VEHICLE DETECTION SYSTEM

Submitted in partial fulfilment of the requirement

for the award of the degree of

Bachelor of Technology

In

Computer Science & Engineering

Submitted By:

Shivam	**(60176802714/CSE3/2014-18)**
Abhishek Saklani	**(00876802714/CSE3/2014-18)**
Sourajit Sanyal	**(00476802714/CSE3/2014-18)**
Hasneet Singh	**(03676802714/CSE3/2015-18)**

Under The Guidance Of:

Mrs. Neetu Singh

Department of Computer Science & Engineering

Guru Tegh Bahadur Institute of Technology

Guru Gobind Singh Indraprastha University

Dwarka, New Delhi

Year 2014-2018

DECLARATION

We hereby declare that all the work presented in the dissertation entitled "**Vehicle Detection System"** in the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science Engineering**, Guru Tegh Bahadur Institute of Technology, affiliated to Guru Gobind Singh Indraprastha University Delhi is an authentic record of our work carried out under the guidance of **Mrs**. **Neetu Singh.**

Date: Shivam (60176802174/CSE3)

Abhishek Saklani (00876802714/CSE3)

Sourajit Sanyal (00476802714/CSE3)

Hasneet Singh (03676802714/CSE3)

CERTIFICATE

This is to certify that dissertation entitled **“Vehicle Detection System”** which is submitted by **Mr. Shivam, Mr. Abhishek Saklani, Mr. Sourajit Sanyal** and **Mr. Hasneet Singh** in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science Engineering**, Guru Tegh Bahadur Institute of Technology, New Delhi is an authentic record of the candidate’s own work carried out by them under our guidance. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

Mrs. Neetu Singh

(Project Guide)

Mr. Ashish Bhardwaj

(Head Of Department)

Computer Science Engineering

Date:

ACKNOWLEDGEMENTS

We would like to express our great gratitude to **Mrs. Neetu Singh** to provide us the opportunity to conduct this project as the minor project which not only increased our awareness about the latest field but also taught us the importance of discipline & dedication towards our work. Without their help we could not have presented this dissertation up to the present standard. We are highly thankful to **Mr. Ashish Bhardwaj** (HOD) for giving us the opportunity.

We choose this moment to acknowledge the contribution of each team member gratefully.

Date:

Shivam
(60176802714/CSE3/2014-18)
shivamsardanaavi@gmail.com

Abhishek Saklani
(00876802714/CSE3/2014-18)
abhisheksaklani96@gmail.com

Sourajit Sanyal
(00476802714/CSE3/2015-18)
sjsanyal96@gmail.com

Hasneet Singh
(03676802714/CSE3/2014-18)
hasneet.singh96@gmail.com

ABSTRACT

The project is Vehicle Detection System focusing on vehicle detection which can be used in autonomous vehicle, self-driving cars, drones. By this system we can reduce the accidents a very large rate and help in making the travel safer. We do not have to do everything manually in nowadays, we can do it easily with the help of artificial intelligence, which will be very fast and easy to use and make travel safe.

The system would be responsible for detecting the vehicles on road and calculating the distance between two vehicles and hence making the vehicle to slow down the speed and avoiding the chances of accident. It is one of the major module in Advanced Driver Assistance Systems.

.

LIST OF FIGURES

S.NO.	FIGURE NAME	PAGE NO.
1.	Classification vs Regression	2
2.	YOLO Architecture	3
3.	YOLO CNN Output Format	3
4.	Level 0 DFD	15
5.	Level 1 YOLO DFD	16
6.	Level 1 HOG+SVM DFD	17
7.	Use case Diagram	18
8.	Activity Diagram	20
9.	YOLO original image and detected vehicle	73
10.	More examples of detected vehicles by YOLO	73
11.	HOG View Dataset	74
12.	Potential Search Area for Car	74
13.	Combine various sliding window approaches	75
14.	Heatmap	75
15.	SciPy Labels to Heatmap	76
16.	Draw bounding box for Labels	76
17.	More examples of detected vehicles by HOG-SVM	77

INDEX

CONTENTS

Chapter	Page No.
Title Page	i
Declaration	ii
Certificate	iii
Acknowledgement	iv
Abstract	v
List of Figures	vi
Index	vii
1. Introduction	1
2. Software Requirements Specification	5
2.1 Introduction about SRS	6
2.2 Goals of SRS	6
2.3 SRS About Vehicle Detection System	6
2.3.1 Purpose	7
2.3.2 Scope	7
2.3.3 Functional Requirements	7
2.3.4 Non-Functional Requirements	7
2.3.5 Functional Requirements For Tool For Learning	8
2.3.6 Non-Functional Requirements For Tool For Learning	8
2.4 Functionality	9
2.4.1 Usability	9
2.4.2 Mean Time between Failure (MTBF)	9
2.4.3 Access Reliability	9
2.5 Performance	9
2.5.1 Response Time	9

2.5.2 Capacity 9
2.5.4 Resource Utilization 9
2.6 Supportability 10
2.6.1 Hardware Support 10
2.6.2 Software Support 10
2.7 Technologies Used System Design 10
3. System Design 12
3.1 Client Server Architecture 13
3.2 Data Flow Diagram 13
3.2.1 Level 0 DFD 15
3.2.2 Level 1 DFD (YOLO) 16
3.2.3 Level 1 DFD (HOG+SVM) 17
3.3 Use Case Diagram 18
3.4 Activity Diagram 20
3.5 Feasibility Study 21
3.5.1 Technical feasibility 21
3.5.2 Operational feasibility 21
3.5.3 Economical feasibility 22
4. System Testing 23
4.1 Introduction 24
4.2 Testing Methodologies 24
4.2.1 Unit Testing 25
4.2.2 System Testing 25
4.2.3 Performance Testing 25
5. Conclusion and Future Scope 26
5.1 Conclusion 27
5.2 Future Scope 27
➢ References 28
➢ Appendix A-Source Code 29
1. YOLO 30
2. HOG+SVM 41
➢ Appendix B- Screenshots 72

CHAPTER ONE

INTRODUCTION

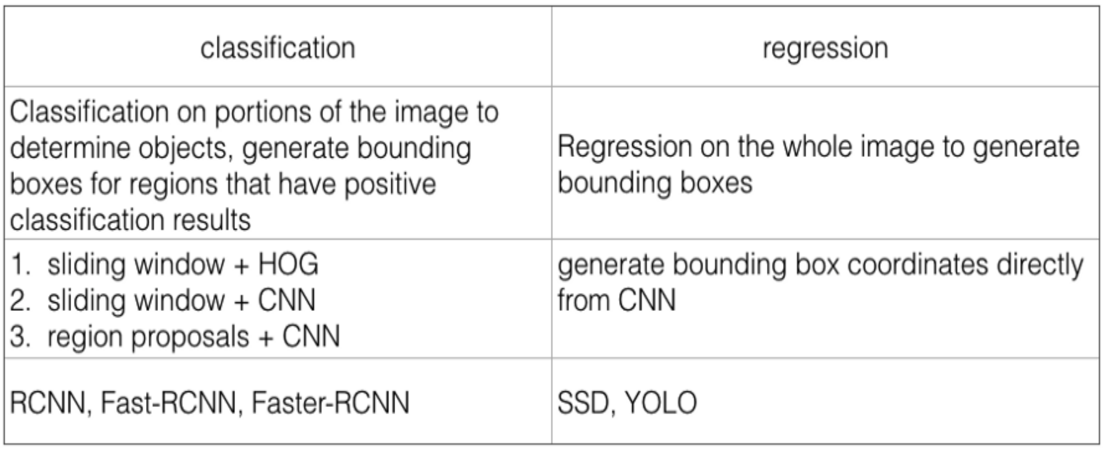
Introduction:

This project is for VEHICLE DETECTION SYSTEM which can be used in self-driving cars, Advanced Driver Assistance Systems(ADAS). By this system we can avoid the accident in collision with other vehicles. We do not have to do everything manually in nowadays, we can do it easily with the help of advanced computers, which will be very fast and easy to use in ADAS.

This project consists of two object detection algorithms used for detection of vehicles.

- YOLO (You Only Look Once) Object Detection
- HOG (Histogram of Gradients) + SVM (Support Vector Machine)

Detecting vehicles in a video stream is an object detection problem. An object detection problem can be approached as either a classification problem or a regression problem. In the classification approach, the image are divided into small patches, each of which will be run through a classifier to determine whether there are objects in the patch. The bounding boxes will be assigned to patches with positive classification results. In the regression approach, the whole image will be run through a convolutional neural network directly to generate one or more bounding boxes for objects in the images.



classification	regression
Classification on portions of the image to determine objects, generate bounding boxes for regions that have positive classification results	Regression on the whole image to generate bounding boxes
1. sliding window + HOG 2. sliding window + CNN 3. region proposals + CNN	generate bounding box coordinates directly from CNN
RCNN, Fast-RCNN, Faster-RCNN	SSD, YOLO

Fig: 1 Classification vs Regression

- **The YOLO approach** of the object detection is consisting of two parts: the neural network part that predicts a vector from an image, and the postprocessing part that interpolates the vector as boxes coordinates and class probabilities.
  For the neural network, the tiny YOLO v1 is consist of 9 convolution layers and 3 full connected layers. Each convolution layer consists of convolution, leaky relu and max pooling operations. The first 9 convolution layers can be understood as the feature extractor, whereas the last three fully-connected layers can be understood as the "regression head" that predicts the bounding boxes. There are a total of 45,089,374 parameters in the model.

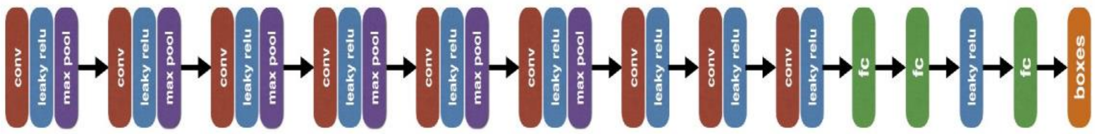


Fig2: YOLO Architecture

The output of this network is a 1470 vector, which contains the information for the predicted bounding boxes. The information is organized in the following way.

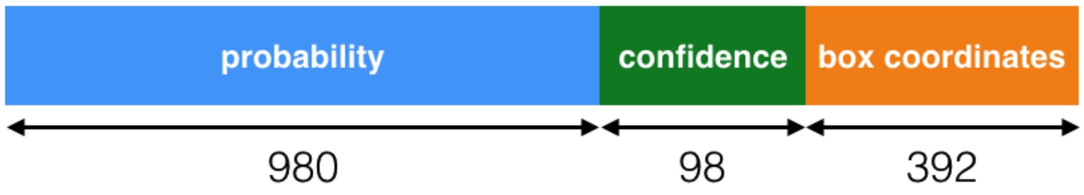


Fig3: YOLO CNN Output Format

The 1470 vector output is divided into three parts, giving the probability, confidence and box coordinates. Each of these three parts is also further divided into 49 small regions, corresponding to the predictions at the 49 cells that form the original image. In postprocessing steps, we take this 1470 vector output from the network to generate the boxes that with a probability higher than a certain threshold.

- **The HOG+SVM approach** uses structural cues like shape give a more robust representation. Gradients of specific directions captures some notion of shape. To allow for some variability in shape, Histogram of Oriented Gradients (HOG) is used.

  Number of orientations, pixels per cell, and cells per block for computing the HOG features of a single channel of an image. The number of orientations is the number of orientation bins that the gradients of the pixels of each cell will be split up in the histogram. The pixels per cells is the number of pixels of each row and column per cell over each gradient the histogram is computed. The cells per block specifies the local area over which the histogram counts in each cell will be normalized. Having this parameter is said to generally lead to a more robust feature set.

