DEVELOPMENT OF A DEEP LEARNING APPROACH FOR SEMANTIC UNDERSTANDING OF VEHICLE ACTIVITY

A PROJECT REPORT

Submitted by

R HARISH KUMAR [RA1511018010120]
KORADA MADHU [RA1511018010203]
S GOKULL [RA1511018010211]

Under the guidance of

Dr. R. SENTHILNATHAN, Ph.D
(Associate Professor, Department of Mechatronics Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

MECHATRONICS ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2019

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**DEVELOPMENT OF A DEEP LEARNING APPROACH FOR SEMANTIC UNDERSTANDING OF VEHICLE ACTIVITY** " is the bonafide work of "**R HARISH KUMAR [RA1511018010120], KORADA MADHU [RA1511018010203], S GOKULL[RA1511018010211]**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. R. SENTHILNATHAN, Ph.D
GUIDE
Associate Professor
Dept. of Mechatronics Engineering

SIGNATURE

Dr. G. MURALI, Ph.D
HEAD OF THE DEPARTMENT
Dept. of Mechatronics Engineering

Signature of the Internal Examiner

Signature of the External Examiner

ABSTRACT

Autonomous vehicles as well as vehicles with ADAS features use multiple sensors and integrate their data to take decisions and assist in the driving process as well as in automating it. By using LiDAR, RADAR, etc., data can be acquired in metric units which helps in taking decisions in real world environment. However, the high cost of such sensors is a major drawback. In the recent years, researchers as well as autonomous vehicle manufacturers have been looking at the feasibility of using only RGB cameras for perception. This is largely attributed to the influence of deep learning on computer vision in the current decade.

In that context, this project work has been focussed on the perception aspect of autonomous vehicles with data from a single RGB camera. The primary objective is to predict the semantic activity of the ego vehicle as well as the vehicles on the scene. It is achieved using deep neural networks for three major tasks namely, object detection and tracking, lane detection and optical flow estimation. Object detection and tracking is used to detect and track the vehicles on the scene. Lane detection is used to understand the context of the road. Optical flow estimation is used to capture the motion of vehicles. The outputs of the three networks are manually integrated to interpret the semantic vehicle activity. The secondary objective is to run this integration on an embedded board. The performance of the same is evaluated and the results are presented.

This work is a step in the shift towards perception with only RGB camera. An end-to-end implementation of the same could be used as a warning system as part of an ADAS feature in vehicles to implement collision warning, lane departure warning, etc. The major benefit of such a system is the reduced cost which makes it easier for the manufacturers to equip their vehicles with these systems at a reasonable price.

ACKNOWLEDGEMENTS

It has been a great honor and privilege to undergo **B.Tech** in **MECHATRONICS** at **SRM Institute of Science and Technology**. We are very much thankful to the Department of Mechatronics, SRM Institute of Science and Technology for providing all facilities and support to meet our project requirements.

We would like to express our heartfelt, sincere thanks to our project guide, **Dr. R. SENTHILNATHAN, Ph.D.** for encouraging us to take up a challenging project and for offering his continuous support and exceptional ideas throughout the course of the project.

We would like to express our sincere gratitude to the Head of Department **Dr. G. MURALI, Ph.D.** and all the staff who offered their support for the success of the project.

We would also like to thank **Mr. R. Prakash**, Lab Assistant, for offering his support during our work in the Motion Analysis Laboratory.

Finally, we wish to express how grateful we are to our family and friends for their unfailing support and strong encouragement throughout the course of the project.

R HARISH KUMAR

KORADA MADHU

S GOKULL

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IR	Infrared Radiation
AP	Average Precision
CNN	Convolutional Neural Network
ADAS	Advanced Driver Assistance System
PC	Personal Computer
GPU	Graphics Processing Unit
CPU	Central Processing Unit
DNN	Deep Neural Network
YOLO	You Only Look Once
MOT	Multiple Object Tracking
FP	False Positives
FN	False Negatives
SCNN	Spatial Convolutional Neural Network
IoU	Intersection over Union
NMS	Non Max Suppression
EPE	End-to-end Point Error
APE	Angular Point Error
fps	frames per second
RCNN	Region based Convolution Neural Network

SORT	Simple Online Real-time Tracking
SSD	Single Shot Multibox Detector
2-D	2-dimensional
3-D	3-dimensional
NCS	Neural Compute Stick

CHAPTER 1

INTRODUCTION

1.1 AUTONOMOUS DRIVING AND ADAS

Vehicles have played a significant role in transforming our lifestyle ever since their inception by making transportation easier and faster. During the early years, major efforts were made to make them more efficient, powerful and suitable for long distance transportation. Over the last few decades further emphasis has been laid on reducing human effort by making intelligent systems. These changes started as minimal feature additions which assisted the driver in various tasks and have evolved to become Driver Assistance Systems as observed today.

In an autonomous vehicle, perception, cognition and action take place with little or no human interference. The standard followed throughout the world for automation in vehicles is provided by Society of Automotive Engineering (SAE) as shown in Figure 1.1.

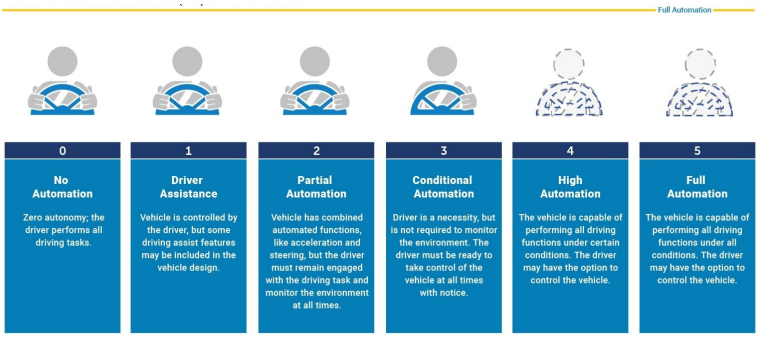


Figure 1.1: Stages of Autonomy by SAE

Advanced Driver Assistance System (ADAS) is one of the beginning steps in the process of achieving Autonomous Driving. Features of ADAS fall within Level 1 or Level 2 depending on the level of control they have over the vehicle. Level 1 features such as Lane Departure Warning System mostly consist of perception which warns the

driver in case of deviation of the ego vehicle from the ego lane. Level 2 features such as Lane Centering System does perception as well as partial control over the ego vehicle such that it stays in the centre of the lane. The amount of cognition and control increases with appropriate perception. For attaining stage 3 autonomy and above, perception in higher dimensions is mandatory.

ADAS features are available in most commercial vehicles right now either as a built-in package or as an add-on. They assist the driver in the process reducing the effort and skill required to perform tasks such as parallel parking, lane maintenance, etc. These features serve as building blocks which when integrated help to achieve complete autonomy of the vehicle in the near future.

1.2 TASKS ADDRESSED BY COMPUTER VISION

Computer Vision is the science of making machines see and make sense of the environment. It is different from digital image processing as it takes image as the input and provides an inference as the output. Some of the tasks addressed by computer vision include classification, object detection, segmentation, pose estimation, 3-dimensional (3-D) reconstruction, etc. With the arrival of deep learning, there was a major shift from feature based algorithms to learning based algorithms. Almost every task could be taught to a computer without explicitly mentioning the features provided enough data is available for training.

Many industries benefited from this transition, especially the automotive industry. Autonomous driving became the most important topic of research. Perception is the primary task in autonomous driving as the subsequent tasks such as cognition and action are dependent on the results of perception. It is directly based on computer vision and deep learning. The tasks addressed by computer vision are used in this application. Some of the tasks which found their application in autonomous vehicles are as follows.

1.2.1 Object Detection

It involves classification and localization of the objects in the frame. Classification provides the class information i.e. car, truck, person, etc. while localization provides their bounding box coordinates. It also provides the confidence score with which each object is predicted as a particular class. Tracking is used to maintain the identity of a specific object across frames. It provides a relation between the same object along successive frames.

1.2.2 Segmentation

It gives pixel based classification where each pixel is allocated to a class. Each class of pixels is specified with a different colour. The advantage of this over standard object detection is that this provides an outline of the object whereas the former provides rectangle boxes for everything.

1.2.3 Optical Flow

It provides the apparent movement of pixels across frames to estimate motion. The motion of the objects which are moving in the real world can be measured in the image plane. The flow vector will have X and Y components. Optical flow estimation can be used to predict relative motion between vehicles, the motion of other vehicles in general, etc.

1.3 MODALITIES IN SENSING

Imaging Modalities vary throughout the electromagnetic spectrum right from gamma rays to radio waves. It also includes acoustic imaging. While each modality has its own application where it is best suited, generally, visible light (RGB) and infrared are considered for most tasks.

Sensors in autonomous vehicles use modalities such as Infrared Radiation (IR), vis-

ible light and radio waves. IR and radio waves are used in sensors which provide 3-D map of the environment. Even though they help in path planning and navigation, RGB is the most important modality in this application as we need the vehicle to view its surroundings as humans view it, i.e. in colour. Depth data alone is not sufficient to understand the environment completely, as a lot of information such as traffic signals, sign boards, etc. are colour coded. In order to understand them, RGB sensor camera is a must and it becomes an important modality. The fact that a pair of stereo cameras can provide depth using a disparity map supports the fact that RGB sensors can be used to measure depth too but the other sensors cannot provide colour data. Majority of the computer vision tasks such as object detection, pixel segmentation, optical flow estimation work on RGB images.

1.4 INTRODUCTION TO DEEP LEARNING

Artificial Intelligence is the science of providing machines with the ability to think and make decisions on their own. Contrary to popular belief, the theory behind most fundamentals of artificial intelligence were laid back in the 20th century. Neural networks are set of algorithms loosely modelled after the human brain, that are designed to recognize pattern. The theory behind the first neural networks were published back in 1943 by Warren McCulloch [36]. However, as the number of layers increased, it became increasingly difficult to perform the computations with the hardware of that time. Hence, they remained a theory for a very long time.

In the 1990s, a number of neural network architectures were proposed and active research was conducted. Convolutional Neural Network (CNN) were used to detect handwritten digits on cheques in the US from the early 2000s. During this period, machine learning techniques such as Support Vector Machines were already in use for various applications where the Machine Learning Engineer had to tune the hyperparameters to classify the data into different categories.

It was in 2012, Alex Krizhevsky and his team won the ImageNet challenge [3] using CNN named AlexNet [15] for image classification. AlexNet achieved state of the art results by reducing the error dramatically, proving the accuracy of such networks.

The beauty of deep learning is that the programmer need not create the feature space on his own. The network will identify these features on its own and classify them accordingly if it was provided with enough data (input and corresponding output labels) which made deep learning as black box. In fact, 'deep' in deep learning meant the large number of layers the network was made of. This enabled the network to process the input and create more and more abstract features of the given input with increase in layers. The time and effort required to generate this level of abstraction using machine learning techniques is very large.

With the advent of Graphics Processing Unit (GPU) and availability of large datasets, deep learning immediately took off and became a popular technique among researchers and companies. It is used in various industries such as automotive, e-commerce, social media, finance, etc., for applications involving sound, time series, text, image and video.

One of the most influenced industries by the arrival of deep learning is the automotive industry. Autonomous vehicles garnered a huge popularity as it was one of the most direct applications of deep learning. It helped solve the problem of perception exceptionally well in autonomous vehicles. Vehicle detection, pedestrian detection, traffic sign recognition, drivable area segmentation are some of the problems being solved by deep learning. This success is largely due to the ability of deep neural networks to make use of the large data available in the form of dashcam videos, etc. Even though people are cynical about providing control to autonomous vehicles, tests are being implemented by many researchers and companies to prove that it could be as reliable as a human driver, if not better.

1.5 OBJECTIVES

The main objectives of this project work are as follows:

- To develop a deep learning algorithm based on vision data to perform.
  - Vehicle Detection and Tracking
  - Lane Detection

  - Optical Flow Estimation

- To integrate the outputs of the above networks to infer semantic vehicle activity.
- To implement the inference algorithm on an standard embedded platform

1.6 CONTEXT AND MOTIVATION

There are certain conditions for which the final expected demo of the project will work. It involves some environmental constraints which are listed as follows:

- Paved roads
- Presence of lane markings
- Less traffic congestion

Currently, the systems deployed in autonomous cars use sensors such as RGB camera, LiDAR, RADAR, etc. for perception. Each of these sensors has its own advantages and limitations, but when used in the right combination, provides the best estimate of the vehicle's environment. However, humans can drive vehicles predominantly using our vision for perception. Artificial Neural Networks were inspired by the working of the brain of human beings and hence, even they can be trained in such a way that complete sensing is performed using RGB cameras alone. This is the upcoming trend as companies are trying to use only cameras for perception as other sensors such as LiDARs are costly too. Most companies working on autonomous vehicles - including Ford, GM Cruise, Uber and Waymo - think LiDAR is an essential part of the sensor suite whereas Tesla's vehicles don't have LiDAR and rely on radar, GPS, maps and other cameras and sensors. Researchers at Cornell University agree with this LiDAR-less approach. Using two inexpensive cameras on either side of a vehicle's windshield, Cornell researchers have discovered they can detect objects with nearly LiDAR's accuracy and at a fraction of the cost [9] .

