles. Contrast- adjustment is one method we tried to reduce glare.

3.4.1 Methods tried to increase contrast of Night time images

Adaptive histogram Equalisation, Gamma Equalisation, Unsharp Masking, Flat-field correction, Reduce atmosperic gaze, Edge-aware local contrast manipulation , Decorrelation stretch are some of the filters which we used to make vehicles in the night-time images more clearer. **Figure 3.6** shows output of various filters applied on the first image in the figure. We could see that 4th image in the set has better clarity. It is output from adaptive histogram equalization over gamma equalization.

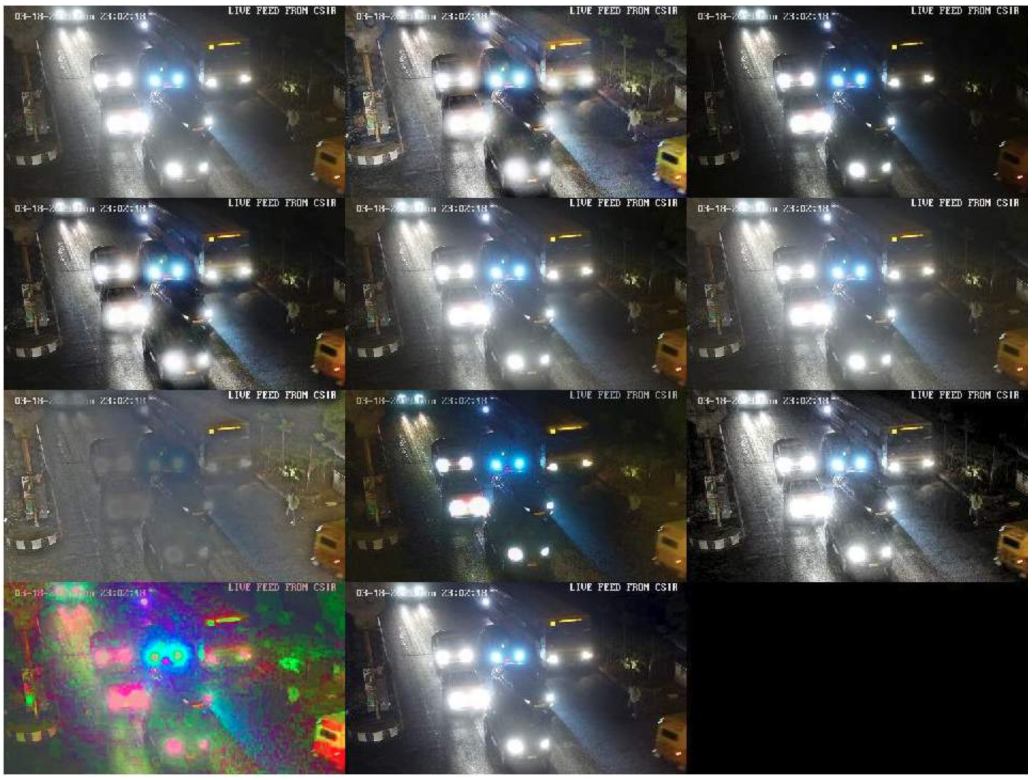


Figure 3.6: Results of contrast enhancement filters applied on a night time image

And we also observe that reduce haze has reduced the glare from headlights (8th image in Figure 3.6), but it also made the image more darker. But as compared to main image, all these changes are very small, and for deep-learning techniques using these filters will not affect the results. Hence we tried to train the network directly on night-time images without applying any filters.

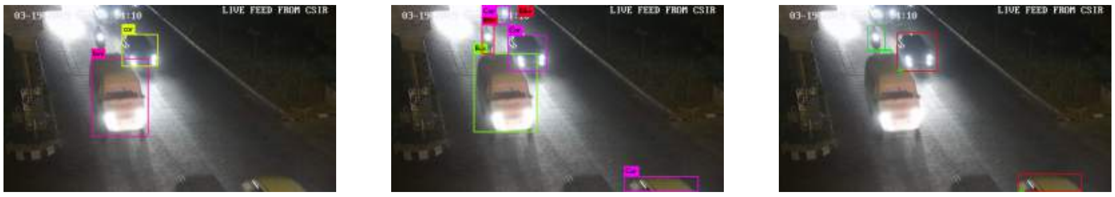
We tested pre-trained YOLO , finetuned YOLO and PyFasterRCNN on night time front-view and night time back-view images and compared the results.

