University of Mumbai

A Project Report on

AUTOMATED TRAFFIC MANAGEMENT SYSTEM

Submitted in partial fulfilment of the requirements

Of the degree of

Bachelor of Engineering

By

Sarvesh Patil
Riddhi Sanghani
Sameer Motiramani
Rajeshree Ghanwat

Under the guidance of
Dr. Nadir Charniya

Department of Electronics and Telecommunication
Vivekanand Education Society's Institute of Technology 2018-2019

Collectors Colony, Chembur, Mumbai-400071

University of Mumbai
Year 2018-2019

CERTIFICATE OF APPROVAL OF PROJECT

This is to certify that Riddhi Sanghani, Sarvesh Patil, Sameer Motiramani, Rajeshree Ghanawat have completed the project report on the topic "Automated Traffic Management System" satisfactorily in partial fulfillment for the Bachelor's Degree in Electronics and Telecommunication Engineering. Under the guidance of Dr. Nadir Charniya during the year 2018-19 as prescribed by the University of Mumbai.

Guide
Dr. Nadir Charniya

Head of department
Prof.(Mrs.) Shobha Krishnan

Principal
Dr. (Mrs.) Jayalakshmi M. Nair

Examiner1

Examiner 2

Project Report Approval for B. E.

This project report entitled **Automated Traffic Management System** by **Sarvesh Patil, Riddhi Sanghani, Sameer Motiramani, Rajeshree Ghanwat** is approved for the degree of B.E. (Discipline of Electronics and Telecommunication).

Examiners:

1.----------

2.----------

Project Guide:

Principal:

Date:

Place: Mumbai

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Riddhi Sanghani
(Roll no. 60)

Sarvesh Patil
(Roll no. 47)

Sameer Motiramani
(Roll no. 37)

Rajeshree Ghanwat
(Roll no. 18)

Date:

Place: Mumbai

Acknowledgement

I would like to take this opportunity to express my sincere thanks with deep sense of gratitude to my project guide Prof. Dr. Nadir N. Charniya, for his valuable support, co-operation, and guidance which helped me in all the time of evolution of this report.

I sincerely thank my H.O.D Prof. Mrs. Shoba Krishnan and all the department staff for giving me the valuable suggestion for my report.

I would also like to convey my gratitude to our principal Dr. Mrs. J.M. Nair and the teaching and non-teaching staff of Vivekanand Education Society's Institute of Technology for permitting to carry out this work in this prestigious institute. The library and lab facilities were of enormous value to this work.

I would like to express my heartfelt thanks to my loving parents for their blessings and all other concerned individuals who have directly or indirectly extended their valuable support and contribution for completion of my work.

Riddhi Sanghani
(Roll no. 60)

Sarvesh Patil
(Roll no. 47)

Sameer Motiramani
(Roll no. 37)

Rajeshree Ghanwat
(Roll no. 18)

Date:

Place: Mumbai

Contents

Abstract

The Automated Traffic Management System (ATMS) aims at undertaking the road and the transport sector, particularly the traffic management system employed in urban areas with high density of population. While the current standard in India deploys remote surveillance and hardcoded traffic management at intersections, it still requires human intervention at a large scale to monitor the feed as well as traffic signals in a multitude of areas. With the use of artificial intelligence, especially Deep Learning and Image processing, the entire traffic management can be automated to detect the traffic law breakers as well as the more serious crimes, especially at busy intersections. It also incorporates a system with dynamic traffic signal controller which controls the traffic light timers depending upon the congestion on the roads at the intersection.

List of Figures

Chapter 1

Introduction

1.1 Motivation

1.2 Project Definition

1.3 Relevance of the Project

1.4 Methodology Used

INTRODUCTION

1.1 Motivation

The entire world is now more connected than ever. Every aspect of the society has been witnessing tremendous growth in terms of finance, infrastructure and most of all, technology. Technology is transforming our lives at a tremendous pace. This has brought along with it a milieu of automation. The Indian traffic system, however, has not been revolutionised by this technical storm. According to National Crime Records Bureau, 1214 road crashes occur every day in India. The traffic congestion problems occur in almost all the urban cities leading to higher pollution levels and inefficient fuel consumption. By using models like ATMS we can limit these by employing stricter monitoring of the traffic rules as well as dynamic traffic controlling to reduce the pollution and fuel consumption levels.

1.2 Problem Definition

In modern day traffic management system, a variety of problems are encountered like monitoring the breaching of the traffic rules at various intersections as well as traffic congestion in the lanes due to the fixed or static timers associated with the traffic signals. Due to the poor monitoring and not so stringent actions based on it, a life is lost every four minutes in India. According to the Global status report on road safety 2013, over 1,37,000 people were killed in road accidents in 2013 alone. Our project aims at employing stricter traffic rules through automation to help provide safer roads in India.

1.3 Relevance of the project

Artificial intelligence (AI) is the simulation of human intelligence processes by machines, especially computer systems. These processes include learning (the acquisition of information and rules for using the information), reasoning (using rules to reach approximate or definite conclusions) and self-correction. Particular application of AI is used to deploy the training and testing of our system.

YOLO is one of the best algorithms for object detection and is a widely used algorithm. The YOLO framework (You Only Look Once), deals with object detection in a different way. It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes. The biggest advantage of using YOLO is its speed – it's incredibly fast and can process 45 frames per second. YOLO also understands generalized object representation and hence is an integral algorithm for ATMS.

1.4 Methodology used

Step 1 Concept Statement:

- Need for traffic regulation and control system

Step 2 Proposal:

- What is the problem being addressed?
- Why is it important and interesting?
- Why ATMS?

Step 3 Methodology:

We propose a system a system which will help monitor the traffic laws with minimum human intervention and solve the traffic congestion problem. By introducing this we can monitor the breach of traffic law in an area. We can also automate the traffic signals according to the congestion on roads at intersections.

Step 4 Deployment

The system proposed implements various algorithms for object detection and congestion detection. The YOLO algorithm is used for object recognition. This will help in real time detection of vehicle incase when a traffic law has been breached. The vehicle's picture will be sent to the control room for further investigation. The traffic congestion algorithm will compare traffic density from different roads on an intersection and will manipulate the traffic timer controls accordingly. Thus, dynamic traffic lights will be deployed.

Chapter 2

Review of Literature Survey

ATMS has been built upon the baselines provided by the consistent efforts of the artificial intelligence research community through the years 1960s until 2018. For the current implementation of the project, a myriad of previous technologies has to be understood and applied and this literature survey aims at bringing forth the papers that have had a major influence on this project either directly or indirectly.

1. Simon Haykin (Neural networks: a comprehensive foundation, 2002)

The beginning of Neurocomputing is often taken to be the research article of McCulloch and Pitts[1] published in 1943, which showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function, was widely read and had great influence. Other researchers, principally Norbert Wiener and von Neumann, wrote a book and research paper [2, 3] in which the suggestion was made that the research into the design of brain-like or brain-inspired computers might be interesting.

In 1949 Hebb wrote a book [4] entitled The Organization of Behaviour which pursued the idea that classical psychological conditioning is ubiquitous in animals because it is a property of individual neurons. This idea was not itself new, but Hebb took it further than anyone before him had by proposing a specific learning law for the synapses of neurons. Hebb then used this learning law to build a qualitative explanation of some experimental results from psychology. Although there were many other people examining the issues surrounding the neurocomputing in the 1940s and early 1950s, their work had more the effect of setting the stage for later developments than of actually causing those developments.

Typical of this era was the construction of first neurocomputer (the Snark) by Marvin Minsky in 1951. The Snark did operate successfully from a technical stand point but it never actually carried out any particularly interesting information processing functions. The first successful neuro-computer (the Mark I perceptron) was developed during 1957 and 1958 by

Frank Rosenblatt, Charles Wightman, and others. As we know it today, Rosenblatt as the founder of Neurocomputing. His primary interest was pattern recognition. Besides inventing the perceptron, Rosenblatt also wrote an early book on Neurocomputing, Principles of Neurodynamics[5] . Slightly later than Rosenblatt, but cut from similar cloth, was Bernard Widrow. Widrow, working with his graduate students (most notably Marcian E. "Ted" Hoff, who later went on to invent the microprocessor) developed a different type of neural network processing element called ADALINE, which was equipped with a powerful new learning law which, unlike the perceptron learning law, is still in widespread use. Widrow and his students applied the ADALINE successfully to a large number of toy problems, and produced several films of their successes. Besides Rosenblatt and Widrow, there were a number of other people during the late 1950s and early 1960s who had substantial success in the development of neural network architectures and implementation concepts.

Notwithstanding the considerable success of these early Neurocomputing researchers, the field suffered from two glaringly obvious problems. First, the majority of researchers approached the subject from a qualitative and experimental point of view. This experimental emphasis resulted in a significant lack of rigor and a looseness of thought that bothered many established scientists and engineers who established the field. Second, an unfortunate large fraction of neural networks researchers was carried away by their enthusiasm in their statements and their writings. For example, there were widely publicized predictions that artificial brains were just a few years away from development, and other incredible statements. Besides the hype and general lack of rigor, by the mid-1960s researchers had run out of good ideas.

The final episode of this era was a campaign led by Marvin Minsky and Seymour Papert to discredit neural network research and divert neural network research funding to the field of "Artificial Intelligence". The campaign was waged by the means of personal persuasion by Minsky and Papert and their allies, as well as by limited circulation of unpublished technical manuscript (which was further published in 1969 by Minsky and Papert as the book Perceptrons [6]). The implicit thesis of Perceptrons was that essentially all neural networks suffer from the same "fatal flaw" as the perceptron; namely the inability to usefullycompute certain essentials predicates such as XOR. To make this point the authors reviewed several proposed improvements to the perceptron and showed that these were also unable to perform well. They left the impression that neural network research had been proven to be a dead end.

In spite of Minsky and Papert's demonstration of the limitations of perceptrons, research on neural network continued. A great deal of neural network research went on under the headings of adaptive signal processing, pattern recognition, and biological modelling. In fact, many of the current leaders in the field began to publish their work during 1970s. Examples include Amari[7] , Fukushima[8] , Grossberg[9] and Klopf[10] and Gose. These people, and those who came in over the next 13 years, were the people who put the field of neural network on a firm footing and prepared the way for the renaissance of the field.

By the early 1980s many Neurocomputing researchers became bold enough to begin submitting proposals to explore the development of neuro-computers and of neural network applications. In the years 1983–1986 John Hopfield, an established physicist of worldwide reputation, had become interested in neural networks a few years earlier. Hopfield wrote two highly readable papers on neural networks in 1982 [11] and 1984 [12] and these, together with his many lectures all over the world, persuaded hundreds of highly qualified scientists, mathematicians, and technologists to join the emerging field of neural networks. In 1986, with the publication of the "PDP books" (Parallel Distributed Processing, Volumes I and II, edited by Rumelhart and McClelland[13]), the field exploded. In 1987, the first open conference on neural networks in modern times, the IEEE International Conference on Neural Networks was held in San Diego, and the International Neural Network Society (INNS) was formed. In 1988 the INNS journal Neural Networks was founded, followed by Neural Computation in 1989 and the IEEE Transactions on Neural Networks in 1990.

2.Yann Le Cun, A theoretical framework for Back-Propagation

Among all the supervised learning algorithms, back propagation is the most widely used. Although numerous experimental works have demonstrated its capabilities, a deeper theoretical understanding is definitely needed. This paper presents a mathematical framework for studying back-propagation based on the Lagrangian formalism. In this framework, inspired by control theory, back-propagation is formulated as an optimization problem with non-linear constraints. The Lagrange function is the sum of all output objective function is the sum of an output objective function and a constraint term which describes the network dynamics.

This approach suggests many natural extensions to the basic algorithm. It also provides an extremely simple formulation (and derivation) of continuous recurrent network equations as described by Pineda[14]. Other easily described variations involve either additional terms if

the error function, additional constraints on the set of solutions, or transforming of the parameter space. An interesting kind of constraint is an equality constraint among the weights, which can be implemented with little overhead. This sort of constraint provides a way of putting *a priori* knowledge into the network while reducing the number of free parameters.

Back Propagation is one of the most efficient learning procedures for multi-layer networks of neuron-like units. One of the reasons for its success is its incredible simplicity. In fact, back propagation is little more than an extremely judicious application of the chain rule and gradient descent. Derivation of the back-propagation algorithm is given by Rumelhart et al[15]. Many variations have been put forth by Yann Le Cun in 1985 [16], 1986 [17], 1987 [18].

The goal of back propagation is to compute the partial derivatives $\partial C/\partial w$ and $\partial C/\partial b$ of the cost function C with respect to any weight w or bias b in the network. For back propagation to work we need to make two assumptions about the form of the cost function. Before stating those assumptions, though, its useful to have an example cost function in mind. We'll use the quadratic cost function which has the form:

$$C = \frac{1}{2n}\sum_{x}^{\cdot} \|y(x) - a^{L}(x)\|^{2}$$

Where n is the total number of training examples; the sum is over individual training examples, x; y = y(x) is the corresponding desired output; L denotes the number of layers in the network; and $a^L = a^L(x)$ is the vector of activations output from the network when x is input. The Hadamard product is denoted by $s \odot t$ and represents elementwise product between two matrices. For e.g.

$$[1\ 2\ ] \odot [3\ 4\ ] = [1 * 3\ 2 * 4\ ] = [3\ 8\ ]$$

3.Leon Bottou, large-Scale Machine Learning with Stochastic Gradient Descent

With the growth spurt happening in the early decades of 21st century, data sizes have grown faster than the speed of processors. In this context, capabilities of statistical machine learning methods is limited by the computing time rather than the sample size. A more precise analysis uncovers qualitatively different trade-offs for the case of small-scale and large-scale learning problems. The large-scale case involves the computational complexity of the underlying optimization algorithm in non-trivial ways. Unlikely optimization algorithms such

as stochastic gradient descent show amazing performance for large-scale problems. In particular, second order stochastic gradient and averaged stochastic gradient are asymptotically efficient after a single pass on the training set.

The computational complexity of learning algorithm becomes the critical limiting factor when one envisions very large datasets. This contribution advocates stochastic gradient algorithms for large scale machine learning problems.Let us first consider a simple supervised learning setup. Each example z is a pair (x, y) composed of an arbitrary input x and a scalar output y. We consider a $loss\ function\ l(\hat{y}, y)$that measures the cost of predicting $\hat{y}$ when the actual answer is y, and we choose a family F of functions $fw(x)$parametrized by a weight vector w. We seek the function f∈ F that minimizes the loss Q(z, w) = $l(fw(x), y)$averaged on the examples. Although we would like to average over the unknown distribution $dP(z)$ that embodies the Laws of nature, we must often settle for computing the average on a sample $z_1 \ldots z_n$.

$$E(f) = \int_{\cdot}^{\cdot} l(f(x), y) dP(z) E_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$

The *empirical risk* $E_n(f)$ measures the training set performance. The expected *riskE*(f) measures the generalization performance, that is, the expected performance on future examples. The statistical learning theory Vapnik et al., [19] justifies minimizing the empirical risk instead of the expected risk when the chosen family F is sufficiently restrictive.

To minimize empirical risk $E_n(f_w)$ using gradient descent (GD), each iteration updates weights w on the basis of gradient of $E_n(f_n)$

$$\omega_{t+1} = \omega_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_\omega Q(z_i, \omega_t)$$

This second order gradient descent (2GD) is a variant of the well-known Newton algorithm. Under sufficiently optimistic regularity assumptions, and provided that w0 is sufficiently close to the optimum, second order gradient descent achieves quadratic convergence. When the cost is quadratic and the scaling matrix Γ is exact, the algorithm reaches the optimum after a single iteration.

The stochastic gradient descent (SGD) algorithm is a drastic simplification. Instead of computing the gradient of $E_n(f_w)$ exactly, each iteration estimates this gradient on the basis of a single randomly picked example z_t.

4.Diederik P. Kingma, Jimmy Lei ba, Adam: A method for stochastic optimization

Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which Adam was inspired, are discussed in this report.We also analyse the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best-known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favourably to other stochastic optimization methods. Finally, we discuss AdaMax, a variant of Adam based on the infinity norm.

Stochastic gradient-based optimization is of core practical importance in many fields of science and engineering. Many problems in these fields can be cast as the optimization of some scalar parameterized objective function requiring maximization or minimization with respect to its parameters. If the function is differentiable w.r.t. its parameters, gradient descent is a relatively efficient optimization method, since the computation of first-order partial derivatives w.r.t. all the parameters is of the same computational complexity as just evaluating the function. Often, objective functions are stochastic. For example, many objective functions are composed of a sum of subfunctions evaluated at different subsamples of data; in this case optimization can be made more efficient by taking gradient steps w.r.t. individual subfunctions, i.e. stochastic gradient descent (SGD) or ascent. SGD proved itself as an efficient and effective optimization method that was central in many machine learning success stories, such as recent advances in deep learning (Deng et al., 2013 [20]; Krizhevsky et al., 2012 [21]; Hinton & Salakhutdinov, 2006 [22]; Hinton et al., 2012a [23]; Graves et al., 2013 [25]). Objectives may also have other sources of noise than data subsampling, such as dropout (Hinton et al., 2012b[24]) regularization. For all such noisy objectives, efficient stochastic

optimization techniques are required. The focus of this paper is on the optimization of stochastic objectives with high-dimensional parameters spaces. In these cases, higher-order optimization methods are ill-suited, and discussion in this paper will be restricted to first-order methods. We propose Adam, a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name Adam is derived from adaptive moment estimation. Our method is designed to combine the advantages of two recently popular methods: AdaGrad (Duchi et al., 2011[26]), which works well with sparse gradients, and RMSProp (Tieleman& Hinton, 2012 [27]), which works well in on-line and non-stationary settings; important connections to these and other stochastic optimization methods are clarified in section 5. Some of Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its step sizes are approximately bounded by the step size hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing.

In Adam, the update rule for individual weights is to scale their gradients inversely proportional to a (scaled) L^2 norm of their individual current and past gradients. We can generalize the L^2norm-based update rule to a L^p norm-based update rule. Such variants become numerically unstable for large p. However, in the special case where we let $p \to \infty$, a surprisingly simple and stable algorithm emerges. Let, in case of the L^p norm, the step size at time t be inversely proportional to $v_t^{1/p}$, where:

$$v_t = \beta_2^p v_{t-1} + (1-\beta_2^p)|g_t|^p$$

$$= (1-\beta_2^p)\sum_{i=1}^{t}\beta_2^{p(t-i)}.|g_i|^p$$

$$v_t = (v_t)^{\frac{1}{p}} = \left((1-\beta_2^p)\sum_{i=1}^{t}\beta_2^{p(t-i)}.|g_i|^p\right)^{\frac{1}{p}}$$

$$= (1-\beta_2^p)^{\frac{1}{p}}\left(\sum_{i=1}^{t}\beta_2^{p(t-i)}.|g_i|^p\right)^{\frac{1}{p}}$$

$$= \lim_{p\to\infty}\left(\sum_{i=1}^{t}(\beta_2^{p(t-i)}.|g_i|)^p\right)^{\frac{1}{p}}$$

$$=\max\left(\beta_2^{t-1}|g_1|, \beta_2^{t-2}|g_2|, \ldots\ldots, \beta_2|g_{t-1}|, |g_t|\right)$$

Which corresponds to the remarkably simple recursive formula:

$$u_t = max(\beta_2 . u_{t-1}, |g_t|)$$

with initial value u0 = 0. Note that, conveniently enough, we don't need to correct for initialization bias in this case. Also note that the magnitude of parameter updates has a simpler bound with AdaMax than Adam, namely: $|\Delta t| \leq \alpha$

5.Alex Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labelled images were relatively small — on the order of tens of thousands of images. Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the current best error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [28]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [29]), but it has only recently become possible to collect labelled datasets with millions of images. The new larger datasets include LabelMe[30], which consists of hundreds of thousands of fully-segmented images, and ImageNet[31], which consists of over 15 million labelled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models[32, 33, 34, 35, 36, 37, 38]. Their capacity can be controlled by varying their depth and breadth, and they also

make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Despite the attractive qualities of CNNs, and despite the relative efficiency of their local architecture, they have still been prohibitively expensive to apply in large scale to high-resolution images. Luckily, current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to facilitate the training of interestingly-large CNNs, and recent datasets such as ImageNet contain enough labelled examples to train such models without severe overfitting.

