A Project Report

3D OBJECT DETECTION FOR AUTONOMOUS DRIVING

Submitted in partial fulfillment of the requirements for the award of degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

SUSHANTH SAMALA(160115733115)

Department of Computer Science and Engineering,
Chaitanya Bharathi Institute of Technology (Autonomous),
(Affiliated to Osmania University, Hyderabad)
Hyderabad, TELANGANA (INDIA) – 500 075
April/May-2019

CERTIFICATE

This is to certify that the project titled "**3D Object Detection For Autonomous Driving**" is the bonafide work carried out by **Sushanth Samala** (160115733115), student of B.E.(CSE) of Chaitanya Bharathi Institute of Technology, Hyderabad, affiliated to Osmania University, Hyderabad, Telangana(India) during the academic year 2018-19, submitted for the partial fulfillment of the requirements for the award of the degree in **Bachelor of Engineering** (**Computer Science and Engineering**) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Project Supervisor

Dr.S.China Ramu
Professor
Department of CSE
CBIT, Hyderabad

Head of the Dept

Dr. M. Swamy Das
Professor and Head
Department of CSE
CBIT, Hyderabad

Place: Hyderabad
Date:

External Examiner

DECLARATION

We hereby declare that the project entitled **3D Object Detection For Autonomous Driving**" submitted for the B. E (CSE) degree is our original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

SAMALA SUSHANTH

(160115733115)

Place: Hyderabad

Date:

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any work would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the project.

We would like to express our sincere gratitude and indebtedness to **Dr. T. Sridevi**, our project guide, for her skilful guidance, constant supervision, timely suggestion, keen interest and encouragement in completing the individual seminar within the stipulated time.

We wish to express our gratitude to **Dr S China Ramu** and **Ms K Spandana**, Project Coordinators, who have shown keen interest and even rendered their valuable time and guidance in terms of suggestions and encouragement.

We are honoured to express our profound sense of gratitude to **Dr. M Swamydas**, Head of Department, CSE, who has served as a host of valuable corrections and for providing us time and amenities to complete this project.

We gratefully express our thanks to **Dr. P Ravinder Reddy**, Principal of our college and the management of CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY for providing excellent academic and learning environment in the college.

We wish to express our heartfelt gratitude to the Members of Staff and all others who helped us in bringing up our project. We would also like to thank the Lab assistants and Programmers for helping us through our project.

Sushanth Samala

160115733115

ABSTRACT

Computer vision in cars doesn't have insights the same way humans do per se. A computer needs to assign specific features to objects to recognize them and understand what's happening or what will happen in the next moment. This process of recognition includes semantic segmentation, creation of a detailed 3D map, and object detection in it. After an AI system has recognized objects on the detailed map, deep learning teaches car how to behave in a particular situation based on what it has detected. Further action is a path planning based on a virtual picture with recognized objects and assigned reactions on them.

This project goals at high-accuracy 3D object detection in self sufficient using scenario. We proposed Histogram Oriented Gradients(HOG) coupled with a sliding window approach which takes a video as input and predicts oriented bounding boxes. We have experimented on the challenging KITTI benchmark and outperform the ultra-modern overall performance on the duties of 3D localization, detection.

Experiments on the challenging KITTI benchmark show that our approach gets an accuracy of 99.2% on the tasks of object detection. In addition, the time taken to train our SVM model is 3.2 seconds on a system with 8 GB ram and an Intel Iris Pro graphics chip. This time can be further reduced by using a more powerful machine.

LIST OF FIGURES

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

Autonomous driving has been a promising industry in recent years. Both car manufacturers and IT companies are competitively investing to self-driving field. Companies like Google, Uber, Ford, BMW have already been testing their self-driving vehicles on the road. Optical vision is an essential component of autonomous car. Accurate real-time object detection, such as vehicles, pedestrians, animals, and road signs, could accelerate the pace of building a self-driving car as safe as human drivers. Image understanding has been a challenging task for decades. Unlike geometric figures, objects in the real world are always irregular figures. And the same object appears in various shapes when capturing from different angles or the object itself is changing its shape. Besides, images of objects in the real world environment are variant to illumination, rotation, scale and occlusion, which makes object detection task more challenging. In recent years, a large improvement in image recognition was made by a series of CNN based solutions.

Autonomous Vehicles (AVs) are widely anticipated to alleviate road congestion through higher throughput, improve road safety by eliminating human error, and free drivers from the burden of driving, allowing greater productivity and/or time for rest, along with a myriad of other foreseen benefits. The past three decades have seen steadily increasing research efforts in developing self-driving vehicle technology, in part fuelled by advances in sensing and computing technologies which have resulted in reduced size and price of necessary hardware. Furthermore, the perceived societal benefits continue to grow in scale along with the rapid global increase of vehicle ownership. As of 2010, the number of vehicles in use in the world was estimated to be 1.015 billion [1], while the world population was estimated to be 6.916 billion [2]. This translates to one vehicle for every seven persons. The societal cost of traffic crashes in the United States was approximately 300 billion USD in 2009 [3]. The financial cost of congestion is continually increasing each year, with the cost estimate for United States reaching as high as 160 billion USD in 2014 [4]. The associated health cost of congestions in United States was estimated to be over 20 billion USD in

2010 from premature deaths resulting from pollution inhalation [5]. While it is uncertain just how much these ongoing costs can be reduced through autonomous vehicle deployment, attempting to curtail the massive scale of these numbers serves as great motivation for the research.

A future with self-driving cars was first envisioned[6], with the idea even broadcasted over television as early as 1958 [7]. By 1988, Carnegie Mellon's NAVLAB vehicle was being demonstrated to perform lane-following using camera images [8]. Development was accelerated when several research teams later developed more advanced driverless vehicles to traverse desert terrain in the 2004 and 2005 DARPA Grand Challenges [9], and then urban roads in the 2007 DARPA Urban Challenge (DUC) [10]. Research related to self-driving has since continued at a fast pace in academic settings, but furthermore is now receiving considerable attention in industry as well.

As research in the field of autonomous vehicles has matured, a wide variety of impressive demonstrations have been made on full-scale vehicle platforms. Recent studies have also been conducted to model and anticipate the social impact of implementing autonomous Mobility-on- Demand (MoD) [11]. The case studies have shown that MoD system would make access to mobility more affordable and convenient compared to traditional mobility system characterized by extensive private vehicle ownership.

Autonomous driving on urban roads has seen tremendous progress in recent years, with several commercial entities pushing the bounds alongside academia. Google has perhaps the most experience in the area, having tested its fleet of autonomous vehicles for more than 2 million miles, with expectation to soon launch a pilot MoD service project using 100 self-driving vehicles [12]. Tesla is early to market their work, having already provided an autopilot feature in their 2016 Model S cars [13]. Uber's mobility service has grown to upset the taxi markets in numerous cities worldwide, and has furthermore recently indicated plans to eventually replace all their human driven fleet with self-driving cars [14], with their first self-driving vehicle pilot program already underway [15].

There are several places where automated road shuttles are in commercial operations. Examples include deployments at Rivium Business Park, Masdar City, and Heathrow Airport [16,17]. The common feature of these operations is that road vehicles are certified as a rail system meaning that vehicles operate in a segregated space [17]. This approach has been necessary due to legal uncertainty around liability in the event of an accident involving an autonomous vehicle. To address this, governments around the world are reviewing and implementing new laws. Part of this process has involved extended public trials of automated shuttles, with CityMobil and CityMobil2 being among the largest of such projects [17].

1.1 Problem Statement

In this project we detect the precise location of objects on a road. This provides the autonomous driving systems with a virtual interpretation of the real-world.

1.2 Objective

In this project, we have developed the perception module of the autonomous driving system. The main function required to build to develop the perception module is the environmental perception function. Environment perception is a fundamental function to enable autonomous vehicles, which provides the vehicle with crucial information on the driving environment, including the free drivable areas and surrounding obstacles' locations, velocities, and even predictions of their future states. Based on the sensors implemented, the environment perception task can be tackled by using LIDARs, cameras, or a fusion between these two kinds of devices. Some other traditional approaches may also involve the use of short/long-range radars and ultrasonic sensors.

1.3 Motivation

The goals of our project are to

(i) Help improve the accuracy of the autonomous driving systems.

(ii) Reduce the computation power required for the perception module of self-driving cars.

1.4 Existing Systems and Problems

One of the major open challenges in self-driving cars is the ability to detect cars and pedestrians to safely navigate in the world. Deep learning-based object detector approaches have enabled great advances in using camera imagery to detect and classify objects. But for a safety critical application such as autonomous driving, the error rates of the current state-of- the-art are still too high to enable safe operation. Moreover, the characterization of object detector performance is primarily limited to testing on prerecorded datasets. Errors that occur on novel data go undetected without additional human labels.

Object detection in self-driving cars is one of the most challenging and important impediments to full autonomy. Self-driving cars need to be able to detect cars and pedestrians to safely navigate their environment. In recent years, state-of-the-art deep learning approaches such as Convolutional Neural Networks (CNNs) have enabled great advances in using camera imagery to detect and classify objects. In part these advances have been driven by benchmark datasets that have large amounts of labeled training data[18]. Our understanding of how well we solve the object detection task has largely been measured by assessing how well novel detectors perform on this pre-recorded human labelled data. Alternatives such as simulation have been proposed to address the lack of extensive labelled data [19]. However, such solutions do not directly address how to find errors in streams of novel data logged from fleets of deployed autonomous vehicles (AVs).

1.5 Proposed System

- Work on labelled datasets to define training and testing sets.
- Extract features from dataset images.
- Train the classifier.
- For each video frame: run a search using a sliding window technique and filter out false positives.

1.6 Organisation of Report

This project is mainly divided into 3 modules as follows:

- Literature Survey discusses about the literature survey of this project which includes an insight into the core part of our project along with the technologies used.
- The System Architecture part deals with the design of our proposed system. The Implementation part deals with the implementation of our system which discusses about the algorithms used in building our system.
- The Result section displays our results and discussions through a series of screenshots. The final part talks about the conclusions and the future scope of our project.

Chapter 2

LITERATURE SURVEY

Literature survey is the most important step in any research or study in a particular domain. Before proceeding onto the work, it is necessary to research and understand the efforts put in previously by various researchers/companies/professionals in solving a specific problem. Their documentation contains the success of their project along with various difficulties they faced.

In this section, we will first go over the evolution of image recognition in recent years. Later, we will review some dominant approaches for object detection. These methods can be divided into two classes namely, traditional (Non-CNN based) methods and CNN based methods. We will discuss the pros and cons in two aspects, detection accuracy, detecting speed.

Evolution Of Image Recognition

Before 2012, object detection methods follow the feature-extraction-plus-classifier paradigm. First, people need to deliberately define specific feature for a category of objects in order to accurately represent the object. After extracting enough features from training dataset, the object can be represented by a vector of features, which is used to train a classifier in training time and also can be used to perform detection task in testing time. If people want to build a multiple objects detector, more consideration is needed in choosing a general feature so that the common feature can fit different objects. The disadvantage is obvious since the feature definition is so complex and the model is hard to extend when new object is added to the detecting list. Besides, the detecting accuracy is unsatisfactory.