The main objective of this project is to perform vehicle activity understanding using vision data alone which is acquired from a single RGB camera. Vehicle activity understanding, implies semantic understanding as obtaining metric data using just a single

camera is not accurate or feasible. The final output will be in terms of classes and their activities such as "car-oncoming-no change", "truck-forward-left to right", etc. This provides a rough estimate of vehicle activity and can be used to provide warnings in case there is a possibility of an accident. The motivation behind taking up this project is to prove the feasibility of a single camera for perception in autonomous driving.

1.7 PERSPECTIVE OF THE PROJECT

The project is a part of perception involved in the complete system of an autonomous vehicle. It acquires data from a sensor and passes it through different networks to obtain an output to understand semantically, the vehicle activity in terms of their position and velocity. These motion related parameters are observed in the image plane and the labels are identified based on their corresponding image plane components.

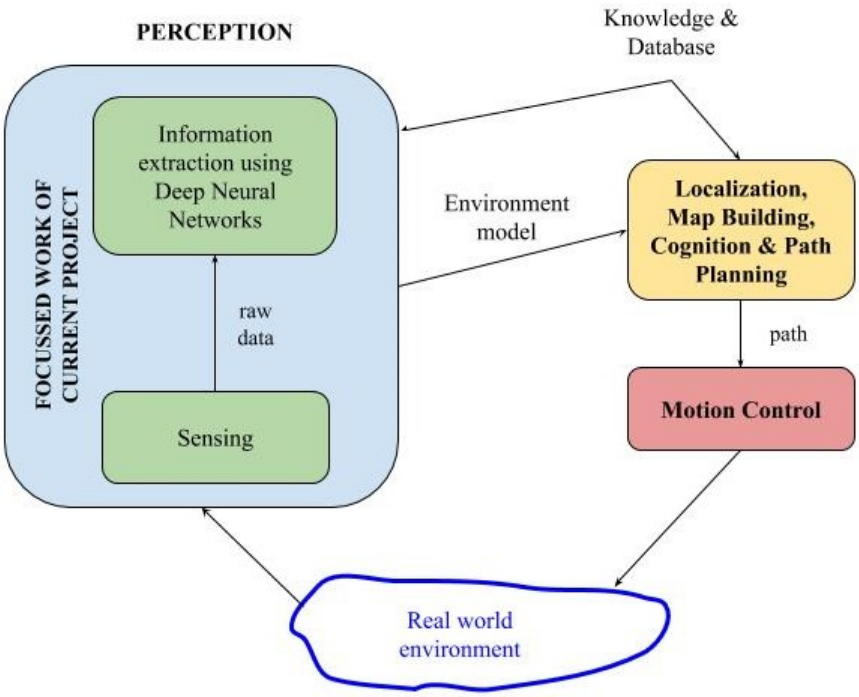


Figure 1.2: Illustration of Mechatronics Perspective

An autonomous vehicle is a complex system involving integration of various systems. System integration is synonymous with mechatronics. Hence, Mechatronics engineers have an edge over the rest in handling of data between units as the needs of the

cognition and actuation are taken into account while obtaining the output from perception unit Figure 1.2.

1.8 OVERVIEW OF THE PROJECT

Firstly, theoretical knowledge about deep learning had to be gained as it was not a part of the B. Tech. Mechatronics engineering curriculum during the course of study of the project members. Learning was an important process in this project. It continued with the course of the project as it is a vast field.

As soon as the main objectives of the project were decided, it was decided to use Deep learning for all the three tasks as those algorithms provided fast and accurate results. Literature survey was the immediate step as a lot of networks had to be analysed for each task and the best one for autonomous driving application had to be chosen.

The framework was also decided based on availability of online community support, embedded deployment capability, optimization of model, etc. The pre-trained networks of the chosen networks were obtained and inference was performed on PC for all the three networks.

The inference is made on an embedded board. Raspberry pi was chosen as the embedded board and Intel Neural Compute Stick-2 was added to support the inference process. Inferencing of the networks were performed on the embedded board.

1.9 ORGANISATION OF THE REPORT

This report consists of five key chapters and their contents are as follows :

Chapter 1 is Introduction which provides an introduction to autonomous driving and deep learning and shows the need for the project. The perspective of the project is also discussed along with the context and motivation for the project.

Chapter 2 is Literature Survey. All the literature referred for the project such as research articles etc. for the three tasks are presented in detail.

Chapter 3 is the Deep Learning Pipeline. The hardware and software tools setup for the training is mentioned. The three networks chosen are discussed in detail along with the network architecture, the reason for choosing them, etc.

Chapter 4 is about the Integration and Inferencing. The merging of outputs of the three networks and the process of extracting the final label for semantic vehicle activity has been discussed. The hardware and software setup for embedded inferencing is also mentioned.

Chapter 5 is about conclusion and future scope of the project.

CHAPTER 2

LITERATURE SURVEY

2.1 OBJECTIVE OF THE SURVEY

The most significant part of developing a deep learning project is literature survey. Building a deep neural network from scratch is difficult and it consumes a lot of time. Hence, it is better to refer to all the existing networks for the same application, shortlist a few and tweak them to work for the required application. This provides a base to work with and saves a lot of time to begin with. Hence, literature survey had to be done to choose the best network for all three tasks. The main objective of the survey is to find out the best network for each task in terms of accuracy, time complexity, space complexity, etc.

2.2 STRATEGIES FOR FILTERING

The performance metrics of many networks were compared to shortlist 2-3 best ones for each task. The main parameter for shortlisting was the runtime of the network for predicting output for one frame. Hardware-independent runtime is very crucial in this application since the final output has to be run on an embedded board. The second parameter was accuracy provided by the network.

Time vs accuracy trade-off is always present in every algorithms. But in autonomous driving scenarios real-time performance is crucial though accuracy has been given almost equal importance.

2.3 EVALUATION METRICS

The evaluation metrics which were used to compare the networks for each task are discussed in this section.

2.3.1 Intersection over Union

Intersection over Union (IoU) is used to measure the accuracy of an object detector on a dataset. It can be used as an evaluation metric for any algorithm that outputs bounding boxes.

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{2.1}$$

2.3.2 F1-measure

F1-measure (also F-score) is a measure of a test's accuracy. It considers both the precision *p* and the recall *r* of the test to compute the score: *p* is the number of correct positive results divided by the number of all positive results returned by the classifier, and *r* is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$F1measure = 2\frac{pr}{p+r} \tag{2.2}$$

2.3.3 Mean Average Precision

Average Precision (AP) is an evaluation metric in measuring the accuracy of object detectors like You Only Look Once (YOLO). AP computes the average of precision value for recall value over 0 to 1. This can be calculated by area under the curve drawn

from precision and recall. mAP is the average of AP.

$$AP = \int_0^1 p(r)dr \tag{2.3}$$

Here, *p* and *r* are precision and recall as mentioned in F1-measure 2.3.2.

2.3.4 Angular Point Error

Angular Point Error (APE) is the difference in angle between the correct and estimated flow vectors.

$$APE = cos^{-1}(c.e) \tag{2.4}$$

Here, *c* is the vector normalized correct motion vector and *e* is the vector normalized estimate optical flow vector.

2.3.5 End-to-end Point Error

End-to-end Point Error (EPE) is calculated by comparing an estimated optical flow vector (V_{est}) with a groundtruth optical flow vector (V_{gt}). EPE is defined as the Euclidean distance between these two:

$$EPE = |V_{est} - V_{gt}| \tag{2.5}$$

2.4 OBJECT DETECTION

The following deep learning networks were shortlisted after literature survey for object detection task. These three networks have been compared in Table 2.1 considering only the speed and accuracy as metrics. All three networks were trained with the same dataset MS COCO [34] and inference has been done on Titan X GPU as the candidate hardware.

Faster RCNN [33] uses a two-staged approach to first get region proposals and then classifying the regions with a secondary neural network. It is the most accurate with

Table 2.1: CNNs for Object Detection

Algorithm	mAP	Speed (fps)	Image Resolution
Faster RCNN [33]	34.9	5	600×600
SSD [37]	31.2	8	512×512
YOLO v3 [32]	33	20	608×608

the highest mAP, but considering speed, it is very slow, about 5 frames per second (fps) while 300 region proposals are considered. Though speed can be increased by decreasing region proposals and by reducing the image resolution but it can affect the overall accuracy of the algorithm.

Unlike Faster RCNN, Single Shot Multibox Detector (SSD) uses a single shot object detector to perform both classification and localization. Single shot object detector is quite similar to region proposal network used in Faster RCNN to find region proposals in the first step, but here object detectors are used to find the object classes as well as the bounding box coordinates. SSD uses default anchor boxes for regions in an image. More anchor boxes tend to give more accurate bounding box prediction but computation cost will be increased. SSD has good mean average precision (mAP) and speed, but is slower than YOLO (You Only Look Once) which proved to be the deciding factor in choosing YOLO.

YOLO v3 architecture is quite similar to SSD. In YOLO, propagated image uses DarkNet-53 as base network for classification and then YOLO head (a set of detection layers) to generate feature map. This feature map contains confidence scores, class labels as well as bounding box coordinates. Because of its single shot nature it gives real-time performance with very good accuracy which led to selection of YOLO for object detection. YOLO v3, a predecessor of YOLO [23], YOLO 9000 has shown better results in accuracy with a very small trade-off with speed.

There are other networks such as RetinaNet [35] with state-of-the-art mean average precision (mAP) results, but were not considered due to speed being the primary metric.

2.5 LANE DETECTION

Lane detection expands the horizon of ADAS applications vastly. There has been constant improvement in the field of lane detection with each year. Semantic segmentation is implemented for detecting lanes. Semantic image segmentation partitions an image into regions of meaningful objects. Fully convolutional networks proposed by Long et al. [22] from Berkeley, has an upper hand over fully connected layers in terms of computation power and produces accurate results.

One of the major problems in using CNN is addressing the "where" problem thrown by pooling layers. Pooling layers increase the field of view and aggregate the image information while discarding the "where" information. The encoder-decoder architecture was proposed to address this problem. Encoder gradually reduces the spatial dimensions and decoder reconstructs the image. There are shortcut connections between encoder and corresponding decoder layer. These connections pass on the information for the decoder layer to reconstruct the image to its initial spatial dimensions. One such network which is based on the above architecture is MultiNet by Marvin et al. [28], University of Toronto. It is an end-to-end joint inferencing network able to perform object detection, drivable path segmentation and classification parallelly albeit sharing a common encoding network. Since, lane detection is required for the application intended, this network requires several modifications to the architecture and training.

Spatial As Deep: Spatial CNN for Traffic Scene Understanding proposed by Xingang et al [16], a generalization of deep neural network to a rich spatial level. This network outperformed other algorithms in TuSimple lane detection challenge [10] and ended up securing first place. Unlike the previous model described above, this network offers message passing. Message passing is useful in cases where the object of interest is occluded partially. In message passing, the information from the surrounding pixels is sent to the target pixel. Message passing is realized in a sequential propagation scheme to make it a computationally efficient process. The published results of the network is mentioned in the Chapter 3. Since this network straight away gives the pixel locations of detected lane points and has the inbuilt capability to distinguish ego lane from the other lanes, this algorithm has been chosen for detecting the lanes for the intended

application.

2.6 OPTICAL FLOW ESTIMATION

Traditionally, optical flow algorithms are divided into derivative based matching, region based matching, energy based matching and phase based techniques. Horn and Schunck [17], Lucas-Kanade [27] and Nagel [21] are some algorithms which use derivative based methods. Anandan [30] employs region based matching for flow estimation. Lucas-Kanade has low computational cost and good noise tolerance but it produces sparse depth maps. Horn and Schnuck provided very good results with suitable approximations. The drawback with all these algorithms are that they are not real-time. Hence, it is not feasible to use them for an application like autonomous driving.