  Linear SVM classifier algorithm is used to classify the car, Sliding Window and Heatmap algorithm are used to make it more robust.

CHAPTER TWO

SOFTWARE REQUIREMENTS SPECIFICATION

SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Introduction about SRS

A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

The software requirements specification document enlists enough and necessary requirements that are required for the project development. To derive the requirements, we need to have clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

2.2 Goals of SRS

The Software Requirements Specification (SRS) is a communication tool between stakeholders and software designers. The specific goals of the SRS are:

- Facilitating reviews.
- Describing the scope of work.
- Providing a reference to software designers (i.e. navigation aids, document structure).
- Providing a framework for testing primary and secondary use cases.
- Including features to customer requirements.
- Providing a platform for on-going refinement (via incomplete specs or questions).

2.3 SRS About Vehicle Detection System

The Software Requirement Specification will provide a detailed description of the requirements of the vehicle detection system. This SRS will allow for the complete understanding of what is to be expected from the newly introduced system which is to be constructed. The clear understanding of the system and its functionality will allow for the correct software to be developed for the end user and will be used for the development of the future stages of the project.

2.3.1 Purpose

This specification document describes the capabilities that will be provided by the software application “VEHICLE DETECTION SYSTEM”. It also takes the various required constraints by which the system will abide. The intended audience for this document are the development team, testing team and end users of the product.

2.3.2 Scope

The software product “VEHICLE DETECTION SYSTEM” is a system which is developed to provide real-time detection of the vehicles (weather stationary or mobile) on the roads. An input stream is passed into the system which is capturing the video from the vehicle dashboard. At the time of input the system will detect the all the vehicles occurring in the field of view and creating bounding boxes around them, and calculating the distance between the autonomous vehicle and the tracked vehicles so that the autonomous vehicle can limit its speed.

2.3.3 Functional Requirements

Functional requirements specify the business requirements of the project in detail. Usually business requirements are specified in terms of the actions that user performs on the software system. This is known as the use case model. But not all requirements need to be specified as use cases. Functional requirements should contain a combination of use cases and plain textual description of system features. System features are specified at a higher level and use cases attempt to translate into user actions.

2.3.4 Non-Functional Requirements

Non-functional or technical requirements specify how the software system should operate. In contrast functional requirements specify what a software system should do. Some of the non-functional requirements are derived from the functional requirements. Non-functional requirements captured include performance requirements, application scalability, application security, maintainability, usability, availability, logging etc

2.3.5 Functional Requirements for Tool for learning language

1. Fast Image Processing

The algorithm can run several threads on multi-core graphics processors making the recognition several times faster.

2. Accurate object detection

- Whether a particular object is present in a scene;
- Where the object is located within the scene;
- How many instances of the object occur in the scene.

3. Objects tracking

The tracking works with complex backgrounds and fast-moving objects. Tracking is initialized when an object is recognized and located using bounding boxes, and then tracks the object until it changes somewhat in appearance, at which point tracking is reinitialized by recognition.

2.3.6 Non-Functional Requirements for Tool for learning language

- **Reliability**

The system allows the user to track moving and stationary vehicles in real-time by creating bounding boxes around the vehicle, with great precision.

- **Availability**

The availability should be 24X7. In case of any hardware or software failure the backup of the database and code should be maintained.

- **Maintainability**

A system is used for maintaining the files along with the customized name. In case of a failure, a re-initialization of the program will be done. Also the software design is being done with modularity in mind so that maintainability can be done efficiently.

- **Portability**

The system is built using the concepts of Python and Artificial Neural Networks. The application will run on any OS having 8GB RAM support and with external graphics.

2.4 Functionality

2.4.1 Usability

• The system allow users to track vehicles in real time weather stationary or mobile.

• Since all users are familiar with the general usage video camera, so no specific training is required.

• The system is user friendly and self- explanatory.

2.4.2 Mean Time between Failures (MTBF)

The system is developed in such a way that it may not be failed.

2.4.3 Access Reliability

The system shall provide 100% access reliability.

2.5 Performance

2.5.1 Response Time

The response time of the system depends mostly on the ability of the graphics processors to quickly process the input stream. The information is refreshed every time a new file is added. The access time for a computer should be less than 1 minute. The system shall respond to the member in not less than two seconds from the time of the request submittal. The system shall be allowed to take more time when doing large processing jobs.

2.5.2 Capacity

The system is capable of handling one user at a time.

2.5.3 Resource Utilization

The resources are modified according the user requirements.

2.6 Supportability

The system designers shall take into consideration the following supportability and technical limitations.

2.6.1 Hardware Support

Processor	: Intel i5 2.1 GHZ or above
Clock speed	: 2.1 GHZ
System bus	: 64 bits
RAM	: 8GB of RAM
HDD	: 40 GB or higher
Monitor	: SVGA COLOR
Keyboard	: 108 keys
Mouse	: 2 button mouse
Graphic Card	: Nvidia GTX 980 Graphics Card

2.7 Technologies Used

1. Computer Vision:

Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

Computer Vision Libraries used:

- OpenCV

2. Machine Learning:

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the develop the learning ability of model.

Machine Learning Libraries used:

- Scikit-Learn
- NumPy

3. Deep Learning:

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, partially supervised or unsupervised.

Deep Learning Libraries used:

- Keras

4. Language Used: Python

CHAPTER 3
SYSTEM DESIGN

SYSTEM DESIGN

3.1 CLIENT SERVER ARCHITETURE

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced.

In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

3.2 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top-down approach to Systems Design. This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modelled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users are able to visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's dataflow diagrams can be drawn up and compared with the new system's data flow diagrams to draw comparisons to implement a more efficient system. Data flow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately has an effect upon the structure of the whole system from order to dispatch to report. How any system is developed can be determined through a data flow diagram model.