In Large Scale Visual Recognition Challenge 2012 (ILSVRC, 2012), Krizhevsky's CNN based model outperformed all the other models by a big margin[20]. Even the similar CNN structure was presented in 1990s, its potential was hidden by the lack of training examples and weak hardware. In ILSVRC (2012), a subset of ImageNet[21] was used for classification which contains 1.2 million images from 1,000 categories. Moreover, GPUs became more powerful. With sufficient training images and

powerful GPUs, Krizhevsky's experiment proved CNN's powerful ability in images classification. The main advantage that CNN has over traditional methods, is the ability to construct feature filters during the training process. Researchers are set free from tedious feature engineering work. Benefited from the self-learning ability, CNN models are much more friendly to extend to other categories if enough training data is provided. From then on, CNN became the major tool for image classification task and people made better results in image classification related problems. Since CNN has proved its huge advantage on image classification, people began to revise previous CNNs to realize image localization and classification. The simple recipe for localization-plus-classification model is to attach another group of fully- connected layers (regression head) to the current CNN model. The new regression head is trained separately to predict coordinates. Thus, the CNN has a classification head and a regression head. At test time, two heads are working together, one for predicting class score and one for position. From 2012 to 2015, Researches successively drafted regression head to the present remarkable CNNs, such as Overfeat-Net[22], VGG-Net[23] and ResNet[24], reducing the single object localization error from 34% to 9%. Obviously, CNN based methods have done very well in localization-plus-classification task.

Later in 2014, researchers began to move to multiple objects detection task. There are more than one object in a single image and the system needs to find out each object's class and location. Many CNN based region proposals-plus-classification approaches came out [25] [26] [27] [28]. The main idea is to utilize region proposal methods to generate candidate regions that possibly contain objects, and later perform classification on each of the regions. In practice, these region-proposals-plus-classification approaches could achieve very high precision [22] [23] [24]. However, the region proposal part is very time consuming, that slows down the speed of the whole system. The slow detection speed prevents these methods from applying to time critical applications, such as auto-driving, surveillance system etc.

Recently, a unified object detection model, YOLO[29], was proposed. It frames detection as a regression problem. The model directly regresses from input image to a

tensor which represents object class score and digits of each object's position[29]. The input images just need to go through the network once and as a result, the model processes images faster. YOLO has achieved an accuracy over 50 percent in real-time detection on VOC 2007 and 2012 dataset, which makes it the best choice for any real-time object detection tasks.

2.1 Convolutional Neural Network Based Object Detection

Unlike image classification, object detection requires multi-object localization. Because regression tool is used to predict fixed number of outputs, the classification network cannot be used to directly regress from the input image to object's location. In this case, the network does not know how many objects are present in the input images. Thus, researchers repurposed the current single object classifier to perform multiple objects detection and came up with region proposal-plus-classification solutions to change the new task to familiar single object classification. Many models were proposed following this two-stage solution.

Regions-Based Convolutional Neural Network (R-CNN)[25]

Girshick, Ross et al. (2014) proposed a multi-stage approach following classification using regions paradigm. The system consists of three components, region proposal component, feature vector extraction by CNN and Support Vector Machine (SVM) as classifier. During training time, the CNN is supervised pre-trained on a large dataset (ILSVRC) and fine-tuned on a small dataset (PASCAL). Therefore, the CNN is more efficient to extract feature vectors for both positive ground truth regions and negative regions from background and save to disk for next training step. Then, these feature vectors are used to train SVM classifier. At test time, an external region proposal component, Selective Search[30], is utilized to generate 2000 fixed-size category-independent regions that possibly contain objects. Then the well-trained CNN extractor will convert all the potential regions into feature vectors, base on which the SVM is used for domain-specific classification. At the end, a linear regression model is used to refine the bounding box and apply a greedy non-maximum suppression to eliminate duplicate detections as well base on the intersection-over-union (IOU) overlap with a higher scoring region.

R-CNN achieves excellent detection accuracy comparing to other method at that time. However, R-CNN's complex multi-stage pipeline bring with notable drawbacks as well. The CNN components plays as a classifier. The region prediction still relies on external region proposal method, which is very slow and slow down the whole system on both training and detecting periods. Besides, the separated training manner of each components results in the CNN part is hard to optimize. Namely, when training the SVM classifier, the CNN part cannot be updated.

2.2 Spatial Pyramid Pooling Network (SPP-Net)[26]

The SPP-Net was proposed for speeding up convolution computation and remove the constrain of fixed-size input. Before SPP-Net, CNNs required fixed-size input images. In general, a CNN always consists of two parts, Convolutional layers for outputting feature map and Fully-connected (FC) layers or a SVM for classification. Inside the Convolutional layers, there are always alternatively layout convolutional layer and pooling (Sliding Window Pooling) layer. In fact, the Convolutional layers can process any size of input images. On the other hand, the FC layers or SVM requires fixed-size input. Thus, the whole network can only process fixed-size input images. To fulfill this requirement, the common practice is to either crop or warp the region proposals before feeding them into the Convolutional layers. Both the solutions could affect the CNNs' performance. It is easy to understand that cropping part of the object may result in failing to recognize the object, and the consequence of warping is loss of the original ratio which is essential information for ratio sensitive objects.

The first progress that SPP-Net made is to remove the constrain of fixed-size input. In SPP-Net, they replaced the last sliding window pooling layer of Convolutional part with a spatial pyramid pooling (SPP) layer. Spatial pyramid pooling can be thought of the spatial version of Bag of Words (BoW)[31] which can extract feature at multi scales. The number of bins is fixed to the input size of FC layers or SVM rather than considering the input image size. Therefore, SPP-Net, and the whole network can accept images of arbitrary sizes.

Another advantage of this spatial pooling after convolution manner of SPP-Net is sharing the computation of all the region proposals. As mentioned earlier, previous

CNNs crop or warp the regions first and let each of them go through the Convolutional layers to extract features. The duplicate computing of overlap regions wastes a lot of time. In SPP-Net, let the whole image go through the Convolutional layers just once to generate a feature map, and use projecting function to project all the regions to the last convolutional layer. So, the feature extraction of each region is only performed on the feature map. The idea of SPP-Net could in general speed up all CNN-based image classification methods at that time.

Even SPP-Net has made such an improvement to the CNNs in speed and detection accuracy, just like R-CNN, it still has same drawbacks. Region proposals still rely on external methods. The multi stage structure require the Convolutional layers and classifier to be trained individually. Since SPP layers does not allow the loss error back-propagation, the Convolutional layers still cannot get updated.

Fast Region-Based Convolutional Network (Fast R-CNN)[27]

The Fast R-CNN was built to realize the training and testing end-to-end. It can be thought as the extension of R-CNN and SPP-Net. Just as SPP-Net, Fast R-CNN swaps the order of extracting region feature and running through the CNN for sharing computation. It also revises the last pooling later in order to process any size input images. But, the difference from SPP-Net is that Fast R-CNN only uses a region of interest (RoI) pooling layer to pool at one scale, in contrast to SPP-Net's multi-scale pooling. This trick is quite important for training, since loss error can back-propagate through RoI pooling layers to update the Convolutional layers. Also, Fast R-CNN use multi-task loss on each labelled RoI by combining the class score loss and bounding box loss. Thus, the whole network can be trained end-to-end.

In practice, the results prove that these innovations are efficient in accelerating training time and testing time while also improving detection accuracy. At the same time, fast running time in networks exposes slow region proposal as a bottleneck.

Faster-RCNN[28]

Faster R-CNN mainly solved the slow region proposal problem. Instead of using external region proposals methods, a Region Proposal Network (RPN) was introduced to perform the region proposal task, which shares the image convolutional

computation with the detection network. A RPN basically is a class-agnostic Fast R-CNN. The feature map is divided by n x n grid and at each region give out 9 region proposals of different ratios and scale. All the regions proposals will be fed in to RPN to predict existence score of objects and their positions. Then, let the high-score output regions of RPN be the input of the second Fast R-CNN to further perform class-specific classification and bounding box refinement. Now, all the region proposal work and classification work is done within the network and the training is end-to-end. As the result, The Faster R-CNN achieved state-of-the-art object detection accuracy on multiple dataset at that time, and it became the foundation of subsequence outstanding detection methods. Even the detection speed is about 10 fps, it is the fastest detection model comparing to other models with same accuracy.

2.3 Real-Time Object Detection

Rapid object detection using a boosted cascade of simple features [32]

This paper proposed an object detection approach which processes image near real-time. At that time, CNN had not caught common attention and this approach still rely on engineered feature to represent image and multi-level classifier to increase accuracy. Three main contributions were made by this paper. Firstly, in order to compute Harr-like feature at different scale rapidly, it introduced integral image, which was computed once and allows Harr-like feature computation in constant time. Secondly, using AdaBoost[9] to select critical features to construct a classifier, which ensure fast classification. Lastly, constructing a cascading classifier to speed up detection time by fast focus on promising regions. In the real-time face detection application, the system could run at 15 fps, which is comparable to the best contemporaneous systems.

Deformable Parts Model - 30hz (DPM)

In [33], a revised DPM[34] was implemented with a variety of speed up strategies. They used (Histogram of oriented gradients) (HoG) features at few scales to reduce the feature computation time. Then a hierarchical vector quantization method compresses the HoG features for further HoG templates evaluation and the HoG templates will be evaluated by hash-table methods to identify the most useful

locations which would be stored to several template priority lists. A cascade frame work was also utilized to improve detecting speed. As the fast speed and high accuracy cannot obtain at same time, the method achieved 30 fps with mAP of 26% on PASCAL 2007 object detection.

3D Fully Convolutional Network for Vehicle Detection in Point Cloud[35]

Bo Li basically applies the DenseBox fully convolutional network (FCN) architecture(Fig.2.1) to a three-dimensional point cloud.

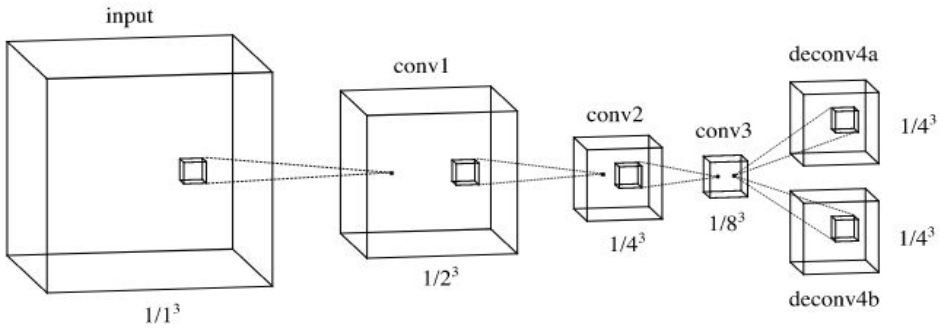

Figure 2.1: 3D Fully Convolutional Network For Vehicle Detection Architecture

Li has proposed the architecture in the following manner:

- Divides the point cloud into voxels. So instead of running 2D pixels through a network, we're running 3D voxels.
- Trains an FCN to identify features in the voxelized point cloud.
- Upsamples the FCN to produce two output tensors: an objectness tensor, and a bounding box tensor.
- The bounding box tensor is probably more interesting for perception purposes. It draws a bounding box around cars on the road.

VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection[36]

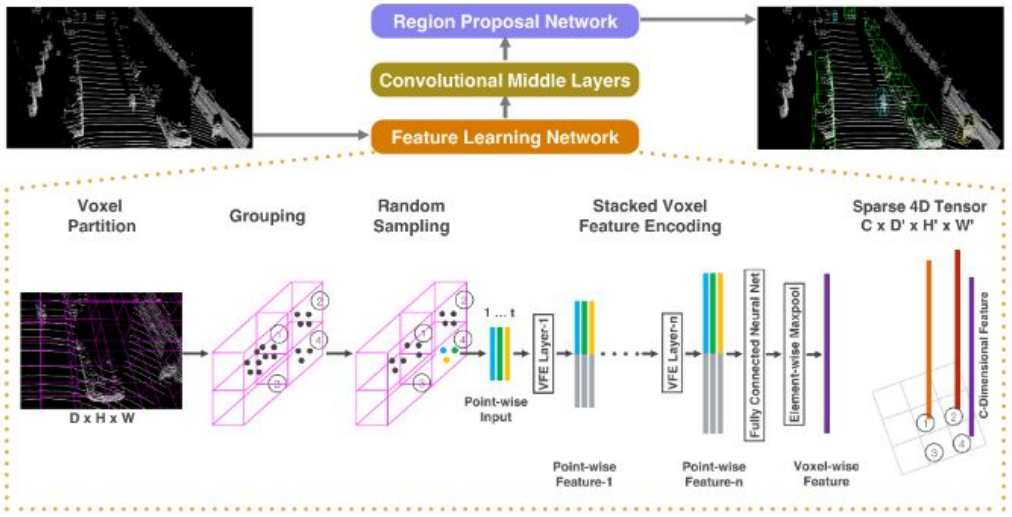

Figure 2.2: Point Cloud Based 3D Object Detection Architecture

That brings us to VoxelNet(Figure 2.2), from Apple, which got so much press recently.

VoxelNet has three components, in order:

- Feature Learning Network
- Convolutional Middle Layers
- Region Proposal Network

The Feature Learning Network seems to be the main "contribution to knowledge", as the scholars say. It seems that what this network does is start with a semi-random sample of points from within "interesting" (my word, not theirs) voxels. This sample of points gets run through a fully-connected (not fully-convolutional) network. This network learns point-wise features which are relevant to the voxel from which the points came.

The network, in fact, uses these point-wise features to develop voxel-wise features that describe each of the "interesting" voxels. We're oversimplifying wildly, but think of this as learning features that describe each voxel and are relevant to classifying the part of the vehicle that is in that voxel. So a voxel might have features like "black", "rubber", and "treads", and so you could guess that the voxel captures part of a tire.

Of course, the real features won't necessarily be intelligible by humans, but that's the idea.

2.4 Technologies

The following sections deal with the technologies we used in our project. Each of them has features that are discussed briefly.

Python

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly braces or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.

Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Python's features include:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintain.

- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street-view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

Scikit-Learn

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy.The

original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from INRIA took leadership of the project and made the first public release on February the 1st 2010. Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012.

NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply

represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

The core functionality of NumPy is its "ndarray", for *n*-dimensional array, data structure. These arrays are strided views on memory. In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type.

Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays.

Inserting or appending entries to an array is not as trivially possible as it is with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy's np.concatenate([a1,a2]) operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...)is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation[17].

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy include scipy.weave, numexpr and Numba. Cython and Pythran are static-compiling alternatives to these.

Example:

Array creation

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> x
array([1, 2, 3])
>>> y = np.arange(10)  # like Python's range, but returns an array
>>> y
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Example:

Line plot

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> a = np.linspace(0, 10, 100)
>>> b = np.exp(-a)
>>> plt.plot(a, b)
>>> plt.show()
```

Linear Support Vector Classification

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.

- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered numpy.ndarray (dense) orscipy.sparse.csr_matrix (sparse) with dtype=float64.

Similar to SVC with parameter kernel=’linear’, but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.

Chapter 3

SYSTEM ARCHITECTURE

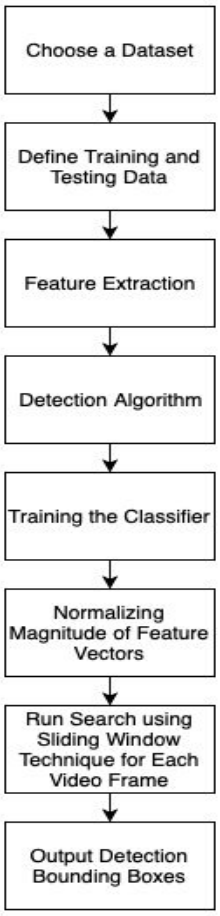

Figure 3.1: Architecture of the Proposed System

As shown in above figure 3.1, in our project, we first choose a dataset to train our model on. We then extract features from the chosen dataset by segregating it into training and testing data. The extracted features are then passed to the detection algorithm to detect the objects. After training the classifier and normalizing the magnitude of feature vectors we run a search on each individual video frame using sliding window approach then we output the detection values using bounding boxes.

3.1 Data Flow Diagrams

Data flow Diagrams give an overview of the entire system. It is a way of representing a flow of data of a process or a system. Data flow diagrams are represented using processes and entities.

DFD Level 0

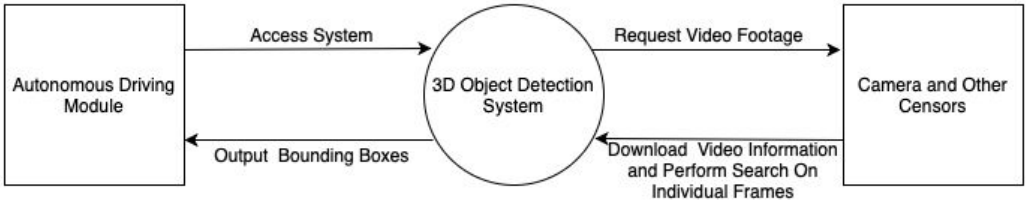

Figure 3.2: DFD Level 0 for Proposed System

Figure 3.2 is Level 0 DFD, which is a context level DFD consisting of the overall process. In the 3D Object Detection System, Driving Module and Camera and other sensors are external entities that interact with the system. 3D Object Detection System is a process. The autonomous driving module accesses the 3D Object Detection System and the System provides it with the object detections on the road, by training the classifier using KITTI dataset and detecting objects in the video data provided by the camera and other sensors.

DFD Level 1

Figure 3.3 represents Level 1 DFD, which is a detailed representation of the level 0. In the 3D Object Detection System, Driving Module and Camera and other sensors are entities that interact with the system. 3D Object Detection System is a process. The autonomous driving module accesses the 3D Object Detection System and the System provides it with the object detections on the road, by training the classifier using KITTI dataset and detecting objects in the video data provided by the camera and other sensors.

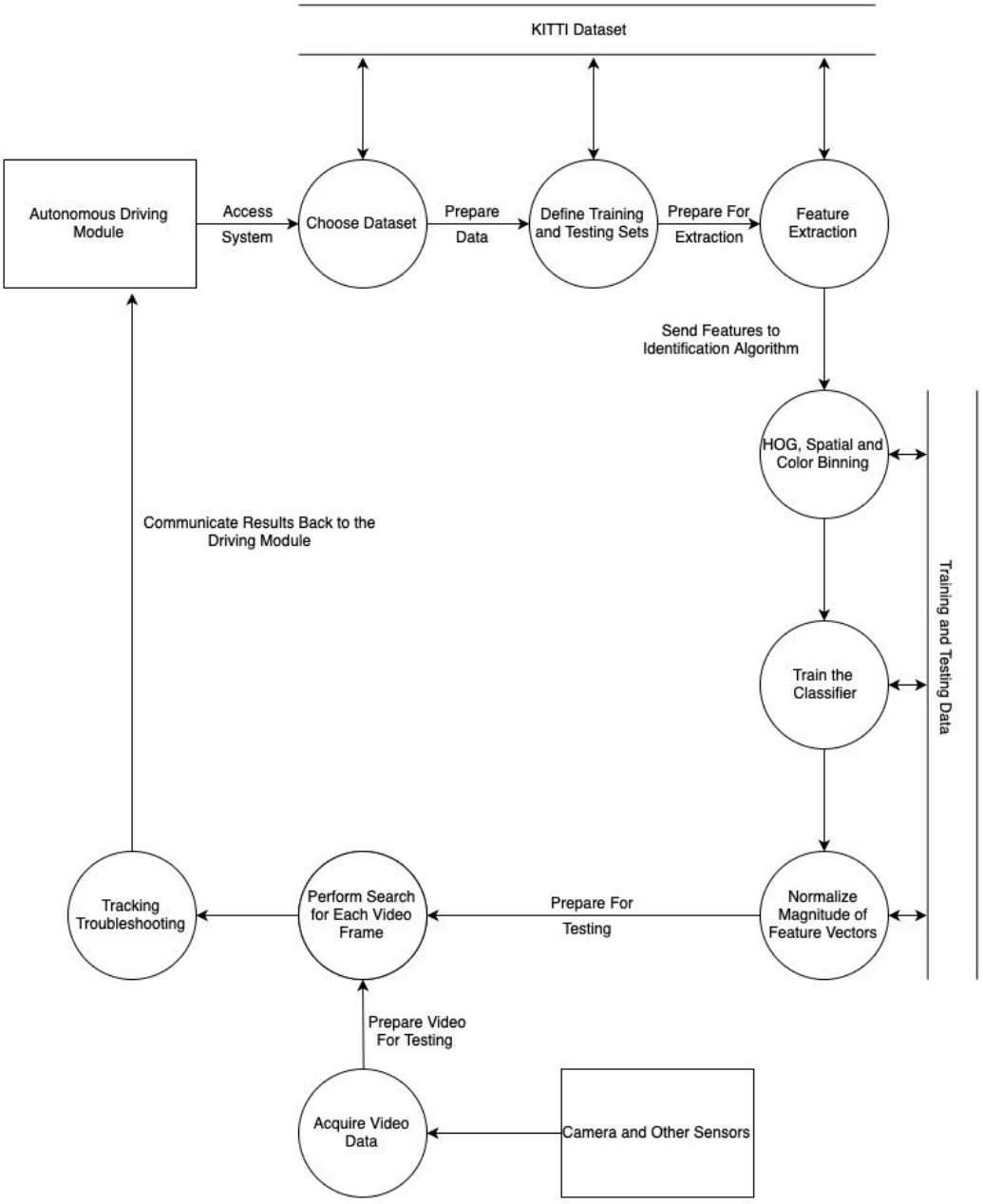

Figure 3.3: DFD level 1 for Proposed System

3.2 UML Diagrams

Unified Modelling Language (UML) is a standard language for creating blueprints that depicts the structure and design of the software system. It is the international standard notation for object-oriented analysis and design. The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential

details of the underlying problem from its usually complicated real world. It is a language for visualizing, specifying, constructing, documenting.

Class Diagram

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

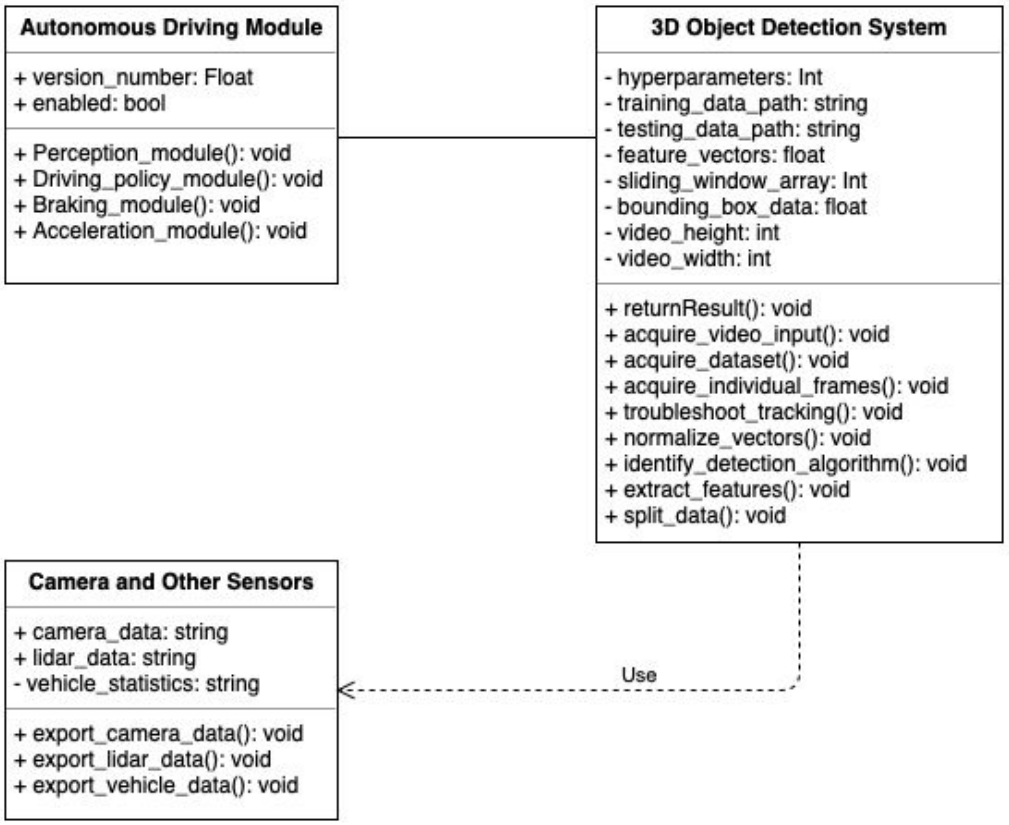

Figure 3.4: Class Diagram of the Proposed System

Figure 3.4 depicts the class diagram of the proposed the system, it has following classes Autonomous Driving Module, 3D Object Detection System and Camera and Other Sensors, along with their attributes and relationships. The Autonomous Driving Module class represents the end user. The Autonomous Driving Module can access the system using the perception_module() operation. The Camera and Other Sensors class represents the parts from which video input is taken which has the features to

extract video and other information from the surroundings. The 3D Object Detection System class has the operations to detect objects, acquire dataset, troubleshoot tracking, split the data and many others.

Use-Case Diagram

A use case diagram depicts the various operations that a system performs. It contains use cases, actors and their relationships.

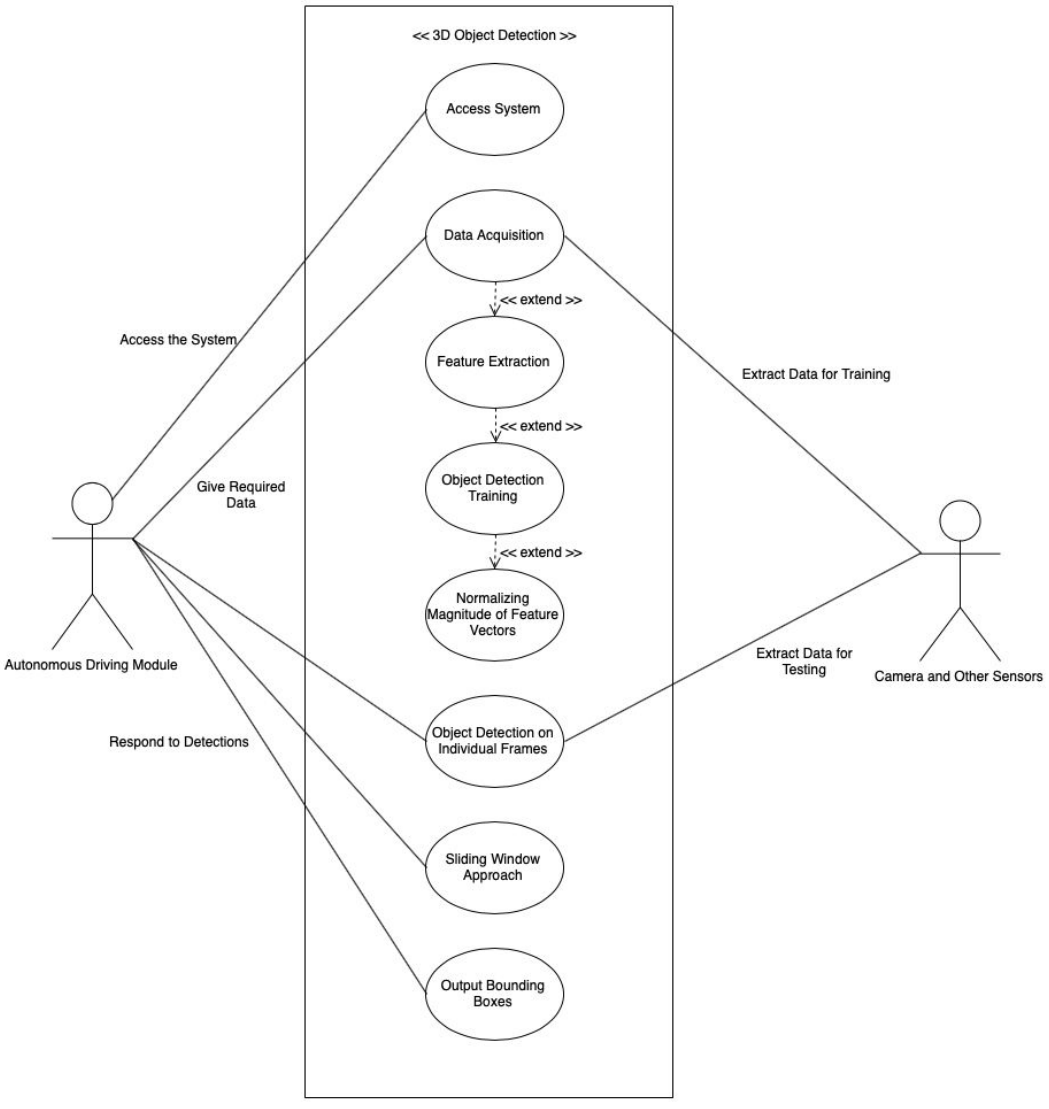

Figure 3.5: Use Case Diagram of the Proposed System

Figure 3.5 depicts use-case diagram of the proposed system, it has Autonomous Driving Module and Camera and Other Sensors as the actors. The Autonomous performs the tasks of accessing the system and viewing the result of testing data. The

extracted features are passed to the detection algorithm to detect the objects. After training the classifier and normalizing the magnitude of feature vectors we run a search on each individual video frame using sliding window approach then we output the detection values using bounding boxes.

Sequence Diagram

A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.

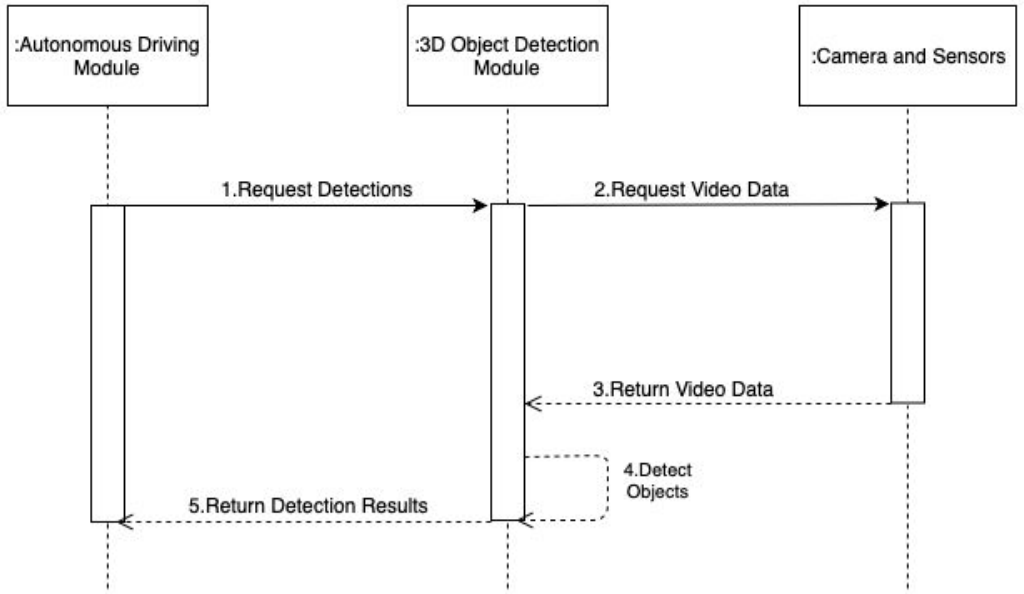

Figure 3.6: Sequence Diagram of the Proposed System

Figure 3.6 depicts the sequence diagram of the proposed system. The objects are of the Autonomous Driving Module, 3D Object Detection Module, Camera and Other Sensors. The object of 'Autonomous Driving Module' class performs the first operation 'Request Detections' in the system by sending a message to object of '3D Object Detection Module' class that begins training the classifier. The object of '3D Object Detection Module' class then sends a message 'Request Video Data' to an object of 'Camera and Other Sensors' class.

The 'Camera and Other Sensors' class object sends the video data back to the '3D Object Detection Module' class object, the '3D Object Detection Module' object

performs detection in the newly acquired video data and returns the results to the ‘Autonomous Driving Module’ object.

Chapter 4

IMPLEMENTATION

In this section, we discuss about the different techniques and methodologies used in our project.

4.1 Feature Extraction

In order to detect vehicles or any other objects we need to know what differentiates them from the rest of the image captured by the camera. Colors and gradients are good differentiators but the most important features will depend on the appearance of the objects.

Color alone as a feature can be problematic. Relying on the distribution of color values (or color histograms) may end up finding matches in unwanted regions of image. Gradients can offer a more robust presentation. Their presence in specific directions around the center could translate into a notion of shape. However a problem with using gradient is that they make the signature too sensitive.

HOG - Histogram of Oriented Gradients

HOG is a Computer Vision technique that counts occurrences of gradient orientation in localized portions of an image. If we compute the gradient magnitudes and directions at each pixel and then group them up into small cells we can use this “star” histogram to establish the dominant gradient direction for that cell.

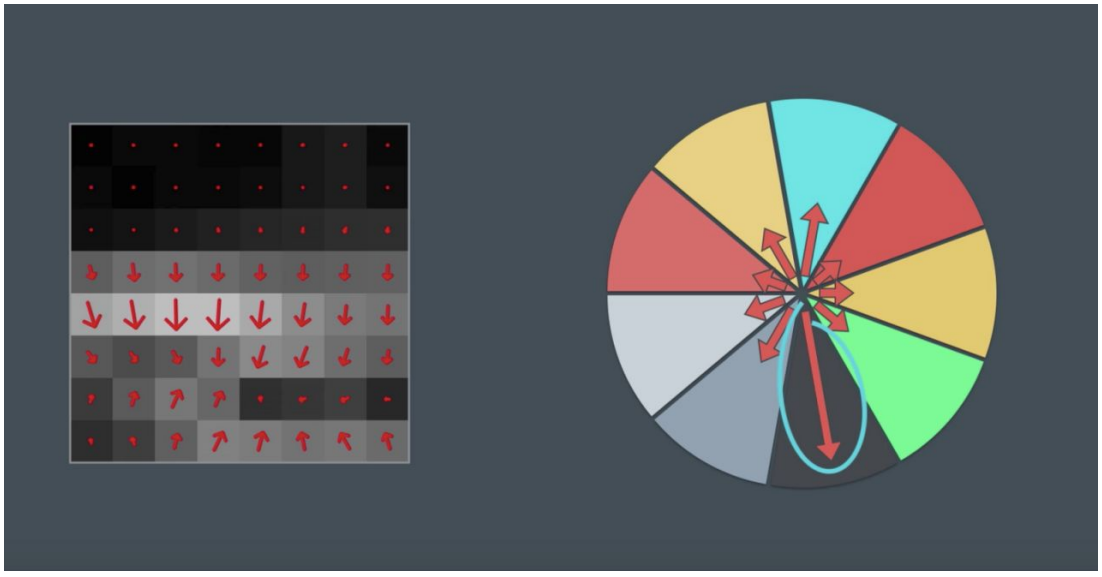

Figure 4.1: Star Histogram Of Gradient Directions

With that, even small shape variations will keep the signature unique for matching. The various parameters like the number of orientation bins, size of grids (of cells) , sizes of the cells, overlap between cells, are great to fine tune the algorithm.

HOG, Color Space, Spatial and Color Binning Parameters

In the final feature extraction function we applied a color transformation to YCrCb . We then concatenated the binned color feature vector, the histograms of color vector with the HOG feature vector. The HOG computation was done on all color channels.