Optical flow was one of the few areas of computer vision which did not get largely influenced by deep learning since generation of ground truth is a time consuming and laborious task. Deep neural networks require ground truth for learning under the supervised learning framework. A ground truth here is the motion field of each and every pixel on the image which is difficult to generate. However, datasets like Middlebury [6], FlyingChairs [2] were created and the algorithms were evaluated based on them. These datasets have very limited images and are not suitable for all applications. MPI Sintel [8] dataset, created from a 3-D animated movie Sintel, became a standard for Optical Flow Evaluation since it encompassed a variety of movements at different scales. KITTI Stereo/Optical Flow Dataset [4] was created specifically for autonomous driving scenes. The 2012 version consists only static scenes. The 2015 version consists dynamic scenes too. The shortlisted algorithms were compared using the benchmark [5] provided by the same dataset.

FlowNetS and FlowNetC [13] were one of the first CNNs proposed to estimate Optical flow. It showed the feasibility of estimating optical flow from raw images. FlowNet2 [19] was a combination of FlowNetS and FlowNetC which runs much faster but requires a lot of memory making it unsuitable for embedded deployment. SpyNet [31] combines deep learning with classical principles but, performs a little slower compared to FlowNet2 due to the same reason. LiteFlowNet [12] is a smaller version of

FlowNet2 and it performs 1.36 times faster with a model size 30 times smaller.

PWC-Net [18] is a CNN for optical flow which does pyramid processing, warping and use of a cost volume. It is 17 times smaller in size compared to FlowNet2 model. It is the state-of-the-art network and is the best on MPI Sintel final pass [7] and KITTI 2015 benchmarks [5]. It runs at 35 fps on Sintel resolution (1024×436) images. Hence, it is the most suitable network for this application.

CHAPTER 3

DNN PIPELINE

Understanding of semantic vehicle activity from the scene requires inputs from three deep neural networks. The accuracy and robustness of the final output depends on the outputs provided by the networks. The networks have to be trained with appropriate datasets so that they can provide good results as inputs to the integration. The set-up of tools i.e. hardware and software is discussed in this section.

3.1 HARDWARE SETUP

Hardware aspect for the development of the project is limited to Personal Computer (PC). The intended application requires more cores for fast computation, hence a dedicated GPU has been chosen for reducing the computation time. CPU is chosen in a way that it is compatible with the chosen GPU. Based on the power requirement of both GPU and CPU, power supply unit is chosen. The specifications of the chosen CPU and GPU are mentioned in the Table 3.1 and Table 3.2

.

Table 3.1: CPU Specifications

Motherboard	ASUS A68HM-K
Processor	AMD A6-7400K Radeon R5, 6 Compute Cores 2C+4G
Frequency	3500 MHz
Datapath width	64-bit
System memory	DIMM DDR3 16GB (2x8GB), 600 MHz
Storage	Kingston A400 120GB SATA 3 2.5 Solid State Drive

Table 3.2: GPU Specifications

Number of CUDA cores	4352
Number of tensor cores	544
Single precision performance	13.4 TFLOPs
Memory	11GB GDDR6
Memory Speed	14 Gbps
Base Clock	1350 MHz
Boost Clock	1545 MHz

3.2 SOFTWARE SETUP

In setting up of software tools, the version number of packages and drivers plays an integral part. The required packages should be compatible with each other. The driver version should be compatible with the hardware model.

3.2.1 Operating System

Generally, most of the deep learning frameworks are designed to work on Linux first, followed by other operating systems. Ubuntu was chosen as the Operating System as it is an open source distribution of Linux based on Debian. The version chosen was 16.04 as support from the developer community was sufficiently available for all modules.

3.2.2 Drivers

The GPU driver is a way for the operating system to communicate with the graphics card. As mentioned in the hardware specifications, the GPU used is NVIDIA GeForce RTX 2080 Ti. The driver version is 418.40.04.

3.2.3 Compute Unified Device Architecture (CUDA)

CUDA is a parallel computing architecture provided by NVIDIA in its GPUs for general purpose computing. It enhances the computing ability of the GPU significantly by using the computing cores of the GPU parallely to increase the computation speed. The CUDA toolkit version is 9.2 which is compatible with the NVIDIA driver.

3.2.4 CUDA Deep Neural Network (cuDNN)

cudNN stands for CUDA deep neural network. It is a library which is specifically used for GPU performance tuning for a variety of deep learning frameworks. It enables the developer to work on a higher level API rather than work at low-level GPU tuning. cuDNN version installed is 7.3.1.

3.2.5 Anaconda

Anaconda is a package manager for python that provides tools such as Spyder, Jupyter Notebook, etc. for scientific computing. The advantage of using such a distribution is that it takes care of handling the compatibilities between different modules, provides a more interactive user interface to help with debugging on its tools such as Spyder. It is similar to that of MATLAB. The Anaconda version is 4.6.8.

3.2.6 Python

Python is widely known and preferred for its simplicity in syntax but the availability of extensive libraries and frameworks and community support in the area of Deep Learning makes it the best programming language to use for this project. The version of Python is 3.6.8.

3.2.7 Python Libraries

As mentioned earlier, Python has many libraries which provide a variety of functions. These Python libraries make the work of developers easier by providing complex functionalities and they are open source. Some of the more important libraries used are as follows.

3.2.7.1 Numpy

Numpy is a package used for scientific computing with python. It is a powerful tool for dealing with multi-dimensional arrays. As we deal with images and multi dimensional arrays, Numpy package is an absolute necessity. The version used is 1.15.4.

3.2.7.2 Tensorflow

Tensorflow is a machine learning end-to-end platform which allows to build and deploy machine learning and deep learning models in embedded platforms with ease. It provides both higher and lower level API which makes it easy for a beginner to start with. Python is one of the most compatible languages with tensorflow. It's high performance in terms of speed and reduced model size. Tensorflow is the best choice among deep learning framework for working with Deep Neural Network (DNN)s and deploying them on embedded platforms. The version of Tensorflow is 1.12.

3.2.7.3 Open source Computer Vision Library (OpenCV)

OpenCV is a computer vision library used to accelerate the use of computer vision applications with ease. Having many functionalities in OpenCV makes the task easier and productive. The version of OpenCV is 4.0.0.

3.3 PROPOSED PIPELINE

Understanding of semantic vehicle activity required information about the objects in the scene, their motion and the information about the road itself. This information should also be temporal (related across frames) which is why tracking plays an important role too. The object detection network gives the classes of the detected objects and provides bounding boxes to the tracking algorithm as input. The tracking algorithm uses these bounding box coordinates to track these objects across frames. The lane detection algorithm detects all the visible lane markings on the scene in a distinct manner such that the ego lane can be detected in every frame. Optical flow estimation provides

the flow vectors of each pixel in the frame as input to manual integration. An abstract model of the proposed pipeline is given in Figure 3.1.

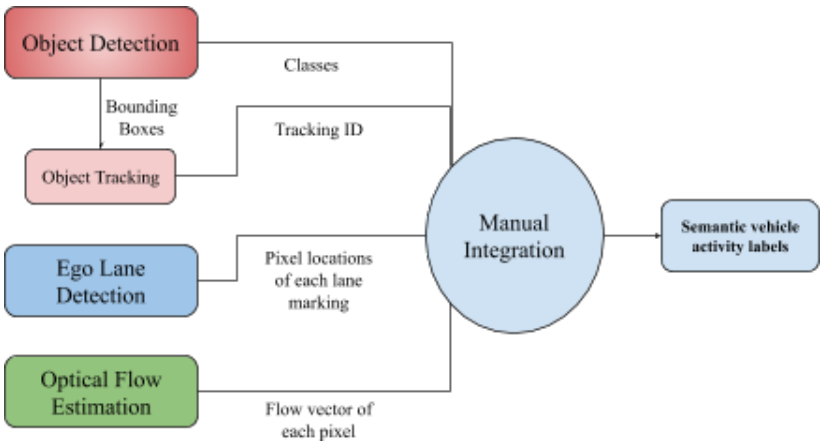


Figure 3.1: Proposed Pipeline

3.4 OBJECT DETECTION AND TRACKING

3.4.1 Problem Definition

Semantic understanding of vehicle activity demands identification of the vehicles in videos. Object detection can be used to detect objects in each frame and tracking can be done to associate the data acquired from object detection along all the frames.

A tracking by detection approach has been proposed to do object detection and tracking as two different tasks. Object detection involves classification and localization of objects in an image. Tracking algorithm tracks multiple objects across all the frames.

3.4.2 Need for the Task

In autonomous driving, scene understanding is a very important task and cannot be done without semantic information. Object detection is an important task in scene perception and it is done across all frames for the following reasons:

- To classify the objects present in the set of pixel areas.
- To localize the objects using a bounding box in an image.

Tracking is essential for the following reasons:

- In order to associate the information acquired from object detection from all frames and get a tracking ID for each object.
- To track the objects along successive frames until they exit the field of view of camera.
- To mitigate the effect of identity switches caused by occlusion, motion blur, lighting conditions

Object detection and tracking gives the bounding box information of object along all the frames with tracking ID.

3.4.3 Preamble of the Networks Used

3.4.3.1 Detection algorithm

Object detection can be done by many ways such as 2-dimensional (2-D) bounding box, 3-D bounding box and instance segmentation methods. In bounding box methods generally a regression-based CNN is used to get the bounding box coordinates. But instance segmentation method uses a pixel level segmentation approach to classify object pixels.

The introduction of CNN, AlexNet opened up a huge research space of semantic understanding in computer vision. But the first object detection approach Region based Convolution Neural Network (RCNN) was published in 2013, which does object detection by two stages. In the first stage it generates region proposals using a regression network, and in the later stage each region of the image subjects to a smaller CNN to classify the object. RCNN was the stepping stone to many object detection approaches. Later in 2015 a new single-staged approach YOLO [23] was proposed, which was fast and accurate. Over the years YOLO has been improved significantly and YOLOv3 [32] is being used in this application.

3.4.3.2 Tracking algorithm

Tracking algorithms have been around for decades, but after the introduction of deep learning they have been improved significantly. Simple Online Real-time Tracking (SORT) [14] was one of the best Multiple Object Tracking (MOT) algorithm with very good speed, but DeepSORT [29], which uses a small CNN to incorporate appearance information using a pre-trained association metric to handle long term occlusion very well.

3.4.4 Positive Attributes

The approach of tracking by detection shows us significant improvement over the pure detection method by reducing False Positives (FP) and False Negatives (FN). The tracking algorithm is very fast, which shows no overall reduce in speed of tracking by detection approach.

3.4.5 Published Results

YOLOv3 has been compared with few state-of-the-art detectors such as SSD, RetinaNet considering mAP and inference time as evaluation metrics. The comparison has been done with Titan X GPU as the candidate hardware as shown in Table 3.3.

Table 3.3: Performance of CNNs for Object Detection

CNN	mAP	Inference time (ms)
SSD	28.0	61
R-FCN	29.9	85
RetinaNet-50-100	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLO v3-320	28.2	**22**
YOLO v3-416	31.0	29
YOLO v3-608	33.0	51

Though the accuracy of RetinaNet [35] is slightly better than YOLO, YOLO surpasses RetinaNet's inference speed with ease.

3.4.6 YOLO Network Architecture

YOLO is a single shot object detector, which takes image as an input and gives a tensor containing all bounding box information, classes and confidence scores.

To understand YOLO network architecture [Figure 3.3], it can be divided into two parts such as YOLO body and YOLO head. First part of the architecture is used to extract features, which is considered base network. In YOLOv3, Darknet-53 has been used as base network, also called as YOLO body, to extract features from the input data. Then the extracted features will be subjected to YOLO head, which does the detection of objects. This detection involves the localization and classification of objects.

YOLO body (DarkNet-53) extracts features from an image in three different scales for detecting small, medium and large objects accurately, which will be fed into YOLO head.

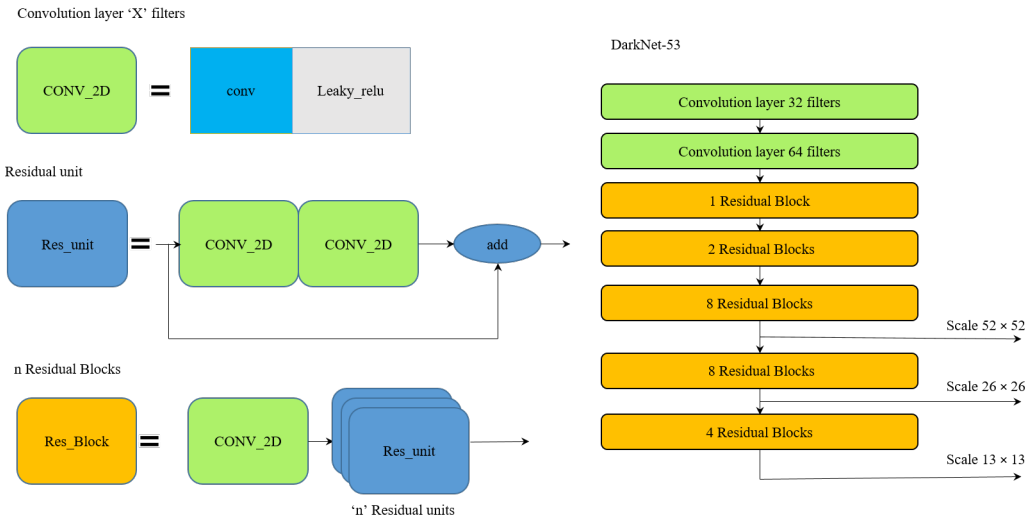


Figure 3.2: DarkNet-53 Architecture

To simplify DarkNet-53 architecture, it has been divided into CONV2D blocks and Residual blocks as shown in Figure 3.2.