Figure below illustrates a sample Convolutional Neural Network

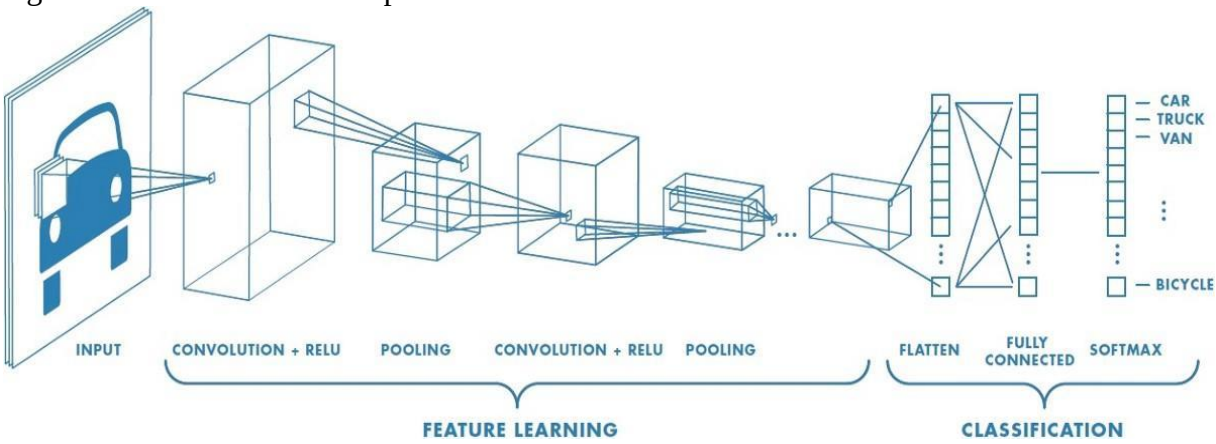


Figure 2.1: neural network with many convolutional layers

The standard way to model a neuron's output f as a function of its input x is with $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. In terms of training time with gradient descent, these saturating nonlinearities are much slower than the non-saturating nonlinearity $f(x) = \max(0, x)$. Following Nair and Hinton [39], we refer to neurons with this nonlinearity as Rectified Linear Units (ReLUs). Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units. This is demonstrated in Figure 1, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network. This plot shows that we would not have been able to experiment with such large neural networks for this work if we had used traditional saturating neuron models. We are not the first to consider alternatives to traditional neuron models in CNNs. For example, Jarrett et al. claim that the nonlinearity $f(x) = |\tanh(x)|$ works particularly well with their type of contrast normalization followed by local average pooling on the Caltech-101 dataset. However, on this dataset the primary concern is preventing

overfitting, so the effect they are observing is different from the accelerated ability to fit the training set which we report when using ReLUs. Faster learning has a great influence on the performance of large models trained on large datasets.

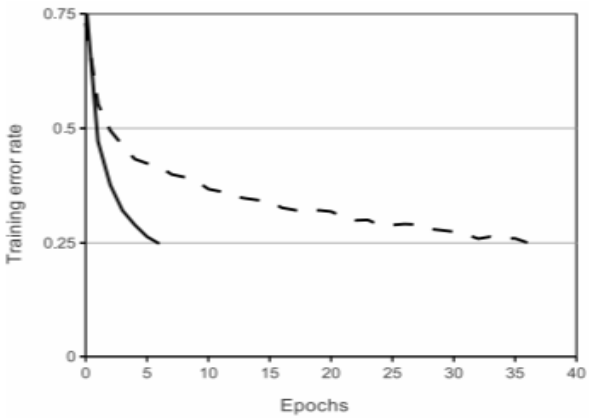


Figure 2.2:A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line).

The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons [40]

ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that the following local normalization scheme aids generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel *i* at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$is given by the expression

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} . (a^j_{x,y})^2\right)^\beta$$

where the sum runs over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set; we used k = 2, n =

5, $\alpha = 10^{-4}$, and $\beta = 0.75$. We applied this normalization after applying the ReLU nonlinearity in certain layers.

6.Parts of Convolutional Neural Networks:

Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 2.3: Effect of convolution of various filters on an image.

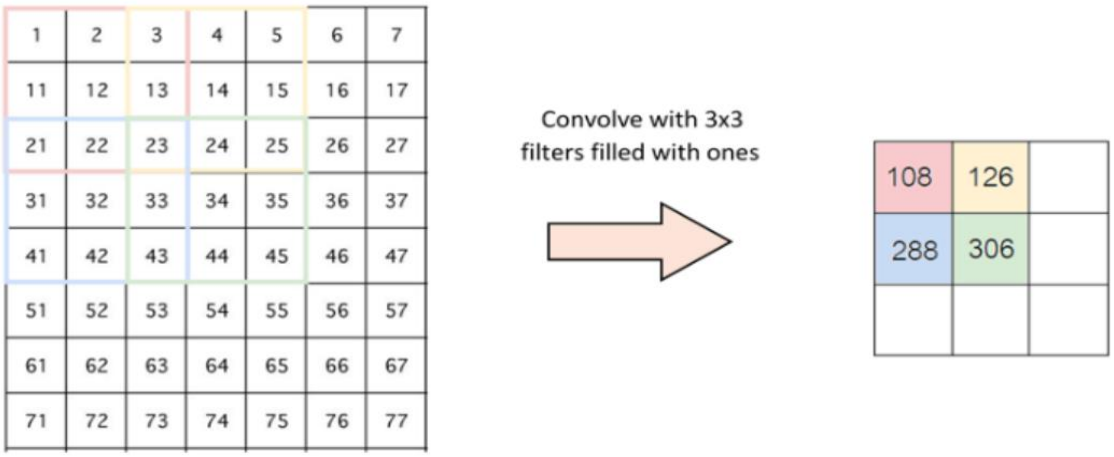


Figure 2.4: Stride of 2 pixels

Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map called as sum pooling.

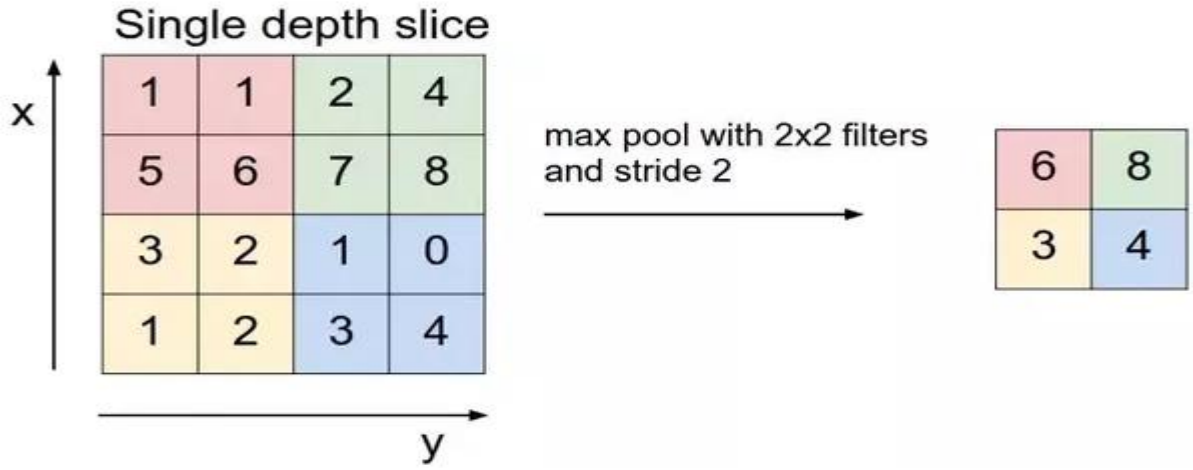


Figure 2.5: Max Pooling

7.Ross Girshick et al., Regional CNN (R-CNN)

Computer vision is an interdisciplinary field that has been gaining huge amounts of traction in the recent years(since CNN) and self-driving cars have taken centre stage. Another integral part of computer vision is object detection. Object detection aids in pose estimation, vehicle detection, surveillance etc. The difference between object detection algorithms and classification algorithms is that in detection algorithms, we try to draw a bounding box around the object of interest to locate it within the image. Also, you might not necessarily draw just one bounding box in an object detection case, there could be many bounding boxes representing different objects of interest within the image and you would not know how many beforehand.

The major reason why you cannot proceed with this problem by building a standard convolutional network followed by a fully connected layer is that, the length of the output layer is variable—not constant, this is because the number of occurrences of the objects of interest is not fixed. A naive approach to solve this problem would be to take different regions of interest from the image, and use a CNN to classify the presence of the object within that region. The problem with this approach is that the objects of interest might have different spatial locations within the image and different aspect ratios. Hence, you would have to select a huge number of regions and this could computationally blow up. Therefore, algorithms like R-CNN, YOLO etc have been developed to find these occurrences and find them fast.

To bypass the problem of selecting a huge number of regions, Ross Girshick et al. [41] proposed a method where we use selective search to extract just 2000 regions from the image and he called them region proposals. Therefore, now, instead of trying to classify a huge number of regions, you can just work with 2000 regions. These 2000 region proposals are generated using the selective search algorithm which is written below.

Selective Search:

1. Generate initial sub-segmentation, we generate many candidate regions
2. Use greedy algorithm to recursively combine similar regions into larger ones
3. Use the generated regions to produce the final candidate region proposals

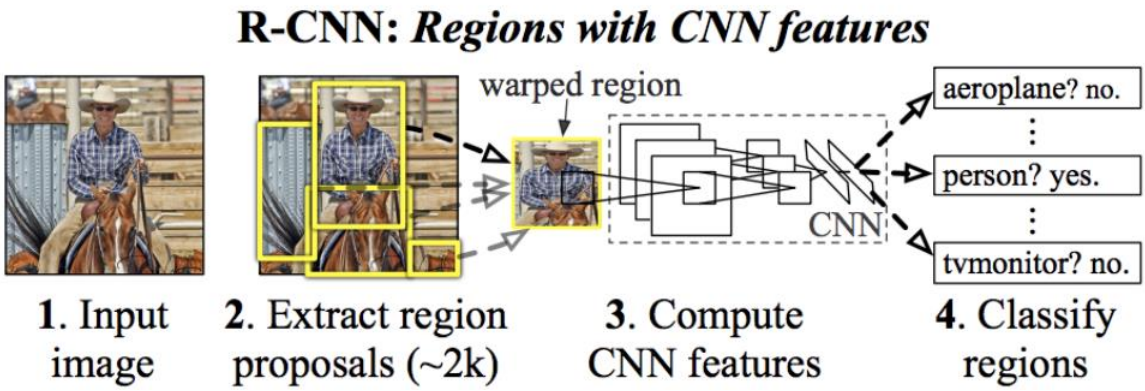


Figure 2.6: R-CNN classification using random regions

The CNN acts as a feature extractor and the output dense layer consists of the features extracted from the image and the extracted features are fed into an SVM to classify the presence of the object within that candidate region proposal. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values to increase the precision of the bounding box. For example, given a region proposal, the algorithm would have predicted the presence of a person but the face of that person within that region proposal could've been cut in half. Therefore, the offset values help in adjusting the bounding box of the region proposal.

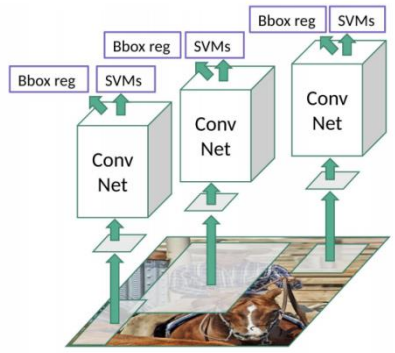


Figure 7:The regions over the image are randomly selected to be fed into a convolutional neural network. The CNN performs convolution and max pooling operations and feed the last layer to a support vector machine (SVM) for classification. The boxes with similar activations are merged together to form one segmented image[41]

Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

8.Ross Girshick, Fast R-CNN

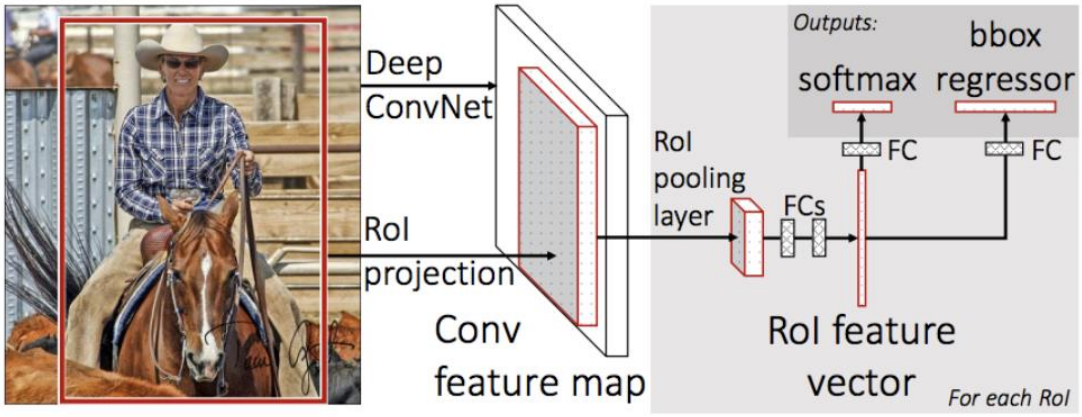


Figure 2.8: Working of a faster R-CNN[43]

The same author of the previous paper(R-CNN) solved some of the drawbacks of R-CNN to build a faster object detection algorithm and it was called Fast R-CNN. The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, we use a SoftMax layer to predict the class of the proposed region and also the offset values for the bounding box.

The reason “Fast R-CNN” is faster than R-CNN is because you don’t have to feed 2000 region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it. Figure 2.9 shows a comparison between R-CNN and Fast R-CNN.

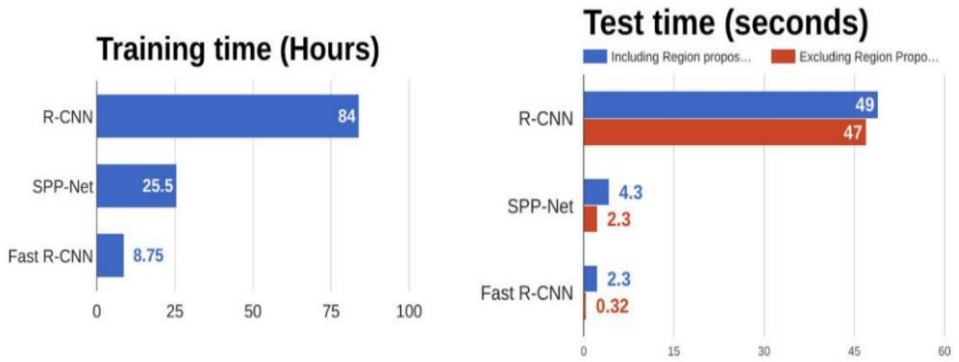


Figure 2.9: Comparison between training and testing time of object detection algorithms

9.Shaoqing Ren et al., Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

One may note that fast region-based CNNs take advantage of GPUs, while the region proposal methods used in research are implemented on the CPU, making such runtime comparisons inequitable. An obvious way to accelerate proposal computation is to

reimplement it for the GPU. This may be an effective engineering solution, but re-implementation ignores the down-stream detection network and therefore misses important opportunities for sharing computation. In this paper, it is shown that an algorithmic change - computing proposals with a deep convolutional neural network -leads to an elegant and effective solution where proposal computation is nearly cost-free given the detection network's computation.

Both of the above algorithms (R-CNN & Fast R-CNN) uses selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. Therefore, Shaoqing Ren et al. came up with an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals.

Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map. Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals. The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

To this end, we introduce novel Region Proposal Networks (RPNs) that share convolutional layers with state-of-the-art object detection networks [42], [43]. By sharing convolutions at test-time, the marginal cost for computing proposals is small (e.g., 10ms per image). Our observation is that the convolutional feature maps used by region-based detectors, like Fast RCNN, can also be used for generating region proposals. On top of these convolutional features, we construct an RPN by adding a few additional convolutional layers that simultaneously regress region bounds and objectness scores at each location on a regular grid. The RPN is thus a kind of fully convolutional network (FCN) [44] and can be trained end-to-end specifically for the task for generating detection proposals.

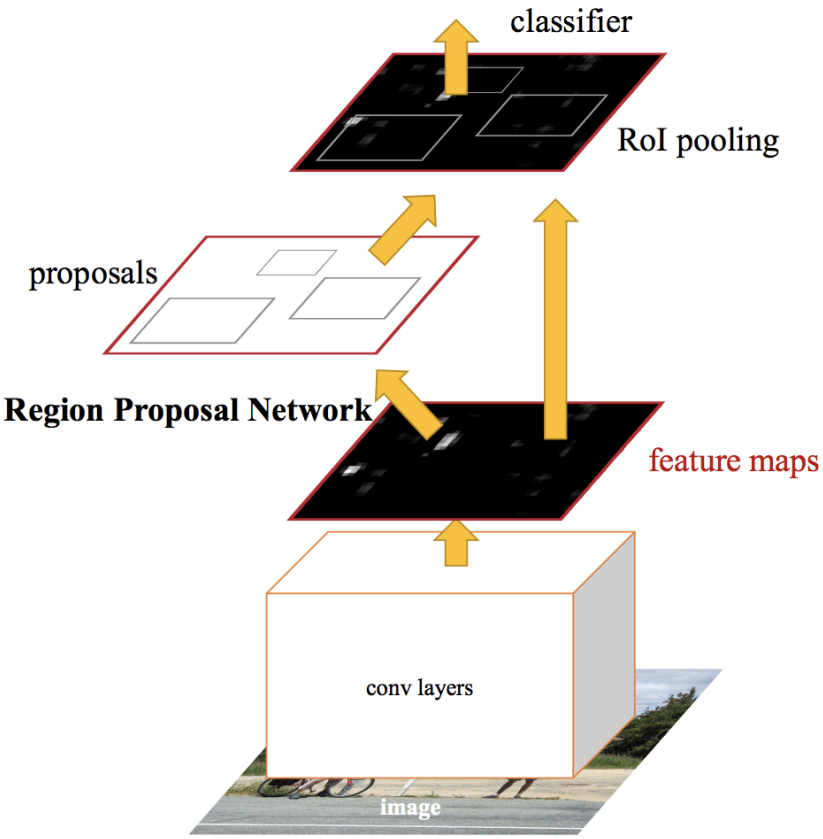


Figure 2.10: Conceptual diagram of a Region Proposal Network (RPN) [44]

RPNs are designed to efficiently predict region proposals with a wide range of scales and aspect ratios. Instead of a pyramid of filters as in RCNN and Fast RCNN, Faster R-CNN is a novel approach to introduce anchor boxes that serve as references at multiple scales and aspect ratios. It can be thought of a pyramid of regression references, which avoids enumerating images or filters of multiple scales or aspect ratios. The model performs well when trained and tested using single-scale images and thus benefits running speed.

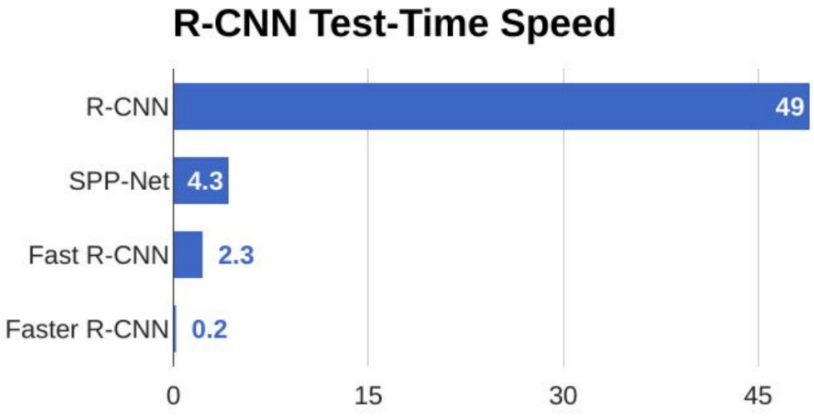


Figure 2.11: Ultimate Comparison among object detection algorithms on basis of their speed. [43]

From the above graph, you can see that Faster R-CNN is much faster than its predecessors. Therefore, it can even be used for real-time object detection. While faster R-CNN looks like the ultimate champion of object detection and segmentation algorithms, it has a few drawbacks that can be traced back to the theory on basis of which R-CNN themselves were created. These drawbacks dictate that while faster R-CNN is the fastest algorithm, it still uses searching operations on randomly selected boxes which are scattered over the image. This leaves behind a probability, albeit a small one, that some object if by pure chance does not fall under any of the windows defined by Faster R-CNN, it won't be detected. This small probability can prove fatal in applications like self-driving cars.Hence we need a more robust algorithm that can work with approximately same time complexity and give better results at detecting objects in an image.