```
color_space = 'YCrCb'
spatial_size = (16, 16)          # Spatial binning dimensions
hist_bins = 32                   # Number of histogram bins
orient = 9                       # HOG orientations
pix_per_cell = 8                 # HOG pixels per cell
cell_per_block = 2               # HOG cells per block
hog_channel = "ALL"              # Can be 0, 1, 2, or "ALL"
```

4.2 Training the Classifier

Once the features were extracted it was time to build and train the classifier. The approach is to build a classifier that can make a distinction between car and non-car

images. This classifier is then ran across the entire picture by sampling small patches. Each patch is then classified as car or non-car.

Datasets

We used labelled data for vehicle and non-vehicle examples in order to train the classifier. These example images come from a combination of the GTI vehicle image database, the KITTI vision benchmark suite.

Figure 4.2: Sample Data From Dataset

The data was split into a training set and a test set after being randomly shuffled to avoid possible ordering effects in the data.

Training is basically extracting the features vectors for every image in the training set. These vectors and their respective labels feed the training algorithm which iteratively changes the model until the error between predicted and actual labels is small enough. (Or the error stops decreasing after a number of iterations.)

4.3 Normalizing Magnitude of Feature Vectors

Before starting to train the classifier we normalized the feature vectors to zero mean and unit variance. This is necessary because there's a difference in magnitude between the color-based and gradient-based features and this can cause problems.

```
# Fit a per-column scaler
X_scaler = StandardScaler().fit(X)
# Apply the scaler to X
scaled_X = X_scaler.transform(X)
```

Support Vector Machine Classifier

We used support vector machines to classify the data. We implemented a Decision Tree classifier as well, however the accuracy wasn't looking promising so we decided to continue fine tuning the SVM.

4.4 The Sliding Window Approach

Now that we trained the classifier we will have to search for cars in the frame. The premise is that we define patches in the image and then run the classifier against each patch. The classifier will then decide if that patch "is" a car or not.

In the sliding window technique, we define a grid onto the image and move across it extracting the trained features. The classifier will give a prediction at each step and tell if that grid element contains car features.

Given the perspective of the images, objects far from the car camera will appear smaller and cars close by will appear larger. So it makes sense that the grid subdivision consider different sizes depending on the position on the image. We also do not consider any area above the horizon, ignoring regions where there's sky, mountains and trees.

Figure 4.3: Sliding Window Approach Output

Tracking Issues

It not surprising that the classifier will return a good number of false positives. These are regions with no cars but lighting or texture in the image will fool the classifier into calling it a car. This is obviously a big issue for a self-driving car application. False positives can cause the car to change direction or activate the brakes, potential cause for accidents.

To filter out the false positives we record the positions of all detections for each frame and compare with detections found in subsequent frames. Clusters of detections are likely an actual car. On the other side, detections that appear in one frame and not again in the the next, will be false positives.

4.5 Heatmap and Bounding Boxes

The search find_cars returns an array of hot boxes that classifier has predicted contains a vehicle. We created a heatmap to identify the clusters of overlapping boxes. With apply_threshold we then remove assumed false positives.

```
import collections
heatmaps = collections.deque(maxlen=10)

def process_frame(source_img):
   out_img, boxes = find_cars(source_img, ystart,
                      ystop, scale, svc,
                      X_scaler, orient, pix_per_cell,
                      cell_per_block, spatial_size,
                      hist_bins, False)
  current_heatmap =
             np.zeros_like(source_img[:,:,0]).astype(np.float)
  current_heatmap = add_heat(current_heatmap, boxes)
  heatmaps.append(current_heatmap)
  heatmap_sum = sum(heatmaps)
  heat_map = apply_threshold(heatmap_sum, 2)
  heatmap = np.clip(heat_map, 0, 255)
  labels = label(heatmap)
  labeled_box_img = draw_labeled_bboxes(source_img, labels)
return labeled_box_img
```

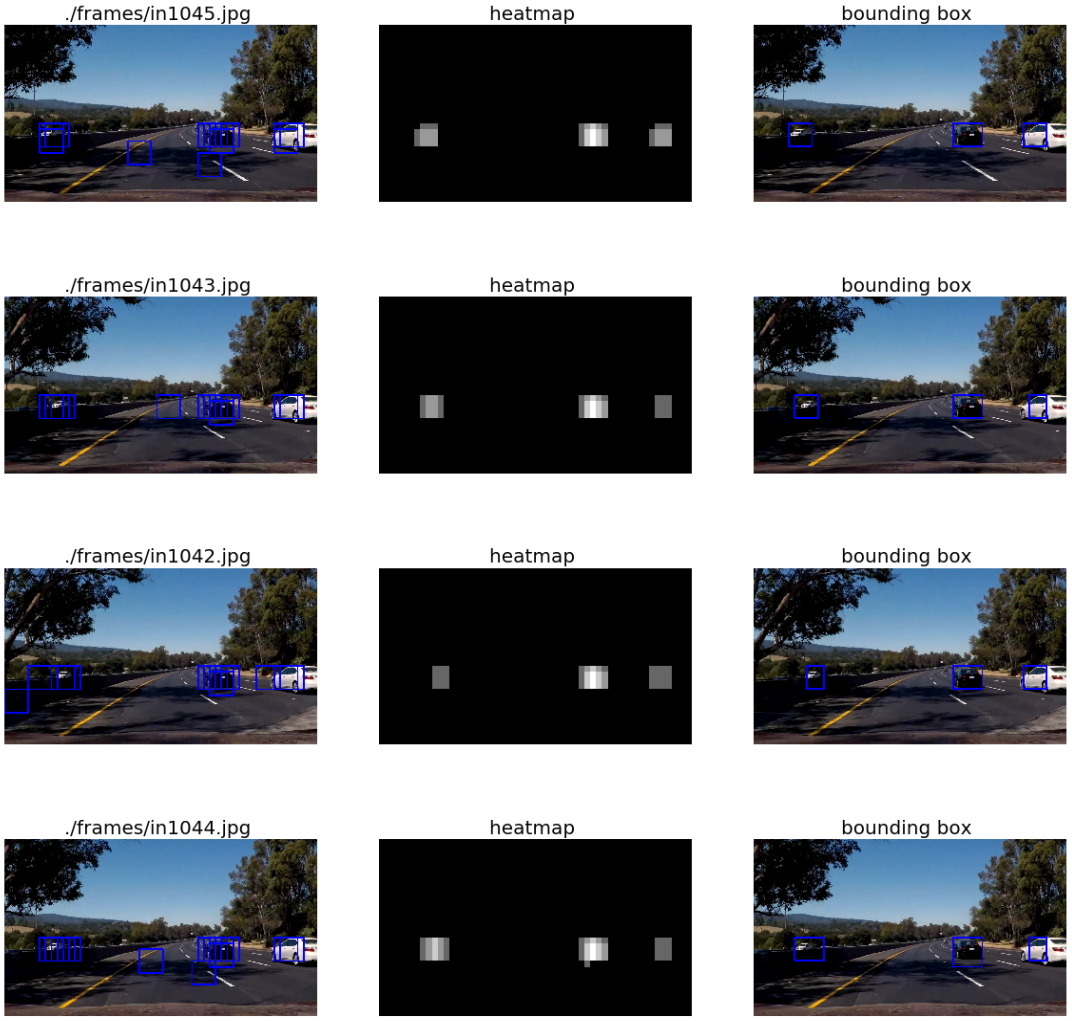

Figure 4.4: Heatmap And Bounding Boxes

The examples above are for individual frames. As mentioned before we want to take into account the time sequence to avoid one-offs to be considered. We used a deque data structure to always keep the boxes from the last 10 frames.

4.6 Algorithm

- Input: GTI vehicle image database, KITTI vision benchmark suite.
- Step 1: Work on labelled datasets to define training and testing sets.
- Step 2: Extract features from dataset images.
- Step 3: Train the classifier.
- Step 4: For each video frame: run a search using a sliding window technique and filter out false positives.
- Output: Video with vehicle detection bounding boxes.

Chapter 5

RESULTS

In this section we are going to discuss about our results.

Figure 5.1: Sample Test Data Output

The figure 5.1 shows the output of the classifier after training the classifier to distinguish between car and non-car images, this is done by running the classifier across the entire picture by sampling small patches, each patch was then classified as car or non-car.