Each CONV2D block consists of a convolution layer along with a leaky ReLU layer as in Figure 3.2. Residual blocks (Res-Block as per the Figure 3.2) are made of one

CONV2D block followed by n residual units. Here, n defines the number of residual blocks. Each Residual unit can be made by two CONV2D blocks stacked together and added with the input itself.

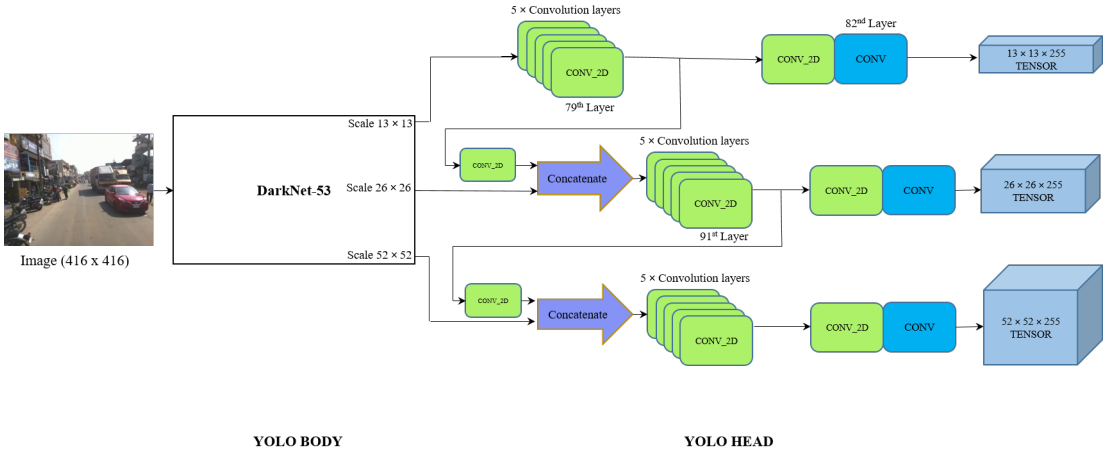


Figure 3.3: YOLO v3 Architecture

From the YOLO body three different scales of output (in our case 13 × 13, 26 × 26, 52 × 52) will be found. At 82nd layer first detection is made. 81st layer has a stride of 32, which makes the input to downsample to the 13 × 13 (considering input image size is 416 × 416). Output feature map will be of 13 × 13 . This will be fed into convolution layer of 1 × 1 × (B × (5 + C)) kernel to get output tensor of size 13 × 13 × 255. Where B is the number of bounding boxes a grid cell on feature map can predict, C number of output classes. For pretrained weights this is 1 × 1 × 255 because of values of B and C. Here, B represents the number of anchor boxes, which is 3, C represents the COCO number of classes, which is 80, and, 5 represents bounding box information and predicted score, which is constant.

Feature map from 79^{th} layer will be subjected to a convolution layer and then concatenated with medium scale output of YOLO body. The output will be subjected to a few convolution layers again and then downsampled to 26 × 26 followed by a convolution layer of 1 × 1 × 255 kernel to get the output tensor of size 26 × 26 × 255.

A similar procedure takes place for small scale also, but feature map will be from 91^{st} layer, at last feature map will be downsampled to 52 × 52. Output tensor size will be 52 × 52 × 255.

It can be inferred from the above that YOLOv3 downsamples input image by the scale of 32, 16 and 8 respectively. One more point to note is functions such as softmax are not used in YOLO, as the output tensors contain all the necessary information.

YOLOv3 uses a total of 9 anchor boxes, 3 for each scale. To use BDD dataset [20], new anchor boxes can be generated using k-means clustering.

Input image is converted into S $\times$ S grid cells (S defines image size / scale). Each grid cell is responsible for predicting the image when the object centre lies on the grid cell. Each grid cell predicts more than one bounding box for the same object with the help of the predefined anchor boxes, later the optimal bounding box can be found using IoU.

YOLOv3 generates 3 different scales of output tensors containing the bounding boxes, confidence scores and class names. This output tensor will be subjected to few post-processing steps such as IoU and Non Max Suppression (NMS) to remove the unnecessary bounding boxes. Bounding boxes, class names and confidence scores will be found from object detection. Bounding boxes from object detection will be used for tracking.

3.4.7 Deep SORT Algorithm

Deep SORT (Simple Online Real-time Tracking with a deep association metric) [29] is an improved version of SORT [14], is used as a tracker in our tracking by detection approach. It uses conventional vision algorithms to do tracking but by adding deep association metric long term occlusions can be sustained. It is generally assumed the noise to be present in our input data and camera to be uncalibrated. The Deep SORT pipeline is shown in the Figure 3.4.

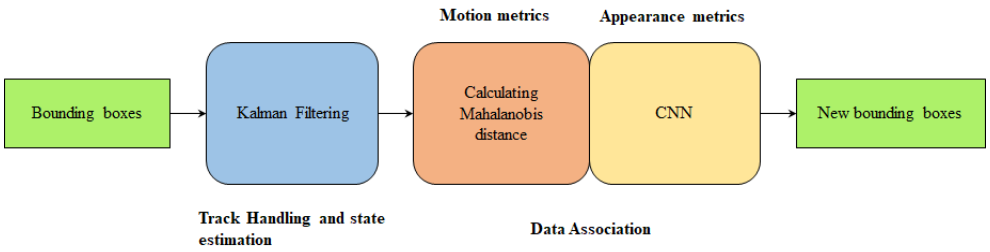


Figure 3.4: Deep SORT Pipeline

3.4.7.1 Track handling and state estimation

An eight dimensional state space model as *(u, v, r, h, u', v', r', h')* is defined, where *u, v* are bounding box centers, *r* is aspect ratio, *h* is height of the box and *(u', v', r', h')* are their velocities in image space. Kalman filter is used to solve the velocity components. For each track *k*, number of frames since the last successful association a_k is counted. This counter will be incremented for each Kalman filter prediction. When tracks exceed their predefined age A_{max} track will be deleted. The algorithm requires at least three frames to successfully associate to a measurement.

3.4.7.2 Data association

SORT Hungarian algorithm is used to solve the assignment problem between predicted Kalman states and newly arrived measurements. But in Deep SORT, motion and appearance information are incorporated using combination of two appropriate metrics:

- For motion information, Mahalanobis distance is calculated between predicted Kalman state and new state.
- For appearance information, a CNN is used to find bounding box appearance descriptors. The architecture of network is described in Figure 3.5. The above mentioned informations will be combined using a weighted sum to build the association problem as shown in Figure 3.4.

This CNN has been trained on a large-scale person re-identification dataset. Though it is trained to track pedestrians it gives decent performance boost over SORT in our driving case scenario. We can train it to improve the accuracy for autonomous driving cases.

Architecture of CNN in Figure 3.5 is a wide residual network starts with two convolution layers followed by six residual blocks. Then, a dense layer is added to reduce the dimensionality to 128. A final batch and L2 normalization converts the feature map of dimension 128 to unit hypersphere to be compatible with our appearance metric.

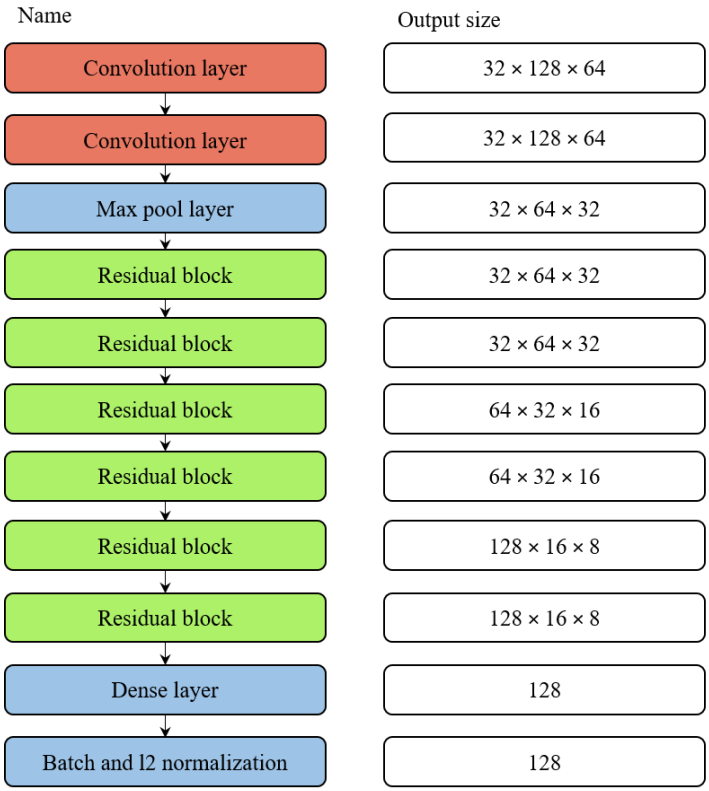


Figure 3.5: CNN Architecture for Appearance Metric

3.4.8 Integration of Detection and Tracking

In tracking-by-detection approach, bounding boxes and tracking IDs are taken from tracking output, but tracking algorithm has generally no influence on class labels. So output class labels have to be fetched from the YOLO output. Number of bounding boxes from YOLO and from tracking will not always be same, as tracking takes care of FPs and FNs . The association between class names and bounding boxes will be a challenge. To mitigate this problem, storing of all class labels will be necessary. For each track, the first frame in which it is being tracked is found and class labels are taken from the stored list. This class label from the first tracked frame will be passed to all consecutive frames until the object is lost.

3.4.9 Results

The results of the object detection using YOLO have been shown in Figure 3.6. On the top of the bounding box, class is shown. Different colors are assigned for various

classes.

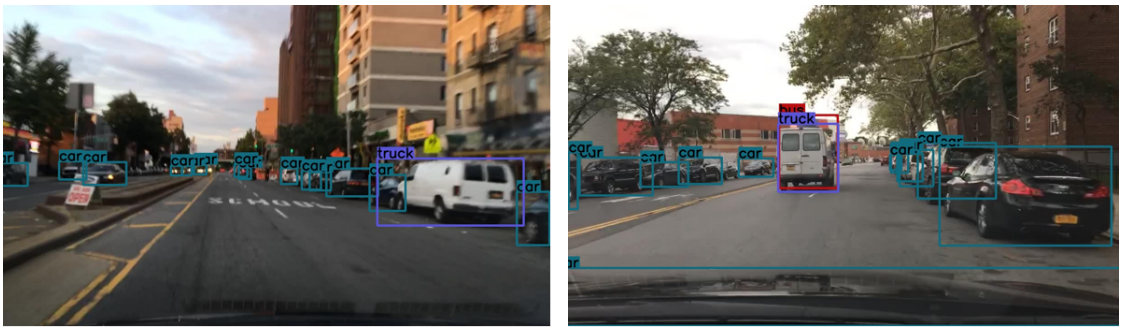


Figure 3.6: Output of YOLO

Tracking ID and bounding boxes are generated from tracking. The results of only tracking are shown in Figure 3.7.

Figure 3.7: Output of Deep SORT

Results of both tracking and object detection are overlapped in Figure 3.8. Blue box represents the bounding box generated by YOLO while white bounding box is from the tracking algorithm.

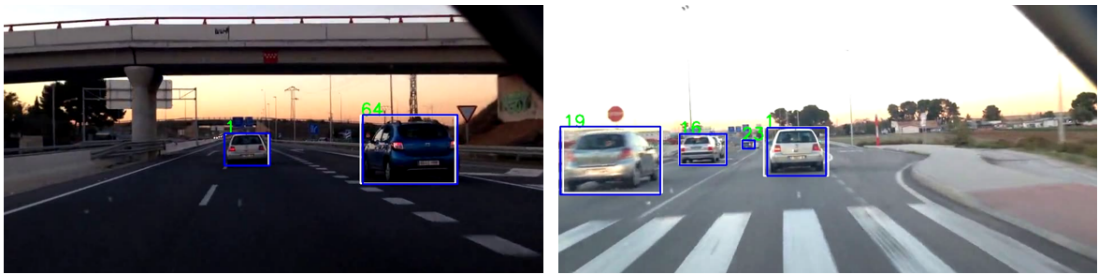


Figure 3.8: Output of YOLO and Deep SORT

After integration between tracking and detection labels will be shown in Figure 3.9. Label contains the class label from YOLO and tracking ID and bounding box from tracker.