3.2.1 Level 0 DFD

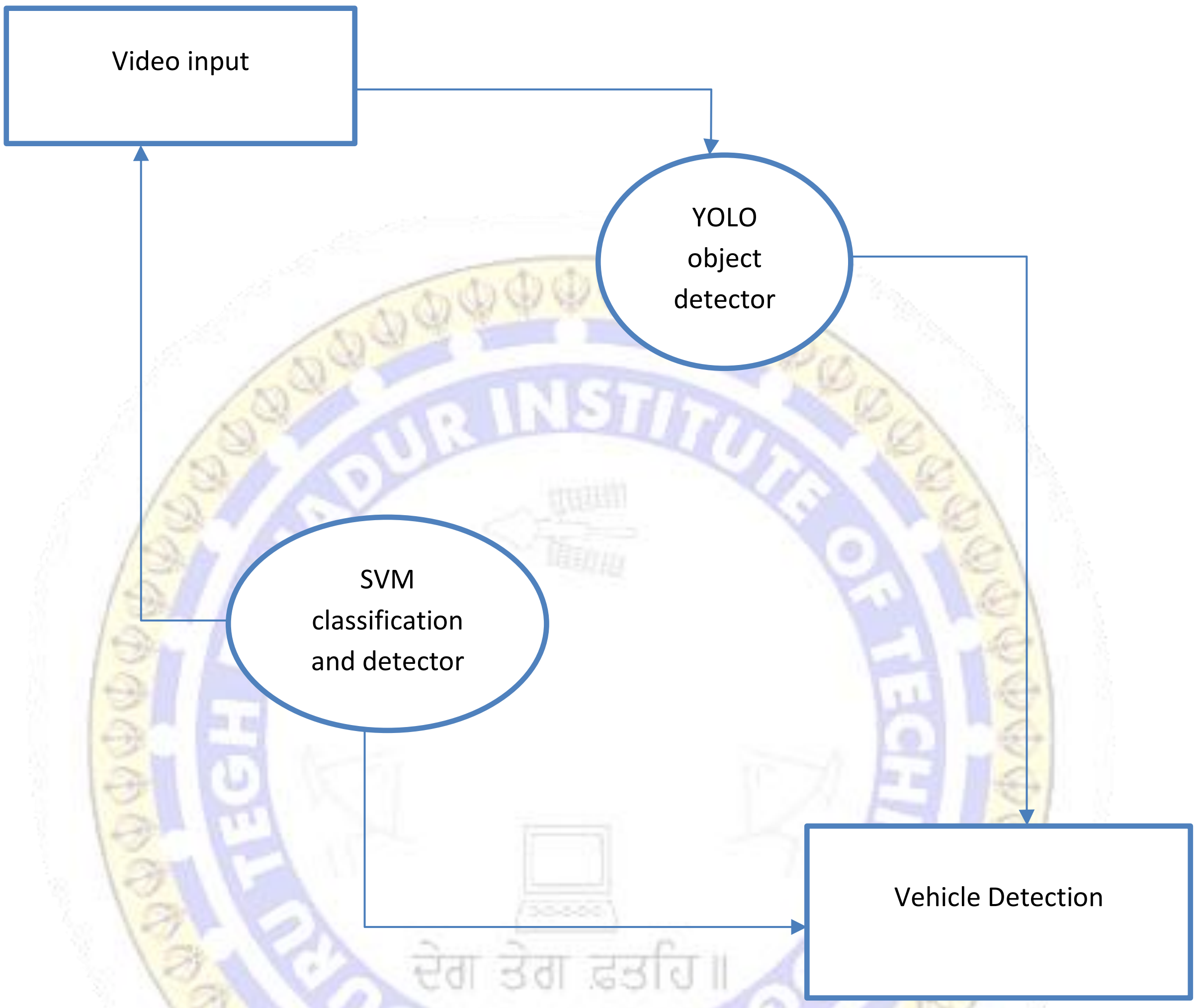


Fig.4 Level 0 DFD

3.2.2.1 Level 1 DFD (YOLO)

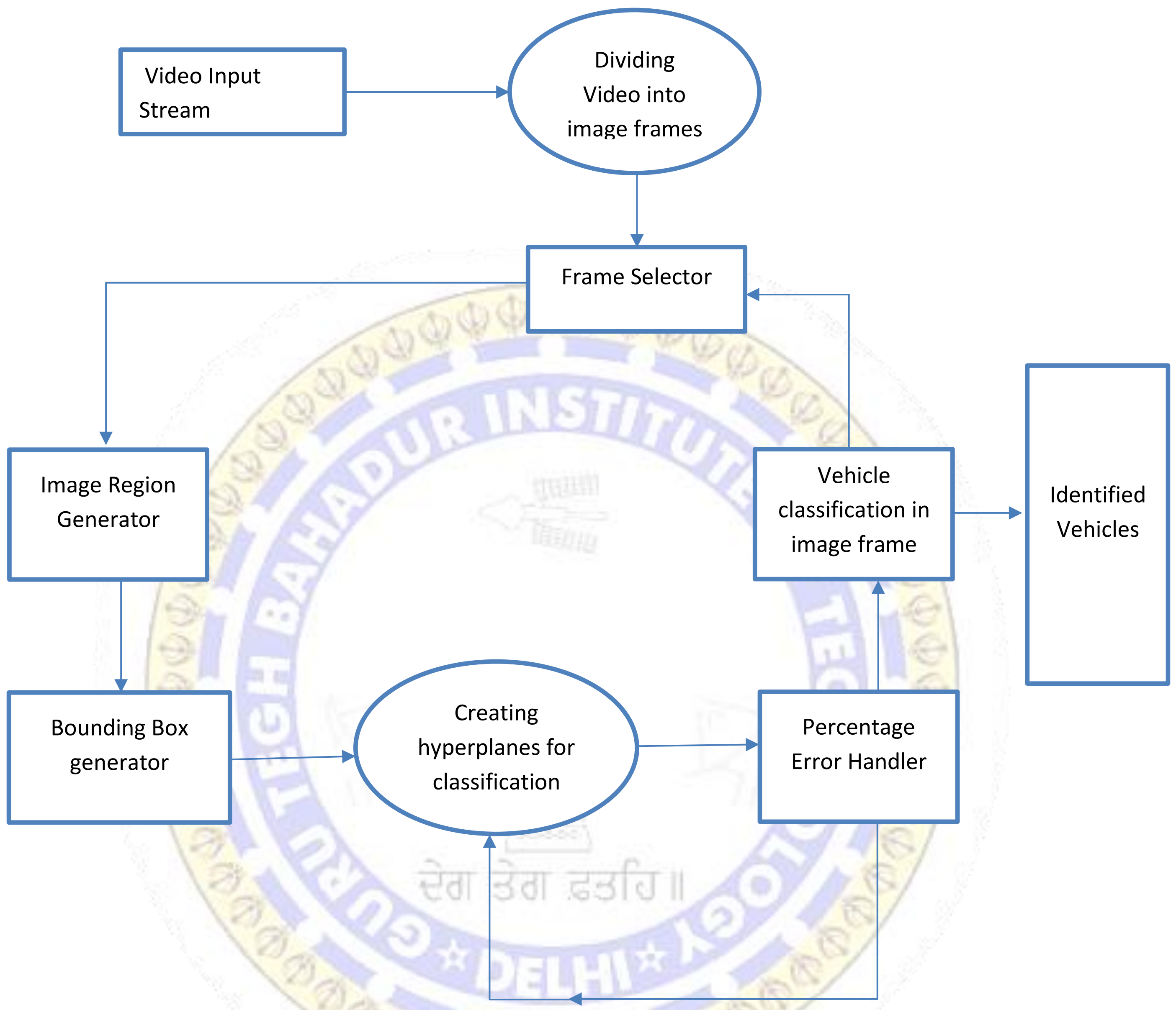


Fig.5 Level 1 YOLO DFD

3.2.2.2 Level 1 DFD (HOG+SVM)

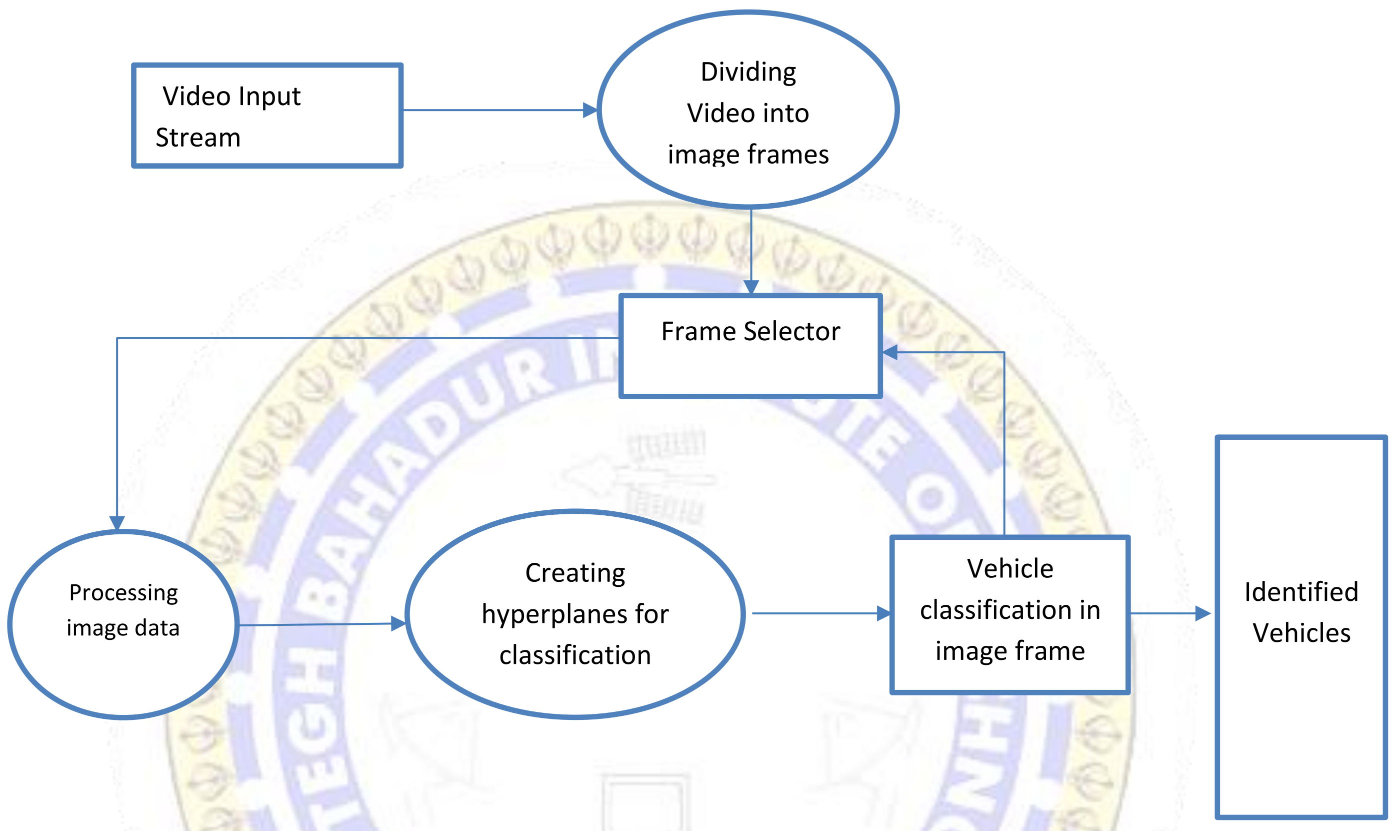

Video Input Stream
Dividing Video into image frames
Frame Selector
Processing image data
Creating hyperplanes for classification
Vehicle classification in image frame
Identified Vehicles

Fig.6: Level 1 HOG+SVM DFD

3.3 USE CASE DIAGRAM

Use case diagrams are usually referred to as behaviour diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system. Note, that UML 2.0 to 2.4 specifications also described use case diagram as a specialization of a class diagram, and class diagram is a structure diagram.

.

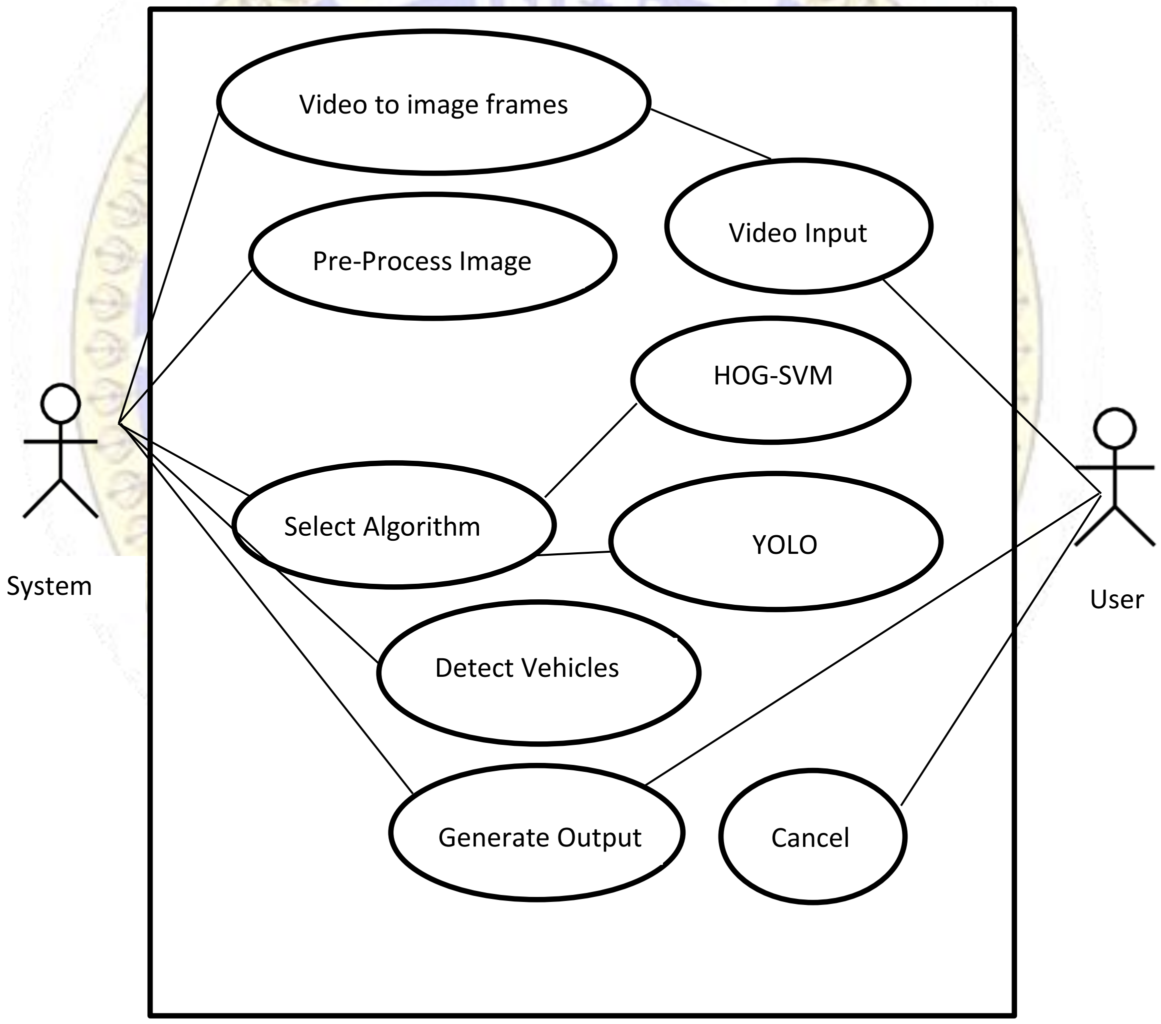


Fig.7: Use Case Diagram

3.4 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows). Activity diagrams show the overall flow of control.

Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:

- rounded rectangles represent actions;
- diamonds represent decisions;
- bars represent the start (split) or end (join) of concurrent activities;
- a black circle represents the start (initial node) of the workflow;
- an encircled black circle represents the end (final node).

Arrows run from the start towards the end and represent the order in which activities happen.

Activity diagrams may be regarded as a form of flowchart. Typical flowchart techniques lack constructs for expressing concurrency. However, the join and split symbols in activity diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops.