10.Kaiming He, Mask R-CNN

So far, we've seen how we've been able to use CNN features in many interesting ways to effectively locate different objects in an image with bounding boxes.

Can we extend such techniques to go one step further and locate exact pixels of each object instead of just bounding boxes? This problem, known as image segmentation, is what Kaiming He and a team of researchers, including Girshick, explored at Facebook AI using an architecture known as Mask R-CNN.Much like Fast R-CNN, and Faster R-CNN, Mask R-CNN's underlying intuition is straight forward. Given that Faster R-CNN works so well for object detection, could we extend it to also carry out pixel level segmentation?

In Mask R-CNN, a Fully Convolutional Network (FCN) is added on top of the CNN features of Faster R-CNN to generate a mask (segmentation output). Notice how this is in parallel to the classification and bounding box regression network of Faster R-CNN.Mask R-CNN does this by adding a branch to Faster R-CNN that outputs a binary mask that says whether or not a given pixel is part of an object. The branch (in white in the above image), as before, is just a Fully Convolutional Network on top of a CNN based feature map.

Here are its inputs and outputs:

- **Inputs**: CNN Feature Map.
- **Outputs**: Matrix with 1s on all locations where the pixel belongs to the object and 0s elsewhere (this is known as a binary mask).

But the Mask R-CNN authors had to make one small adjustment to make this pipeline work as expected.

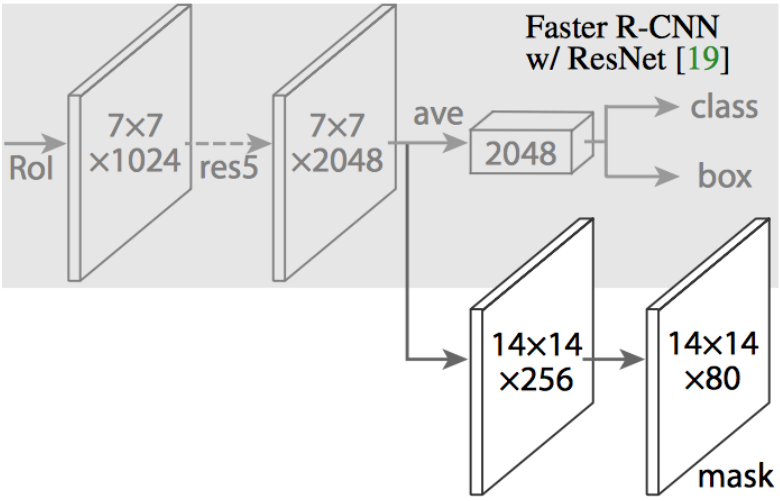


Figure 2.12: Extracting features as a trainable mask before applying classification.[45]

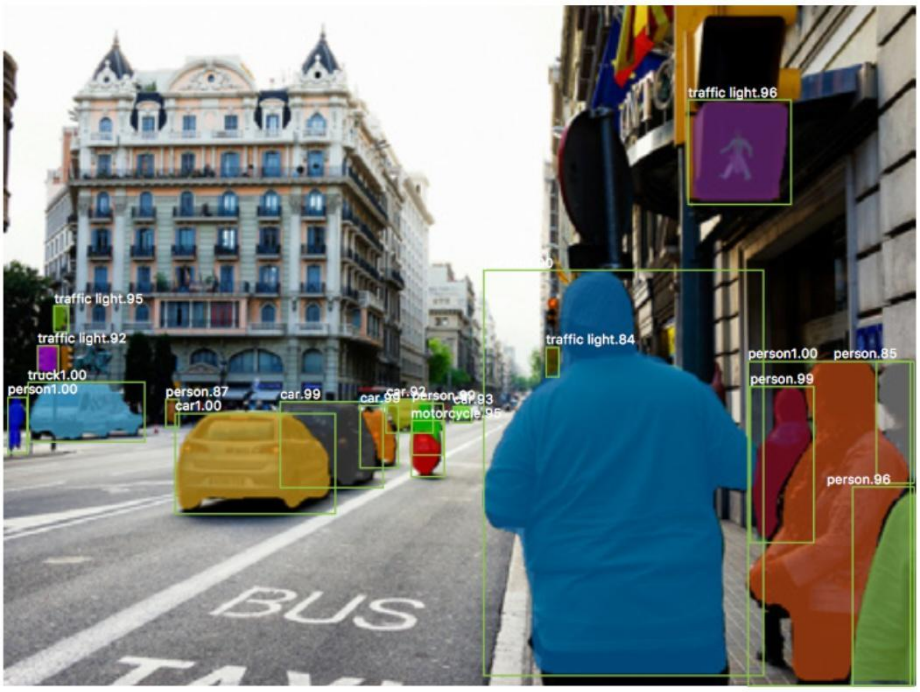


Figure 2.13: Working of Mask R-CNN[46]

Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5 fps. Moreover, Mask R-CNN is easy to generalize to other tasks, e.g., allowing us to estimate human poses in the same framework.Mask R-CNN, extends Faster R-CNN by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box the mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. Mask R-CNN is simple to implement and train given the Faster R-CNN framework, which

facilitates a wide range of flexible architecture designs. Additionally, the mask branch only adds a small computational overhead, enabling a fast system and rapid experimentation.

11.Joseph Redmon et al., YOLO – You Only Look Once

All of the previous object detection algorithms use regions to localize the object within the image. The network does not look at the complete image. Instead, parts of the image which have high probabilities of containing the object. YOLO or You Only Look Once is an object detection algorithm much different from the region-based algorithms seen above. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.

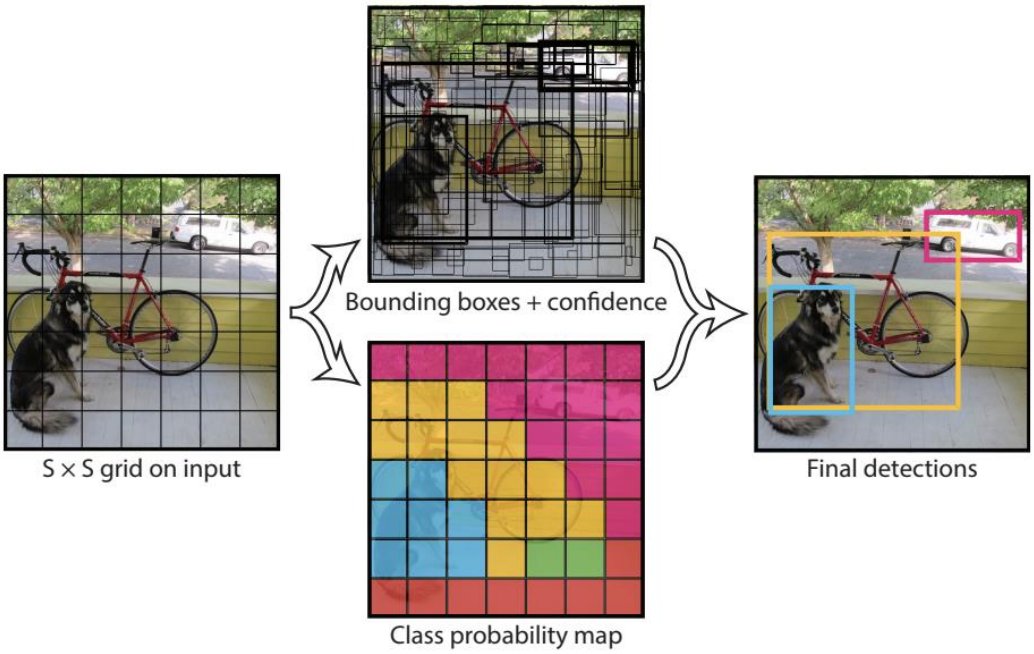


Figure 2.12: Working of YOLO [46]

How YOLO works is that we take an image and split it into an SxS grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network outputs a class probability and offset values for the bounding box. The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image.

YOLO is orders of magnitude faster (45 frames per second) than other object detection algorithms. The limitation of YOLO algorithm is that it struggles with small objects within the image, for example it might have difficulties in detecting a flock of birds. This is due to the spatial constraints of the algorithm.

YOLO is refreshingly. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and

directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection. First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict detections. A base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems.

Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method [45], mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs. YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones.

We unify the separate components of object detection into a single neural network. The network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means the network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and Realtime speeds while maintaining high average precision. The system divides the input image into an S × 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

$$\Pr(\text{Object}) * IOU_{pred}^{truth}$$

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, $\Pr(\text{Class}_i|\text{Object})$. These probabilities are conditioned on the grid cell containing an object. We only predictone 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,

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

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.

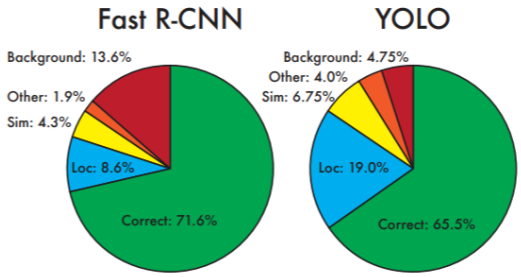


Figure 2.13: Comparison between error rates of Fast R-CNN and YOLO architecture [46]

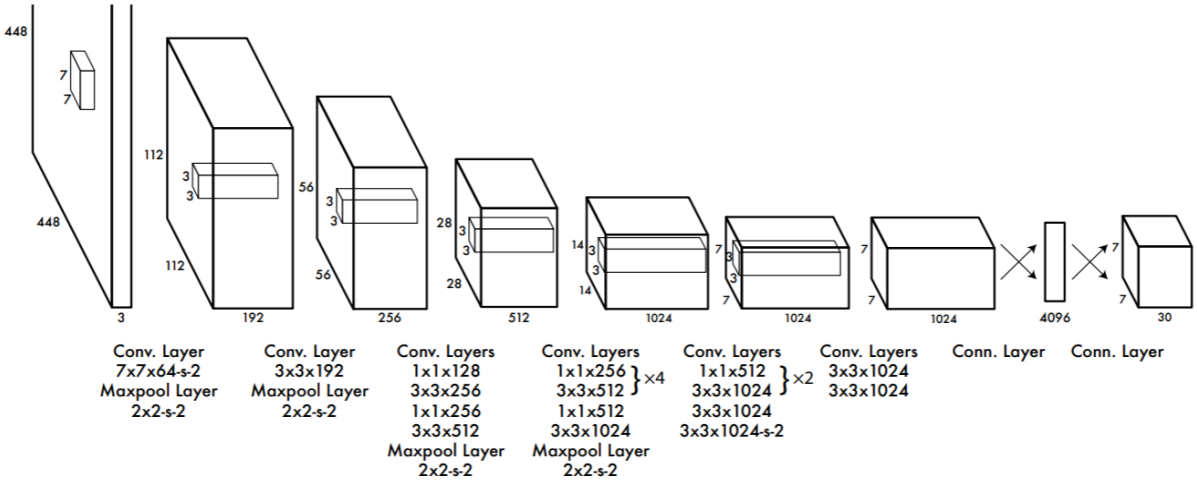


Figure 2.14: Architecture of YOLO network [46]

Thus, as HOG is a method of the past and methods like SIFT, Haar, convolutional features and localizers have proven to be very poor in achieving human level performance, it is but obvious that the task be handed out to neural networks. By modifying the existing

convolutional neural networks, we can generate image segmentation models using methods like regional CNN. The improvement between R-CNN, Fast R-CNN and Faster R-CNN has been of a great impetus and Fast R-CNN achieves the best possible accuracy for object detection and segmentation. In spite of all the advances in R-CNN networks, the fundamental flaw was overcome by the YOLO architecture, by eliminating the need of stacking multiple layers on top of each other to create a state of the art object detection algorithm for use in real time applications. Hence, in this project, we have decided to go through with YOLO architecture as a model of our choice and all the work done by us will be pushing the limits of this network in the best possible way that we can, by implementing it on CCTV cameras in urban environments to automate traffic management and helping catch law breakers.

13. Image processing based Adaptive Traffic Control System, Arif A. Bookseller and Rupali R Jagtap:

Objective of proposed system[49] is to improve efficiency of existing automatic traffic signaling system. The system will be image processing based adaptive signal controlling. The timing will be calculated each time change automatically depending upon the traffic load.

Controlling of signal lights: The signaling is cyclic in clockwise direction starting from first road, through fourth sequentially. The timing is set after its calculation from the estimated density with the help of image processing technique on the captured image. Digital monochrome or color camera is used. The timing calculated is passed on to Microcontroller 89s52, 40 pin DIP with necessary RS232 interface. Microcontroller operates the signal lights with the help of necessary driver circuitry.

Image processing: The steps followed in processing of a capture image includes experimentally found out results along with simplified consideration.

i. Fixing queue area: The predefined length L1 as queue length and road width gives maximum area of queue.T, is time required to clear the 100% queue area, and this will be maximum time setting for the control of signal at that intersection. This area of interest can be obtained by installation of camera at height and angle facing towards road. Hence L and T are experimentally found.

ii. Region of interest (ROI) : Area of queue is region of interest (ROI) and could be freezed by generating a cropped image of empty road or normal image scene. Captured image when saved is in matrix form Imn.. Final cropped image Fmn will be product of Cmn.

Figure 17: cropped out image, ROI [49]

iii. Conversion to gray scale :Capture an image, if it is color, convert it into gray scale image. MATLAB® functions „rgb2gray(x)" and adapthisteq(x)" could be used. CONVERSION TO BINARY Some threshold could be used while conversion from grayscale to binary. MATLAB® function „im2bw(x,0.3)" could be used. This conversion may be dropped and directly we can go to next step.

iv. Subtracting : Subtract cropped image of scene from the ROI to get area occupied by vehicle .This can be achieved by anding operation on crop captured image scene and ROI. The logic is, structure generated by asphalt image in ROI and the changed structure due to occupied vehicle. The changed structure could be separated from unchanged structure after Anding.

v. Time calculation: We know maximum "T", for complete ROI. Now in above step we have found out the percentage occupancy. Hence percentage of „T" will be final time Tf , that would be applicable for the control of signal. The microcontroller drives the final signal lights for the calculated time.

14. Learning Mixtures of Gaussians:

Mixtures of Gaussians are among the most fundamental and widely used statistical models.This algorithm is very simple and returns the true centers of the Gaussians to within the precision specified by the user, with high probability. Two Gaussians $N(\mu 1; \sigma 2In)$ and $N(\mu 2; \sigma 2In)$are considered c-separated if $k\mu 1 - \mu 2k \geq c\sigma\sqrt{n}$. More generally, Gaussians $N(\mu 1, \Sigma 1)$ and $N(\mu 2, \Sigma 2)$ in Rn are c-separated if $k\mu 1 - \mu 2k \geq cpn \max(\lambda max(\Sigma 1), \lambda max(\Sigma 2))$, where $\lambda max(\Sigma)$ is shorthand for the largest eigenvalue of Σ.A mixture of Gaussians is c-separated if its component Gaussians are pairwise c-separated.A 2-separated mixture corresponds roughly to almost completely separated Gaussians, whereas a mixture that is 1-

or 1/2-separated contains Gaussians which overlap significantly. We will be able to deal with Gaussians that are arbitrarily close together; the running time will, however, inevitably depend upon their radius of separation.

This method helps calculate centers of gaussian in consecutive frame structure and can help remove background and extract ROI. This can be used in videos to remove background by using a simple command in high-level programming language with integrated dynamic semantics.[51]

15. Single Shot Detector:

We use a method developed by Liu et al., for detecting objects in images using a single deep neural network. The approach, named SSD, discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. SSD is simple relative to methods that require object proposals because it completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network. This makes SSD easy to train and straightforward to integrate into systems that require a detection component. Experimental results on the PASCAL VOC, COCO, and ILSVRC datasets confirm that SSD has competitive accuracy to methods that utilize an additional object proposal step and is much faster, while providing a unified framework for both training and inference. For 300×300 input, SSD achieves 74.3% mAP on VOC2007 test at 59FPS on a Nvidia Titan X and for 512×512 input, SSD achieves 76.9% mAP, outperforming a comparable state of the art Faster R-CNN model. Compared to other single stage methods, SSD has much better accuracy even with a smaller input image size.

Previously, state-of-the-art algorithms for tasks like object detection, recognition and segmentation are mostly build upon Faster R-CNN model. Eventhough it gives highest accuracy and sophistication, it lacks efficiency. While accurate, these approaches have been too computationally intensive for embedded systems and, even with highend hardware, too slow for real-time applications. Often detection speed for these approaches is measured in frames per second, and even the fastest high-accuracy detector, Faster R-CNN, operates at

only 7 frames per second (FPS). There have been many attempts to build faster detectors by attacking each stage of the detection pipeline, but so far, significantly increased speed comes only at the cost of significantly decreased detection accuracy.

The fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature resampling stage. By adding a series of improvements, we manage to increase the accuracy significantly over previous attempts. The improvements include using a small convolutional filter to predict object categories and offsets in bounding box locations, using separate predictors (filters) for different aspect ratio detections, and applying these filters to multiple feature maps from the later stages of a network in order to perform detection at multiple scales. With these modifications—especially using multiple layers for prediction at different scales—we can achieve high-accuracy using relatively low resolution input, further increasing detection speed. While these contributions may seem small independently, we note that the resulting system improves accuracy on real-time detection. This is a larger relative improvement in detection accuracy than that from the recent, very high-profile work on residual networks. Furthermore, significantly improving the speed of high-quality detection can broaden the range of settings where computer vision is useful.
In practice, one can also design a distribution of default boxes to best fit a specific dataset. By combining predictions for all default boxes with different scales and aspect ratios from all locations of many feature maps, we have a diverse set of predictions, covering various input object sizes and shapes. For example, in Fig. 1, the dog is matched to a default box in the 4 × 4 feature map, but not to any default boxes in the 8 × 8 feature map. This is because those boxes have different scales and do not match the dog box, and therefore are considered as negatives during training.

Thus, because of all the great features of SSD, to provide accurate Object Detection on high frame rates, we decided to use this Algorithm on videos with high fps rates.

Chapter 3

Methodology

3.1 Overview

The video feed obtained from the roads will be fed to the ATMS system where the object detection will take place to detect objects of interest. These objects when cross an imaginary line are detected when a red light was on i.e. when the vehicle should have been behind the imaginary line, the screenshot of the object will be taken and the object will be classified further as which model of car/bus etc it is. The vehicle's license plate information can be extracted. This information will then be carried onto to the control or the management unit where the concerned authorities will be notified of the breach of the law and necessary action can be taken.

The congestion detection of the lanes is also computed on the lanes and the traffic timers are manipulated accordingly. This makes the system dynamic.

3.2 Congestion Detection Algorithm

The congestion detection algorithm detects the amount of congestion that is present on a lane and compares it with the neighbouring lanes to manipulate the traffic timers to remove the congestion. The algorithm uses Mask-RCNN which uses object detection and semantic segmentation. Using Mask R-CNN one can automatically segment and construct pixel-wise masks for every object in an image.

Object detection builds on image classification, but this time allows us to localize each object in an image. The image is now characterized by:

1. Bounding box (x, y)-coordinates for each object
2. An associated class label for each bounding box

Fast R-CNN still utilizes Selective Search to obtain region proposals; however, the novel contribution from the paper was Region of Interest (ROI) Pooling module.

ROI Pooling works by extracting a fixed-size window from the feature map and using these features to obtain the final class label and bounding box. The primary benefit here is that the network is now, effectively, end-to-end trainable:

1. We input an image and associated ground-truth bounding boxes
2. Extract the feature map
3. Apply ROI pooling and obtain the ROI feature vector
4. And finally, use the two sets of fully-connected layers to obtain the class label predictions and the bounding box locations for each proposal.

While the network is now end-to-end trainable, performance suffered dramatically at inference (i.e., prediction) by being dependent on Selective Search.