Figure 3.9: Integration of Object Detection and Tracking Outputs

Information to be passed into integration algorithm will be bounding boxes, tracking IDs and class labels.

3.5 LANE DETECTION

3.5.1 Problem Definition

Deep learning architecture should be able to detect all the lanes and be able to differentiate ego lane from the detected lanes despite partial occlusion. It should be able to predict the precise lane curve.

3.5.2 Need for the Task

The interest to develop lane detection solutions increased with the demand for ADAS and self driving cars. Drivers not only depend on the lanes for safe driving but also for visual cues (e.g., pavement markings) to understand what it is and what is not allowed (e.g., lane change,direction change). The integration part discussed in the later section of the report assumes the lane as an object of reference for detecting ego motion, i.e motion of the camera on board vehicle, which will be discussed elaborately in Chapter 4. Lane detection plays an important part in vehicle lane change activity analysis.

3.5.3 Preamble of the Network Used

To compute spatial relationships, traditionally Markov Random Fields and Conditional Random Fields were used. Message passing, is another spatial relationship computation process wherein each pixel gets information from the pixels around it. It is computationally expensive and harder to be implemented in real time. Generally, these methods are applied to the output of the CNN models. The top hidden layer comprises of rich information which could be a better place for placing the spatial relationship model.The Spatial Convolutional Neural Network (SCNN) proposed by Xingang et al [16] offers better run time and spatial relationship model runs over information rich top layer.

3.5.4 Positive Attributes

The positive attributes of SCNN network are as follows :

- The adopted network has the capability to detect the lanes despite partial occlusion of lanes.
- SCNN is computationally efficient where message passing is realized in a sequential propagation scheme rather than each pixel receiving information from the pixels around it.
- It shows good ability to predict fine lane curves and offers good balance between speed (fps) and accuracy.
- It is capable of detecting upto four lanes and differentiates ego lane from the others.
- It gives output in the form of pixel coordinate location of the detected lanes points which is easier for integration.

3.5.5 Published Performance Results

The SCNN was tested on three different testing sets which are TuSimple dataset [11], CULane [1] and BDD100K [20]. The results of SCNN (Library-Torch) network, based on ResNet-101 [24], tested on TuSimple dataset is given in the Table 3.4. The

results of SCNN (Library-Tensorflow/Torch), based on VGG-16 [25], tested on CULane and BDD100k testing test are given in the Table 3.5 and Table 3.6 respectively.

Table 3.4: TuSimple Dataset Results

Model	Accuracy	FP	FN
SCNN (Library-Torch)	96.53%	0.0617	0.0180

Table 3.5: CULane Testing Set Results

Category	SCNN (Library-Torch) F1-measure	SCNN(Library-Tensorflow) F1-measure
Normal	90.6	90.2
Crowded	69.7	71.9
Night	66.1	64.6
No line	43.4	45.8
Shadow	66.9	73.8
Arrow	84.1	83.8
Dazzle light	58.5	59.5
Curve	64.4	63.4
Crossroad	1990	4137
Total	71.6	71.3

Table 3.6: BDD100K Dataset Results

Model	Accuracy	IoU
SCNN-Torch	35.79%	15.84

3.5.6 Network Architecture

SCNN views rows or columns of feature maps as layers and applies convolution, nonlinear activation, and sum operations sequentially, which forms a deep neural network. This makes it possible for the information to be passed between the neurons in the same layer. The word spatial in SCNN, denotes propagating spatial information via specific CNN structure design.

The architecture of the adopted network is shown in the Figure 3.10. The network resizes the input image size to 800×288 by linear interpolation function using OpenCV library and sends it as a 3-D tensor input of size $C \times H \times W$, where *C, H,* and *W* denote the number of channel, rows and columns respectively. The input tensor is then passed to the first 13 layers of VGG16 and the weights are initialized accordingly from VGG16

model. Followed by atrous convolution of rate 4 which strikes a good balance between efficiency and accuracy [26]. Then fast bilinear interpolation by an additional factor of 8 is done to recover the feature maps at the original resolution [26]. The probability maps from softmax layer is passed over to another small network to predict existence of lane markings. For lanes with more than 0.5 for lane existence value, the network searches every row in the corresponding probability map for the pixel locations with highest response and these locations are then connected by cubic splines. The detected lane pixel coordinates is then overlaid over the actual input image with different color codes for the four lanes. The region between detected blue lane and detected green lane is the ego lane. The metric evaluation of the network is not within the scope of this project.

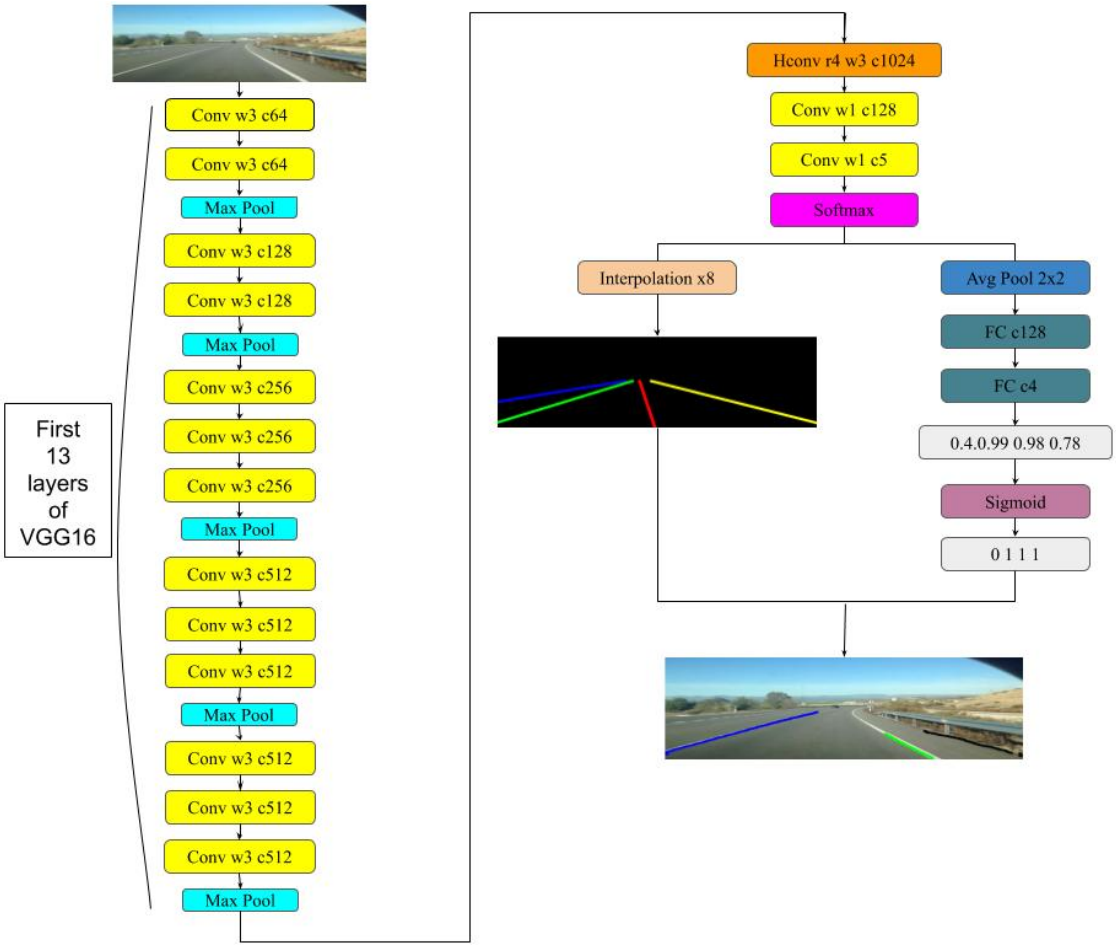


Figure 3.10: SCNN Lane Detection Architecture

3.5.7 Non Destructive Overlay for Visualization

SCNN gives output in list format comprising of pixel location of the detected lanes. For visualization using OpenCV library function *cv2.circle*, circles of radius 5 pixels were plotted over the detected lane pixel locations in the input frame. As the number of detected lane pixels increases, circles starts to overlap with each other and it appears as a line plotted over the lane.

3.5.8 Results

The SCNN algorithm was run on the videosets downloaded from the internet. The ego lane is the path between blue and green detected lanes. The algorithm performed impressively considering the ability to detect lanes despite partial occlusion and different lighting conditions. Figure 3.11, Figure 3.12, Figure 3.13 and Figure 3.14 shows detection results of different lanes subjected to different conditions. There were a few frames with unsatisfactory results Figure 3.15 in which two different lane colors were overlaid on the same lane. When the separation between the dashed lane increases, the algorithm performs unreliably. Also, eccentricity of the lanes has a huge impact in the lane detection.

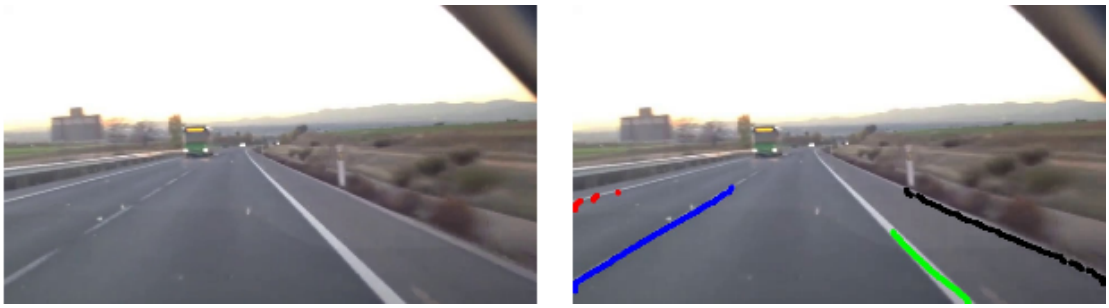


Figure 3.11: SCNN Result for Straight Lanes

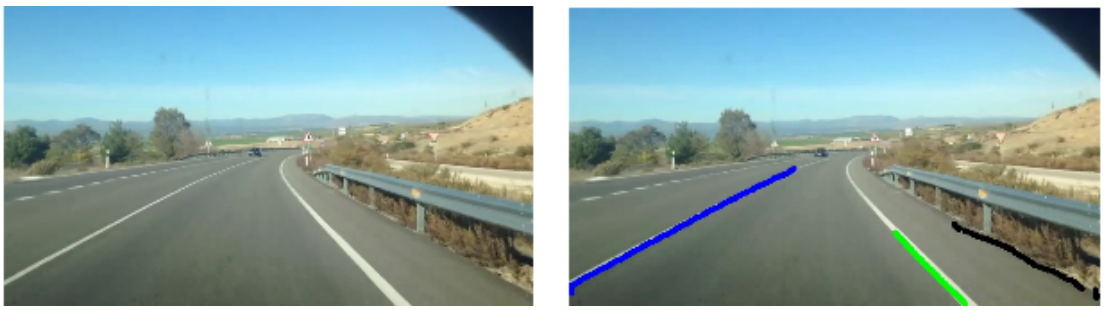


Figure 3.12: SCNN Result for Curved Lanes

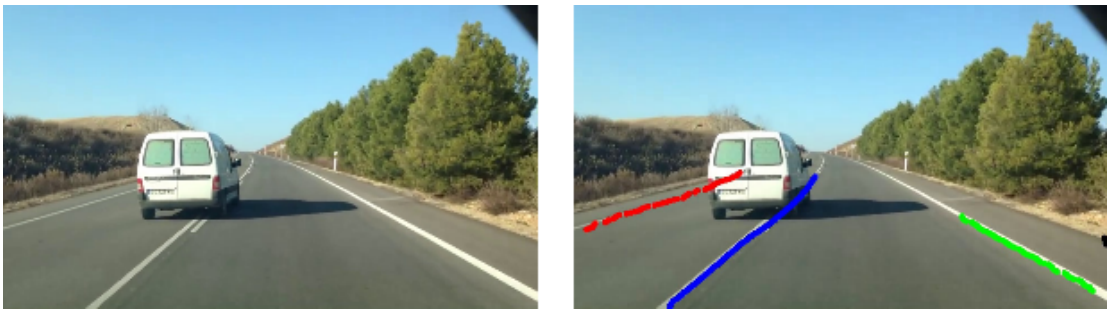


Figure 3.13: SCNN Result for Partially Occluded Lanes

Figure 3.14: SCNN Result for Lanes Under Shadow

Figure 3.15: Poor Results from SCNN

3.6 OPTICAL FLOW ESTIMATION

3.6.1 Problem Definition

Optical flow is the movement of the brightness patterns across frames caused due to the apparent movement of objects on the real world. The optical flow vector of every pixel in every frame of the acquired video has to be determined by the network.