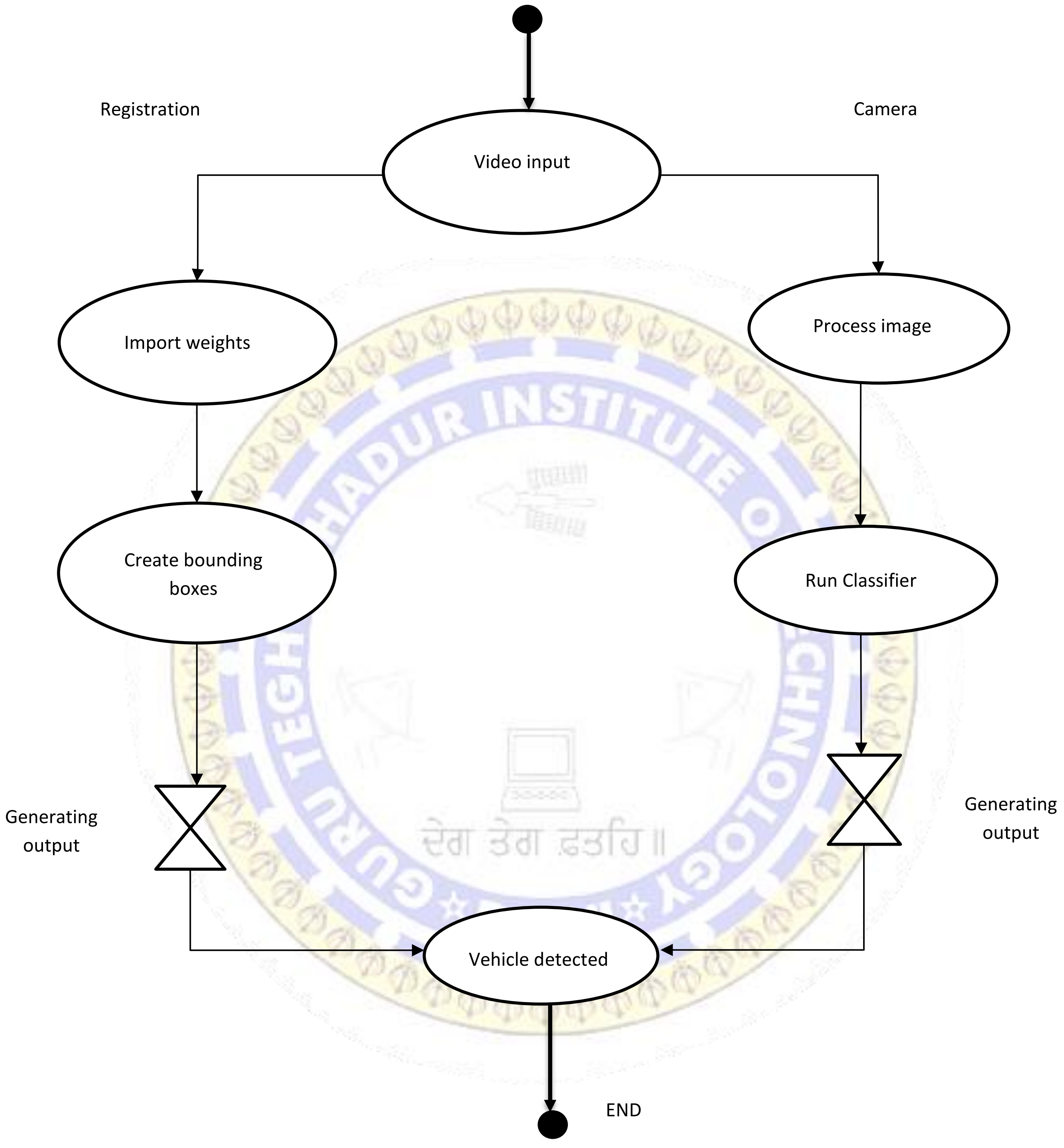


Fig.8 Activity Diagram

3.5 FEASIBILITY STUDY

An important outcome of the preliminary investigation is the determination that the system requested is feasible. Feasibility study is carried out to select the best system that meets the performance requirements.

Feasibility study is both necessary and prudent to evaluate the feasibility of the project at the earliest possible time. It involves preliminary investigation of the project and examines whether the designed system will be useful to the organization.

The different types of feasibility are:

Technical feasibility,

Operational feasibility,

Economic feasibility.

3.5.1 Technical feasibility

Technical Feasibility deals with the hardware as well as software requirements. Technology is not a constraint to type system development. We have to find out whether the necessary technology, the proposed equipment's have the capacity to hold the data, which is used in the project, should be checked to carry out this technical feasibility.

The technical feasibility issues usually raised during the feasibility stage of investigation includes these

a. This software is running in windows 2007 Operating System, which can be easily installed

b. The hardware required is Pentium based server.

c. The system can be expanded.

3.5.2 Operational feasibility

The proposed system offers greater level of user-friendliness. The proposed system produces best results and gives high performance. It can be implemented easily. So this project is operationally feasible.

3.5.3 Economical feasibility

Economical feasibility deals about the economic impact faced by the organization to implement a new system. Financial benefits must equal or exceed the costs. The cost of conducting a full system, including software and hardware cost for the class of application being considered should be evaluated. Economic Feasibility in this project:

a. The cost to conduct a full system investigation is possible.

b. There is no additional manpower requirement.

c. There is no additional cost involved in maintaining the proposed system.

CHAPTER FOUR

SYSTEM TESTING

SYSTEM TESTING

4.1 Introduction

Once source code has been generated, software must be tested to uncover (and correct) as many errors as possible before delivery to customer. Our goal is to design a series of test cases that have a high likelihood of finding errors. To uncover the errors software techniques are used. These techniques provide systematic guidance for Designing test that

(1) Exercise the internal logic of software components, and

(2) Exercise the input and output domains of the program to uncover errors In program function, behaviour and performance.

Steps: Software is tested from two different perspectives:

(1) Internal program logic is exercised using -White box test case design Techniques.

(2) Software requirements are exercised using -black box test case design techniques.

In both cases, the intent is to find the maximum number of errors with the Minimum amount of effort and time.

4.2 Testing Methodologies

A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements. A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when deadline pressure begins to rise, progress must be measurable and problems must surface as early as possible. Following testing techniques are well known and the same strategy is adopted during this project testing.

4.2.1 Unit testing

Unit testing focuses verification effort on the smallest unit of software design. Using the procedural design description, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of test and uncovered errors is limited by the constrained scope established for unit testing. The unit test is normally white box oriented, and the step can be conducted in parallel for multiple modules.

4.2.2 System testing

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Below we have described the two types of testing which have been taken for this project. it is to check all modules worked on input basis. if you want change any values or inputs will change all information. So specified input is must. After the unit and integration testing system testing is done on the software as a whole. This type of testing is closer to everyday experience. Goal of system testing is not to find faults but to demonstrate performance

4.2.3 Performance Testing

Performance testing is designed to test the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as white box tests are conducted. It will generate report fast. Entered correct data will show result in few milliseconds. Just uses only low memory of our system. Automatically do not getting access at another software.

CHAPTER FIVE

CONCLUSION & FUTURE SCOPE

CONCLUSION & FUTURE SCOPE

5.1 Conclusion

This project is VEHICLE DETECION SYSTEM which can be used in autonomous vehicles, self-driving car and Advanced Driver Assistance Systems. By this system we can reduce our work. We do not have to do everything manually in nowadays, we can do it easily with the help of pcs, which will be very fast and easy to use. A forward pass of an entire image through the network is more expensive than extracting a feature vector of an image patch and passing it through an SVM. However, this operation needs to be done exactly once for an entire image, as opposed to the roughly 150 times in the SVM+HOG approach. A linear SVM processes video at a measly 3FPS on an i5 CPU.YOLO, a blazingly fast convolutional neural network for object detection on a fast GPU (GTX 1080) the video gets processed at about 65FPS. YOLO is more than 20x faster than the SVM+HOG and at least as accurate.

The problem of implementing object detection and classification on Indian roads is that Indian road signs are not well organized and printed. There is blurring and occlusion which lead to noise in images. One model which works one foreign roads necessarily may not work for Indian roads. This project can be extended to many dimensions such as self-driving vehicles, autonomous drones, robotics, ADAS. For its enhancement it will need more mathematical advancement and better hardware.

5.2 Future Scope

- Traffic Sign detection can be made
- Object detection could be made which could help in parking and obstacle detection
- Path/lane finding could be made so that can be helpful as lane keeping assist
- Can be used in autonomous drones, robotics, self-driving cars
- Limitation is it fails to identity some cars that are far away or not in dataset therefore it will be improved in future versions.
- Get tighter bounding boxes on vehicles themselves and to detect indian vehicles also and it could be solved by acquiring a training image set with more vehicle images

REFERENCES

- J. Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016
- Jiang Guo, Jun Cheng, Jinxing Pang, Yu Guo, "Real-time hand detection based on multi-stage HOG-SVM classifier"
- Heong-tae Kim and Bongsob Song, “Vehicle recognition based on radar
- and vision sensor fusion for automatic emergency braking,”
- http://cs231n.stanford.edu/reports/2016/pdfs/263_Report.pdf
- http://scikit-learn.org/stable/modules/svm.html
- https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
- https://keras.io/11
- https://docs.python.org/2/library/glob.html
- https://matplotlib.org/api/pyplot_summary.html
- https://docs.scipy.org/doc/
- http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/
- https://www.coursera.org/learn/convolutional-neural-networks
- http://deeplearning.net/tutorial/lenet.html
- https://in.udacity.com/course/deep-learning-nanodegree-foundation--nd101
- https://github.com/thtrieu/darkflow

APPENDIX – A

SOURCE CODE

YOLO

vechicle_detection.py

```
import numpy as np

import matplotlib.pyplot as plt

import cv2

import glob

import keras

from keras.models import Sequential

from keras.layers.convolutional import Convolution2D, MaxPooling2D

from keras.layers.advanced_activations import LeakyReLU

from keras.layers.core import Flatten, Dense, Activation, Reshape

from utils import load_weights, Box, yolo_net_out_to_car_boxes, draw_box

keras.backend.set_image_dim_ordering('th')


model = Sequential()

model.add(Convolution2D(16, (3, 3),input_shape=(3,448,448),padding='same',strides=(1,1)))

model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Convolution2D(32,(3,3) ,padding='same'))

model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D(pool_size=(2, 2),padding='valid'))

model.add(Convolution2D(64,(3,3) ,padding='same'))

model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D(pool_size=(2, 2),padding='valid'))

model.add(Convolution2D(128,(3,3) ,padding='same'))

model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2, 2),padding='valid'))

model.add(Convolution2D(256,(3,3) ,padding='same'))

model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D(pool_size=(2, 2),padding='valid'))

model.add(Convolution2D(512,(3,3) ,padding='same'))

model.add(LeakyReLU(alpha=0.1))

model.add(MaxPooling2D(pool_size=(2, 2),padding='valid'))

model.add(Convolution2D(1024,(3,3) ,padding='same'))

model.add(LeakyReLU(alpha=0.1))

model.add(Convolution2D(1024,(3,3) ,padding='same'))

model.add(LeakyReLU(alpha=0.1))

model.add(Convolution2D(1024,(3,3) ,padding='same'))

model.add(LeakyReLU(alpha=0.1))

model.add(Flatten())

model.add(Dense(256))

model.add(Dense(4096))

model.add(LeakyReLU(alpha=0.1))

model.add(Dense(1470))

model.summary()

load_weights(model,'./yolo-tiny.weights')


imagePath = './test_images/test1.jpg'

image = plt.imread(imagePath)

image_crop = image[300:650,500:,:]

resized = cv2.resize(image_crop,(448,448))

batch = np.transpose(resized,(2,0,1))
batch = 2*(batch/255.) - 1

batch = np.expand_dims(batch, axis=0)

out = model.predict(batch)

boxes = yolo_net_out_to_car_boxes(out[0], threshold = 0.17)

f,(ax1,ax2) = plt.subplots(1,2,figsize=(16,6))

ax1.imshow(image)

ax2.imshow(draw_box(boxes,plt.imread(imagePath),[[500,1280],[300,650]]))

plt.show()
```