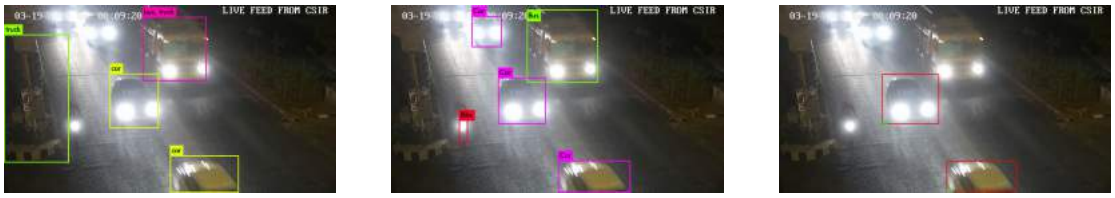
3.4.2 On Night-Time Front-View Images

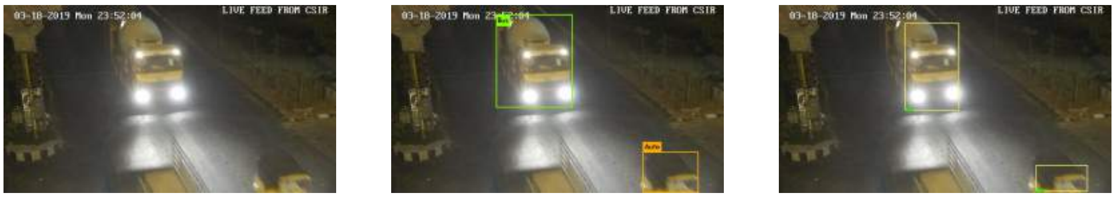
From **Figure 3.7** we can infer that :

- Clearly Fine-Tuned YOLO completely outperformed Pre-Trained YOLO and did better than trained Py-Faster-RCNN.
- It is able to detect vehicles which are difficult to recognise even with a human eye.

- It is observed that by training with night-time images makes a clear increase in the accuracy of detection and recognition of YOLO network


(a)


(b)


(c)

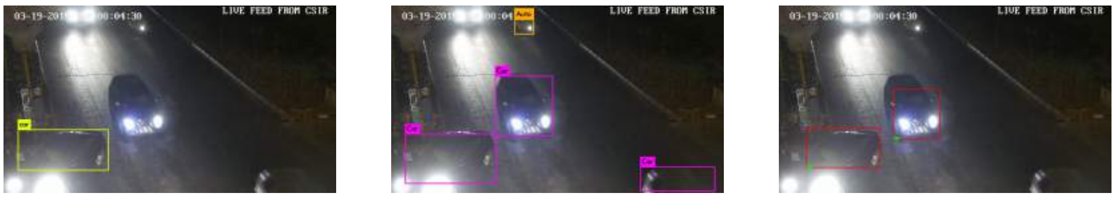
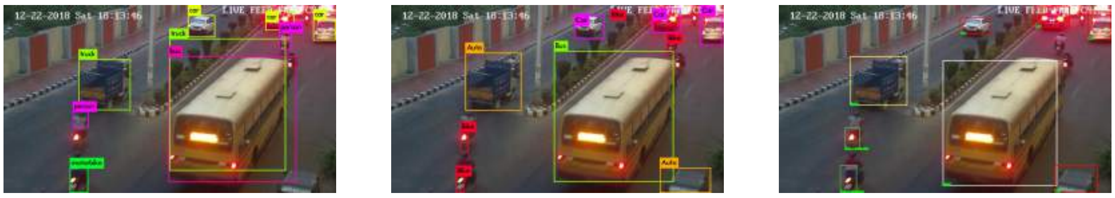

(d)

Figure 3.7: PreTrained-YOLO vs FineTuned YOLO vs PyFaster-RCNN

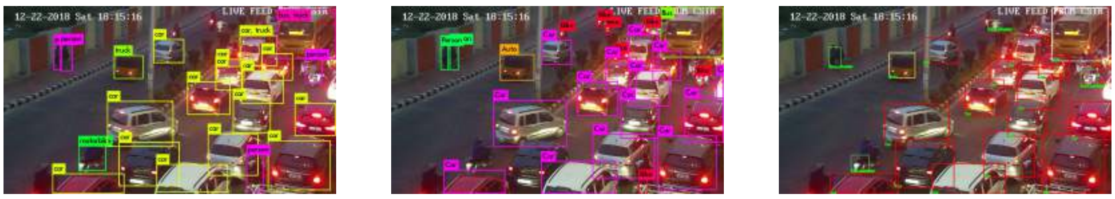
3.4.3 On Night-Time Back-View Images

From Figure 3.8 we can infer that :