3.6.2 Need for the Task

Each pixel will have a flow vector will have two components along X and Y directions which are nothing but projections of the actual 3-D flow of the objects on the image. Estimation of this flow is essential as we can extract information about the motion of every object on the scene using this information. The motion of the ego vehicle can also be determined using the static objects (known beforehand) on the scene. This will be explained in detail in Chapter 4.

3.6.3 Preamble of the Network Used

Traditionally, optical flow estimation has always been a problem which has been solved using image processing techniques. It was one of the areas of computer vision where deep learning could not make a quick impact. It was largely due to the lack of availability of ground truth. Manual labelling was a laborious and time consuming task as it involved labelling every pixel motion.

Optimizing a complex energy function was the approach used by many of the traditional algorithms but it was computationally expensive. It assumed brightness constancy and spatial smoothness constraints to predict the optical flow. CNNs were initially used as a component in the algorithms to perform tasks such as sparse to dense interpolation, construction of cost volume and sparse matching. The most recent methods used cost volumes, pyramid creation and warping methods but they were not real-time.

3.6.4 Positive Attributes

The positive attributes of PWC-Net are as follows :

- PWC-Net model has computationally light CNN layers, cost volumes and warping compared to energy minimization approaches.
- It constructs only partial cost volume making it more memory and computation efficient.
- It used feature pyramids instead of image pyramids making it invariant to shadows and lighting changes.
- It combines deep learning with domain knowledge to reduce model size as well as improve performance.

3.6.5 Published Results

Since its arrival, the PWC-Net has been the state-of-the-art network for optical flow estimation. It is 17 times smaller in size compared to the second best network and twice as faster. It runs at 35 fps for the Sintel resolution (1024×436) images. It is the top rated network on the KITTI 2015 benchmark as shown in Table 3.7.

Table 3.7: KITTI 2015 Benchmark Results for Optical Flow

	Non-occluded pixels			All pixels		
Method	Flo-bcg	Flo-frg	Flo-all	Flo-bcg	Flo-frg	Flo-all
PWC-Net	6.14%	5.98%	6.12%	9.66%	9.31%	9.60%

Flo - % of optical flow outliers
bcg - % of outliers averaged only over background regions
frg - % of outliers averaged only over foreground regions
all - % of outliers averaged over all ground truth pixels

3.6.6 Network Architecture

PWC stands for pyramidal processing, warping and cost volume. The method has been designed based on these simple principles. The method can be divided into five major parts - feature pyramid extractor, warping layer, cost volume layer, optical flow estimator and context network.

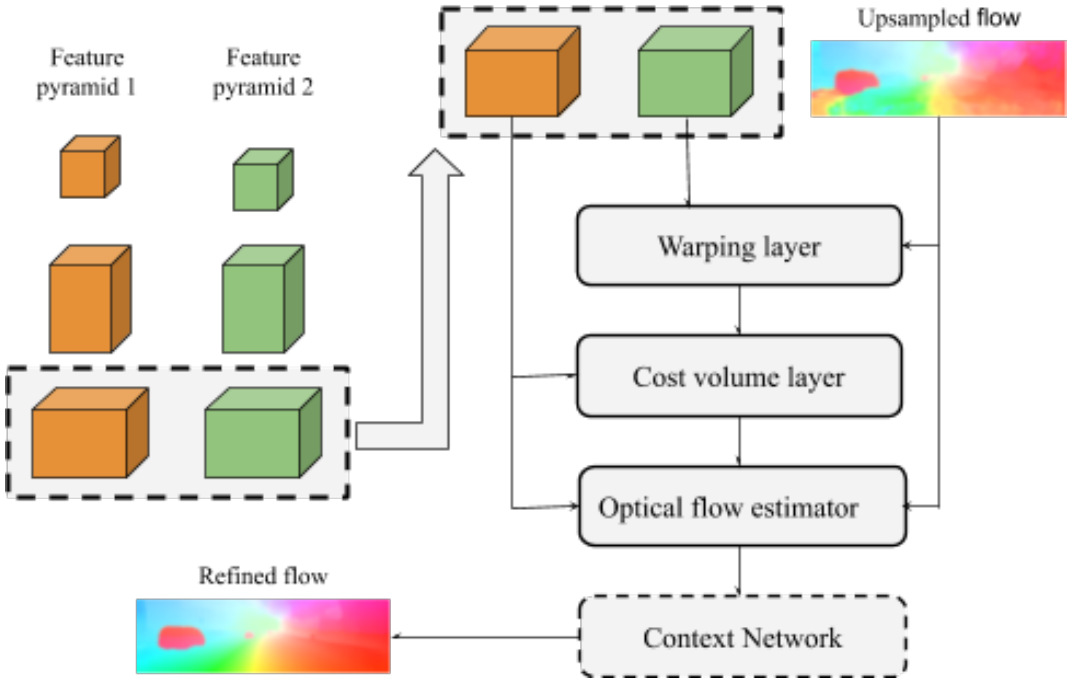


Figure 3.16: PWC-Net Architecture

3.6.6.1 Feature pyramid extractor

Two consecutive frames are taken as the two images and n-level pyramids of feature representations are created. The input images are the bottommost images. Layers of convolutional filters are used to downsample the features at each pyramid level by 2. With each level, the number of feature channels keep on doubling starting from 16 for the first level to 196 for the sixth level. Siamese network is used to encode the frames. Leaky ReLU is used as the activation function after every convolutional layer.

3.6.6.2 Warping layer

At nth level, bilinear interpolation is used to warp the features of the second frame on the first frame using *x2* upsampled flow from the $n+1^{th}$ level. The flow for back-propagation and gradients to CNN features'inputs are computed.

3.6.6.3 Cost volume layer

Cost volume defines the range of search for corresponding features between the consecutive frames. It stores the costs for matching them appropriately. It is defined as the correlation between the first frame and warped features of second frame. An important thing to note is that the motion at the topmost level of the pyramid amplifies with the decrease in level. A small motion at the top might amplify, to become a significant motion at the actual resolution.

3.6.6.4 Optical flow estimator

The optical flow estimator is a CNN on its own. Its architecture is shown in Figure 3.17. It is fixed at second level of the pyramid. In this network too, Leaky ReLU is used as the activation function that follows the convolutional layer. The cost volume, first image's features and upsampled optical flow are the inputs of the network.

The number of feature channels keeps on reducing from 128 to 32 with the layers. The final layer does not have any activation function as it provides the output i.e. optical

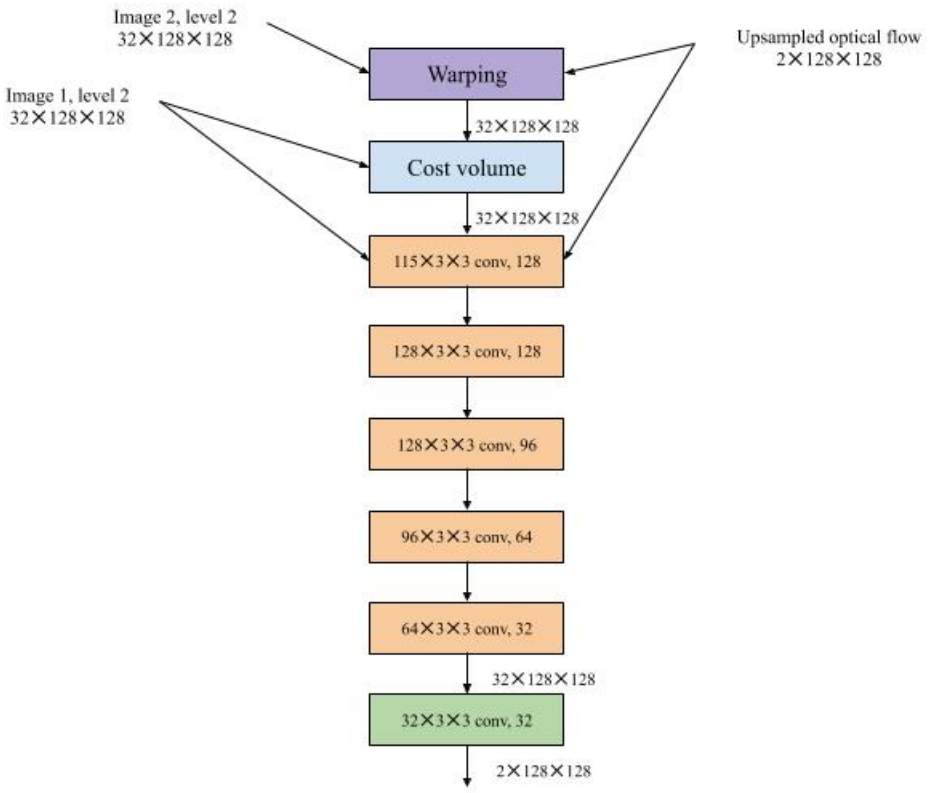


Figure 3.17: Optical Flow Estimator Network

flow at the n^{th} level.

3.6.6.5 Context network

The context network is used to post process the flow [Figure 3.18]. This is also applied at the second pyramid level and a leaky ReLU follows each convolutional layer. Its inputs are the estimated optical flow and the penultimate layer's features from the optical flow estimator network.

The dilation constants given at the end handles the separation between the input units in horizontal and vertical directions. The output of the context network is the refined optical flow.

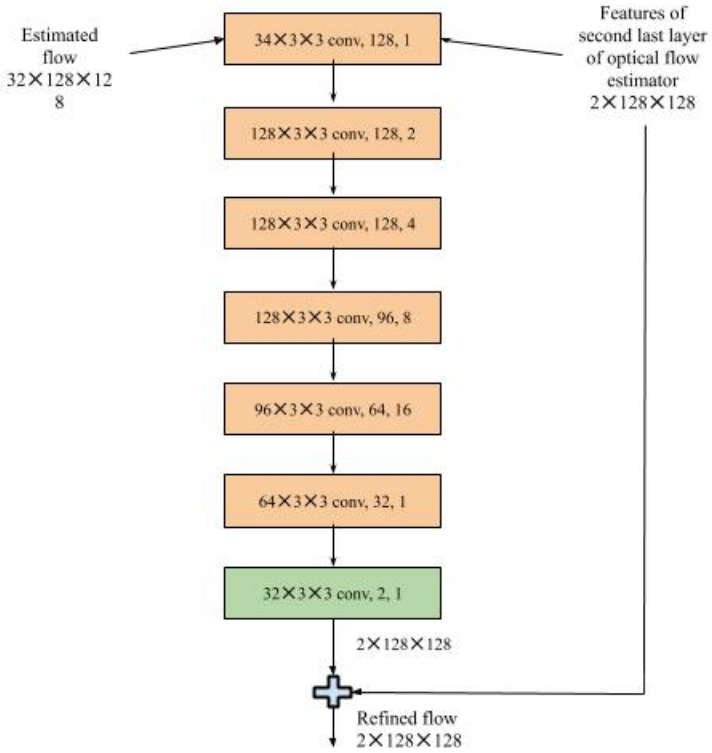


Figure 3.18: Context Network

3.6.7 Results

Each pixel will have a 2-D vector which can be converted to polar coordinates for visualization. The visualization is colour coded in such a manner that hue represents direction and saturation represents magnitude of the flow vector as shown in the Figure 3.19. The optical flow output for a sample frame is shown in the Figure 3.20

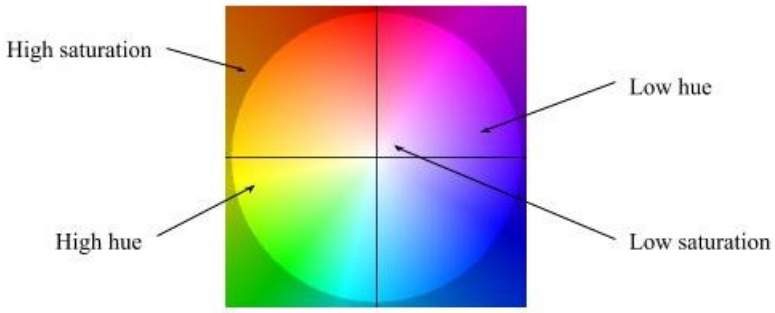


Figure 3.19: Colour Coding for Optical Flow Visualization

Figure 3.20: Sample Frame and its Optical Flow Output from PWC-Net

CHAPTER 4

INTEGRATION AND INFERENCE

Object detection and tracking, lane detection and optical flow estimation cannot provide meaningful data on their own separately. Their outputs have to be made use of to extract meaningful data about vehicle activity and acquire the final labels. This integration of outputs from all the three aforementioned networks majorly involves the use of image processing techniques and pixel level processing.