/utils

utils.py

```
import numpy as np

import cv2




def load_weights(model, yolo_weight_file):

    tiny_data = np.fromfile(yolo_weight_file, np.float32)[4:]



    index = 0

    for layer in model.layers:

        weights = layer.get_weights()

        if len(weights) > 0:

            filter_shape, bias_shape = [w.shape for w in weights]

            if len(filter_shape) > 2:  # For convolutional layers

                filter_shape_i = filter_shape[::-1]

                bias_weight = tiny_data[index:index + np.prod(bias_shape)].reshape(bias_shape)

                index += np.prod(bias_shape)

                filter_weight = tiny_data[index:index + np.prod(filter_shape_i)].reshape(filter_shape_i)

                filter_weight = np.transpose(filter_weight, (2, 3, 1, 0))

                index += np.prod(filter_shape)

                layer.set_weights([filter_weight, bias_weight])

            else:  # For regular hidden layers

                bias_weight = tiny_data[index:index + np.prod(bias_shape)].reshape(bias_shape)

                index += np.prod(bias_shape)

                filter_weight = tiny_data[index:index + np.prod(filter_shape)].reshape(filter_shape)
            index += np.prod(filter_shape)

            layer.set_weights([filter_weight, bias_weight])



class Box:

    def __init__(self):

        self.x, self.y = float(), float()

        self.w, self.h = float(), float()

        self.c = float()

        self.prob = float()



def overlap(x1, w1, x2, w2):

    l1 = x1 - w1 / 2.;

    l2 = x2 - w2 / 2.;

    left = max(l1, l2)

    r1 = x1 + w1 / 2.;

    r2 = x2 + w2 / 2.;

    right = min(r1, r2)

    return right - left;



def box_intersection(a, b):

    w = overlap(a.x, a.w, b.x, b.w);

    h = overlap(a.y, a.h, b.y, b.h);

    if w < 0 or h < 0: return 0;
    area = w * h;

    return area;



def box_union(a, b):

    i = box_intersection(a, b);

    u = a.w * a.h + b.w * b.h - i;

    return u;



def box_iou(a, b):

    return box_intersection(a, b) / box_union(a, b);



def yolo_net_out_to_car_boxes(net_out, threshold=0.2, sqrt=1.8, C=20, B=2, S=7):

    class_num = 6

    boxes = []

    SS = S * S  # number of grid cells

    prob_size = SS * C  # class probabilities

    conf_size = SS * B  # confidences for each grid cell


    probs = net_out[0: prob_size]

    confs = net_out[prob_size: (prob_size + conf_size)]

    cords = net_out[(prob_size + conf_size):]

    probs = probs.reshape([SS, C])

    confs = confs.reshape([SS, B])
```

```
cords = cords.reshape([SS, B, 4])


for grid in range(SS):
    for b in range(B):
        bx = Box()
        bx.c = confs[grid, b]
        bx.x = (cords[grid, b, 0] + grid % S) / S
        bx.y = (cords[grid, b, 1] + grid // S) / S
        if np.isnan(cords[grid, b, 2] ** sqrt):
            bx.w = np.exp(cords[grid, b, 2] + sqrt)
        else:
            bx.w = cords[grid, b, 2] ** sqrt
        if np.isnan(cords[grid, b, 3] ** sqrt):
            bx.h = np.exp(cords[grid, b, 3] + sqrt)
        else:
            bx.h = cords[grid, b, 3] ** sqrt
        p = probs[grid, :] * bx.c

        if p[class_num] >= threshold:
            bx.prob = p[class_num]
            boxes.append(bx)

# combine boxes that are overlap
boxes.sort(key=lambda b: b.prob, reverse=True)
for i in range(len(boxes)):
    boxi = boxes[i]
```

```
        if boxi.prob == 0: continue
        for j in range(i + 1, len(boxes)):
            boxj = boxes[j]
            if box_iou(boxi, boxj) >= .4:
                boxes[j].prob = 0.
    boxes = [b for b in boxes if b.prob > 0.]

    return boxes


def draw_box(boxes, im, crop_dim):
    imgcv = im
    [xmin, xmax] = crop_dim[0]
    [ymin, ymax] = crop_dim[1]
    for b in boxes:
        h, w, _ = imgcv.shape
        left = int((b.x - b.w / 2.) * w)
        right = int((b.x + b.w / 2.) * w)
        top = int((b.y - b.h / 2.) * h)
        bot = int((b.y + b.h / 2.) * h)
        left = int(left * (xmax - xmin) / w + xmin)
        right = int(right * (xmax - xmin) / w + xmin)
        top = int(top * (ymax - ymin) / h + ymin)
        bot = int(bot * (ymax - ymin) / h + ymin)

        if left < 0:  left = 0
    if right > w - 1: right = w - 1

    if top < 0:   top = 0

    if bot > h - 1:   bot = h - 1

    thick = int((h + w) // 150)

    cv2.rectangle(imgcv, (left, top), (right, bot), (255, 0, 0), thick)


return imgcv
```