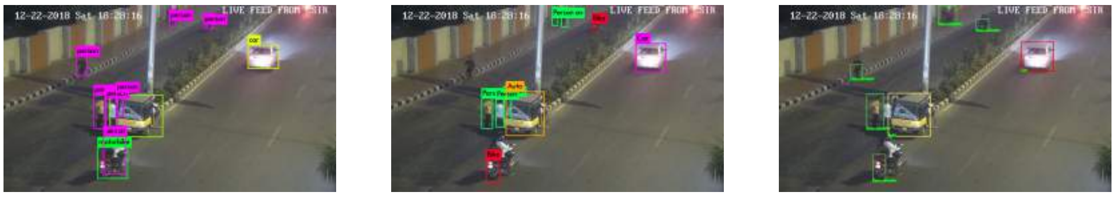
- Again Fine-Tuned YOLO performed better on bikes compared to other architectures.
- It also recognised Auto and Person classes well.



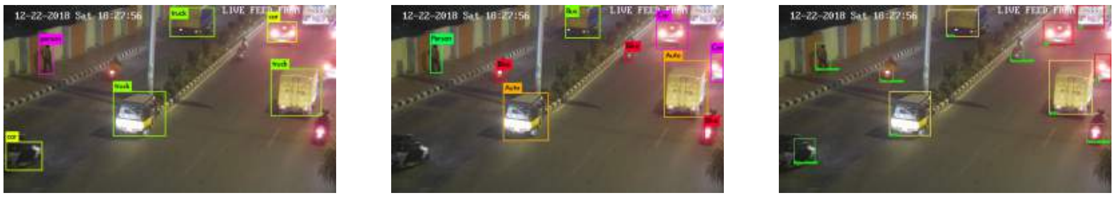
(a)



(b)



(c)



(d)

Figure 3.8: PreTrained-YOLO vs FineTuned YOLO vs PyFaster-RCNN

CHAPTER 4

Tracking

Using Py-Faster-RCNN [1], trained on Pascal VOC [10]+ IITM- HeTra+Front-View+Back-View datasets, as detector, Kalman filter [4] tracker code, counted number of different classes of vehicles in a chennai traffic sample video. Implemented the code memory efficiently by removing all information of a vehicle from memory as soon as it leaves the video. Also implemented different useful applications of this project like to track vehicles moving only in a particular direction and track vehicles only in specific regions of the video. As the bounding box size of an object keeps changing from time to time, the output wouldnt look appealing to eyes. Hence implemented a code to change size of box only at every 12th frame. On an average tracking happens at 13 frames/sec

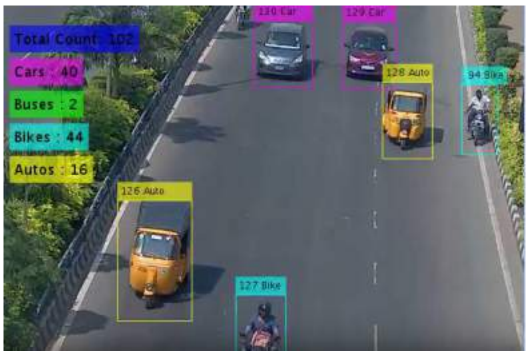


Figure 4.1: An image frame in a tracking Output

4.1 Results

We tested our implementation of tracking code on traffic from different regions, different traffic conditions and different times of the day.

4.1.1 On Chennai Traffic :

4.1.1.1 Track vehicles on a 1-way road :

We tested on 5min Chennai Traffic in which traffic moves in only one direction and has 234 vehicles. Our code could track 230 vehicles in that video.Table 4.1 gives count of different classes vehicles tracked in the video. Figure 4.2 is one of the last frames of the video.