To make the R-CNN architecture even faster we need to incorporate the region proposal directly into the R-CNN:

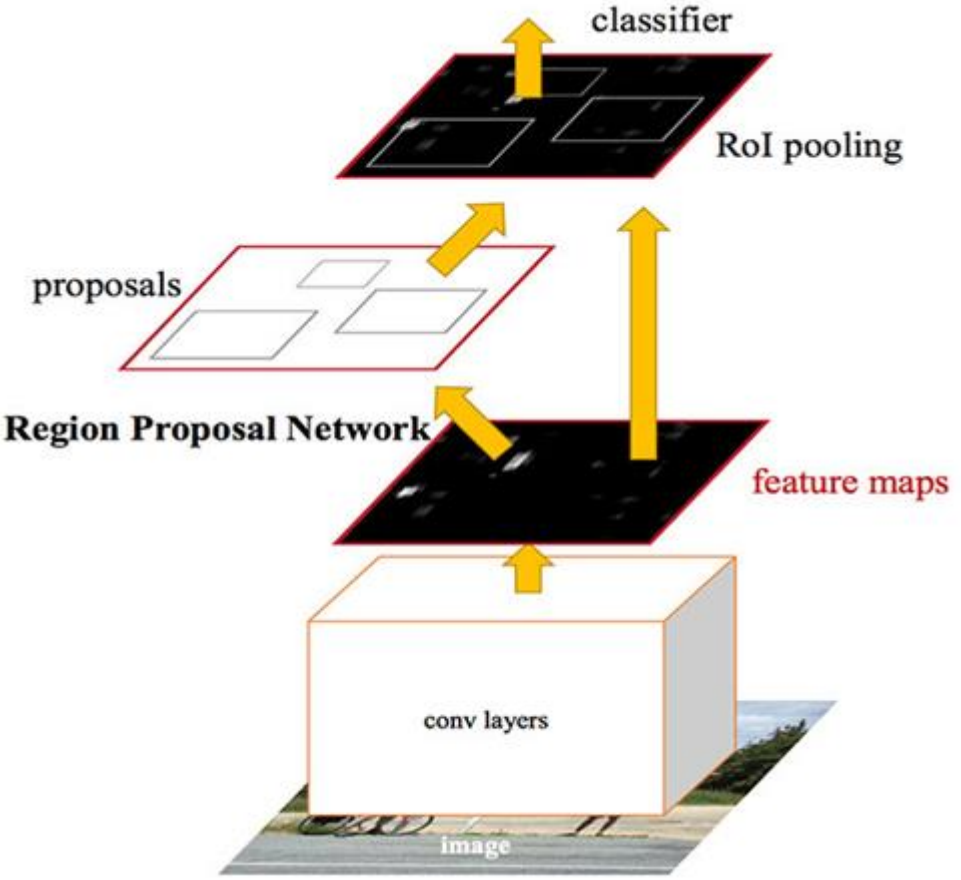


fig 3.1: ROI pooling in Fast RCNN

The Faster R-CNN architecture is capable of running at approximately 7-10 FPS, a huge step towards making real-time object detection with deep learning a reality. The Mask R-CNN algorithm builds on the Faster R-CNN architecture with two major contributions:

1. Replacing the ROI Pooling module with a more accurate ROI Align module
2. Inserting an additional branch out of the ROI Align module

This additional branch accepts the output of the ROI Align and then feeds it into two CONV layers. The output of the CONV layers is the mask itself. We can visualize the Mask R-CNN architecture in the following figure:

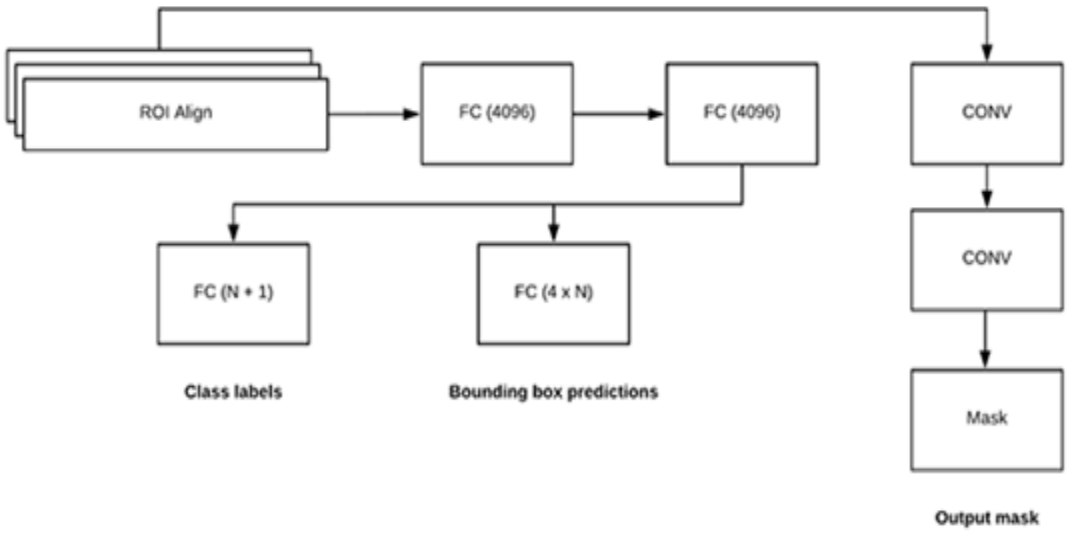


fig 3.2 :Mask RCNN overview

Each of the 300 selected ROIs go through three parallel branches of the network:

1. Label prediction
2. Bounding box prediction
3. Mask prediction

3.2.1 Object Detection

Object detection is the process of finding instances of real-world objects such as faces, bicycles, and buildings in images or videos. The algorithm detects an object from the frames marking it by a bounding box. The objects detected are vehicles on the road. This incorporates the first part of the algorithm and recognises the object and classifies it into a label along with the bounding box prediction.

3.2.2 Semantic Segmentation

Segmentation is essential for image analysis tasks. What happens is that on the final layers each "pixel" represent a larger area of the input image so we can use those cells to infer the object position. One thing to pay attention is that even though we are squeezing the image to a lower spatial dimension, the tensor is quite deep, so not much information is lost. This gives us the exact boundary of the vehicle and helps in calculating the exact areas covered by vehicles during congestion detection.

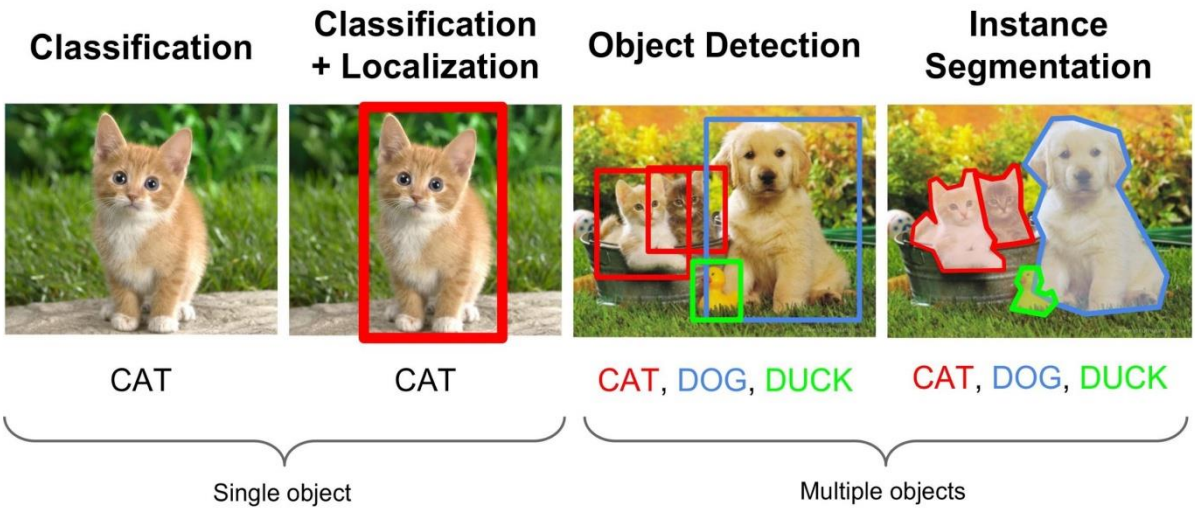


fig 3.3. Object recognition and semantic segmentation

Thus, this algorithm using Mask RCNN detects the exact areas of the vehicles instead of using a bounding box. This makes it efficient when calculating the congestion factor of the lane. It is obtained by adding all the areas of the vehicles in the frame and dividing it by the total area of the frame. This gives us a congestion factor with a value between 0 and 1. The testing results were found to have an accuracy of 97%

3.3 Algorithm for Vehicle detection during breach of law

3.3.1 Working of Convolutional Neural Networks

CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and is trained with either RMSProp or ADAM optimizer to obtain gradient descent for convergence of the neural network.

Convolution: Convolution operation can be explained by the following images.

We take the 5*5*3 filter and slide it over the complete image and along the way take the dot product between the filter and chunks of the input image.

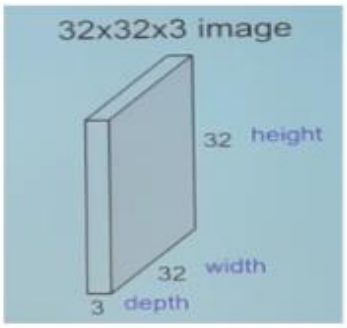


fig 3.4 Example of an RGB image

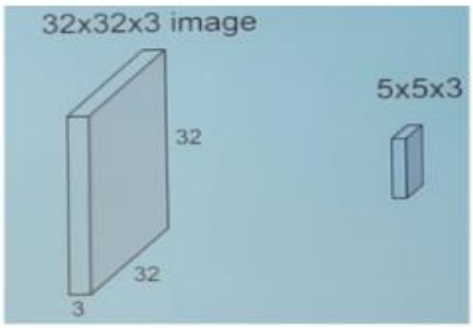


fig 3.5 Convolving an image with a filter

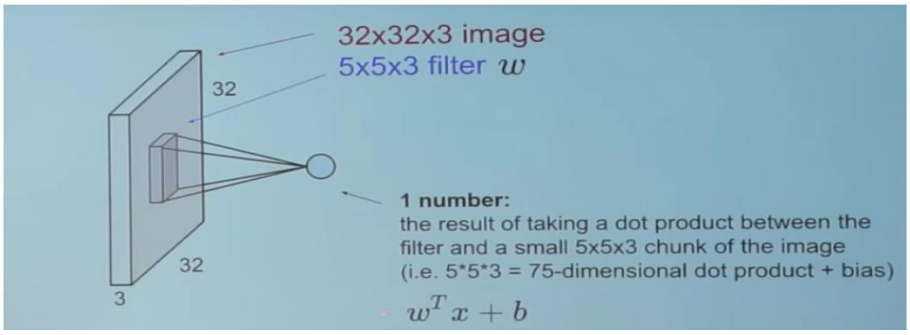


fig 3.6 How it looks

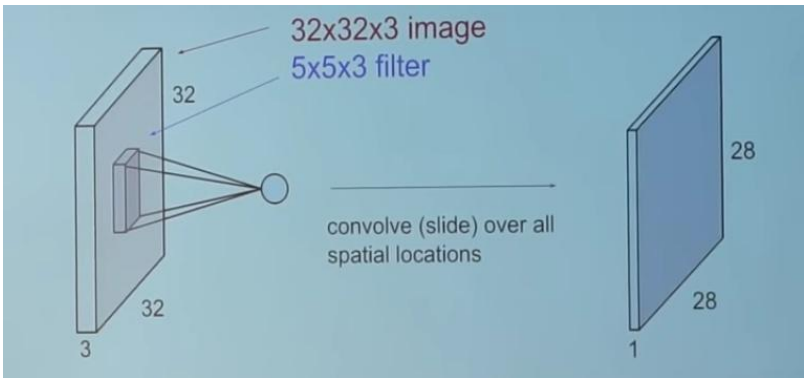


Fig 3.7 The 32x32x3 image is converted to a 28x28x1 image after convolution operation.

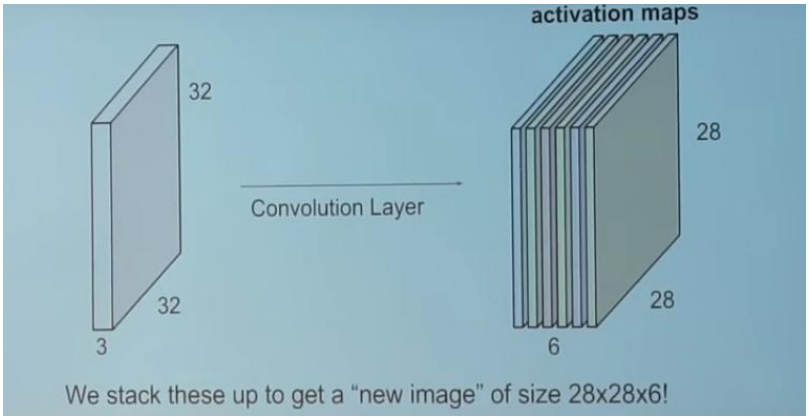


Fig 3.8 Convolution Layer

The convolution layer is the main building block of a convolutional neural network. The convolution layer comprises of a set of independent filters (6 in the example shown). Each filter is independently convolved with the image and we end up with 6 feature maps of shape 28*28*1.

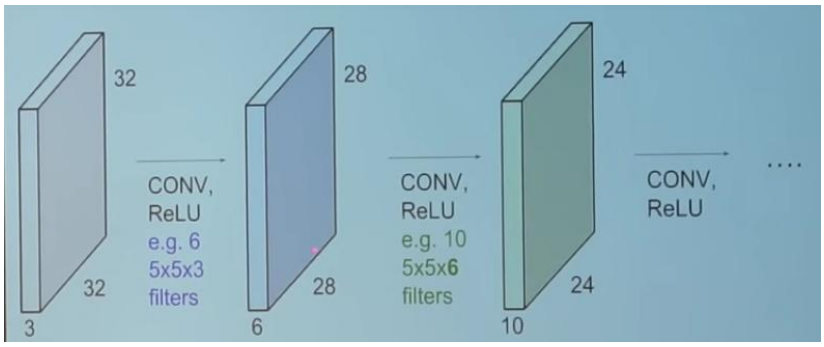


Fig 3.9 Convolution Layers in sequence

All these filters are initialized randomly and become our parameters which will be learned by the network subsequently.

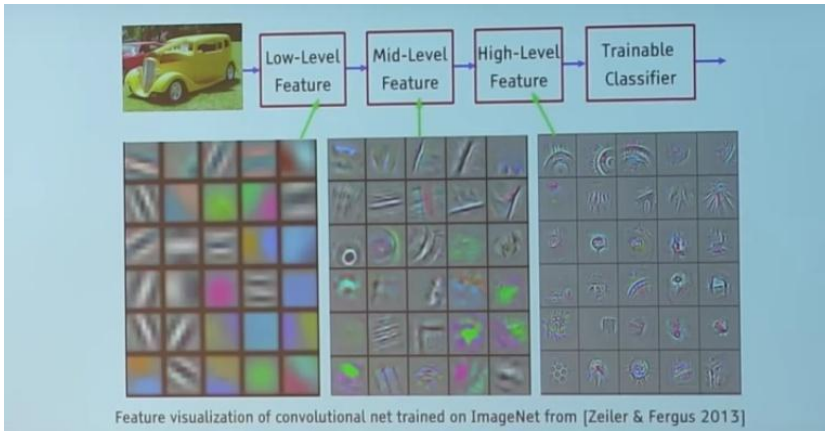


Fig 3.10 Filters in a trained network

Through back propagation, the filters have tuned themselves to become blobs of coloured pieces and edges. As we go deeper to other convolution layers, the filters are doing dot products to the input of the previous convolution layers. So, they are taking the smaller coloured pieces or edges and making larger pieces out of them. In the final layer, the network almost recognizes complex patterns like eyes, honey combs, flowers, wheels, etc. If the network is trained on faces, it looks something like this:

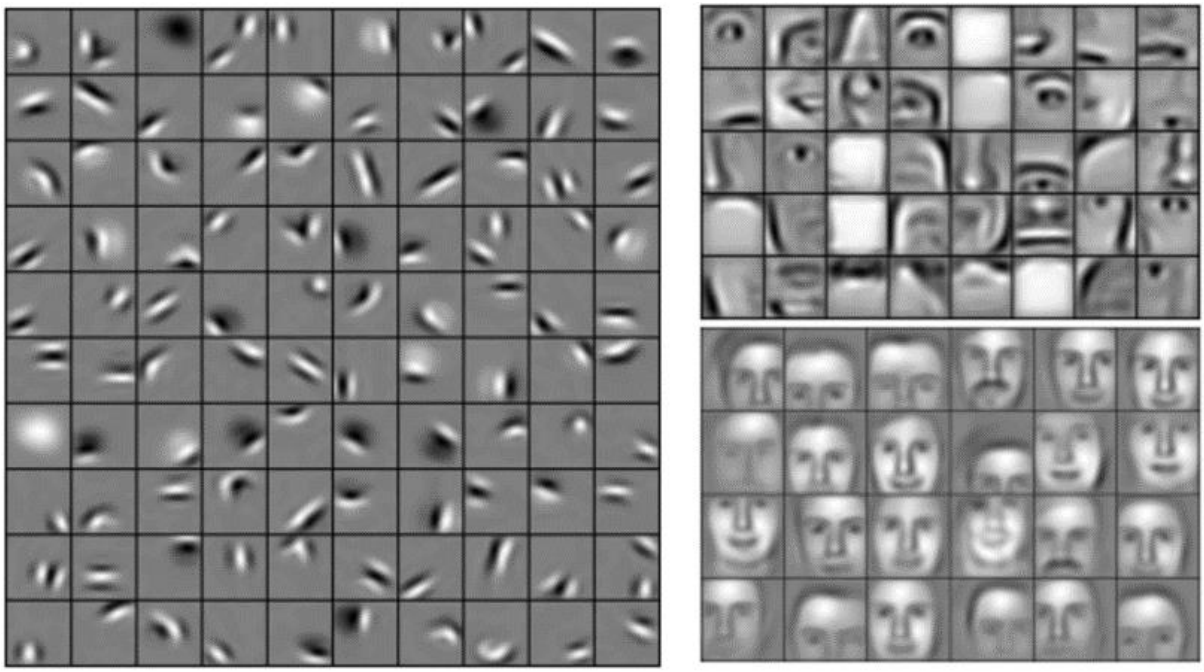


Fig 3.11 A neural network trained on faces dataset learns these features as it goes deeper

Now that we understand how CNNs work, we can use them as backend of high-level applications in this project. We use an inception v4 network for classification purposes in this project as it can take as an input image of any size and perform computations relatively faster than traditional convolutional neural networks.

The algorithm used SSD for detection of the vehicle that crossed a red signal.Then the classification of the vehicle was done using YOLO and conventional neural network (CNN) which was developed by us. The training of the network included a database of approx 20K images.

3.3.2 Single Shot Detector

We use a method developed by Liu et al., for detecting objects in images using a single deep neural network. The approach, named SSD, discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. SSD is simple relative to methods that require object proposals because it completely eliminates proposal generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network. This makes SSD easy to train and straightforward to integrate into systems that require a detection component. Experimental results on the PASCAL VOC, COCO, and ILSVRC datasets confirm that SSD has competitive accuracy to methods that utilize an additional object proposal step and is much faster, while providing a unified framework for both training and inference. For 300×300 input, SSD achieves 74.3% mAP on VOC2007 test at 59FPS on a Nvidia Titan X and for 512×512 input, SSD achieves 76.9% mAP, outperforming a comparable state of the art Faster R-CNN model. Compared to other single stage methods, SSD has much better accuracy even with a smaller input image size.

Previously, state-of-the-art algorithms for tasks like object detection, recognition and segmentation are mostly build upon Faster R-CNN model. Eventhough it gives highest accuracy and sophistication, it lacks efficiency. While accurate, these approaches have been too computationally intensive for embedded systems and, even with highend hardware, too slow for real-time applications. Often detection speed for these approaches is measured in frames per second, and even the fastest high-accuracy detector, Faster R-CNN, operates at only 7 frames per second (FPS). There have been many attempts to build faster detectors by attacking each stage of the detection pipeline, but so far, significantly increased speed comes only at the cost of significantly decreased detection accuracy.

The fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature resampling stage. By adding a series of improvements, we manage to increase the accuracy significantly over previous attempts. The improvements include using a small convolutional filter to predict object categories and offsets in bounding

box locations, using separate predictors (filters) for different aspect ratio detections, and applying these filters to multiple feature maps from the later stages of a network in order to perform detection at multiple scales. With these modifications—especially using multiple layers for prediction at different scales—we can achieve high-accuracy using relatively low resolution input, further increasing detection speed. While these contributions may seem small independently, we note that the resulting system improves accuracy on real-time detection. This is a larger relative improvement in detection accuracy than that from the recent, very high-profile work on residual networks. Furthermore, significantly improving the speed of high-quality detection can broaden the range of settings where computer vision is useful.