Metrics involves quantifying the various numerical parameters such as distance, speed, etc. but semantics involves just qualitative labelling. Semantics can just provide information about whether a change is happening or not. It cannot provide how fast or how slow the change is happening as it would all be relative. The work carried out in this project involves extracting semantic labels for vehicle activity as metric labels are not possible to acquire using the data from just a single camera. There are various techniques to estimate the depth, get metrics about distance of objects on the scene, etc. but they are not reliable and robust to be performed using just a single camera. Moreover, the recent trend in autonomous driving is to do more with just a single camera. Autonomous vehicle makers such as Tesla are aspiring towards it. Hence, the labels given for every vehicle would be semantic and the details are discussed in Figure 4.1.

Figure 4.1 shows how the final label for each vehicle is obtained. The final label consists of two parts.

- The first label describes the motion of the ego vehicle itself, if it is at rest or moving and is displayed at the bottom of each frame
- The second label is for each and every vehicle and follows the following format: *class name - tracking ID - along the road - across the road*

The class name of the object is the first part of the label which is obtained from YOLOv3. The tracking ID for the corresponding object is obtained from DeepSORT and appended to the class name. The labels to describe vehicle motion along the road and across the road are appended at the end.

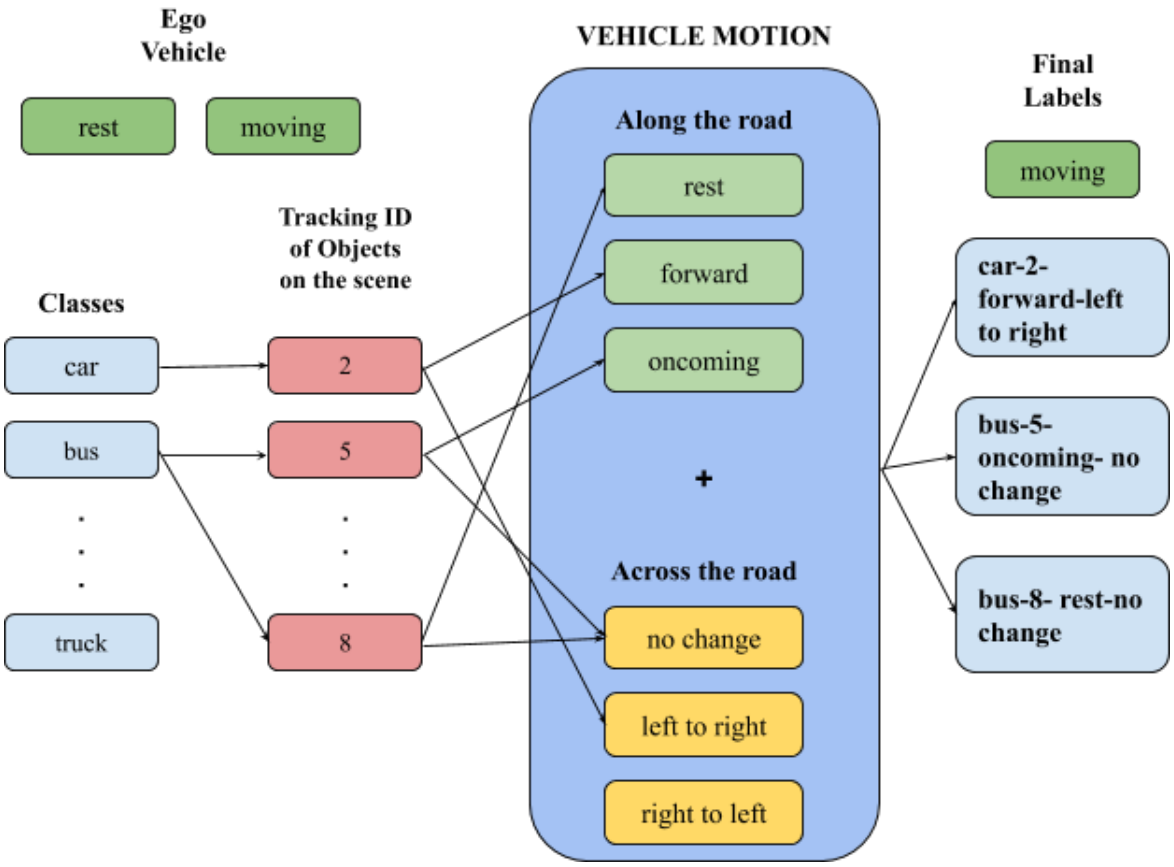


Figure 4.1: Final Output Labels

4.1 VEHICLE MOTION ALONG THE ROAD

The third part of the final label for each vehicle is the motion of the vehicle along the road. This is an important part where the results from optical flow matters a lot. The decision tree for prediction of vehicle motion along the road is shown in Figure 4.2.

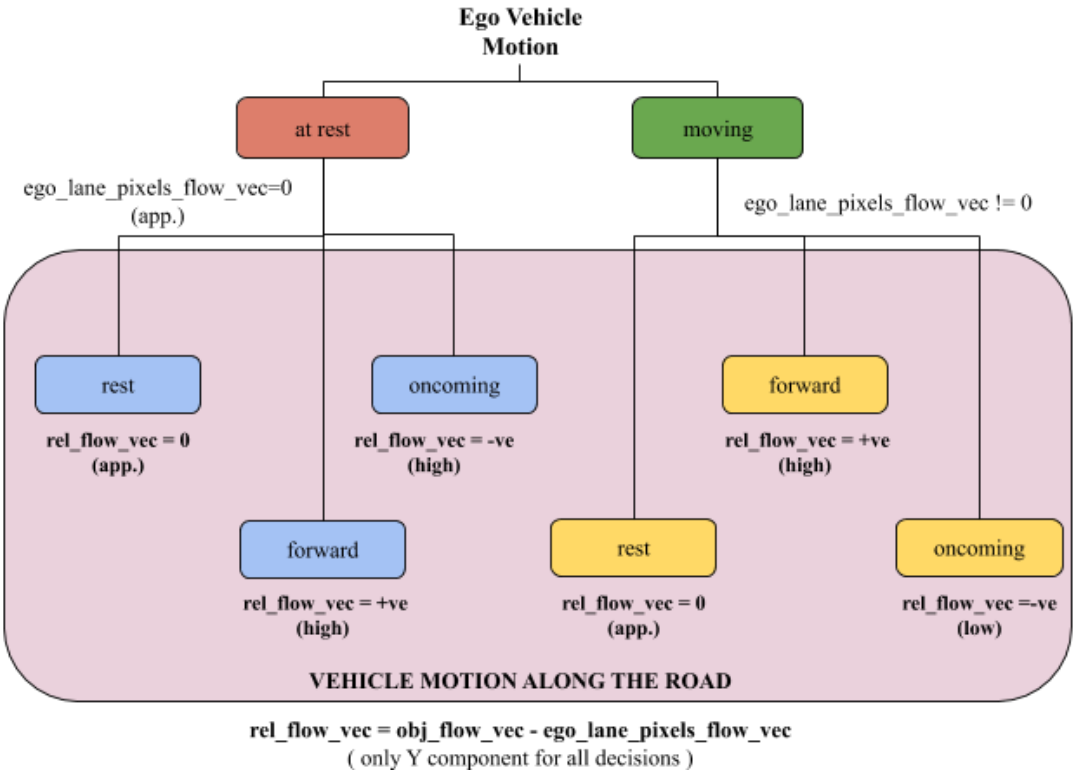


Figure 4.2: Decision Tree for Prediction of Vehicle Motion Along the Road

4.1.1 Reference Frame

First of all, to define any motion, a reference frame has to be set. In this case, the lane markings on the road are considered as the reference for all motion as they are stationary. The ego vehicle motion and motion of the vehicles on the scene are predicted with reference to these lane markings.

4.1.2 Decision Tree Level 0

If the ego vehicle is at rest, the ego lane markings detected by the lane marking detection network will also be stationary. Hence, their flow vector will have zero magnitude. If the vehicle is at motion, the lane markings will have a considerable amount of flow in the negative Y direction.

4.1.3 Decision Tree Level 1

Once the ego vehicle motion is decided, there are three possibilities of motion for the vehicle on scene in each case - rest, forward and oncoming. For this level, a relative optical flow vector is obtained where the Y component of the flow vector of the ego lane pixels are subtracted from the Y component of the flow vector of the object.

Note that in both the levels of the decision tree, the flow vector average of only the ego lane markings are considered as the other lanes are too eccentric and reduce the average magnitude of the lane flow vector.

The relative flow vector obtained is used to decide the motion of the vehicle. It can be almost equal to zero, have a positive value (high or low) or a negative value (high or low) depending on the cases presented in Figure 4.2

4.2 VEHICLE MOTION ACROSS THE ROAD

The fourth and the final label for each vehicle describes its motion across the lane. Due to perspective projection, even if a vehicle moves far away from the ego vehicle without changing lane, still there would be a component of optical flow along the X direction. Hence, using optical flow for this case proves to be ambiguous. Hence, a different approach is taken here given by the following Figure 4.4.

Two conditions have to be satisfied to provide the lane change label. The point of intersection of the lane marking and the bounding box of the object should be between the left and right bottommost corners of the bounding box. This condition proves that

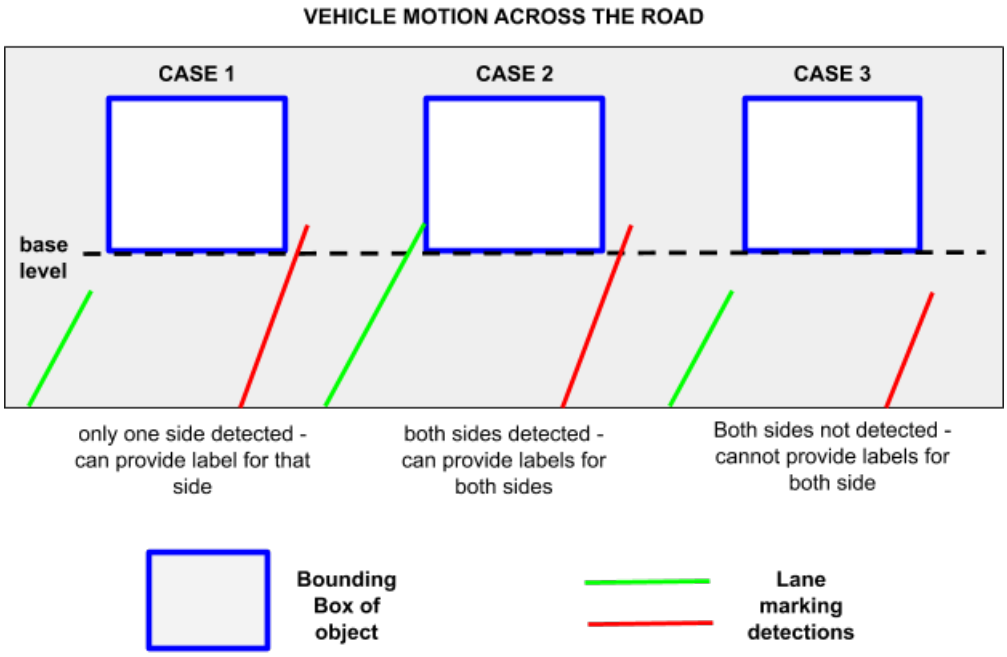


Figure 4.3: Cases for Lane Changing Prediction

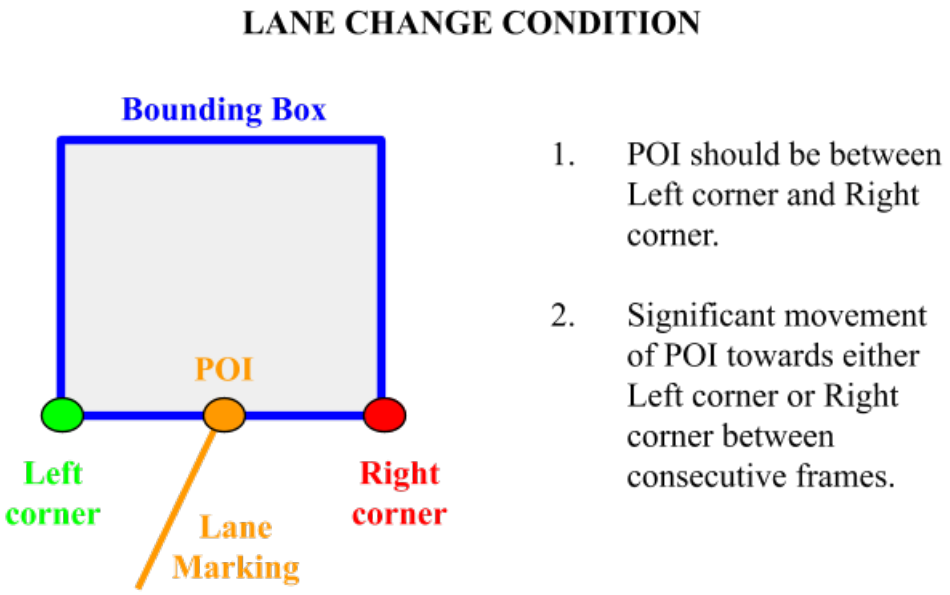


Figure 4.4: Lane Change Condition

the vehicle is not travelling in one specific lane. The second condition is that the point of intersection should move significantly towards either the left or the right corner between successive frames and the labels like (left to right or right to left) can be given according to that. In case there is no significant movement, then no change will be alloted as the label. Various cases for lane changing prediction are shown in the Figure 4.3.