```
3.2 Seconds to train SVC...
Test Accuracy of SVC =  0.9928
```

Figure 5.2: Support Vector Machine Statistics

In figure 5.2 we can see the accuracy and the training time taken by the support vector machine. We used support vector machines to classify the data. We implemented a Decision Tree classifier as well, however the accuracy wasn't looking promising so we decided to continue fine tuning the SVM.

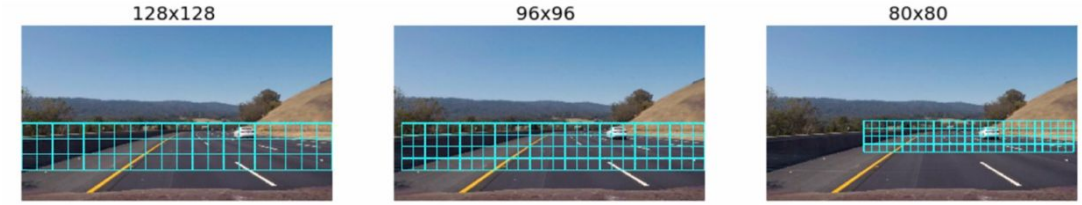

Figure 5.3: Sliding Window Approach Output

In figure 5.3, we can see the output after using sliding window approach. In the sliding window technique, we define a grid onto the image and move across it extracting the trained features. The classifier will give a prediction at each step and tell if that grid element contains car features. Given the perspective of the images, objects far from the car camera will appear smaller and cars close by will appear larger. So it makes sense that the grid subdivision consider different sizes depending on the position on the image. We also do not consider any area above the horizon, ignoring regions where there's sky, mountains and trees.

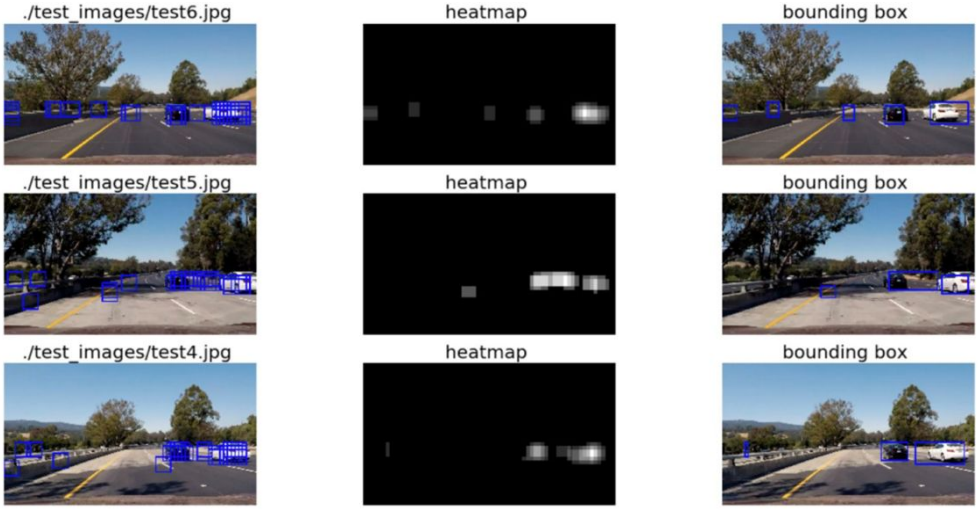

Figure 5.4: Heatmap And Bounding Boxes

In figure 5.4, we can see the heatmap and bounding boxes. The search find_cars returns an array of hot boxes that classifier has predicted contains a vehicle. We created a heatmap to identify the clusters of overlapping boxes. With apply_threshold we then remove assumed false positives.

The examples in figure 5.4 are for individual frames. As mentioned before we want to take into account the time sequence to avoid one-offs to be considered. We used a

deque data structure to always keep the boxes from the last 10 frames.

```
[MoviePy] >>>> Building video output_video_test4.mp4
[MoviePy] Writing video output_video_test4.mp4
100%|████████| 1260/1261 [08:35<00:00,  2.44it/s]
[MoviePy] Done.
[MoviePy] >>>> Video ready: output_video_test4.mp4

CPU times: user 12min 45s, sys: 16.1 s, total: 13min 2s
Wall time: 8min 36s
```

Figure 5.5: Video Writing Output

In figure 5.5, we can see that the CPU takes a total of 13 minutes 2 seconds to write the output video ascribing the bounding boxes where ever the model deems fit.

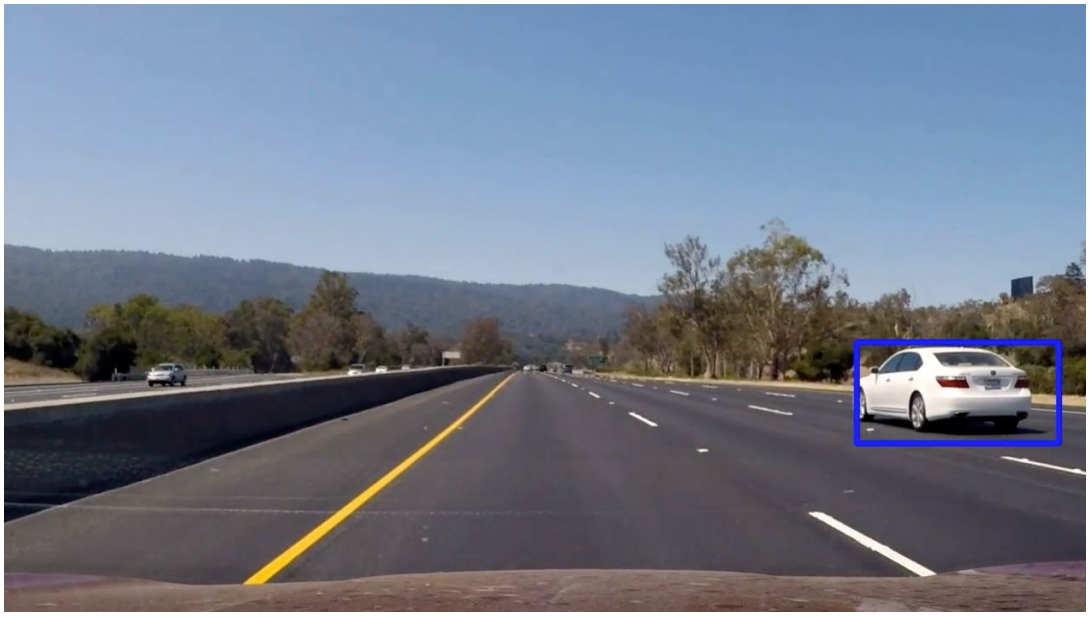

Figure 5.6: Video Output Screenshot-1

In figure 5.6, we have an example of a successful vehicle detection.

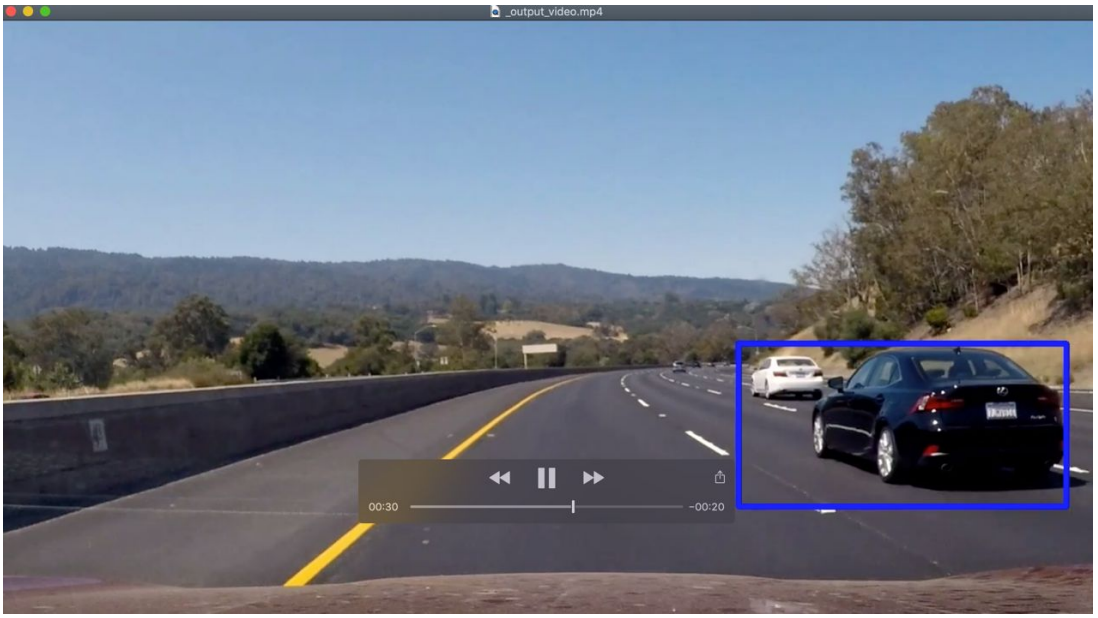

Figure 5.7: Video Output Screenshot-2

In figure 5.7, we have a situation where we have a single prediction for 2 vehicles in place of one prediction for each vehicle. Though this situation could be improved this isn't exceedingly poor behaviour given that the bounding box covers the two vehicles completely hopefully, detection at the next time-step would offer a refined estimation of the bounding box location.

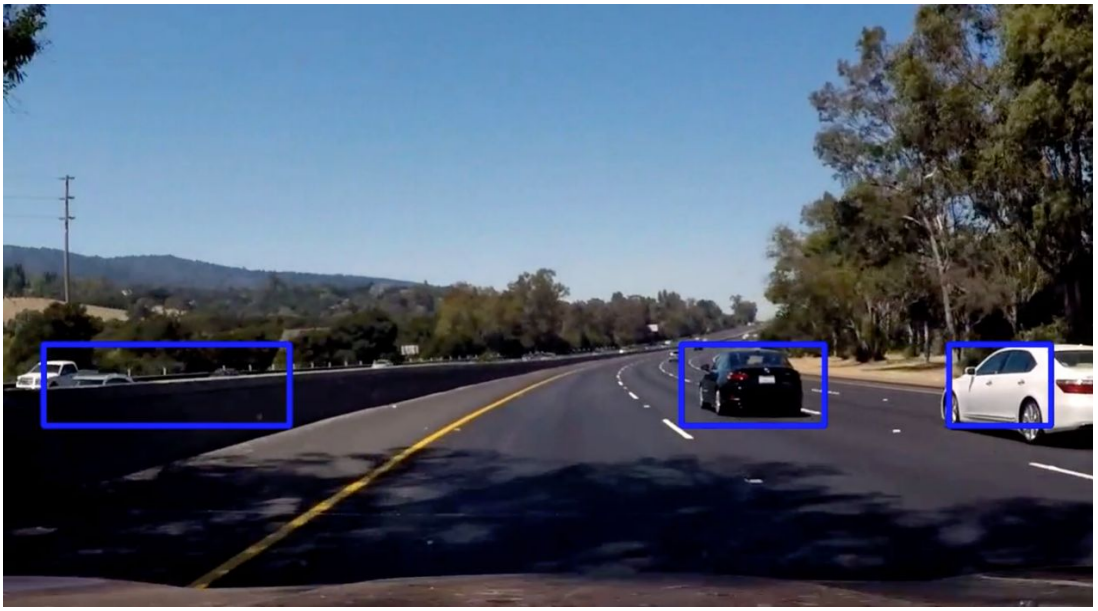

Figure 5.8: Video Output Screenshot-3

In figure 5.8, we have an example of detection where we can see that the vehicle on the opposite side of the road is also detected which we do not want. Such type of

detections occur whenever a vehicle on the opposite side of the road passes too close to our vehicle.

Figure 5.9: Video Output Screenshot-4

In figure 5.9, we have an example of a successful multi-vehicle detection and localization.

Chapter 6

CONCLUSION AND FUTURE ENHANCEMENTS

- In this work, we have examined an approach to deep object detection that makes bounding box predictions for an image without the need for expensive pre-processing or expensive deep evaluations. It's noticeable that some detections were of automobiles on the opposite course of the highway. While this isn't a terrible thing we ought to avoid them by using narrowing the vicinity of consideration. Instances when a automobile enters the body of the video and detection occurs with a little extend can be addressed via tweaking the quantity of frames considered in the average box calculation.
- We have constructed a complete Object Detection model with the use of methods such as HOG(Histogram Oriented Gradients), Sliding Window Search and others.

Future Enhancements

- Interesting future work includes trying to extend the training function to more accurately capture the desired metrics at hand, increasing the prediction speed by sharing computation, and increasing the size of data trained and tested on.
- More current methods make use of deep neural networks which can enhance function detection through increasing accuracy, reducing the occurrence of false positives and boosting performance. Tiny YOLO seems to be a frequent strategy.

REFERENCES

[1]. http://wardsauto.com/ar/ world_vehicle_population_110815.

[2]. http://esa.un.org/unpd/wpp/index.htm.

[3]. Systematics, "C. Crashes vs. Congestion–What's the Cost to Society*"*. American Automobile Association: Chicago, IL, USA, 2011.

[4]. Schrank, D.; Eisele, B.; Lomax, T.; Bak, J. "Urban Mobility Scorecard" Texas A&M Transportation Institute: College Station, TX, USA, 2015.

[5]. Levy, J.I.; Buonocore, J.J.; Von Stackelberg, K. "Evaluation of the public health impacts of traffic congestion: A health risk assessment". Environ. Health, 9, 65, 2010.

[6]. https://www.scribd.com/document/20618885/1918-March-10-Oakland-Tribune-Oakland-CA

[7]. https://www.youtube.com/watch?v=L3funFSRAbU.

[8]. Thorpe, C.; Hebert, M.H.; Kanade, T.; Shafer, S.A. "Vision and navigation for the Carnegie-Mellon Navlab". IEEE Trans. Pattern Anal. Mach. Intell. 10, 362–373, 1988.

[9]. Thrun, S.; Montemerlo, M.; Dahlkamp, H.; Stavens, D.; Aron, A.; Diebel, J.; Fong, P.; Gale, J.; Halpenny, M.; Hoffmann, G.; et al. Stanley: "The robot that won the DARPA Grand Challenge". J. Field Robot, 23, 661–692, 2006.

[10]. Urmson, C.; Anhalt, J.; Bagnell, D.; Baker, C.; Bittner, R.; Clark, M.N.; Dolan, J.; Duggins, D.; Galatali, T.; Geyer, C.; et al. "Autonomous driving in urban environments: Boss and the Urban Challenge". J. Field Robot, 25, 425–466, 2008.

[11]. Zhang, R.; Spieser, K.; Frazzoli, E.; Pavone, M. "Models, Algorithms, and Evaluation for Autonomous Mobility-On-DemandSystems". In Proceedings of the American Control Conference,Chicago,IL,USA; pp. 2573–2587, July 2015.

[12]. https://www.google.com/selfdrivingcar/.

[13]. https://www.tesla.com/presskit/autopilot.

[14]. Newton, C. “Uber will eventually replace all its drivers with self-driving cars”. Verge 2014, 5, 2014.

[15]. https://newsroom.uber.com/pittsburgh-self-driving-uber/.

[16]. De Graaf, M. PRT “Vehicle Architecture and Control in Masdar City”. 13th International Conference on Automated People Movers and Transit Systems, Paris, France; pp. 339–348, 22–25 May 2011.

[17]. Alessandrini, A.; Cattivera, A.; Holguin, C.; Stam, D. CityMobil2: “Challenges and Opportunities of Fully Automated Mobility”. Springer: Berlin, Germany; pp. 169–184, 2014.

[18]. A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” The International Journal of Robotics Research, vol. 32, no. 11, pp. 1231–1237, 2013.

[19]. M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, “Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?” in Robotics and Automation (ICRA), 2017 IEEE International Conference, pp. 746– 753, 2017.

[20]. Krizhevsky, A., Sutskever, I., and Hinton, G. E. “Imagenet classification with deep convolutional neural networks”. In Advances in neural information processing systems, pp. 1097–1105, 2012.

[21]. Deng, J., Berg, A., Satheesh, S., Su, H., Khosla, A., and Fei-Fei, L. Imagenet large scale visual recognition competition, 2012.

[22]. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. “Integrated recognition, localization and detection using convolutional networks”. arXiv preprint arXiv:1312.6229 2013.

[23]. Simonyan, K., and Zisserman, A. “Very deep convolutional networks for large-scale image recognition”. arXiv preprint arXiv:1409.1556 2014.

[24]. He, K., Zhang, X., Ren, S., and Sun, J. “Deep residual learning for image recognition”. arXiv preprint arXiv:1512.03385 2015.

[25]. Girshick, R., Donahue, J., Darrell, T., and Malik, J. “Rich feature hierarchies for accurate object detection and semantic segmentation”. IEEE conference on computer vision and pattern recognition, pp. 580–587, 2014.

[26]. He, K., Zhang, X., Ren, S., and Sun, J. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. European Conference on Computer Vision, Springer, pp. 346–361, 2014.

[27]. Girshick, R. “Fast r-cnn”. IEEE International Conference on Computer Vision, pp. 1440–1448, 2015.

[28]. Ren, S., He, K., Girshick, R., and Sun, J. “Faster r-cnn: Towards real- time object detection with region proposal networks”. In Advances in neural information processing systems, pp. 91–99, 2015.

[29]. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. “You only look once: Unified, real-time object detection”. IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788, 2016.

[30]. Uijlings, J. R., van de Sande, K. E., Gevers, T., and Smeulders, A. W. “Selective search for object recognition”. International journal of computer vision 104, 2, 154–171, 2013.

[31]. Mikolajczyk, K., and Schmid, C. “A performance evaluation of local descriptors”. IEEE transactions on pattern analysis and machine intelligence 27, 10, 1615–1630, 2005.

[32]. Viola, P., and Jones, M. “Rapid object detection using a boosted cascade of simple features”. In Computer Vision and Pattern Recognition, 2001. CVPR 2001. IEEE Computer Society Conference, vol. 1, IEEE, pp. I–511, 2001.

[33]. Sadeghi, M. A., and Forsyth, D. “30hz object detection with dpm v5”. European Conference on Computer Vision, Springer, pp. 65–79, 2014.

[34]. Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ra- manan, D. “Object detection with discriminatively trained part-based models”. IEEE transactions on pattern analysis and machine intelligence, 1627–1645, 2010.

[35]. Li, Bo. “3D Fully Convolutional Network for Vehicle Detection in Point Cloud”. eprint arXiv:1611.08069 2016.

[36]. Zhou, Yin, Tuzel, Oncel. VoxelNet: “End-to-End Learning for Point Cloud Based 3D Object Detection”. eprint arXiv:1711.06396 2017.

APPENDIX

Vehicle Detection and Tracking

Global Variables and Imports

In [1]:

```
import time
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import glob
import cv2
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from skimage.feature import hog
from scipy.ndimage.measurements import label
from sklearn.model_selection import train_test_split

color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
spatial_size = (16, 16) # Spatial binning dimensions
hist_bins = 32    # Number of histogram bins
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = "ALL" # Can be 0, 1, 2, or "ALL"
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
color_space = 'YUV'
orient = 11
pix_per_cell = 16
cell_per_block = 2
hog_channel = "ALL"
%matplotlib inline
```

Feature Extraction

Channel, HOG and Color Functions

In [2]:

```
def bin_spatial(img, size=(32, 32)):
    ch1 = cv2.resize(img[:,:,0], size).ravel()
    ch2 = cv2.resize(img[:,:,1], size).ravel()
```

```
    ch3 = cv2.resize(img[:,:,2], size).ravel()
    features = np.concatenate ([ch1, ch2, ch3])
    return features


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=True,
                                  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=True,
                       visualise=vis, feature_vector=feature_vec)
        return features

def color_hist(img, nbins=32, bins_range=(0, 1)):
    channel1_hist = np.histogram(img[:,:,0], bins=nbins, range=bins_range)
    channel2_hist = np.histogram(img[:,:,1], bins=nbins, range=bins_range)
    channel3_hist = np.histogram(img[:,:,2], bins=nbins, range=bins_range)
     hist_features = np.concatenate((channel1_hist[0], channel2_hist[0],
channel3_hist[0]))
   return hist_features
```

Feature Extraction That Iterates Over Training Data

In [3]:

```
def extract_features(imgs, color_space='RGB',
                     spatial_size=(32, 32),hist_bins=32,
                     orient=9, pix_per_cell=8,
                     cell_per_block=2, hog_channel=0,
                     spatial_feat=True, hist_feat=True,
                     hog_feat=True):


    features = []

    for file in imgs:
```

```
        file_features = []
        image = mpimg.imread(file)

        if color_space != 'RGB':
            feature_image = cv2.cvtColor (image, getattr(cv2, 'COLOR_RGB2' +
color_space))
        else:
            feature_image = np.copy(image)
        if spatial_feat == True:
            spatial_features = bin_spatial(feature_image, size=spatial_size)
            file_features.append(spatial_features)
        if hist_feat == True:
            hist_features = color_hist(feature_image, nbins=hist_bins)
            file_features.append(hist_features)
        if hog_feat == True:
        # 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)
                 file_features.append(hog_features)
        features.append(np.concatenate(file_features))

    return features
```

Training the Classifier

Reading the Training Data

In [4]:

```
def display_image(image, gray=False):
    plt.figure()
    if gray:
        plt.imshow(image, cmap='gray')
    else:
        plt.imshow(image)
    plt.show()

def display_grid_images (images, grid_size, cell_size=(10, 10), cmap=None,
titles=None):
    sizex, sizey = grid_size[0], grid_size[1]
    fig, imtable = plt.subplots (sizey, sizex, figsize=cell_size, squeeze=False)
```

```
    for j in range (sizey):
        for i in range (sizex):
            im_idx=i+j*sizex
            if (isinstance(cmap,(list, tuple))):
                imtable[j][i].imshow(images[im_idx], cmap=cmap[i])
            else:
                im=images[im_idx]
                if len(im.shape) == 3:
                    imtable[j][i].imshow(im)
                else:
                    imtable[j][i].imshow(im, cmap='gray')
            imtable[j][i].axis('off')
            if not titles is None:
                channels = " #str(images[im_idx].shape)
                imtable[j][i].set_title(titles[im_idx]+" "+channels, fontsize=20)

    plt.show ()


def read_image_file(path):
    image = mpimg.imread(path)
    read_image = np.copy(image)

    #if 'png' in path:
    #    read_image = image.astype(np.float32) * 255
    return read_image


def convert_color(img, conv='RGB2YCrCb'):
    if conv == 'RGB2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    if conv == 'BGR2YCrCb':
        return cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
    if conv == 'RGB2LUV':
        return cv2.cvtColor(img, cv2.COLOR_RGB2LUV)



def list_data_files(path, extension='.png'):
    imgs = []
    for root, dirs, files in os.walk(path):
        for file_ in files:
            if file_.endswith(extension):

                imgs.append( os.path.join(root, file_) )
    return imgs


cars_dir = './data/vehicles/'
```

```
cars_list = list_data_files(cars_dir)


notcars_dir = './data/non-vehicles/'
notcars_list = list_data_files(notcars_dir)

print('Number of vehicle files: {}'.format(len(cars_list)))
print('Number of non-vehicle files: {}'.format(len(notcars_list)))
```

Sample Data

In [5]:

```
rand_cars_ind = np.random.randint(0, len(cars_list))
rand_notcars_ind = np.random.randint(0, len(notcars_list))

car_sample_img = read_image_file(cars_list[rand_cars_ind])

not_car_sample_img = read_image_file(notcars_list[rand_notcars_ind])

display_grid_images([car_sample_img, not_car_sample_img], (2,1), cell_size=(15,
10),
                    titles=('This is a car','This is not a car!'))
```

Extracting Features From Training Data

In [6]:

```
car_features = extract_features(cars_list, color_space,spatial_size, hist_bins, orient,
pix_per_cell,
                                cell_per_block, hog_channel, spatial_feat, hist_feat, hog_feat)

notcar_features  = extract_features(notcars_list, color_space, spatial_size, hist_bins,
orient, pix_per_cell,
                                cell_per_block, hog_channel, spatial_feat, hist_feat, hog_feat)


image = np.sqrt(image)
```

Normalize Feature Vectors

In [7]:

```
X = np.vstack((car_features, notcar_features)).astype(np.float64)
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))
print (len(y))

X_scaler = StandardScaler().fit(X)
```

```
scaled_X = X_scaler.transform(X)
```

Split Up Data Into Randomized Training and Test Sets

In [8]:

```
rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(scaled_X, y, test_size=0.2,
random_state=rand_state)


print('Parameters:',orient,'orientations,',pix_per_cell,'pixels per cell,', cell_per_block,
    'cells per block, spatial binning of',spatial_size, 'and', hist_bins,'histogram bins')
print('Feature vector length:', len(X_train[0]))
```

Linear SVM Classifier

In [9]:

```
svc = LinearSVC(C=0.1)

t=time.time()
svc.fit(X_train, y_train)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to train SVC...')