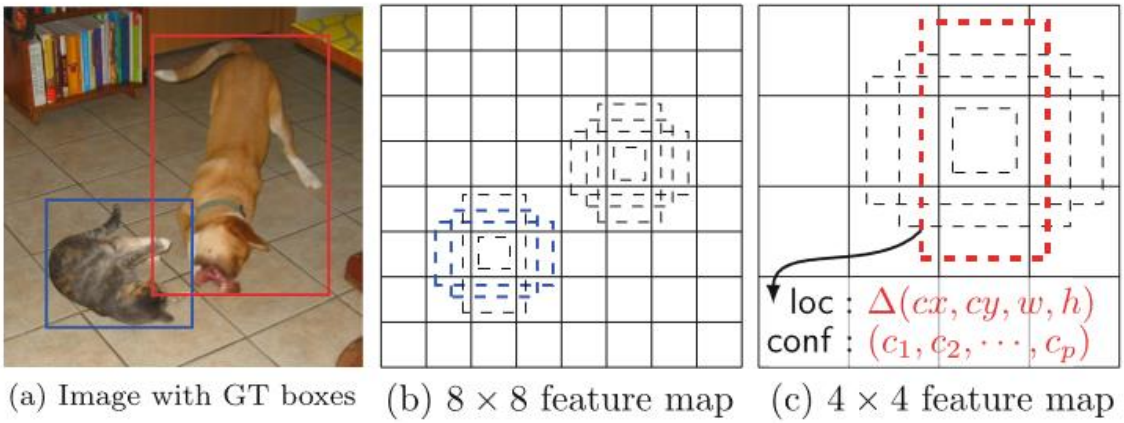


Fig. 3.12 SSD Framework SSD only needs an input image and ground truth boxes for each object during training.

The SSD 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. 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 network1. We then add auxiliary structure to the network to produce detections with the following key features:

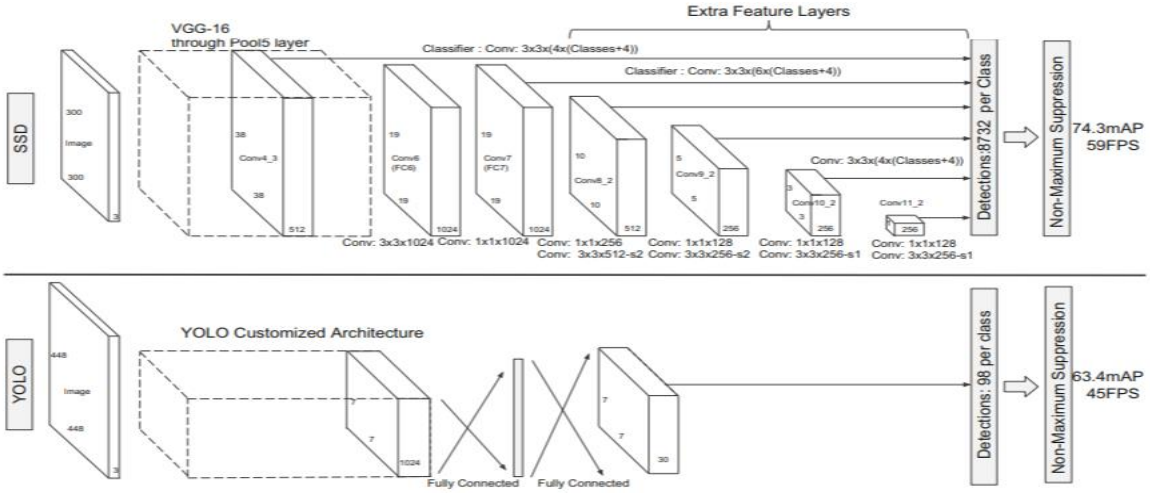


fig 3.13:A comparison between two single shot detection models: SSD and YOLO [5]

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.

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. These are indicated on top of the SSD network architecture in Fig. 3.13. For a feature layer of size m × n with p channels, the basic element for predicting parameters of a potential detection is a 3 × 3 × p small kernel that produces either a score for a category, or a shape offset relative to the default box coordinates. At each of the m × 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 SSD: Single Shot MultiBox Detector 25 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 × n feature map. For an illustration of default boxes, please refer to Fig. 3.12. 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.

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 SSD: Single Shot MultiBox Detector 25 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 × n feature map. For an illustration of default boxes, please refer to Fig. 3.12. 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.

Training The key difference between training SSD and training a typical detector that uses region proposals, is that ground truth information needs to be assigned to specific outputs in the fixed set of detector outputs. Some version of this is also required for training in YOLO [5] and for the region proposal stage of Faster RCNN [2] and MultiBox [7].
Once this assignment is determined, the loss function and back propagation are applied end-to-end. Training also involves choosing the set of default boxes and scales for detection as well as the hard negative mining and data augmentation strategies. Matching Strategy. During training we need to determine which default boxes correspond to a ground truth detection and train the network accordingly. For each ground truth box we are selecting from default boxes that vary over location, aspect ratio, and scale. We begin by matching each ground truth box to the default box with the best Jaccard overlap (as in MultiBox [7]). Unlike MultiBox, we then match default boxes to any ground truth with Jaccard overlap higher than a threshold

(0.5). This simplifies the learning problem, allowing the network to predict high scores for multiple overlapping default boxes rather than requiring it to pick only the one with maximum overlap. Training Objective. The SSD training objective is derived from the MultiBox objective [7,8] but is extended to handle multiple object categories. Let xp ij = {1, 0} be an indicator for matching the i-th default box to the j-th ground truth box of category p. In the matching strategy above, we can have i xp ij ≥ 1. The overall objective loss function is a weighted sum of the localization loss (loc) and the confidence loss (conf):

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

where N is the number of matched default boxes, and the localization loss is the Smooth L1 loss [6] between the predicted box (l) and the ground truth box (g) parameters. Similar to Faster R-CNN [2], we regress to offsets for the center of the bounding box and for its width and height. Our confidence loss is the softmax loss over multiple classes confidences (c) and the weight term α is set to 1 by cross validation. 26 W. Liu et al. Choosing Scales and Aspect Ratios for Default Boxes. To handle different object scales, some methods [4,9] suggest processing the image at different sizes and combining the results afterwards. However, by utilizing feature maps from several different layers in a single network for prediction we can mimic the same effect, while also sharing parameters across all object scales. Previous works [10,11] have shown that using feature maps from the lower layers can improve semantic segmentation quality because the lower layers capture more fine details of the input objects. Similarly, [12] showed that adding global context pooled from a feature map can help smooth the segmentation results. Motivated by these methods, we use both the lower and upper feature maps for detection. Figure 1 shows two exemplar feature maps (8 × 8 and 4 × 4) which are used in the framework. In practice, we can use many more with small computational overhead. We design the tiling of default boxes so that specific feature maps learn to be responsive to particular scales of the objects. Suppose we want to use m feature maps for prediction. The scale of the default boxes for each feature map is computed as:

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m - 1}(k - 1), \quad k \in [1, m]$$

where smin is 0.2 and smax is 0.9, meaning the lowest layer has a scale of 0.2 and the highest layer has a scale of 0.9, and all layers in between are regularly spaced. We impose different aspect ratios for the default boxes, and denote them as ar ∈ {1, 2, 3, 1 2 , 1 3 }. We can compute the width (wa k = sk √ar) and height (ha k = sk/ √ar) for each default box. For the

aspect ratio of 1, we also add a default box whose scale is s k = √sksk+1, resulting in 6 default boxes per feature map location. We set the center of each default box to (i+0.5 |fk| , j+0.5 |fk|), where |fk| is the size of the k-th square feature map, i, j [0, |fk|]. In practice, one can also design a distribution of default boxes to best fit a specific dataset. By combining predictions for all default boxes with different scales and aspect ratios from all locations of many feature maps, we have a diverse set of predictions, covering various input object sizes and shapes. For example, in Fig. 1, the dog is matched to a default box in the 4 × 4 feature map, but not to any default boxes in the 8 × 8 feature map. This is because those boxes have different scales and do not match the dog box, and therefore are considered as negatives during training.

Thus, because of all the great features of SSD, to provide accurate Object Detection on high frame rates, we decided to use this Algorithm on videos with high fps rates.

Chapter 4

IMPLEMENTATION AND RESULTS

We have successfully implemented vehicle detection by implementing YOLO and SSD algorithms. We were able to detect objects as car, bus, bike, etc. We have also developed an algorithm for traffic light timer control based on congestion with the help of Image Processing and artificial intelligence (SSD algorithm)

4.1 Vehicle Detection

As one of our objectives is to classify the brand and type of the vehicles, the process begins by gathering the data sets of several vehicles, which in our case are the photographs of majority of the vehicles that run on Indian roads.

1. To begin with, several popular vehicle brands are to be noted down, and pictures of these vehicles are taken from different angles to ensure that the vehicle should be detected from any angle accurately. For each vehicle, about 300-400 images need to be clicked.
2. Considering the vast number of vehicle models out there, the total number of images that we will end up with is somewhere near 35000. This would consume a lot of memory and make the working of detection system inefficient. This can be taken care of by clicking all images at a lower but acceptable resolution of 800*600 or 640*480, so as to reduce memory usage.
3. The key technology used here for object detection is YOLO i.e. You Only Look Once and SSD i.e. Single Shot Detector.

4.1.1. YOLO technique

We use 5 anchor boxes. So the YOLO architecture is as the following: IMAGE (m, 608, 608, 3) -> DEEP CNN -> ENCODING (m, 19, 19, 5, 85).

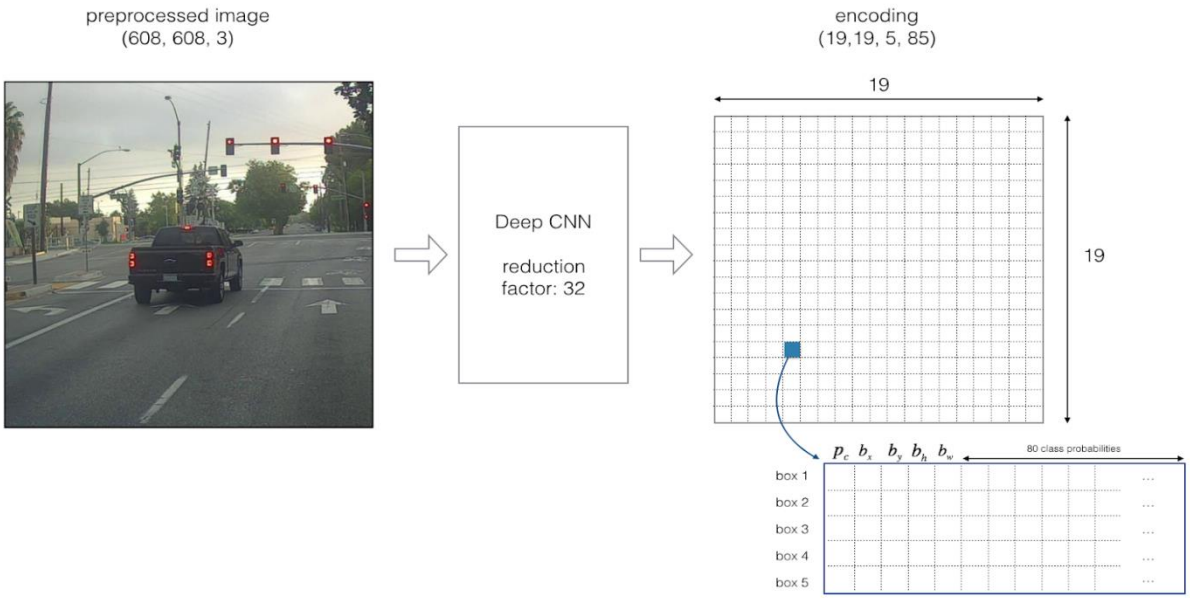


Figure 4.1 Encoding architecture for YOLO[46]

If the center/midpoint of an object falls into a grid cell, that grid cell is responsible for detecting that object. Since we are using 5 anchor boxes, each of the 19 x19 cells thus encodes information about 5 boxes. Anchor boxes are defined only by their width and height.

For simplicity, we will flatten the last two last dimensions of the shape (19, 19, 5, 85) encoding. So the output of the Deep CNN is (19, 19, 425).

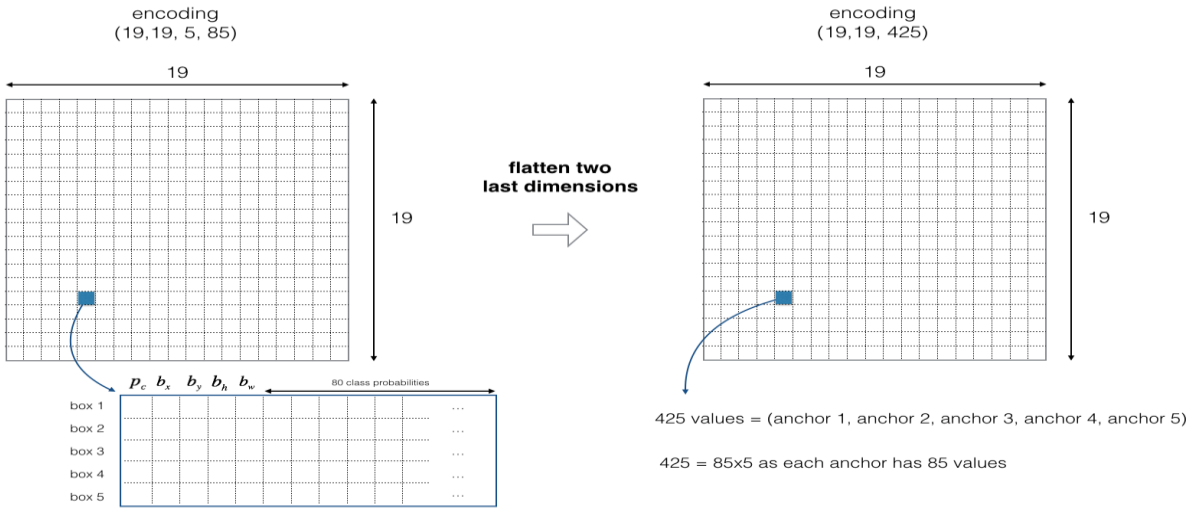


Figure 4.2 Flattening the last two last dimensions[46]

Here's one way to visualize what YOLO is predicting on an image:

- For each of the 19x19 grid cells, find the maximum of the probability scores (taking a max across both the 5 anchor boxes and across different classes).
- Color that grid cell according to what object that grid cell considers the most likely.

It is as shown in figure below

Fig 4.3 : Different classes detected by YOLO[46]

We applied the first filter by thresholding to get rid of any box for which the class "score" is less than a chosen threshold. Even after filtering by thresholding over the classes scores, you still end up a lot of overlapping boxes. A second filter for selecting the right boxes is called non-maximum suppression (NMS).

Fig 4.4: Running non-max suppression (NMS)[46]

With respect to the project, YOLO is used to detect object successfully and can be easily carried out. However the brand of the vehicle cannot be determined

To do that we are to design our own convolutional neural network (CNN) which will be trained using the images we clicked. Also all the images that are given as input data must have a common resolution which makes testing of the CNN easier.

Another part of our data set includes images of riders, with and without helmets, which should be around 400 images.

Output obtained by the implementation of YOLO.

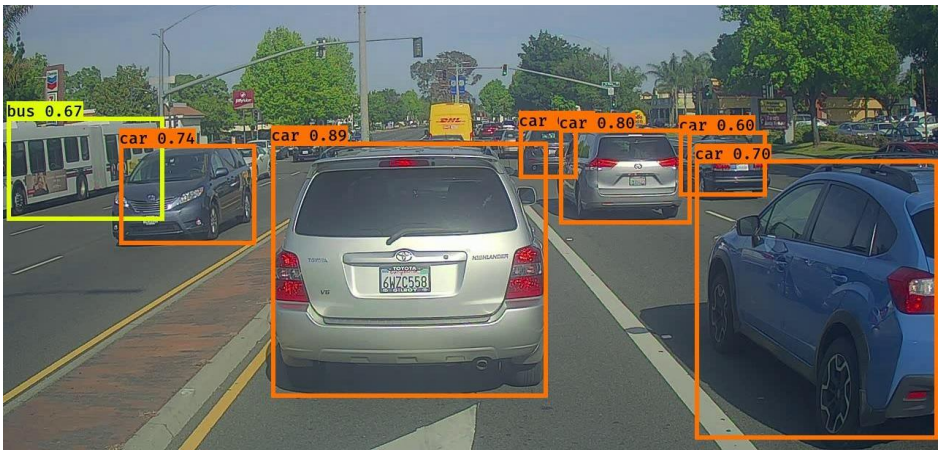


Fig 4.5 Output of YOLO algorithm[46]

4.1.2 Breaking traffic signal algorithm using Single Shot detector (SSD)

An imaginary line is drawn across the video, this line is then activated during a red traffic light. Whenever a vehicle then passes the red light breaking the traffic law or is overspeeding on a road, that vehicle is detected using object detection. This vehicle is then classified by the RCNN defined by us to classify it as a particular brand of vehicle. For example: Volkswagen Polo. This vehicle classifier is built with the help of a database of approx 20,000 images. The information about the vehicle is then stored in a Comma separated value (CSV) file. The direction of the vehicle, the class of the vehicle i.e. the brand and model of the vehicle and speed of the vehicle is noted in this file.

This file can then be easily accessed to obtain all the information stored from the video feed.

4.2 Congestion detection

Traffic in different lanes can be controlled and cleared effectively by using a simple algorithm explained below.

We consider four lanes in this example, denoted by T, B, L and R which stands for top, bottom, left and right respectively.

1. Now, we need to consider the presence of an emergency vehicle like an ambulance, fire brigade etc. on any of these lanes and try to clear the traffic in that lane on an urgent basis.
2. This can be done by using a sound sensor, to detect the sound of siren from such vehicles. Also to ensure if such a vehicle is genuinely present and the sound has not been falsely produced, the roads will be monitored by a CCTV simultaneously. If an emergency vehicle is detected, the red signal timer on that road must be decreased for it to quickly pass. Then wait for a buffer timer for signal lights to be set according to the newly detected traffic status.

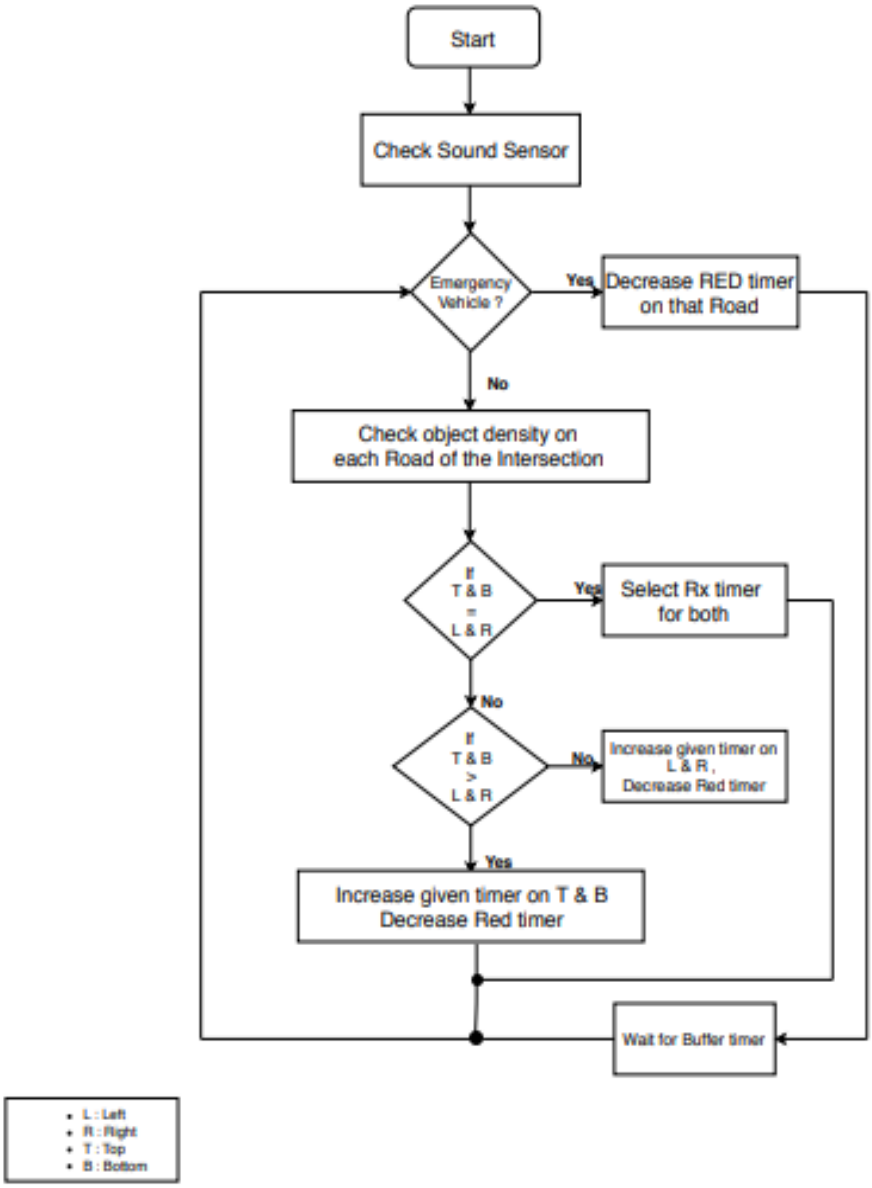


Fig 4.6 Proposed traffic control flowchart.