/utils

__init__.py

```
from .utils import *
```

HOG-SVM

vehicle_detection.py

```
from skimage.feature import hog

from sklearn.svm import LinearSVC

from sklearn.preprocessing import StandardScaler

# for scikit-learn >= 0.18 use:

from sklearn.model_selection import train_test_split

# from sklearn.cross_validation import train_test_split

from scipy.ndimage.measurements import label

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

from moviepy.editor import VideoFileClip

from IPython.display import HTML

import numpy as np

import pickle

import cv2

import glob

import time


get_ipython().magic(u'matplotlib inline')


print('Shivam')




car_images = glob.glob('vehicles/**/*.png')

noncar_images = glob.glob('non-vehicles/**/*.png')
```

```
print(len(car_images), len(noncar_images))



fig, axs = plt.subplots(8,8, figsize=(16, 16))

fig.subplots_adjust(hspace = .2, wspace=.001)

axs = axs.ravel()



for i in np.arange(32):

    img = cv2.imread(car_images[np.random.randint(0,len(car_images))])

    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

    axs[i].axis('off')

    axs[i].set_title('car', fontsize=10)

    axs[i].imshow(img)

for i in np.arange(32,64):

    img = cv2.imread(noncar_images[np.random.randint(0,len(noncar_images))])

    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

    axs[i].axis('off')

    axs[i].set_title('nope', fontsize=10)

    axs[i].imshow(img)



def get_hog_features(img, orient, pix_per_cell, cell_per_block,
                     vis=False, feature_vec=True):
    # Call with two outputs if vis==True
    if vis == True:
        features, hog_image = hog(img, orientations=orient,
                                  pixels_per_cell=(pix_per_cell, pix_per_cell),
                                  cells_per_block=(cell_per_block, cell_per_block),
                                  transform_sqrt=False,
                                  visualise=vis, feature_vector=feature_vec)
        return features, hog_image
    # Otherwise call with one output
    else:
        features = hog(img, orientations=orient,
                       pixels_per_cell=(pix_per_cell, pix_per_cell),
                       cells_per_block=(cell_per_block, cell_per_block),
                       transform_sqrt=False,
                       visualise=vis, feature_vector=feature_vec)
        return features

print('Shivam')



car_img = mpimg.imread(car_images[5])
_, car_dst = get_hog_features(car_img[:,:,2], 9, 8, 8, vis=True, feature_vec=True)
noncar_img = mpimg.imread(noncar_images[5])
```

```
_, noncar_dst = get_hog_features(noncar_img[:,:,2], 9, 8, 8, vis=True, feature_vec=True)


# Visualize
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(7,7))
f.subplots_adjust(hspace = .4, wspace=.2)
ax1.imshow(car_img)
ax1.set_title('Car Image', fontsize=16)
ax2.imshow(car_dst, cmap='gray')
ax2.set_title('Car HOG', fontsize=16)
ax3.imshow(noncar_img)
ax3.set_title('Non-Car Image', fontsize=16)
ax4.imshow(noncar_dst, cmap='gray')
ax4.set_title('Non-Car HOG', fontsize=16)
print('...')



def extract_features(imgs, cspace='RGB', orient=9,
                        pix_per_cell=8, cell_per_block=2, hog_channel=0):
    # Create a list to append feature vectors to
    features = []
    # Iterate through the list of images
    for file in imgs:
        # Read in each one by one
        image = mpimg.imread(file)
        # apply color conversion if other than 'RGB'
        if cspace != 'RGB':
```

```
        if cspace == 'HSV':
            feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
        elif cspace == 'LUV':
            feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2LUV)
        elif cspace == 'HLS':
            feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)
        elif cspace == 'YUV':
            feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
        elif cspace == 'YCrCb':
            feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2YCrCb)
    else: feature_image = np.copy(image)

    # Call get_hog_features() with vis=False, feature_vec=True
    if hog_channel == 'ALL':
        hog_features = []
        for channel in range(feature_image.shape[2]):
            hog_features.append(get_hog_features(feature_image[:,:,channel],
                                orient, pix_per_cell, cell_per_block,
                                vis=False, feature_vec=True))
        hog_features = np.ravel(hog_features)
    else:
        hog_features = get_hog_features(feature_image[:,:,hog_channel], orient,
                    pix_per_cell, cell_per_block, vis=False, feature_vec=True)
    # Append the new feature vector to the features list
    features.append(hog_features)
# Return list of feature vectors
```

```
    return features


print('Shivam')




colorspace = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb

orient = 11

pix_per_cell = 16

cell_per_block = 2

hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"


t = time.time()

car_features = extract_features(car_images, cspace=colorspace, orient=orient,

                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,

                        hog_channel=hog_channel)

notcar_features = extract_features(noncar_images, cspace=colorspace, orient=orient,

                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,

                        hog_channel=hog_channel)

t2 = time.time()

print(round(t2-t, 2), 'Seconds to extract HOG features...')

# Create an array stack of feature vectors

X = np.vstack((car_features, notcar_features)).astype(np.float64)



y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))
```

```
# Split up data into randomized training and test sets

rand_state = np.random.randint(0, 100)

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=rand_state)


print('Using:',orient,'orientations',pix_per_cell,

    'pixels per cell and', cell_per_block,'cells per block')

print('Feature vector length:', len(X_train[0]))




svc = LinearSVC()

# Check the training time for the SVC

t = time.time()

svc.fit(X_train, y_train)

t2 = time.time()

print(round(t2-t, 2), 'Seconds to train SVC...')

# Check the score of the SVC

print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))

# Check the prediction time for a single sample

t=time.time()

n_predict = 10

print('My SVC predicts: ', svc.predict(X_test[0:n_predict]))

print('For these',n_predict, 'labels: ', y_test[0:n_predict])
```

```
t2 = time.time()
print(round(t2-t, 5), 'Seconds to predict', n_predict,'labels with SVC')



# Define a single function that can extract features using hog sub-sampling and make predictions
def find_cars(img, ystart, ystop, scale, cspace, hog_channel, svc, X_scaler, orient,
              pix_per_cell, cell_per_block, spatial_size, hist_bins, show_all_rectangles=False):

    # array of rectangles where cars were detected
    rectangles = []

    img = img.astype(np.float32)/255

    img_tosearch = img[ystart:ystop,:,:]

    # apply color conversion if other than 'RGB'
    if cspace != 'RGB':
        if cspace == 'HSV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2HSV)
        elif cspace == 'LUV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2LUV)
        elif cspace == 'HLS':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2HLS)
        elif cspace == 'YUV':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2YUV)
        elif cspace == 'YCrCb':
            ctrans_tosearch = cv2.cvtColor(img_tosearch, cv2.COLOR_RGB2YCrCb)
    else: ctrans_tosearch = np.copy(image)


    # rescale image if other than 1.0 scale
    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1]/scale),
np.int(imshape[0]/scale)))


    # select colorspace channel for HOG
    if hog_channel == 'ALL':
        ch1 = ctrans_tosearch[:,:,0]
        ch2 = ctrans_tosearch[:,:,1]
        ch3 = ctrans_tosearch[:,:,2]
    else:
        ch1 = ctrans_tosearch[:,:,hog_channel]


    # Define blocks and steps as above
    nxblocks = (ch1.shape[1] // pix_per_cell)+1  #-1
    nyblocks = (ch1.shape[0] // pix_per_cell)+1  #-1
    nfeat_per_block = orient*cell_per_block**2
    # 64 was the orginal sampling rate, with 8 cells and 8 pix per cell
    window = 64
    nblocks_per_window = (window // pix_per_cell)-1
    cells_per_step = 2  # Instead of overlap, define how many cells to step
    nxsteps = (nxblocks - nblocks_per_window) // cells_per_step
    nysteps = (nyblocks - nblocks_per_window) // cells_per_step


    # Compute individual channel HOG features for the entire image

    hog1 = get_hog_features(ch1, orient, pix_per_cell, cell_per_block, feature_vec=False)

    if hog_channel == 'ALL':

        hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block, feature_vec=False)

        hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block, feature_vec=False)


    for xb in range(nxsteps):

        for yb in range(nysteps):

            ypos = yb*cells_per_step

            xpos = xb*cells_per_step

            # Extract HOG for this patch

            hog_feat1 = hog1[ypos:ypos+nblocks_per_window, xpos:xpos+nblocks_per_window].ravel()

            if hog_channel == 'ALL':

                hog_feat2 = hog2[ypos:ypos+nblocks_per_window,
xpos:xpos+nblocks_per_window].ravel()

                hog_feat3 = hog3[ypos:ypos+nblocks_per_window,
xpos:xpos+nblocks_per_window].ravel()

                hog_features = np.hstack((hog_feat1, hog_feat2, hog_feat3))

            else:

                hog_features = hog_feat1


            xleft = xpos*pix_per_cell

            ytop = ypos*pix_per_cell
            ###########################################################################

            test_prediction = svc.predict(hog_features)

            if test_prediction == 1 or show_all_rectangles:
                xbox_left = np.int(xleft*scale)
                ytop_draw = np.int(ytop*scale)
                win_draw = np.int(window*scale)
                rectangles.append(((xbox_left,
ytop_draw+ystart),(xbox_left+win_draw,ytop_draw+win_draw+ystart)))

    return rectangles

print('Shivam')

test_img = mpimg.imread('./test_images/test8.jpg')

ystart = 400
ystop = 656
scale = 1.5
colorspace = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 11
```

```
pix_per_cell = 16

cell_per_block = 2

hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"


rectangles = find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None, orient,
pix_per_cell, cell_per_block, None, None)


print(len(rectangles), 'rectangles found in image')




def draw_boxes(img, bboxes, color=(0, 0, 255), thick=6):

    # Make a copy of the image

    imcopy = np.copy(img)

    random_color = False

    # Iterate through the bounding boxes

    for bbox in bboxes:

        if color == 'random' or random_color:

            color = (np.random.randint(0,255), np.random.randint(0,255), np.random.randint(0,255))

            random_color = True

        # Draw a rectangle given bbox coordinates

        cv2.rectangle(imcopy, bbox[0], bbox[1], color, thick)

    # Return the image copy with boxes drawn

    return imcopy


print('...')
```

```
test_img_rects = draw_boxes(test_img, rectangles)

plt.figure(figsize=(10,10))

plt.imshow(test_img_rects)

print('...')





test_img = mpimg.imread('./test_images/test8.jpg')



rects = []



ystart = 400

ystop = 464

scale = 1.0

rects.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None, show_all_rectangles=True))

ystart = 416

ystop = 480

scale = 1.0

rects.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None, show_all_rectangles=True))
```

```
rectangles = [item for sublist in rects for item in sublist]

test_img_rects = draw_boxes(test_img, rectangles, color='random', thick=2)

plt.figure(figsize=(10,10))

plt.imshow(test_img_rects)

print('Number of boxes: ', len(rectangles))



# In[39]:


test_img = mpimg.imread('./test_images/test8.jpg')


rects = []


ystart = 400

ystop = 496

scale = 1.5

rects.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                       orient, pix_per_cell, cell_per_block, None, None, show_all_rectangles=True))

ystart = 432

ystop = 528

scale = 1.5

rects.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                       orient, pix_per_cell, cell_per_block, None, None, show_all_rectangles=True))


rectangles = [item for sublist in rects for item in sublist]

test_img_rects = draw_boxes(test_img, rectangles, color='random', thick=2)
```

```
plt.figure(figsize=(10,10))

plt.imshow(test_img_rects)

print('Number of boxes: ', len(rectangles))



# In[40]:


test_img = mpimg.imread('./test_images/test8.jpg')


rects = []


ystart = 400

ystop = 528

scale = 2.0

rects.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                       orient, pix_per_cell, cell_per_block, None, None, show_all_rectangles=True))

ystart = 432

ystop = 560

scale = 2.0

rects.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                       orient, pix_per_cell, cell_per_block, None, None, show_all_rectangles=True))


rectangles = [item for sublist in rects for item in sublist]

test_img_rects = draw_boxes(test_img, rectangles, color='random', thick=2)

plt.figure(figsize=(10,10))

plt.imshow(test_img_rects)
```

```
print('Number of boxes: ', len(rectangles))



# In[41]:


test_img = mpimg.imread('./test_images/test8.jpg')


rects = []


ystart = 400
ystop = 596
scale = 3.0
rects.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                       orient, pix_per_cell, cell_per_block, None, None, show_all_rectangles=True))
ystart = 464
ystop = 660
scale = 3.0
rects.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                       orient, pix_per_cell, cell_per_block, None, None, show_all_rectangles=True))


rectangles = [item for sublist in rects for item in sublist]
test_img_rects = draw_boxes(test_img, rectangles, color='random', thick=2)
plt.figure(figsize=(10,10))
plt.imshow(test_img_rects)
print('Number of boxes: ', len(rectangles))
```

```
# ### Combine Various Sliding Window Searches

test_img = mpimg.imread('./test_images/test8.jpg')

rectangles = []

colorspace = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 11
pix_per_cell = 16
cell_per_block = 2
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"

ystart = 400
ystop = 464
scale = 1.0
rectangles.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                       orient, pix_per_cell, cell_per_block, None, None))
ystart = 416
ystop = 480
scale = 1.0
rectangles.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                       orient, pix_per_cell, cell_per_block, None, None))
```

```
ystart = 400
ystop = 496
scale = 1.5
rectangles.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                    orient, pix_per_cell, cell_per_block, None, None))
ystart = 432
ystop = 528
scale = 1.5
rectangles.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                    orient, pix_per_cell, cell_per_block, None, None))
ystart = 400
ystop = 528
scale = 2.0
rectangles.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                    orient, pix_per_cell, cell_per_block, None, None))
ystart = 432
ystop = 560
scale = 2.0
rectangles.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                    orient, pix_per_cell, cell_per_block, None, None))
ystart = 400
ystop = 596
scale = 3.5
rectangles.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                    orient, pix_per_cell, cell_per_block, None, None))
ystart = 464
```

```
ystop = 660

scale = 3.5

rectangles.append(find_cars(test_img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                       orient, pix_per_cell, cell_per_block, None, None))


# apparently this is the best way to flatten a list of lists

rectangles = [item for sublist in rectangles for item in sublist]

test_img_rects = draw_boxes(test_img, rectangles, color='random', thick=2)

plt.figure(figsize=(10,10))

plt.imshow(test_img_rects)

print('...')




def add_heat(heatmap, bbox_list):

    # Iterate through list of bboxes

    for box in bbox_list:

        # Add += 1 for all pixels inside each bbox

        # Assuming each "box" takes the form ((x1, y1), (x2, y2))

        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1


    # Return updated heatmap

    return heatmap


print('...')
```

```
# In[44]:


# Test out the heatmap

heatmap_img = np.zeros_like(test_img[:,:,0])

heatmap_img = add_heat(heatmap_img, rectangles)

plt.figure(figsize=(10,10))

plt.imshow(heatmap_img, cmap='hot')


print('...')




def apply_threshold(heatmap, threshold):

    # Zero out pixels below the threshold

    heatmap[heatmap <= threshold] = 0

    # Return thresholded map

    return heatmap


print('...')



# In[46]:


heatmap_img = apply_threshold(heatmap_img, 1)
```

```
plt.figure(figsize=(10,10))

plt.imshow(heatmap_img, cmap='hot')


print('...')




labels = label(heatmap_img)

plt.figure(figsize=(10,10))

plt.imshow(labels[0], cmap='gray')

print(labels[1], 'cars found')



# ### Draw Bounding Boxes for Lables


# In[48]:


def draw_labeled_bboxes(img, labels):

    # Iterate through all detected cars

    rects = []

    for car_number in range(1, labels[1]+1):

        # Find pixels with each car_number label value

        nonzero = (labels[0] == car_number).nonzero()

        # Identify x and y values of those pixels

        nonzeroy = np.array(nonzero[0])

        nonzerox = np.array(nonzero[1])
        # Define a bounding box based on min/max x and y

        bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzeroy)))

        rects.append(bbox)

        # Draw the box on the image

        cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 6)

    # Return the image and final rectangles

    return img, rects


# Draw bounding boxes on a copy of the image

draw_img, rect = draw_labeled_bboxes(np.copy(test_img), labels)

# Display the image

plt.figure(figsize=(10,10))

plt.imshow(draw_img)

print('...')



# ### Put it All Together


# In[49]:


def process_frame(img):


    rectangles = []


    colorspace = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb

    orient = 11
```

```
pix_per_cell = 16

cell_per_block = 2

hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"


ystart = 400

ystop = 464

scale = 1.0

rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None))

ystart = 416

ystop = 480

scale = 1.0

rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None))

ystart = 400

ystop = 496

scale = 1.5

rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None))

ystart = 432

ystop = 528

scale = 1.5

rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None))

ystart = 400

ystop = 528
```

```
scale = 2.0

rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None))

ystart = 432

ystop = 560

scale = 2.0

rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None))

ystart = 400

ystop = 596

scale = 3.5

rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None))

ystart = 464

ystop = 660

scale = 3.5

rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,

                    orient, pix_per_cell, cell_per_block, None, None))


rectangles = [item for sublist in rectangles for item in sublist]


heatmap_img = np.zeros_like(img[:,:,0])

heatmap_img = add_heat(heatmap_img, rectangles)

heatmap_img = apply_threshold(heatmap_img, 1)

labels = label(heatmap_img)

draw_img, rects = draw_labeled_bboxes(np.copy(img), labels)
```

```
    return draw_img


print('...')



# Run the pipeline on all the test images.


# In[50]:


test_images = glob.glob('./test_images/test*.jpg')


fig, axs = plt.subplots(3, 2, figsize=(16,14))
fig.subplots_adjust(hspace = .004, wspace=.002)
axs = axs.ravel()


for i, im in enumerate(test_images):
    axs[i].imshow(process_frame(mpimg.imread(im)))
    axs[i].axis('off')




test_out_file = 'test_video_out.mp4'
clip_test = VideoFileClip('test_video.mp4')
clip_test_out = clip_test.fl_image(process_frame)
get_ipython().magic(u'time clip_test_out.write_videofile(test_out_file, audio=False)')
```

```
# In[34]:


HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(test_out_file))




class Vehicle_Detect():
    def __init__(self):
        # history of rectangles previous n frames
        self.prev_rects = []

    def add_rects(self, rects):
        self.prev_rects.append(rects)
        if len(self.prev_rects) > 15:
            # throw out oldest rectangle set(s)
            self.prev_rects = self.prev_rects[len(self.prev_rects)-15:]


print('...')
```

```
def process_frame_for_video(img):

    rectangles = []

    colorspace = 'YUV' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
    orient = 11
    pix_per_cell = 16
    cell_per_block = 2
    hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"

    ystart = 400
    ystop = 464
    scale = 1.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                        orient, pix_per_cell, cell_per_block, None, None))
    ystart = 416
    ystop = 480
    scale = 1.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                        orient, pix_per_cell, cell_per_block, None, None))
    ystart = 400
    ystop = 496
    scale = 1.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                        orient, pix_per_cell, cell_per_block, None, None))
    ystart = 432
    ystop = 528
    scale = 1.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                        orient, pix_per_cell, cell_per_block, None, None))
    ystart = 400
    ystop = 528
    scale = 2.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                        orient, pix_per_cell, cell_per_block, None, None))
    ystart = 432
    ystop = 560
    scale = 2.0
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                        orient, pix_per_cell, cell_per_block, None, None))
    ystart = 400
    ystop = 596
    scale = 3.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                        orient, pix_per_cell, cell_per_block, None, None))
    ystart = 464
    ystop = 660
    scale = 3.5
    rectangles.append(find_cars(img, ystart, ystop, scale, colorspace, hog_channel, svc, None,
                        orient, pix_per_cell, cell_per_block, None, None))
```

```
    rectangles = [item for sublist in rectangles for item in sublist]


    # add detections to the history
    if len(rectangles) > 0:
        det.add_rects(rectangles)


    heatmap_img = np.zeros_like(img[:,:,0])
    for rect_set in det.prev_rects:
        heatmap_img = add_heat(heatmap_img, rect_set)
    heatmap_img = apply_threshold(heatmap_img, 1 + len(det.prev_rects)//2)


    labels = label(heatmap_img)
    draw_img, rect = draw_labeled_bboxes(np.copy(img), labels)
    return draw_img


print('...')


det = Vehicle_Detect()


test_out_file2 = 'test_video_out_2.mp4'
clip_test2 = VideoFileClip('test_video.mp4')
clip_test_out2 = clip_test2.fl_image(process_frame_for_video)
get_ipython().magic(u'time clip_test_out2.write_videofile(test_out_file2, audio=False)')
```

```
# In[38]:


HTML("""
<video width="960" height="540" controls>
  <source src="{0}">
</video>
""".format(test_out_file2))



# In[55]:


det = Vehicle_Detect()


proj_out_file = 'project_video_out.mp4'
clip_proj = VideoFileClip('project_video.mp4') #.subclip(23,26)  # subclip = only specified span of
video


clip_proj_out = clip_proj.fl_image(process_frame_for_video)
get_ipython().magic(u'time clip_proj_out.write_videofile(proj_out_file, audio=False)')
```