Class	Actual Count	Observed count
Auto	31	29
Bike	108	107
Bus	8	8
Car	87	88

Table 4.1: Count of instances of each class

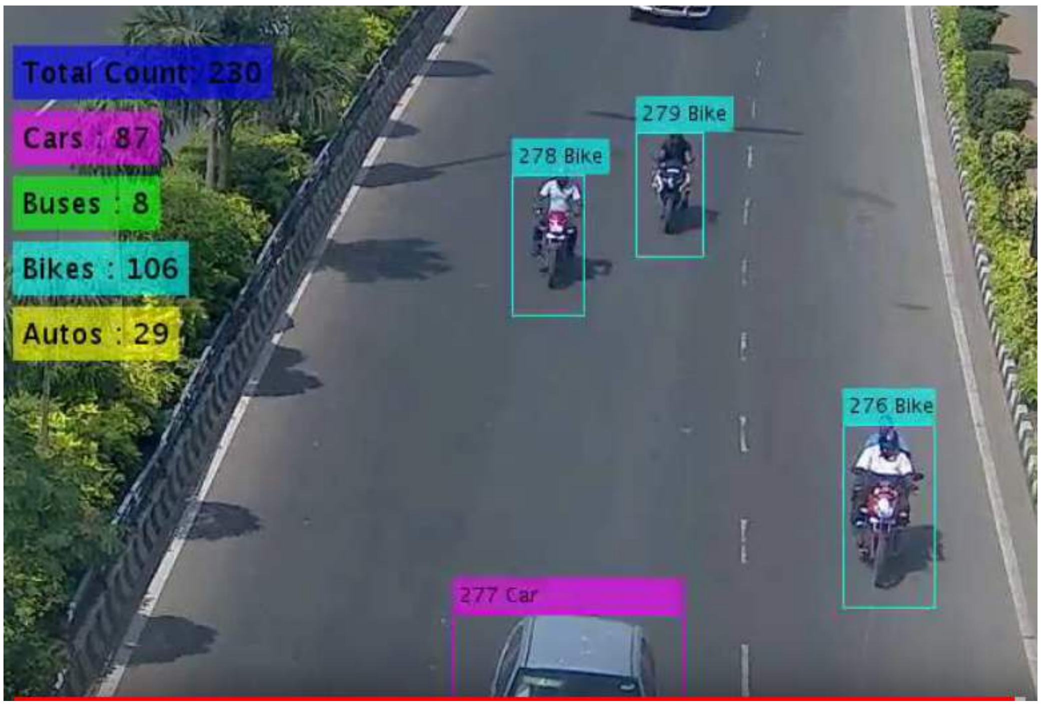


Figure 4.2: Tracking output on Chennai Traffic moving on 1-way road

4.1.1.2 Track vehicles on a 2-way road :

We also tested on traffic on a 2-way road in which vehicles move in both the directions. One of our 4min tested input has 144 vehicles moving towards camera and some vehicles moving away from camera.We implemented a code to track vehicles which are moving only towards camera, the code could count 145 vehicles in that 4min video.Table 4.2 gives count of different classes vehicles tracked in the video. Figure 4.3 is one of the last frames of the video.

Class	Actual Count	Observed count
Auto	13	14
Bike	75	74
Bus	4	4
Car	52	53

Table 4.2: Count of instances of each class

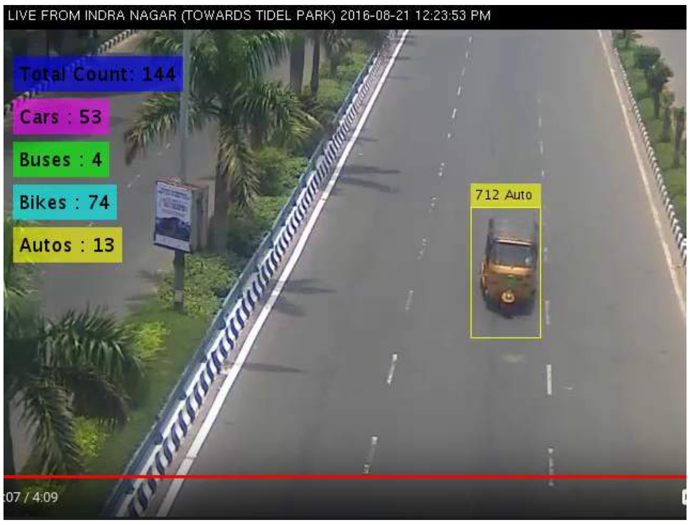


Figure 4.3: Tracking output on Chennai Traffic on a 2-way road

4.1.2 On Foreign Traffic :

Tested our code on many foreign traffic situations too, and we got satisfactory results. Figure 4.4 is an example frame of tracking on foreign traffic.