3. Object density on each road must be determined to control congestion. This is done using image processing techniques, where an image of a road is taken at two different intervals, converted to gray scale, and 1st image is subtracted from the 2nd, and then converted to binary. The end result would look somewhat like a collection of white dots or shapes which are nothing but the vehicles. The area of these white shapes is then calculated to find traffic density. This effectively determines real time traffic congestion.(refer literature survey)

4. If congestion on the top and bottom lanes is equal to the right and left lanes, then a predetermined timer is selected for both.
5. If vehicles on top and bottom lanes are more as compared to left and right lanes, then red signal timer on T and B should be decreased in order clear congestion faster.
6. However if left and right lanes are found to be more blocked, then red timer on these lanes must be decreased, with an increased green timer to ensure neither of these lanes end up getting over crowded.
7. This process of detecting congestion and accordingly increasing or decreasing signal timers is carried out, with simultaneous detection of an emergency vehicles as well.

4.2.1 Image Processing technique

The Image processing technique uses image processing techniques like grayscale conversion, thresholding, binary conversion and Mixture of Gaussians.In order to detect traffic density, the first method we resorted to was using image processing.

Like everything else we started coding this one in python too. The basic idea was to consider the entire video frame to be equivalent to a road, and find a ratio of the area occupied by the vehicles present on the road to the entire area of the frame (i.e. the road).

The video if first converted to binary. This is done by using a function from the opencv library. Opencv is imported by using the command import cv2. This function is called backgroundimagesubtractorMOG2. In order to detect moving objects (i.e. the vehicles) this function blackens the entire frame apart from any objects that are constantly in motion in the video, which get colored in white, thus also making it binary. The way this works is every pixel in the previous frame gets compared with the pixel in the same position, but in the next frame. The pixels that remain constant throughout every frame get blacked out being considered a part of the background. So if the pixel value changes in the next frame the MOG function considers such a line of changing pixels as a boundary of a moving object. Also checks for neighboring pixels of the varying pixels to precisely find out the boundary. The boundary ones constructed, all the pixels inside of the boundary (which are
obviously varying) get whitened out.

An n-dimensional Gaussian $N(\mu; \Sigma)$ has density function:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

For a Gaussian mixture model with components K, the component has a mean of μk and variance of σk. If the number of components k is known, expectation maximization is the technique most commonly used to estimate the mixture model's parameters. In frequentist probability theory, models are typically learned by using maximum likelihood estimation techniques, which seek to maximize the probability, or likelihood, of the observed data given the model parameters.

Expectation maximization (EM) is a numerical technique for maximum likelihood estimation, in this case the maximum likelihood of the matching of pixel properties in succeeding frames.

On applying this function, at times pixelated boundaries were obtained. So these were smoothened out using several steps of erosion, dilation and Gaussian blur commands to obtain an output that suited the best.

Now that the vehicles have been segregated from the rest, the area of vehicles needs to be determined. This has been done using the drawing contours function, again from the cv2 library. Any color can be chosen to draw these contours by typing the color codes. A command called findcontours does the task of finding the boundaries of moving objects, and the command drawcontours draws it. To neglect small faulty white pixel areas, a threshold can be set before drawing areas so as to only consider areas that are big enough to genuinely be that of a vehicle.

The areas are determined by using the cv2.findcontourarea, again an inbuilt function. This sums up areas from each frame (only of the objects in motion), so a list of areas gets listed. Now that we have the area from each frame, it means we can determine the traffic density at any instance. To make things easier these areas can be appended to a list and can be called according to what instance needs to be calculated.

The entire frame area is calculated by simply multiplying the length and breadth of the video frame. Each element of the list of areas can be called, divided by this entire frame area and multiplied by 100 to obtain in percentage the density factor. This would again give a list of

densities for each instance of the video. These values can also be appended to a list to and displayed, or stored for calculation.

To calculate an average density of how much traffic was present throughout the entire video length, an average of all these values can also be taken, to just obtain one value in the end and make computation easier.

However, after getting all these operations perfect, it was observed that as the traffic halts at a red light signal, the entire frame almost blackens out, since the MOG function subtracts the background keeping in account all the movement, and now there is no movement in the video. Hence no contours, no areas and no traffic present was detected, which failed the purpose.

fig 4.7 Congestion detection algorithm using Image processing

4.2.1 Mask R-CNN technique

In principle Mask R-CNN is an intuitive extension of Faster R-CNN, yet constructing the mask branch properly is critical for good results. Most importantly, Faster RCNN was not designed for pixel-to-pixel alignment between network inputs and outputs. This is most evident in how RoIPool, the de facto core operation for attending to instances, performs coarse spatial quantization for feature extraction. To fix the misalignment, we propose a

simple, quantization-free layer, called RoIAlign, that faithfully preserves exact spatial locations.

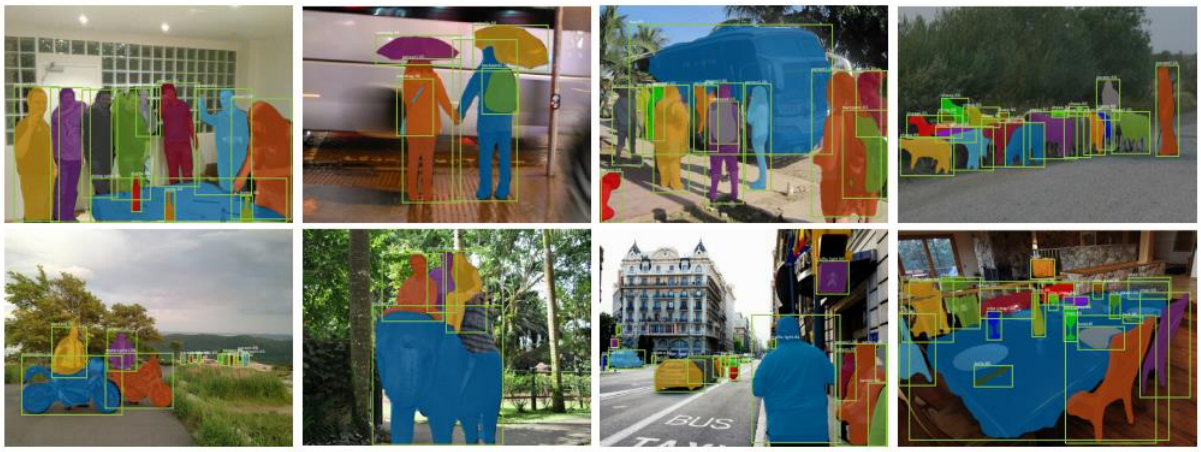


Figure 4.8. Mask R-CNN results on the COCO test set.

Our models run at about 200ms per frame on a GPU, and training on COCO takes one to two days on a single 8-GPU machine. We believe the fast train and test speeds, together with the framework's flexibility and accuracy, will benefit and ease future research on instance segmentation. By viewing each keypoint as a one-hot binary mask, with minimal modification Mask R-CNN can be applied to detect instance-specific vehicles. Mask R-CNN surpasses the winner of the 2016 COCO keypoint competition, and at the same time runs at 5 fps. Mask R-CNN, therefore, can be seen more broadly as a flexible framework for instance-level recognition and can be readily extended to more complex tasks like traffic management which we have used.

A mask encodes an input object's spatial layout. Thus, unlike class labels or box offsets that are inevitably collapsed into short output vectors by fully-connected (fc) layers, extracting the spatial structure of masks can be addressed naturally by the pixel-to-pixel correspondence provided by convolutions.

Specifically, we predict an m × m mask from each RoI using an FCN. This allows each layer in the mask branch to maintain the explicit m × m object spatial layout without collapsing it into a vector representation that lacks spatial dimensions. This pixel-to-pixel behavior requires our RoI features, which themselves are small feature maps, to be well aligned to faithfully preserve the explicit per-pixel spatial correspondence. This motivated us to develop the following RoIAlign layer that plays a key role in mask prediction.

4.3 Results

4.3.1 Congestion algorithm output

4.3.1.1 Image Processing technique

The results obtained from the Image processing technique contained holes and the accuracy while good did not beat the SSD method. The output obtained by using MOG and Image processing techniques is shown in the figure below.

fig 4.9 Congestion detection algorithm results using Image processing

4.3.1.2 Mask R-CNN

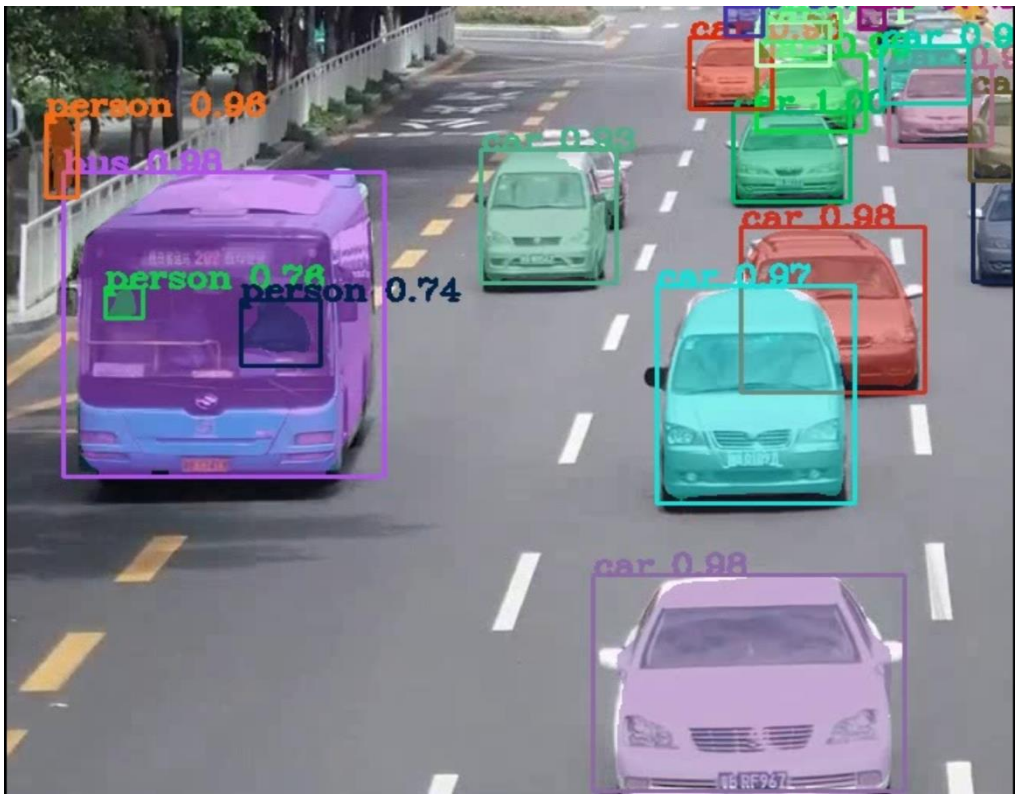


fig 4.10 Congestion detection using Mask R-CNN

4.3.2 Detection of Vehicle After Breach of Law

4.3.2.1 YOLO object detection

In the real time scenario, the YOLO detection failed to incorporate all the vehicles in a congested lane where vehicles often overlap and detected only 1 out of 2 or 3 vehicles in that scenario.

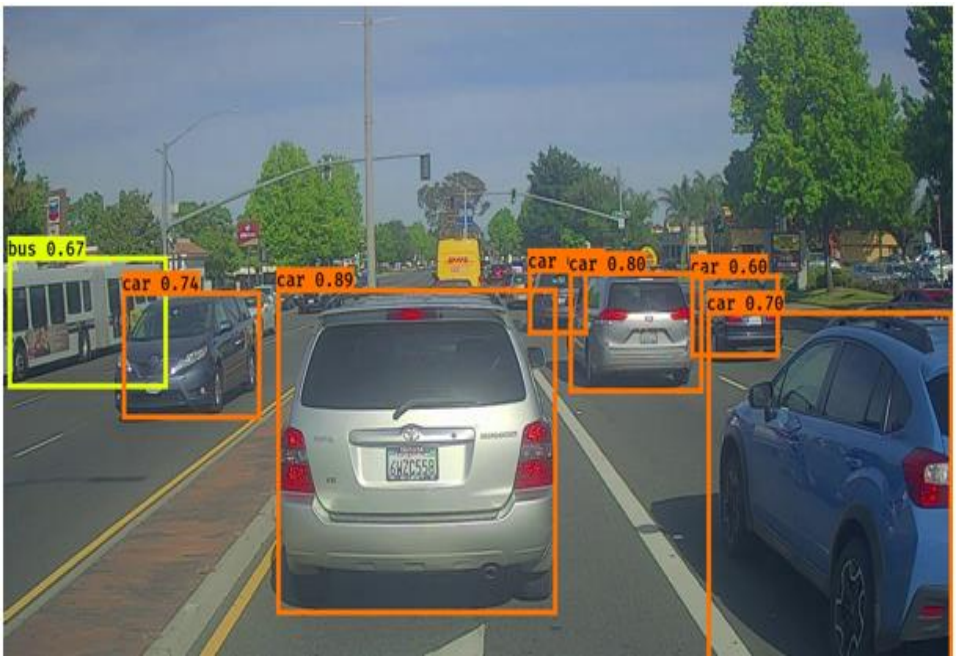


fig 4.11 : Vehicle detection using YOLO

Hence, owing to this disadvantage we moved on to SSD for vehicle detection during crossing of Red light.

4.3.2.2 SSD technique

This technique was used along with an imaginary line which can be activated during a red timer and any vehicle crossing this line will be detected and information of the vehicle as discussed in the Implementation section 4.1.2 will be stored in a CSV file.

The figures below shows the imaginary line drawn over the video feeds and vehicles being detected.

fig 4.12 Detection of vehicles breaking red signal

	A	B	C	D
1	Vehicle Type/Size	Vehicle Color	Vehicle Movement D	Vehicle Speed (km/h
2				
3	car	white	down	35.87156296
4				
5	car	blue	down	48.69215859
6				
7	car	blue	down	55.51172256

fig 4.13 Information of Detected vehicles breaking red signal

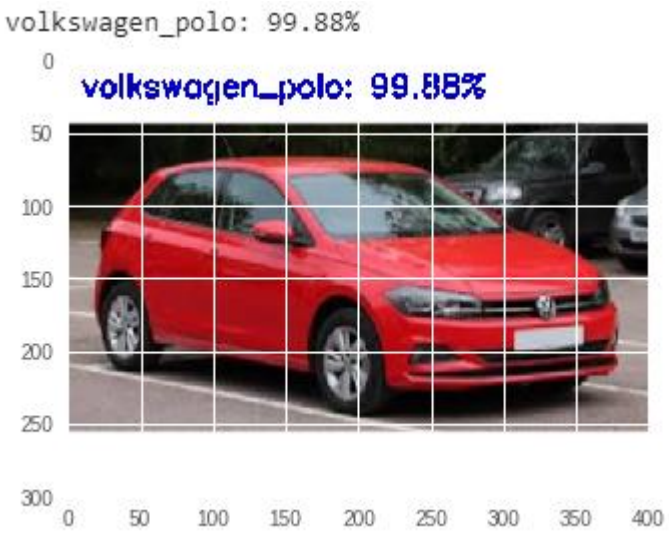


fig 4.14 classification of detected vehicle

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

India has experienced numerous traffic related issues most of which includes accidents due to poor traffic disciplinary. The traffic congestion problem is faced in all the major cities in India on a daily basis wasting precious time. The Automated Traffic Management System(ATMS) has aimed to solve these problems. The vehicle detection algorithm will help the traffic police by easing the processing information from camera footages. It will automatically detect the breach of traffic law and alert the authorities. The congestion algorithm will help in providing dynamic traffic controlling. This will result in fewer traffic jams and also reduce pollution levels considerably. Thus, ATMS will prove to be an efficient model to build upon for real time processing and traffic management.

5.2 Future Scope

The congestion control algorithm works with a single node and an algorithm can be developed to communicate with the neighboring traffic signals to give a more dynamic control over the traffic. Additionally, all the data obtained can be hosted on cloud and a web API can be developed for displaying a full fledged monitoring and control information.

The vehicle detection algorithm can be extended to bikes for helmet detection. Data set can be collected and trained on CNNs in a similar way and desirable results can be achieved.

REFERENCES

1. W.S. McCulloch, W. Pitts, A logical Calculus of the ideas immanent in nervous activity. Bull. Math. Biol. 5, 115–133 (1943)
2. J.V. Neumann, The General and Logical Theory of Automata (Wiley, New York, 1951)
3. J.V. Neumann, Probabilistic logics and the synthesis of reliable organisms from unreliable components, in Automata Studies (Princeton University Press, Princeton, 1956), 43–98
4. D.O. Hebb, The Organization of Behaviour: A Neuropsychological Theory (Wiley, New York, 1949)
5. F. Rosenblatt, Principles of Neurodynamics (Spartan Books, Washington, 1961)
6. M. Minsky, S. Papert, Perceptrons (MIT Press, Cambridge, 1969)
7. S. Amari, A theory of adaptive pattern classifiers. IEEE Trans. Electron. Compute. 16(3), 299–307 (1967)
8. K. Fukushima, Visual feature extraction by multi-layered networks of analog threshold elements. IEEE Trans. Syst. Sci. Cyber 5(4), 322–333 (1969)
9. S. Grossberg, Embedding fields: a theory of learning with physiological implications. J. Math. Psychol. 6, 209–239 (1969)
10. A.H. Klopf, E. Gose, An evolutionary pattern recognition network. IEEE Trans. Syst. Sci. Cyber 53, 247–250 (1969)
11. J.J. Hopfield, Neural Networks and physical systems with emergent collective computational abilities. Proc. Natl Acad. Sci. 79, 2254–2258 (1982)
12. J.J. Hopfield, Neurons with graded response have collective computational properties like those of two state neurons. Proc. Natl. Acad. Sci. 81, 3088–3092 (1984)

13. D.E. Rumelhart, J.L. McClelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, I and II (MIT Press, Cambridge, 1986)
14. F.J. Pineda, generalization of back propagation to recurrent and higher order neural networks. In proceedings of IEEE Conference on Neural Information Processing Systems, Denver, Colorado, November 1987.
15. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representation by error propagation. Parallel distributed processing: Explorations in the microstructure of cognition, volume I. Bradford Books, Cambridge, MA, 1986.
16. Y. le Cun, A learning schemefor asymmetric threshold networks. In Proceedings of Cognitiva 85, pages 599-604, Paris, France, 1985.
17. Y. le Cun, Learning processes in an asymmetric threshold network. In F. Fogelman-Soulie E. Bienenstock and G. Weisuch, editors, Disordered systems and biological organization, pages 233-240, Les Houches, France, 1986. Springer-Verlang
18. Y. le Cun, ModelesConnexionnistes de l'Apprentisage. PhD thesis, Universite Pierre et Marie Curie, Paris, France, 1987
19. VAPNIK, V. N. and CHERVONENKIS, A. YA. (1971): On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. Theory of Probability and its Applications, 16(2):264-280
20. Deng, Li, Li, Jinyu, Huang, Jui-Ting, Yao, Kaisheng, Yu, Dong, Seide, Frank, Seltzer, Michael, Zweig, Geoff, He, Xiaodong, Williams, Jason, et al. Recent advances in deep learning for speech research at microsoft. ICASSP 2013, 2013.
21. Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, 1097–1105, 2012.
22. Hinton, G.E. and Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. Science, 313 (5786):504–507, 2006.
23. Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. Signal Processing Magazine, IEEE, 29(6):82–97, 2012a.