APPENDIX - B

SCREENSHOTS

SCREENSHOTS

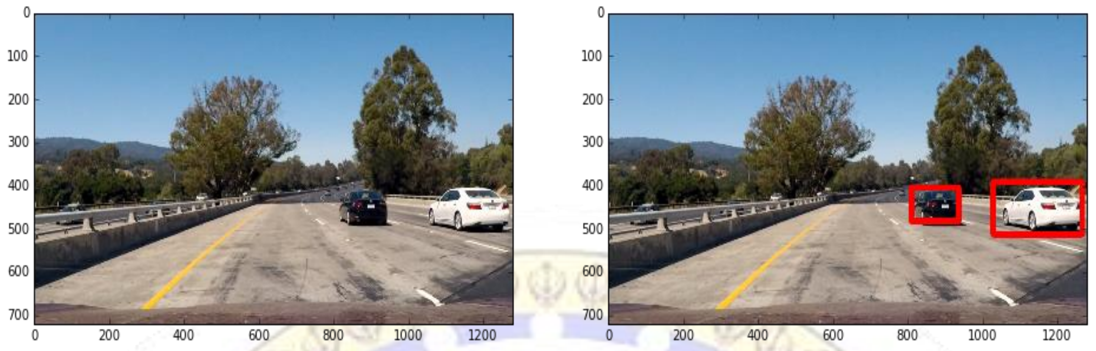


Fig.9 YOLO original image and detected vehicle

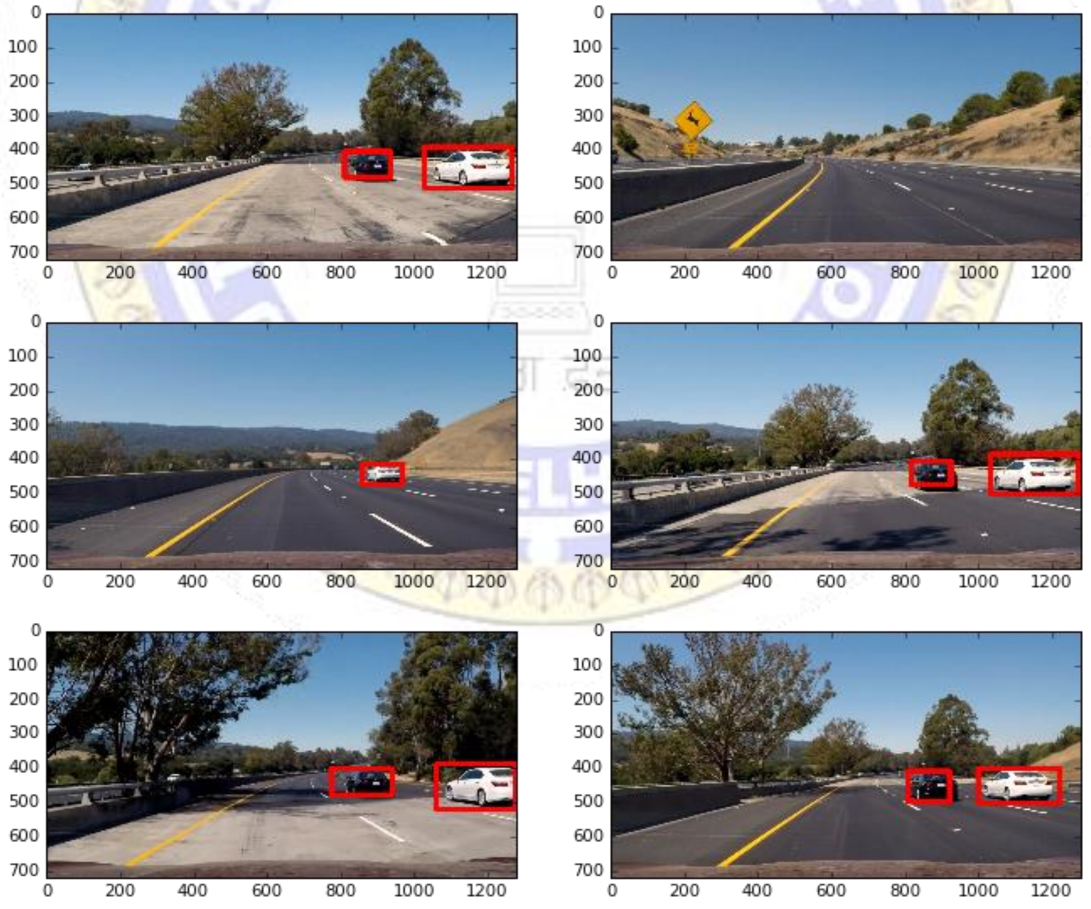


Fig.10 More examples of detected vehicles by YOLO

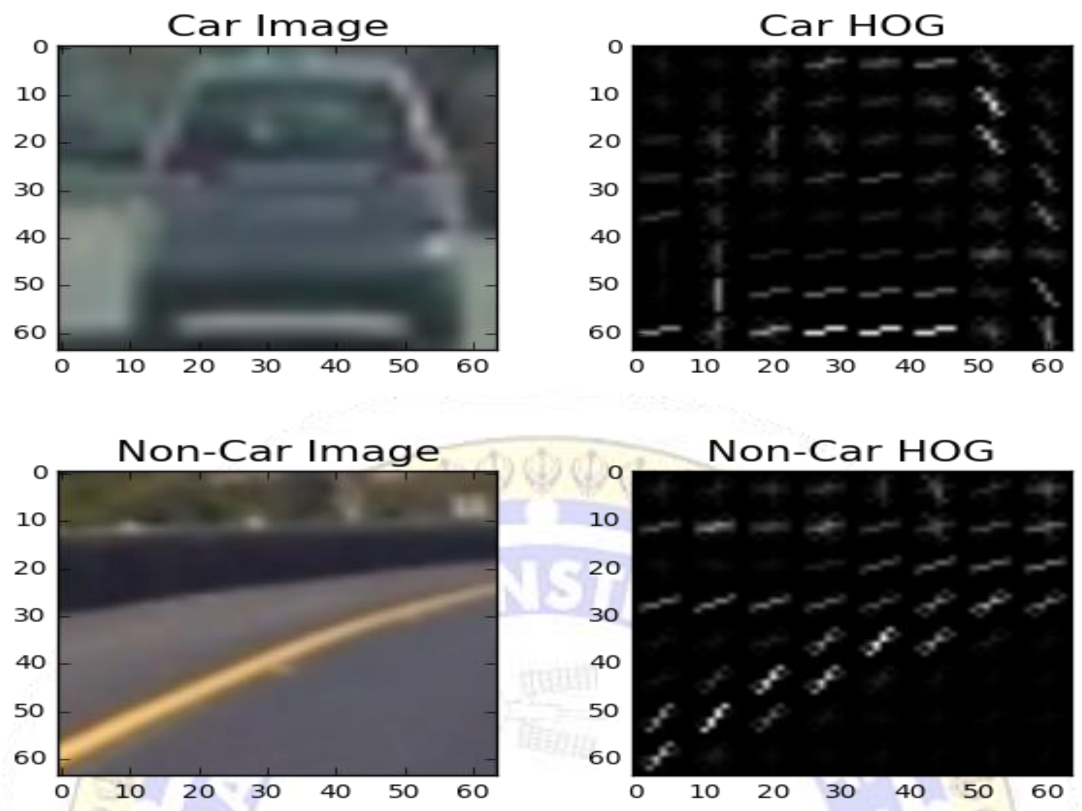


Fig.11 HOG view on dataset

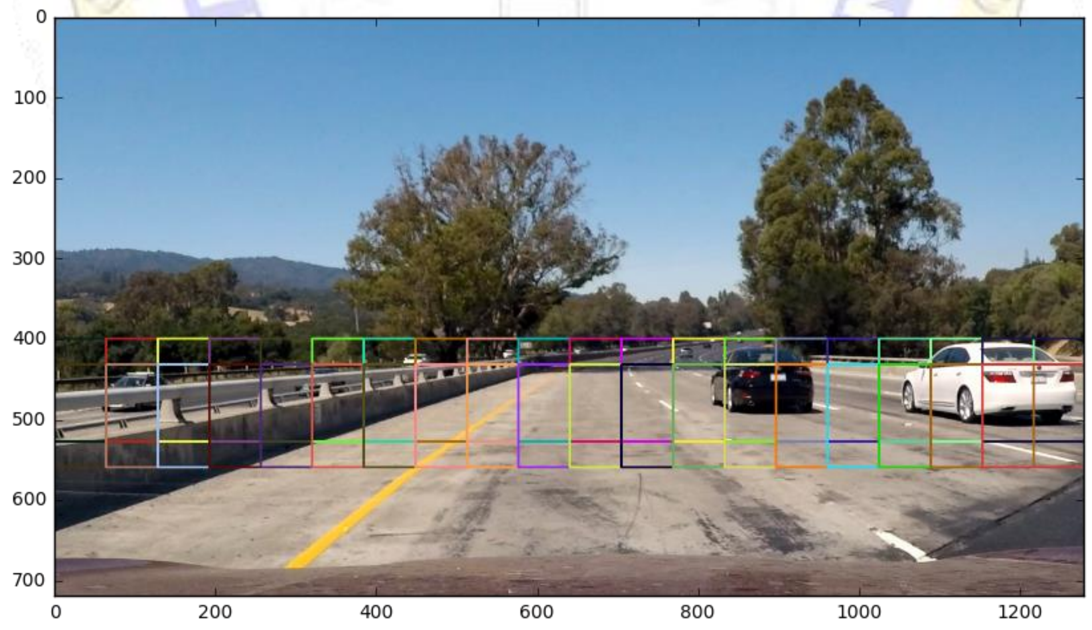


Fig. 12 Potential Search Area for Car