4.1.2.1 Region Select :

In the **Figure 4.5**, we could see that vehicles are occluded when they are far away from camera. In such situations, tracking should be done in the selected regions (need not be rectangles) where there is less occlusion, for more accurate information. Implemented

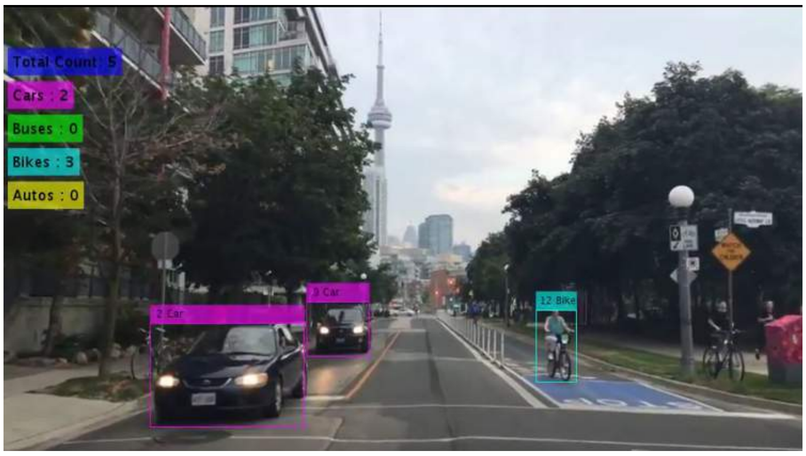


Figure 4.4: Frame of Tracking output on foreign traffic

a code which tracks vehicles only in the red shaded region of the video. Figure 4.5 is one of the frame of output video.

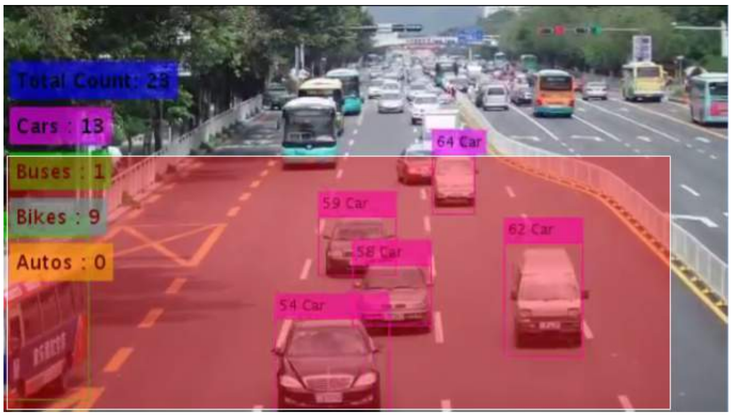


Figure 4.5: Tracking with Region Select

4.2 Tracking at night-time :

Tracking results at night-time,are not that satisfactory, because detector itself is not able to recognise the object continuously in all frames of video in which the object is present. For the kalman algorithm to work properly ,detector should detect the object as long as it is in the video. Figure 4.6 shows an output frame on night-time traffic

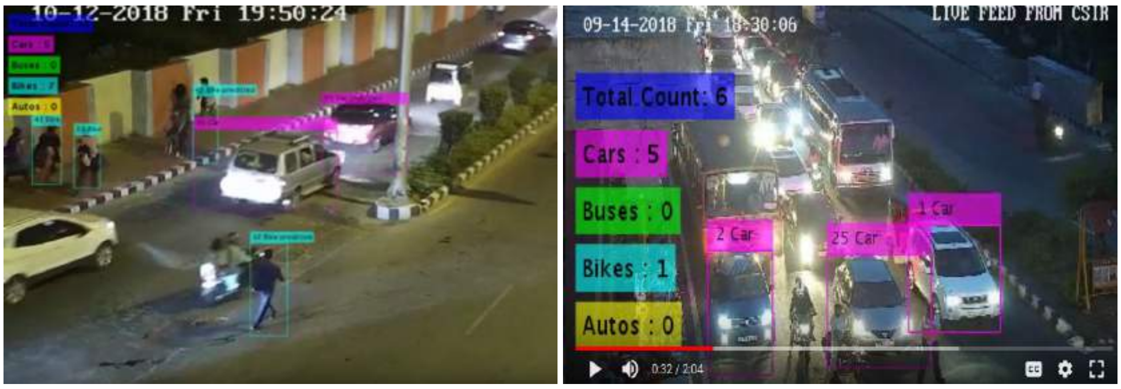


Figure 4.6: Tracking output frame on night-time traffic

CHAPTER 5

Conclusions & Future Work

We need to prefer YOLO over other architectures because :