24. Hinton, Geoffrey E, Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012b.
25. Graves, Alex, Mohamed, Abdel-rahman, and Hinton, Geoffrey. Speech recognition with deep recurrent neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, pp. 6645–6649. IEEE, 2013.
26. Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. The Journal of Machine Learning Research, 12:2121–2159, 2011.
27. Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report, 2012.
28. D. Cire¸san, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. Arxiv preprint arXiv:1202.2745, 2012.
29. N. Pinto, D.D. Cox, and J.J. DiCarlo. Why is real-world visual object recognition hard? PLoS computational biology, 4(1):e27, 2008.
30. B.C. Russell, A. Torralba, K.P. Murphy, and W.T. Freeman. Labelme: a database and web-based tool for image annotation. International journal of computer vision, 77(1):157–173, 2008.
31. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In CVPR09, 2009.
32. Y. LeCun, F.J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on, volume 2, pages II–97. IEEE, 2004.
33. K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In International Conference on Computer Vision, pages 2146–2153. IEEE, 2009..
34. A. Krizhevsky. Convolutional deep belief networks on cifar-10. Unpublished manuscript, 2010.
35. H. Lee, R. Grosse, R. Ranganath, and A.Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In Proceedings of the 26th Annual International Conference on Machine Learning, pages 609–616. ACM, 2009.

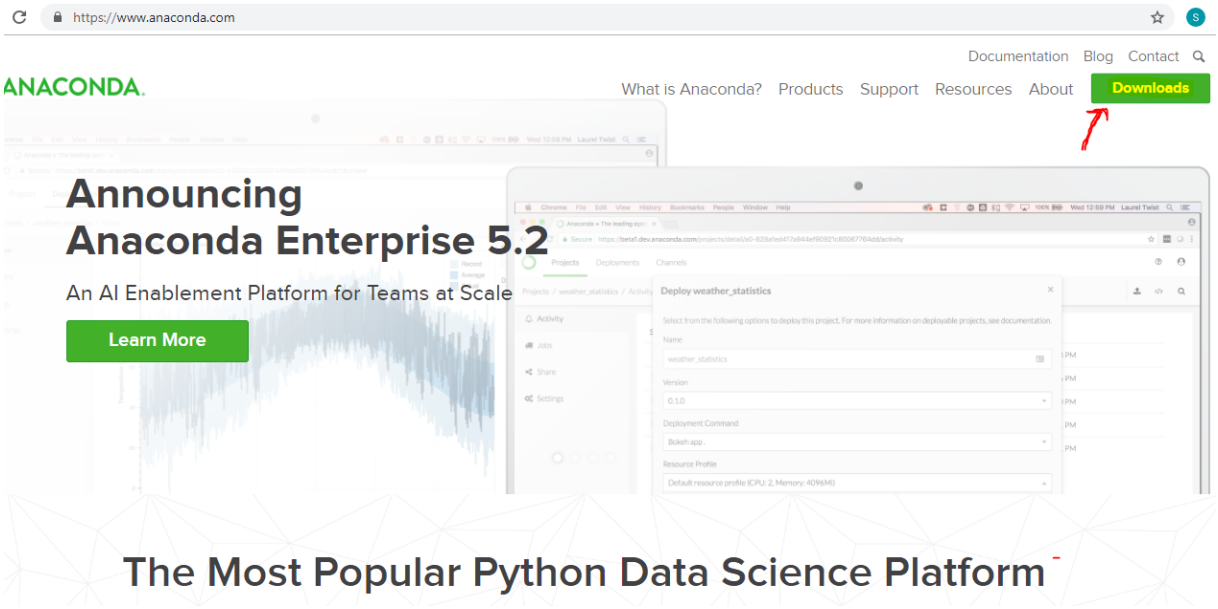
36. D.C. Cire¸san, U. Meier, J. Masci, L.M. Gambardella, and J. Schmidhuber. High-performance neural networks for visual object classification. Arxiv preprint arXiv:1102.0183, 2011.
37. N. Pinto, D. Doukhan, J.J. DiCarlo, and D.D. Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. PLoS computational biology, 5(11):e1000579, 2009.
38. S.C. Turaga, J.F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H.S. Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. Neural Computation, 22(2):511–538, 2010.
39. V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In Proc. 27th International Conference on Machine Learning, 2010.
40. K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In International Conference on Computer Vision, pages 2146–2153. IEEE, 2009.
41. R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
42. K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in European Conference on Computer Vision (ECCV), 2014. [2]
43. R. Girshick, "Fast R-CNN," in IEEE International Conference on Computer Vision (ICCV), 2015
44. J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
45. R. B. Girshick. Fast R-CNN. CoRR, abs/1504.08083, 2015.
46. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", arXiv:1506.02640 [cs.CV].
47. Reulke, S. Bauer, T. D¨oring, F. Meysel, "Traffic surveillance using multi-camera detection and multi-target tracking", Proceedings of Image and Vision Computing New Zealand 2007,175–180, Hamilton, New Zealand, December 2007.
48. Edwin Ospina1, Eliana Tascon1, Juan Valencia1 and Carlos Madrigal1, Member IEEE,"Traffic flow control using artificial vision techniques," 2011 IEEE.

49. Arif A. Bookseller, Rupali R Jagtap “Image processing based Adaptive Traffic Control System”, IOSR Journal of Electronics and Communication Engineering (IOSR-JECE), 33-37.
50. Siddharth Srivastava, Subhadeep Chakraborty, Raj Kamal, Rahil, Minocha, “Adaptive Traffic Light Control System”, IIT Kanpur, Nerd Magazine, India, March 2016.
51. Sanjoy Dasgupta, “Learning Mixtures of Gaussians”, University of California, Berkeley.

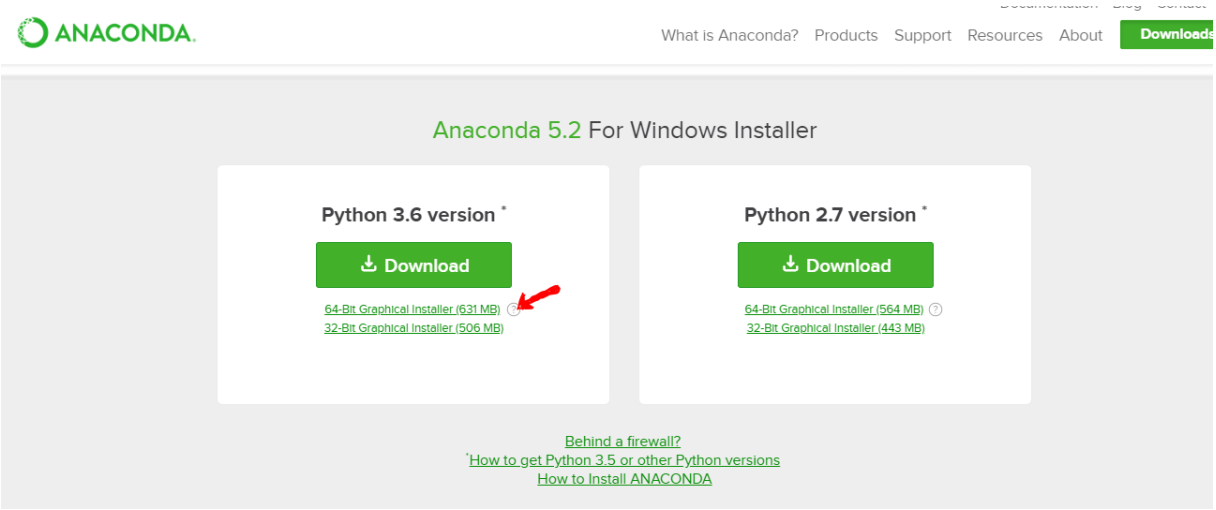
APPENDIX

MANUAL TO SETUP A DEEP-LEARNING WORKING ENVIRONMENT

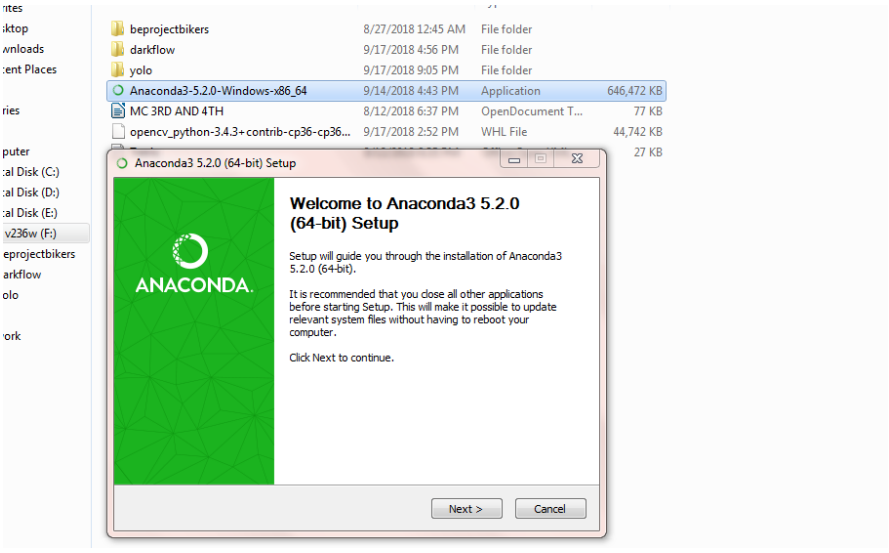
Download Anaconda from the official website of anaconda.



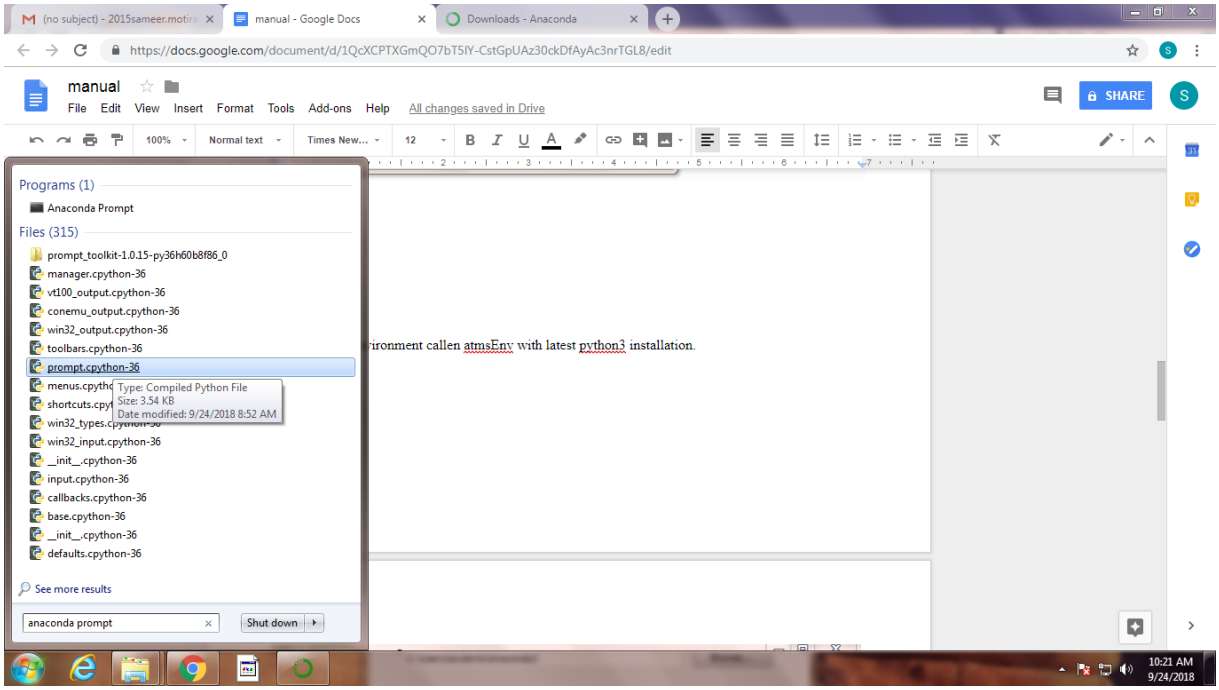
Download the 64 or 32-bit version according to your requirements, with python 3.x



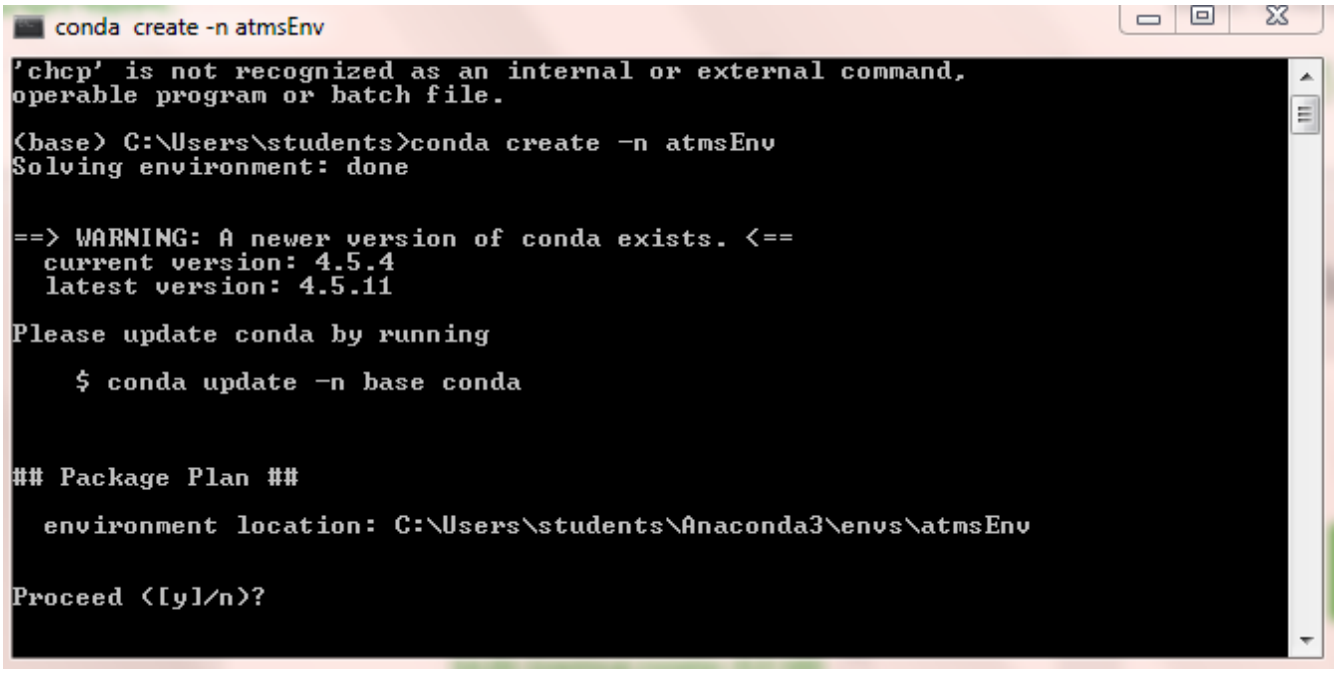
Click on the anaconda application file to start the installation, and keep proceeding until the installation is complete.



Once done installing, open anaconda prompt.



Create a new environment called atmsEnv with latest python3 installation using command conda create -n atmsEnv, where atmsEnv is the environment name.



Press y for every one of the below commands to execute the command successfully.
Activate the environment by using the command “activate atmsEnv”, where atmsEnv in this case is the environment name.

Install numpy scipy matplotlib

```
(base) C:\Users\students>activate atmsEnv
'chcp' is not recognized as an internal or external command,
operable program or batch file.

(atmsEnv) C:\Users\students>conda install numpy scipy matplotlib
```

Install Jupyter notebook

```
(atmsEnv) C:\Users\students> conda install jupyter notebook
```

Install Tensorflow – CPU

```
(atmsEnv) C:\Users\students>pip install tensorflow
```

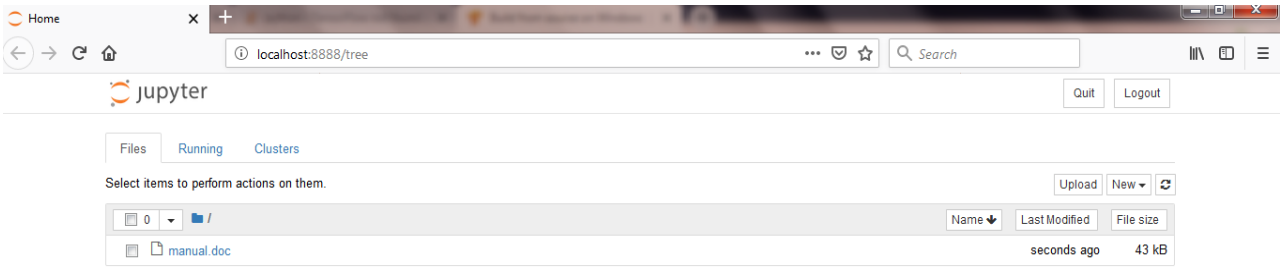
If the above command throws an error, use the command: conda install tensorflow.
Similarly install pillow and keras

Now make a working directory and cd into it. Here the working directory is names ATMS.

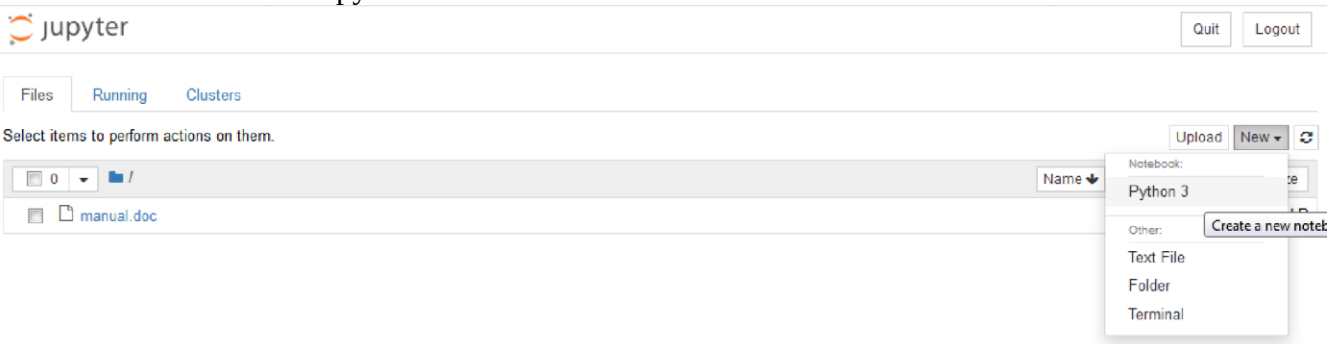
Run jupyter notebook from anaconda prompt

```
(atmsEnv) C:\Users\students>cd D:\ATMS

(atmsEnv) C:\Users\students>D:

(atmsEnv) D:\ATMS>jupyter notebook
```

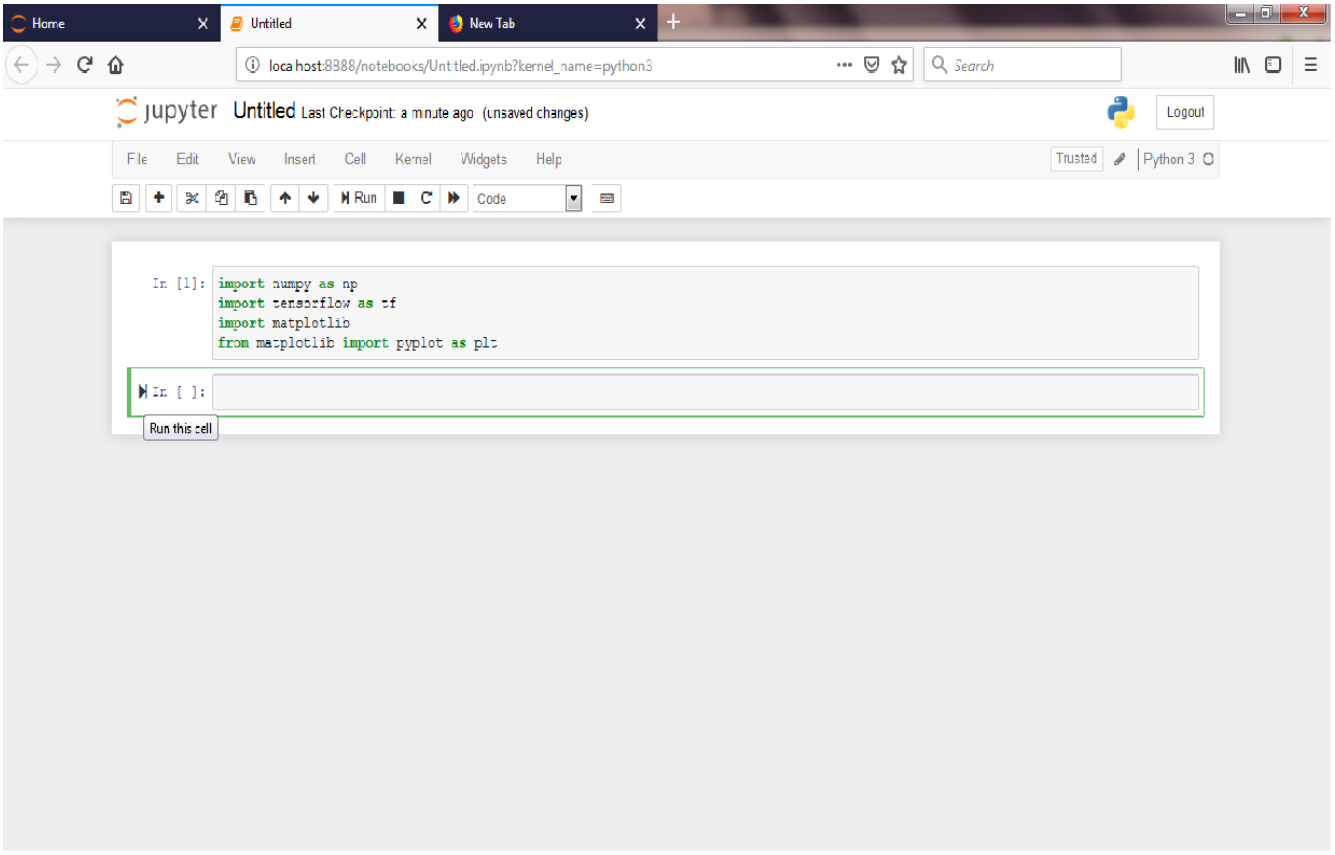
A new web page will be opened in browser.