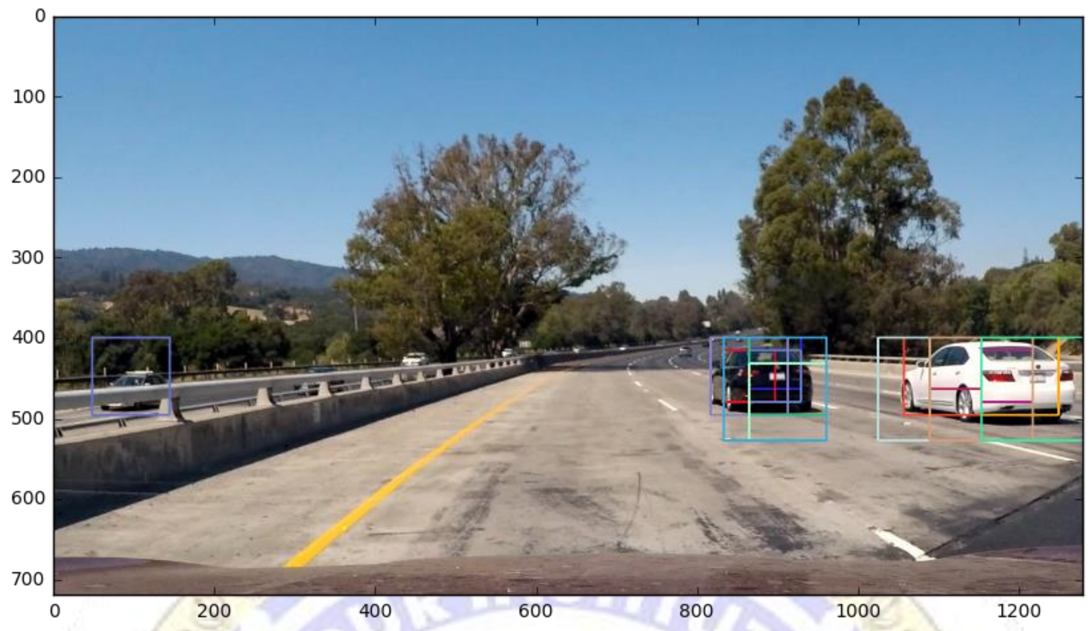


Fig. 13 Combine various sliding window approaches

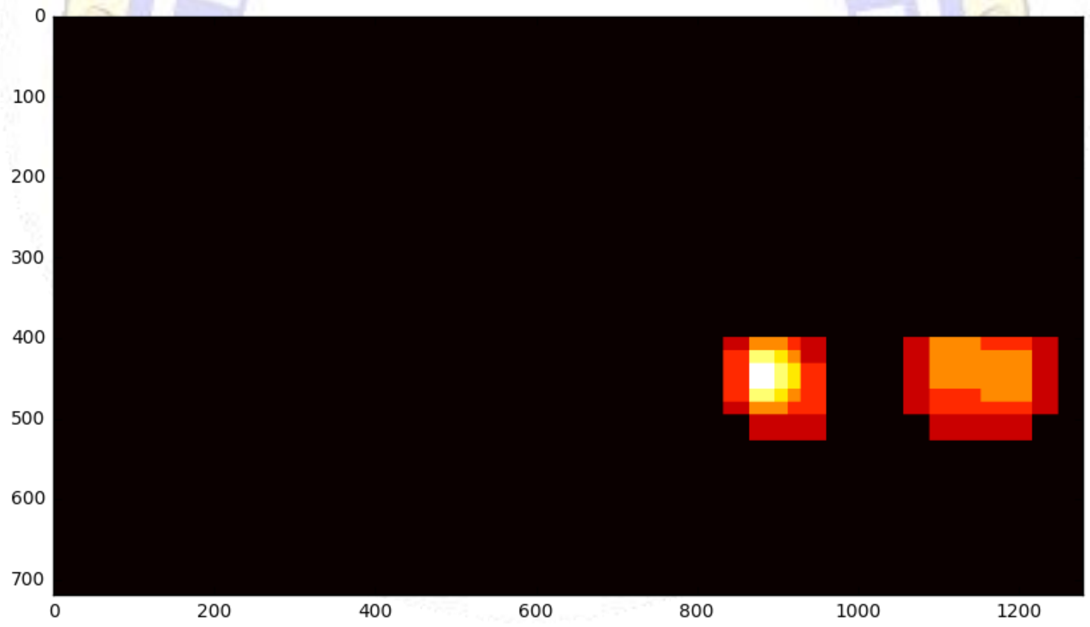


Fig. 14 Heatmap

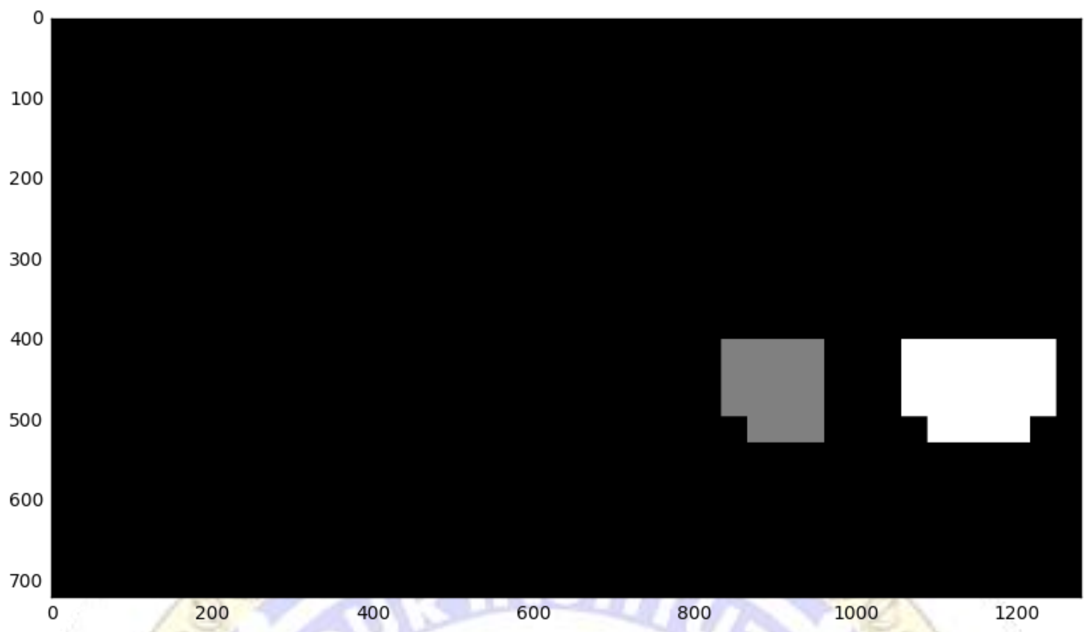


Fig 15 SciPy Labels to Heatmap

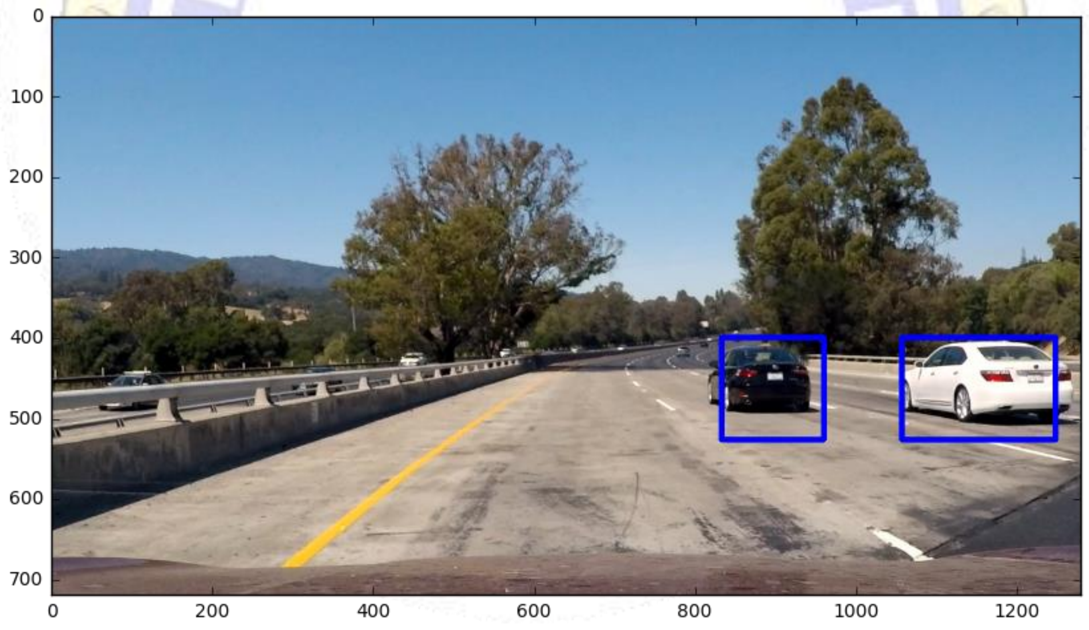


Fig. 16 Draw bounding box for Labels

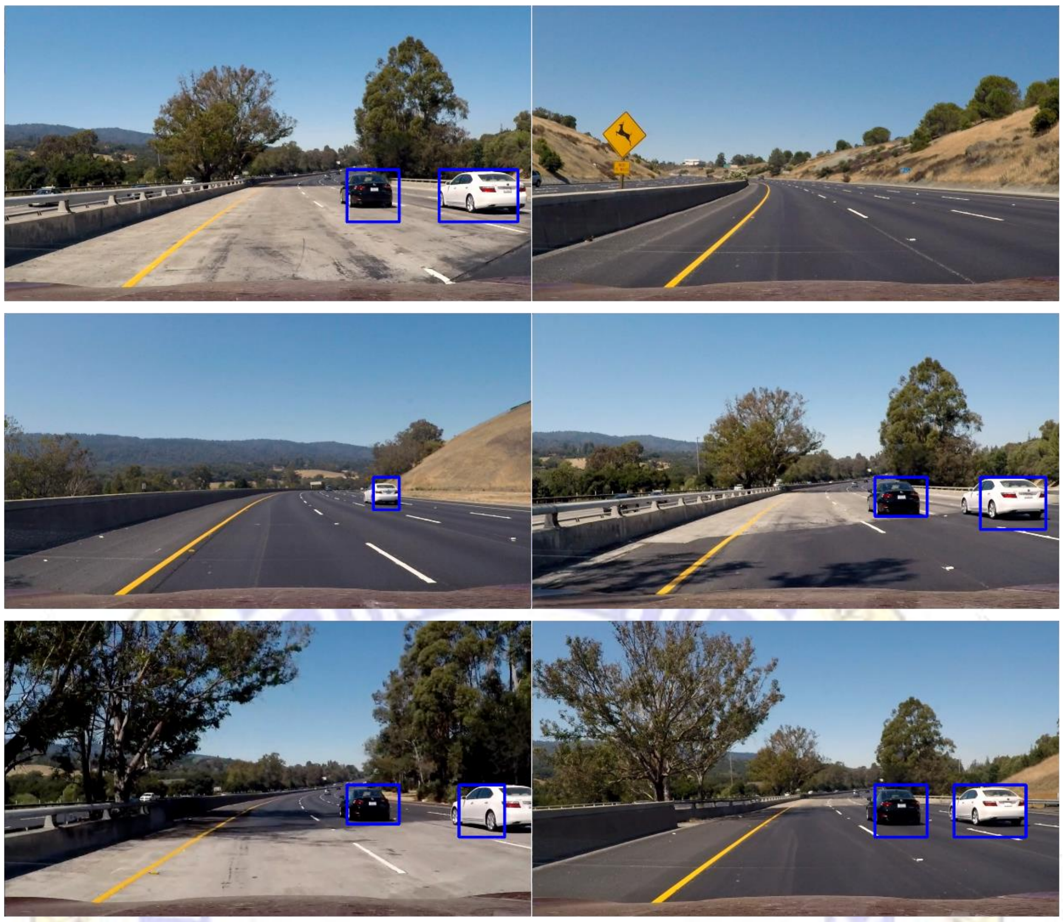


Fig. 17 More examples of detected vehicles by HOG-SVM