- YOLO processes an image much faster than Pyfaster-RCNN .YOLO works at 27fps, while PyFaster RCNN works at 2-3fps i.e YOLO works nearly **10 times** as fast as PyFaster RCNN
- Finetuning takes fewer number of iterations than training. To get good results from PyFaster-RCNN we need to train it from scratch [9] on big Dataset (i.e Pascal VOC + a smaller dataset). Hence working with PyFasterRCNN is computationally expensive.
- Setting up YOLO is much easier than setting up PyFasterRCNN
- YOLO-finetuned model significantly improved detection of vehicles at night time.
- YOLO-finetuned model is doing better than pre-trained model for detecting and recognising truncated vehicles, as we also included truncated vehicles in our dataset.
- YOLO does much better than PyFaster RCNN in detecting number plates.
- Among all the classes , Auto is trained with lowest number of instances, finetuned-YOLO could detect and recognise Auto's well.

The model we built is robust only to day & night time and to front-view & back-view of vehicles. We can make the model more robust by training the model on other situations like raining etc. Fine-Tuned YOLO is not very good with detecting bikes. Decreasing the threshold improved the results but pre-trained YOLO did much better. One way to get as good results as pre-trained YOLO with bikes is to train YOLO from scratch on Pascal VOC + IITM HeTra + Front-View + Back-View datasets. Followed a similar procedure with Py-Faster-RCNN , Py-Faster-RCNN could detect bikes well from both front-view and back-view. License Plate Recognition will be a useful technology which can help transportation management in many ways like we can make highway toll collection systems more automatic, enhance prevention of theft of vehicles etc. Till now we only detected licence plates , recognition of licence plates could be the next step to the project. And also at night times, when glare from vehicles is too high, detectors are not able to detect all vehicles. And as the detector is performing poorly, tracking results are also poor.We should explore methods to reduce this glare from vehicles and improve detection of vehicles at night-time.

Present tracking code works at 12-13 frames/sec. Using deep learning technologies we can make it work in real-time. And when two objects occlude trackers of those objects can get interchanged. We need to find a way for trackers to track objects ,even during occulsions. The present implementation of tracking is not a continuous process i.e we first do detection, then we feed the results of detection to the tracking code.We can join Detection and Tracking code and make the process continuous.

REFERENCES

[1] Ross Girshick and Kaiming He ,"Faster R-CNN: Towards Real-Time Object Detectionwith Region Proposal Networks" https://github.com/rbgirshick/py-faster-rcnn

[2] Joseph Redmon and Ali Farhadi, "YOLOv3: An Incremental Improvement", https://pjreddie.com/darknet/yolo/

[3] Wei Liu, Dragomir Anguelov, "SSD: Single Shot MultiBox Detector", https://www.cs.unc.edu/ wliu/papers/ssd.pdf

[4] R.E Kalman, "A New Approach to Linear Filtering and Prediction Problems" https://www.cs.unc.edu/ welch/kalman/media/pdf/Kalman1960.pdf

[5] Joseph Redmon and Ali Farhadi, "YOLO9000: Better, Faster, Stronger", https://arxiv.org/pdf/1612.08242.pdf

[6] Joseph Redmon and Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", https://arxiv.org/pdf/1506.02640.pdf

[7] IITM HeTra Dataset : https://www.kaggle.com/deepak242424/iitmhetra

[8] Pretrained weights of YOLO trained on COCO dataset https://pjreddie.com/media/files/yolov3.weights

[9] Deepak Mittal and Avinash Reddy, "Training a deep learning architecture for vehicle detection using limited heterogeneous traffic data" https://ieeexplore.ieee.org/document/8328279/references

[10] PascalVOC dataset: http://host.robots.ox.ac.uk/pascal/VOC/

[11] Alex Krizhevsky and Ilya Sutskever, "ImageNet Classification with Deep Convolutional Neural Networks"

[12] J. Redmon. Darknet:Open source neural networks in c https://pjreddie.com/darknet/

[13] Raman Arora, Amitabh basu,"Understanding Deep Neural Networks with Rectified Linear Units" https://arxiv.org/pdf/1803.08375

[14] https://in.mathworks.com/videos/introduction-to-deep-learning-wha

[15] https://en.wikipedia.org/wiki/Kalman_filter