Click on new and create a python 3 notebook.



After creating the notebook, we can type commands in the code blocks and execute the cells individually. This is the importance of using jupyter notebook.



Now we can enter commands sequentially or block wise and execute them seamlessly.

What is Anaconda?

Anaconda is a free and open source distribution of the Python and R programming languages for data science and machine learning related applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Anaconda distribution comes with more than 1,000 data packages as well as the Conda package and virtual environment manager, called Anaconda Navigator , so it eliminates the need to learn to install each library independently.

The following applications are available by default in Navigator:**JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio,Visual Studio Code**

Important libraries used:

1. Numpy: compiles python code for fast multiplication and array manipulation. Much faster than ordinary python data structures and looping mechanisms. Numpy is used almost exclusively in all applications data science, image, and sound processing using python. It's very fast and efficient.

2. Matplotlib: Matplotlib offers way of plotting graphs and images on the command window of a jupyter notebook and separate graphical windows outside of it. For inline plotting, it must be imported inline inside of a jupyter notebook.
3. OpenCV: OpenCV is an open source image-processing library available in C++ and Python. We use the python version here. It has all the tools that MATLAB offers and some more because the library is developer driven and is open source. Thus, it is very fast and easy to use. OpenCV can be used to read and manipulate images for various activities like flipping of the image, rotation, zooming, accentuation of edges, etc. It can also be used for video applications by rendering images using FFMPEG library.
4. Tensorflow: Neural Networks can be implemented in numpy by creating tensors and applying matrix multiplication operations on them. But the process of doing such operating often becomes difficult to debug and defining each and every function for back propagation, activation functions, updating of weights, etc. requires complex coding. This can be bypassed using a high level library like Tensorflow or TF. TF allows execution of these functions by simple calls of methods inside the big library. Tensorflow has some additional benefits of memory management for huge data, fast computation and easy execution as compared to low level code. TF is used almost exclusively for Deep Learning applications.
5. Keras: For creation of even more complex networks, tensorflow code can get large. Even though it is optimized, many small things need to be taken care of. Keras does that for you. Keras is yet higher implementation of tensorflow. Thus, when creating high level networks, when accuracy is all that matters, we use keras to alleviate all the difficulties for us. Code written in keras is coherent with the terminologies used in Deep Learning and makes the code look clean.
6. Jupyter Notebook: Jupyter Notebook is a library based on python and JavaScript. Python code which is executed in command prompt can't be executed in a bunch of commands and stored. And in .py scripts, the commands can't be individually executed. Hence, we need a middle ground to write blocks of code that can be executed individually. Jupyter notebook provides exactly this environment by creation of IPython notebooks with a .ipynb extension. We can define methods, or classes or write simple commands outside those functions using global variables. This freedom to execute commands as we wish makes Jupyter Notebooks very crucial in Deep Learning Projects.

Need of Sophisticated Hardware:

As we have seen above, deep learning applications require high level matrix multiplications and updating operations. GPU or graphics processing unit of a computer is a device which is used primarily for games as there are similar calculations involved. Hence they can be used in Deep Learning applications to speed up the processing of training by a huge factor.

Minimum Hardware requirements:
Intel i5 or equivalent quad core CPU
8 GB RAM
Atleast 100GB harddisk space to store training and testing datasets for subsequent applications.

Recommended Hardware:
Intel i7 CPU, quad core
>16 GB RAM
256 GB SSD
Nvidia GPU with CUDA compute capability >3.0 and >4GB memory

Here is a sample of its execution with reference to this project.

1. Vehicle Classification: Training CNN for multi-class classification (brand of car in our project)
    a. Collect a huge dataset of thousands of images divided into various folders with the name of the folder representing the class name.
    b. Edit "*train.py*" - you can change the size of input for the VGGnet default is (128,128). Images will be stored in RAM with that size. So larger dimensions will lead to a larger use of RAM.
    c. Then vary the hyper-parameters like:
        i. No of epochs
        ii. Learning rate and
        iii. Batch size
    d. Train the network by giving as input

"python train.py --data data --labelbin lb.pickle --model your_model_name.model --plot plot.png"

e. If you have GPU, it will be used, else CPU. The network will start training now.
f. On a GPU for 7000 images and 100 epochs, approximately 40 mins are required.
g. If the training loss decreases very slowly or doesn't decrease, manipulate the learning rate. The Goldilocks learning rate will require some trial and error. Using VGG net architecture, we obtained about 97% accuracy and with RESNET architecture with pretrained weights, around 99.7% can be obtained ideally if the dataset is good.
h. For testing, edit the file "*test.py*" and change the name of model string to "*your_model_name.model"* and label binarizer string to "*lb.pickle*". Replace the default string in the read image function for OpenCV cv2.imread() to the file you want to test inference on.
i. The output will be the image with text written on it dictating the accuracy and confidence score of the CNN that the image belongs to a certain class.
j. This model will be used in the signal breaking code to detect the class of the vehicle and store the data in a CSV file.

Loading Images as Input.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam,Nadam
from keras.preprocessing.image import img_to_array
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from vggnet import SmallerVGGNet
import matplotlib.pyplot as plt
from imutils import paths
import numpy as np
import argparse
import random
import pickle
import cv2
import os
```

Import Libraries For Deep Learning

```
#construct argument parse and parse the arguments
# ap = argparse.ArgumentParser()
# ap.add_argument("-d","--data",required = True,
#                 help = "path to input dataset")
# ap.add_argument("-m","--model",required = True,
#                  help = "path to model")
# ap.add_argument("-l","--labelbin",required=True,
#                 help = "path to output label binarizer")
# ap.add_argument("-p","--plot",type=str,default="plot.png",
#                 help="path to output accuracy vs loss plot")
# args = vars(ap.parse_args())
#initialize epochs learning rate batch size and img dims
epochs = 100
lr = 0.0002
bs = 64
IMAGE_DIMS = (128,128,3)
#initialize data and labels
data = []
labels = []
print("[INFO] loading images ...")
imagePaths = sorted(list(paths.list_images('data')))
random.seed(42)
random.shuffle(imagePaths)
f = 1
#loop over input images
for imagePath in imagePaths:
    #load the image,preprocess it, and store it in data list
    image = cv2.imread(imagePath)
    if image is not None:
        image = cv2.resize(image,(IMAGE_DIMS[1],IMAGE_DIMS[0]))
        image = img_to_array(image)
        data.append(image)

        #extract class label; from the image path and update labels list
        label = imagePath.split(os.path.sep)[-2]
        labels.append(label)
        f+=1

print(f)
#scale pixel intensities from 0-255 to 0-1
data = np.array(data,dtype="float")/255.0
labels = np.array(labels)
print("[INFO] data matrix: {:.2f}MB".format(data.nbytes/(1024*1000.0)))
#binarize labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
print(labels)
```

Hyper Parameters

Training the network from the dataset.

```
#partition the data into training and testing splits using 80%
#of the data for trainning and the remaining 20% for testing
(trainX,testX,trainY,testY) = train_test_split(data,labels,
    test_size=0.2,random_state = 42)
"""Data Augmentation is IMP cos low data"""
#construct image generator for data augmentation
aug = ImageDataGenerator(rotation_range = 25,width_shift_range=0.1,
                         height_shift_range=0.1,shear_range=0.2,zoom_range=0.2,
                         horizontal_flip=True,fill_mode="nearest")
#initialize the model
print("[INFO] compiling model...")
model = SmallerVGGNet.build(width=IMAGE_DIMS[1],height=IMAGE_DIMS[0],
                            depth=IMAGE_DIMS[2],classes=len(lb.classes_))
opt = Nadam(lr=lr)
#categorical crossentropy as multi-class output
#for 2 classes outputs, use binary crossentropy
model.compile(loss="categorical_crossentropy",optimizer=opt,
              metrics=["accuracy"])

#train the network
print("[INFO] training network...")
H = model.fit_generator(aug.flow(trainX,trainY,batch_size=bs),
                        validation_data=(testX,testY),
                        steps_per_epoch=len(trainX)//bs,
                        epochs=epochs,verbose=1)
```

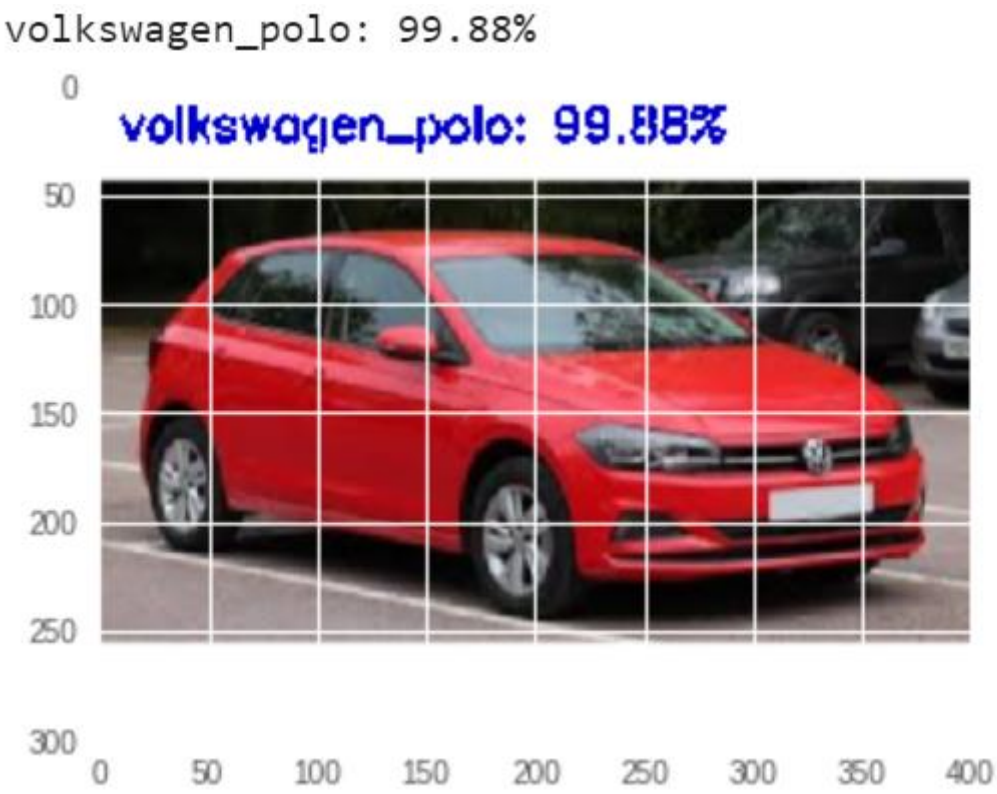
Training, output:

```
[INFO] loading images ...
6126
[INFO] data matrix: 2352.00MB
[[1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [1 0 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 1 0 0]]
[INFO] compiling model...
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be rem
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
[INFO] training network...
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future versi
Instructions for updating:
Use tf.cast instead.
Epoch 1/100
76/76 [==============================] - 37s 491ms/step - loss: 3.3910 - acc: 0.1560 - val_loss: 3.1502 - val_acc: 0.2261
Epoch 2/100
76/76 [==============================] - 29s 381ms/step - loss: 2.7603 - acc: 0.2228 - val_loss: 2.8412 - val_acc: 0.2612
Epoch 3/100
76/76 [==============================] - 29s 381ms/step - loss: 2.5349 - acc: 0.2593 - val_loss: 2.6687 - val_acc: 0.2914
Epoch 4/100
76/76 [==============================] - 29s 379ms/step - loss: 2.3920 - acc: 0.2998 - val_loss: 2.8627 - val_acc: 0.2751
Epoch 5/100
76/76 [==============================] - 29s 382ms/step - loss: 2.2895 - acc: 0.3259 - val_loss: 3.1984 - val_acc: 0.2914
Epoch 6/100
```

Testing Code:

```
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import numpy as np
import argparse
import imutils
import pickle
import cv2
import os


#load the model
model = load_model("vehicle0.model")
lb= pickle.loads(open("lb.pickle","rb").read())

image1 = cv2.imread("examples/IMG_20181105_165644.jpg")
op1 = image1.copy()

image1 = cv2.resize(image1,(128,128))
image1 = image1.astype("float")/255.0
image1 = img_to_array(image1)
image1 = np.expand_dims(image1,axis=0)

print("Classifying images...")
prob1 = model.predict(image1)[0]
idx1 = np.argmax(prob1)
label1 = lb.classes_[idx1]

# we'll mark our prediction as "correct" of the input image filename
# contains the predicted label text (obviously this makes the
# assumption that you have named your testing image files this way)
'''filename = args["image"][args["image"].rfind(os.path.sep) + 1:]
correct = "correct" if filename.rfind(label)!=-1 else "correct"'''

#build label and draw it on image
label1 = "{}: {:.2f}%".format(label1,prob1[idx1]*100)
op1 = imutils.resize(op1,width=400)
cv2.putText(op1,label1,(10,25),cv2.FONT_HERSHEY_SIMPLEX,0.7,(200,0,0),2)
print(label1)
%matplotlib inline
import numpy as np
import skimage
from skimage import data
from matplotlib import pyplot as plt

img1 = op1
plt.imshow(cv2.cvtColor(img1, cv2.COLOR_RGB2BGR))
plt.show()
```

Testing Output:



2. Signal Breaking:

1. Signal Breaking code using Single Shot Detection
2. SSD algorithm is a pretrained model trained in COCO dataset for object detection and has obtained extremely well accuracy results in VOC and ILSVRC competitions.
3. The model developed by Ross Girshick was trained by the darknet and the weights are available from the darknet website.
4. SSD 500 is the model used by us to perform object detection on the video frames. It gives very high mAP values along with decent frame rates up to 20fps, which is quite good for human vision.
5. You can use either the latest YOLO v3 or SSD 500 depending on fps and mAP requirements. Comparison between both was mentioned in the literature survey.
6. To run inference on a video, copy its name and edit the string in the "*vehicle_detection_main.py*" to paste the name there. In the code, the position of ROI

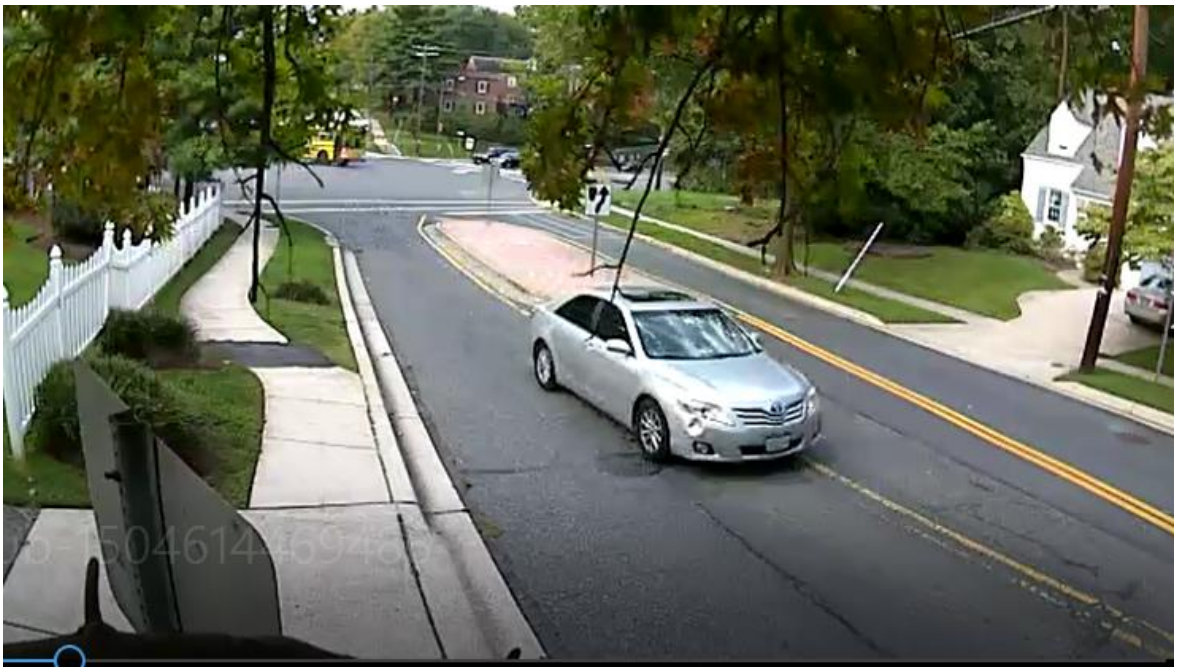
line can be modified by changing the coordinates of the line in function cv2.line() according to the road requirements.

7. Before detecting signal breaking, ensure that all libraries have been installed in the anaconda environment. Then in the anaconda prompt, type the command
8. "*python vehicle_detection_main.py*", ensure a GPU is present in the PC. This will make inference run at ideal speeds. As soon as the command is entered, the video will load and details of all vehicles crossing the ROI will be recorded in the CSV file.
9. All the lines in the code are well commented so that a layman can read them and understand their function in the code.

Input statement to load the input video feed using cv2.VideoCapture:

```
# input video
cap = cv2.VideoCapture('sub-1504614469486.mp4')
```

A sample input video frame is as shown below:



After running through the code you will get a video output. The output video will look like this.

	A	B	C	D
1	Vehicle Type/Size	Vehicle Color	Vehicle Movement D	Vehicle Speed (km/h
2				
3	car	white	down	35.87156296
4				
5	car	blue	down	48.69215859
6				
7	car	blue	down	55.51172256

1. Mask R-CNN:
    a. For running Mask R-CNN on a video, open the notebook in Google colaboratory for inference on Nvidia Titan K80 GPU or in your PC if the hardware is good.
    b. For 12GB GPU, a fully trained Mask R-CNN algorithm works at around 5fps.
    c. On a stack of 5 Titan K80 GPUs connected in parallel processing ecosystem, and on COCO dataset, it takes around 1 week to train, proving that training is not feasible if you have little resources.
    d. The model is very robust and is the current state-of-the-art.

e. Google Colab is just like a Jupyter Notebook running on Google Cloud Servers for free.
f. To run each snippet of code individually, press shift+enter or click on the run icon at the left of the code window.
g. Initial steps in the code import python coco tools, which make inferencing on pretrained codes on COCO easier.
h. If it runs into any error, your workstation doesn't contain some of the other library required for this project.
i. Please install them first.
j. When the environment is ready, provide the path to coco dataset annotations and labels and the folder where the model is stored.
k. Configuration of the network can be seen from Inference Config
l. Next, we recreate the model as demonstrated in the Mark R-CNN paper.
m. We fill the model with pretrained weights.
n. For inference on the video, the frame rate of the video is converted to 5fps so that at a time, Mask R-CNN can provide computation on 5 images at a time.
o. Each batch of images is stored locally. At the end of the training, these images are collated into a video and then the folder is deleted.

As an example, we show a few frames of the input video and their corresponding outputs:

Inputs:


13-10-2016 12:22:


13-10-2016 12:26:

Outputs:


13-10-2016 12:23:
person 0.86
car 0.87
person 0.76
person 0.90
truck


13-10-201 12:22:
car 0.70
person 0.94
bottle 0.77


13-10-2016 12:26:
truck 0.74
car 0.94
car 0.87
chair 0.91
person 0.94
motorcycle 0.91
motorcycle 0.80