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

Sliding Window Search

In [10]:

```
def slide_window(img, x_start_stop=[None, None], y_start_stop=[None, None],
                 xy_window=(64, 64), xy_overlap=(0.5, 0.5)):
    # If x and/or y start/stop positions not defined, set to image size
    if x_start_stop[0] == None:
        x_start_stop[0] = 0
    if x_start_stop[1] == None:
        x_start_stop[1] = img.shape[1]
    if y_start_stop[0] == None:
        y_start_stop[0] = 0
    if y_start_stop[1] == None:
        y_start_stop[1] = img.shape[0]

    xspan = x_start_stop[1] - x_start_stop[0]
    yspan = y_start_stop[1] - y_start_stop[0]

    nx_pix_per_step = np.int(xy_window[0]*(1 - xy_overlap[0]))
    ny_pix_per_step = np.int(xy_window[1]*(1 - xy_overlap[1]))
```

```
    nx_buffer = np.int(xy_window[0]*(xy_overlap[0]))
    ny_buffer = np.int(xy_window[1]*(xy_overlap[1]))
    nx_windows = np.int((xspan-nx_buffer)/nx_pix_per_step)
    ny_windows = np.int((yspan-ny_buffer)/ny_pix_per_step)

    window_list = []
        for ys in range(ny_windows):
        for xs in range(nx_windows):
            startx = xs*nx_pix_per_step + x_start_stop[0]
            endx = startx + xy_window[0]
            starty = ys*ny_pix_per_step + y_start_stop[0]
            endy = starty + xy_window[1]
             window_list.append(((startx, starty), (endx, endy)))

    return window_list

def single_img_features(img, color_space, spatial_size,
                        hist_bins, orient,
                        pix_per_cell, cell_per_block, hog_channel,
                        spatial_feat, hist_feat, hog_feat):


    img_features = []
    # Apply color conversion if other than 'RGB'
    if color_space != 'RGB':
            feature_image = cv2.cvtColor (img, getattr(cv2, 'COLOR_RGB2' +
color_space))
    else:
        feature_image = np.copy(img)
    # Compute spatial features if flag is set
    if spatial_feat == True:
        spatial_features = bin_spatial(feature_image, size=spatial_size)

        img_features.append(spatial_features)

    if hist_feat == True:
        hist_features = color_hist(feature_image, nbins=hist_bins)

        img_features.append(hist_features)
    # Compute HOG features if flag is set
    if hog_feat == True:
        if hog_channel == 'ALL':
            hog_features = []
            for channel in range(feature_image.shape[2]):
                hog_features.extend(get_hog_features(feature_image[:,:,channel],
                                orient, pix_per_cell, cell_per_block,
                                vis=False, feature_vec=True))
```

```
        else:
            hog_features = get_hog_features(feature_image[:,:,hog_channel], orient,
                    pix_per_cell, cell_per_block, vis=False, feature_vec=True)

        img_features.append(hog_features)


    return np.concatenate(img_features)

def search_windows(img, windows, clf, scaler, color_space,
                   spatial_size, hist_bins, orient,
                   pix_per_cell, cell_per_block,
                   hog_channel, spatial_feat,
                   hist_feat, hog_feat):


    on_windows = []

    for window in windows:

        test_img = cv2.resize(img[window[0][1]:window[1][1],
window[0][0]:window[1][0]], (64, 64))

        features = single_img_features(test_img, color_space,
                        spatial_size, hist_bins,orient, pix_per_cell,cell_per_block,
                        hog_channel, spatial_feat, hist_feat, hog_feat)
        test_features = scaler.transform(np.array(features).reshape(1, -1))

        prediction = clf.predict(test_features)

        if prediction == 1:
            on_windows.append(window)

    return on_windows

def draw_boxes(img, bboxes, color, thick):

    imcopy = np.copy(img)

    for bbox in bboxes:

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

    return imcopy
```

Tiling Schemes

In [11]:
```
image = mpimg.imread('./test_images/test3.jpg')
window_img = np.copy(image)

sw_x_limits = [[None, None],[32, None],[400, 1280]]
sw_y_limits = [[400, 640],[400, 600],[390, 540]]
sw_window_size = [(128, 128),(96, 96),(64, 64)]
sw_overlap = [(0.5, 0.5),(0.5, 0.5),(0.5, 0.5)]

display_imgs = []

for i in range(3):
  windows = slide_window(window_img, sw_x_limits[i], sw_y_limits[i],
sw_window_size[i], sw_overlap[i])
  bboxes = search_windows(window_img, windows, svc, X_scaler, color_space,
                    spatial_size, hist_bins, orient,
                    pix_per_cell, cell_per_block,
                    hog_channel, spatial_feat,
                    hist_feat, hog_feat)
  display_img = draw_boxes(window_img, bboxes, color=(0, 255, 255), thick=3)
  display_imgs.append(display_img)


display_grid_images(display_imgs, (3, 1), cell_size=(20, 10),
titles=('128x128','96x96', '80x80'))
```

Optimizing Search Windows

In [12]:
```
def find_cars(img, ystart, ystop, scale, svc, X_scaler, orient, pix_per_cell,
          cell_per_block, spatial_size, hist_bins, display_boxes):

  draw_img = np.copy(img)
  img = img.astype(np.float32)/255
  boxes = []

  img_tosearch = img[ystart:ystop,:,:]
  ctrans_tosearch = convert_color(img_tosearch, conv='RGB2YCrCb')
  if scale != 1:
     imshape = ctrans_tosearch.shape
     ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1]/scale),
np.int(imshape[0]/scale)))

  ch1 = ctrans_tosearch[:,:,0]
  ch2 = ctrans_tosearch[:,:,1]
  ch3 = ctrans_tosearch[:,:,2]
```

```
  nxblocks = (ch1.shape[1] // pix_per_cell) - cell_per_block + 1
  nyblocks = (ch1.shape[0] // pix_per_cell) - cell_per_block + 1
  nfeat_per_block = orient*cell_per_block**2


  window = 64
  nblocks_per_window = (window // pix_per_cell) - cell_per_block + 1
  cells_per_step = 1
  nxsteps = (nxblocks - nblocks_per_window) // cells_per_step
  nysteps = (nyblocks - nblocks_per_window) // cells_per_step


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

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

        xleft = xpos*pix_per_cell
        ytop = ypos*pix_per_cell


        subimg = cv2.resize(ctrans_tosearch[ytop:ytop+window, xleft:xleft+window],
(64,64))


        spatial_features = bin_spatial(subimg, size=spatial_size)
        hist_features = color_hist(subimg, nbins=hist_bins)


        test_features = X_scaler.transform(np.hstack((spatial_features, hist_features,
hog_features)).reshape(1, -1))
```

```
        test_prediction = svc.predict(test_features)

        if test_prediction == 1:
            xbox_left = np.int(xleft*scale)
            ytop_draw = np.int(ytop*scale)
            win_draw = np.int(window*scale)


            if display_boxes:
                cv2.rectangle(draw_img,(xbox_left,
ytop_draw+ystart),(xbox_left+win_draw,ytop_draw+win_draw+ystart),(0,0,255),6)

            boxes.append(((xbox_left,
ytop_draw+ystart),(xbox_left+win_draw,ytop_draw+win_draw+ystart)))

  return draw_img, boxes



def add_heat(heatmap, bbox_list):

   for box in bbox_list:


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


   return heatmap

def apply_threshold(heatmap, threshold):

   heatmap[heatmap <= threshold] = 0
   return heatmap

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

      nonzero = (labels[0] == car_number).nonzero()
      nonzeroy = np.array(nonzero[0])
      nonzerox = np.array(nonzero[1])

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

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

   return img
```

Debugging Some Troublesome Frames

In [13]:

```
images = glob.glob('./test_images/*.jpg')
images = ['./frames/in1045.jpg', './frames/in1043.jpg',
        './frames/in1042.jpg', './frames/in1044.jpg']

ystart = 400
ystop = 656
scale = 1.2

display_images = []
title = []
labeled_box_imgs = []

for idx, fname in enumerate(images):
  source_img = read_image_file(str(fname))
  out_img, boxes = find_cars(source_img, ystart, ystop, scale, svc, X_scaler, orient,
pix_per_cell,
                  cell_per_block, spatial_size, hist_bins, True)

  heat_map = np.zeros_like(source_img[:,:,0]).astype(np.float)
  heat_map = add_heat(heat_map, boxes)
  heat_map = apply_threshold(heat_map, 1)
  heatmap = np.clip(heat_map, 0, 255)

  labels = label(heatmap)
  labeled_box_img = draw_labeled_bboxes(source_img, labels)

  display_images.append(out_img)
  display_images.append(heatmap)
  display_images.append(labeled_box_img)
  title.append(fname)
  title.append('heatmap')
  title.append('bounding box')

display_grid_images(display_images, (3, int(len(display_images)/3)), cell_size=(20,
20), titles=title)
```

In [14]:

```
import collections
heatmaps = collections.deque(maxlen=10)

def process_frame(source_img):

  out_img, boxes = find_cars(source_img, ystart, ystop, scale, svc, X_scaler, orient,
pix_per_cell, cell_per_block, spatial_size, hist_bins, False)
```

```
    current_heatmap = np.zeros_like(source_img[:,:,0]).astype(np.float)
    current_heatmap = add_heat(current_heatmap, boxes)
    heatmaps.append(current_heatmap)
    heatmap_sum = sum(heatmaps)

    heat_map = apply_threshold(heatmap_sum, 2)
    heatmap = np.clip(heat_map, 0, 255)
    labels = label(heatmap)
    labeled_box_img = draw_labeled_bboxes(source_img, labels)


    return labeled_box_img


def save_frame(source_img):

    global iidx
    iidx += 1
    write_name = './frames/out'+str(iidx)+'.jpg'
    img = cv2.cvtColor(source_img, cv2.COLOR_BGR2RGB)
    cv2.imwrite(write_name, img)
```

In[15]

```
from moviepy.editor import VideoFileClip
from IPython.display import HTML
global iidx
iidx = 0
clip = VideoFileClip('project_video.mp4')
out_clip = clip.fl_image(process_frame)
%time out_clip.write_videofile('output_video.mp4', audio=False)
```

In[16]

```
HTML("""
<video width="1280" height="760" controls>
 <source src="{0}">
</video>
""".format('output_video.mp4'))
```