In any case, the tracking ID will always be available on the final label. The class name might be missing in a few frames but it is a rare occurrence as YOLO detects almost all object accurately across all frames. The labels for vehicle motion along and across the road will also be present for all the frames as it covers all the cases.The final output with labels for few sample frames are shown in Figure 4.5.

Figure 4.5: Final Results with Labels

4.3 INFERENCING

Inferencing for final integration was carried out on both embedded platform and PC for video sets obtained from internet sources. The processing was done frame by frame. Since PC offers external GPU support, inferencing was done much faster done on PC.

4.3.1 Inferencing on Embedded Platform

Raspberry Pi model 3B [Table 4.1] was chosen as the primary processor which is assisted by the Intel Neural Compute Stick-2 to run deep neural networks [Figure 4.6]. Intel NCS-2 which is powered by Intel's VPU(Vision Processing Unit) - the Intel Movidius Myriad X, which includes an on-chip neural network accelerator called the Neural Compute Engine. It has 16 programmable SHAVE (Streaming Hybrid Architecture Vector Engine) cores which accelerates the deep learning network performance. Deployment of the models on NCS required installation of the Intel Openvino toolkit and conversion of checkpoint files to XML and binary files.

NCS is a relatively new device specifically designed for the inferencing of deep neural networks. Some of the layers of Neural networks such as argmax, etc. are not supported by it due to reasons not known yet. The available online community support is also less. It has USB form factor and can be plugged directly into the Raspberry Pi.

It ran at 2 fps for YOLOv3. Due to the lack of support of some layers, PWCNet and SCNN could not be deployed on NCS. However, future work could be done to modify these layers to suit the NCS.

Table 4.1: Embedded Platform Specifications

Device Model	Raspberry pi 3B
Central Processing Unit (CPU)	4 X ARM Cortex-A53, 1.2GHz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Storage	32GB micro SD card class 10
Power supply	6V 2-3A
Operating System	Raspbian Stretch

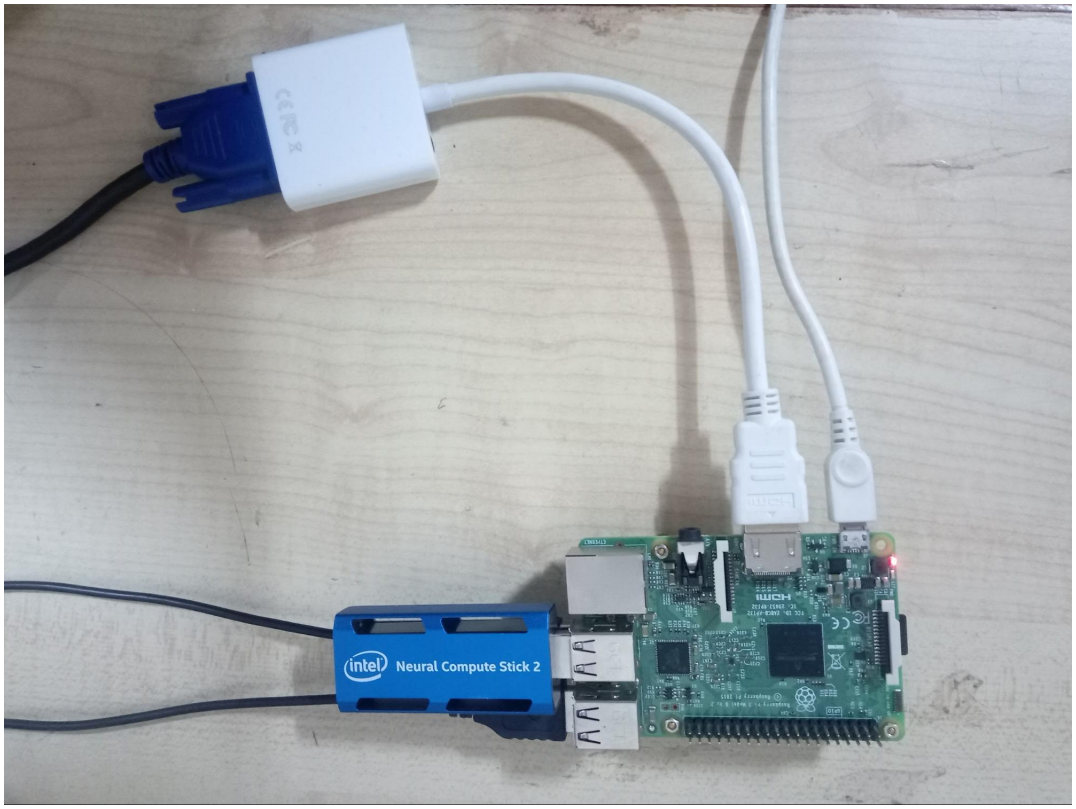


Figure 4.6: Embedded Hardware with Intel NCS-2

4.3.2 Inferencing on PC

The integration script was run on PC which reads the video and processes it frame by frame. Each network is called for every frame and the output data is obtained from them. The CPU and GPU specifications for inference are the same as the one mentioned in Chapter 3 (Table 3.1, Table 3.2). The speed of inference was 5 fps.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

The objective of the project was to interpret the semantic activity of vehicles on the scene. Initially drivable area detection was chosen as the task to understand the road environment but later ego lane detection proved out to be more efficient and computationally less expensive. The three algorithms based on deep neural networks were chosen among the different research articles on the basis of run time, model complexity and accuracy. The outputs of the three networks are integrated to predict the semantic vehicle activity.

The inferred labels were accurate for most test cases but some of the videos had very noisy output, i.e. labels varied in a drastic manner. The result of the integration was heavily dependent on the accuracy of three networks. In some of the frames, YOLO was not able to detect the object but Deep SORT was able to predict the bounding box. As a result, the classes were not identified in those frames. The output of the ego lane detection was affected by eccentric lane markings. The lane detection algorithm struggles to detect when the separation between the dashed lane markings increases.

The embedded implementation was carried on raspberry pi supported by Intel NCS-2 to boost the inference process.The processing frame rate was lower than expected which can be attributed to the complexity of the neural networks and limitations of the processor. One of the major setbacks was that some of the layers of the neural networks were not supported by the Intel NCS-2 and hence some models could not be run on the NCS-2.

This project work can further be enhanced in the following ways :

- Smoothening the output of the integration to make it less noisy
- Training the networks to get best accuracy, preferably with Indian road data.
- Making the DNN pipeline end-to-end, instead of manual integration.
- Making it work in real-time on an embedded board for true real time performance.

REFERENCES

[1] "CULane Dataset" [Online] Available: https://xingangpan.github.io/projects/CULane.html.

[2] "Flying Chairs Optical Flow Dataset [Online] Available: https://lmb.informatik.uni-freiburg.de/resources/datasets/FlyingChairs.en.htmlflyingchairs .

[3] "IMAGENET-Large Scale Visual Recognition Challenge (ILSVRC)" [Online] Available: http://www.image-net.org/challenges/LSVRC/.

[4] "KITTI Optical Flow 2015 Dataset [Online] Available: www.cvlibs.net/publications/Menze2015CVPR.pdf.

[5] "KITTI Vision Benchmark Suite Optical Flow Evaluation 2015 Dataset" [Online] Available: http://www.cvlibs.net/datasets/kitti/eval$_scene_flow.php?benchmark = flow.$

[6] "Middlebury Optical Flow Dataset [Online] Available: http://vision.middlebury.edu/flow/data .

[7] "MPI Sintel Final Pass" [Online] Available: .

[8] "MPI Sintel Flow Dataset [Online] Available: http://sintel.is.tue.mpg.de/ .

[9] "Researchers back Tesla's non-LiDAR approach to self-driving cars." [Online] Available: https://www.therobotreport.com/researchers-back-teslas-non-lidar-approach-to-self-driving-cars/.

[10] "TuSimple Benchmark Platform." [Online] Available: http://benchmark.tusimple.ai//.

[11] "TuSimple Dataset" [Online] Available: https://github.com/TuSimple/tusimple-benchmark/wiki.

[12] (2018). "Liteflownet: A lightweight convolutional neural network for optical flow estimation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[13] A. Dosovitskiy, P. F. (2015). "Flownet: Learning optical flow with convolutional networks." *IEEE International Conference on Computer Vision (ICCV)*.

[14] Alex Bewley, Z. G. (2017). "Simple online and realtime tracking." *CVPR*.

[15] Alex Krizhevsky, Ilya Sutskever, G. E. H. (2012). "Imagenet classification with deep convolutional neural networks." *NIPS*.

[16] Angshuman Parashar, M. R. (2017). "Scnn: An accelerator for compressed-sparse convolutional neural networks." *CVPR*.

[17] B. Horn, B. S. (1981). "Determining optical flow." *Artificial Intelligence*.

[18] Deqing Sun, X. Y. (2018). "Pwc_net: Cnns for optical flow using pyramid, warping, and cost volume." *CVPR*.

[19] E. Ilg, N. M. (2017). "Flownet 2.0: Evolution of optical flow estimation with deep networks.." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[20] Fisher Yu, W. X. (2018). "Bdd100k: A diverse driving video database with scalable annotation tooling." *CVPR*.

[21] H., N. (1987). "On the estimation of optical flow: Relations between different approaches and some new results." *Artificial Intelligence*, 33, 299–324.

[22] Jonathan Long, Evan Shelhamer, T. D. (2015). "Fully convolutional networks for semantic segmentation." *IEEE conference*.

[23] Joseph Redmon, S. D. (2015). "You only look once: Unified, real_time object detection." *CVPR*.

[24] Kaiming He, Xiangyu Zhang, S. R. J. S. (2015). "Deep residual learning for image recognition." *CVPR*.

[25] Karen Simonyan, A. Z. (2015). "Very deep convolutional networks for large-scale image recognition." *ICLR*.

[26] Liang-Chieh Chen, George Papandreou, I. K. K. M. A. L. Y. (2017). "Deeplab: Semantic image segmentation withdeep convolutional nets, atrous convolution,and fully connected crfs." *IEEE transactions on pattern analysis and machine intelligence*.

[27] Lucas B. D., K. T. (1981). "An iterative image-registration technique with an application to stereo vision." *Proceedings of IJCAI*, 674–679.

[28] Marvin Teichmann, M. W. (2016). "Multinet: Real-time joint semantic reasoning for autonomous driving." *CVPR*.

[29] Nicolai Wojke, A. B. (2017). "Simple online and real time tracking with a deep association metric." *CVPR*.

[30] P., A. (1989). "A computational framework and an algorithm for the measurement of visual motion." *IJCV*, 2, 283–310.

[31] Ranjan, A. and Black, M. J. (2017). "Optical flow estimation using a spatial pyramid network." *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[32] Redmon, J. (2017). "Yolov3: An incremental improvement.

[33] Shaoqing Ren, K. H. (2016). "Rich feature hierarchies for accurate object detection and semantic segmentation." *CVPR*.

[34] Tsung-Yi Lin, M. M. (2014). "Microsoft coco: Common objects in context." *CVPR*.

[35] Tsung-Yi Lin, P. G. (2017). "Focal loss for dense object detection." *CVPR*.

[36] Warren Sturgis McCulloch, W. P. (1943). "A logical calculus of the ideas immanent in nervous activity." 5, 115–133.

[37] Wei Liu, D. A. (2015). "Ssd: Single shot multibox detector." *CVPR*.

APPENDIX A

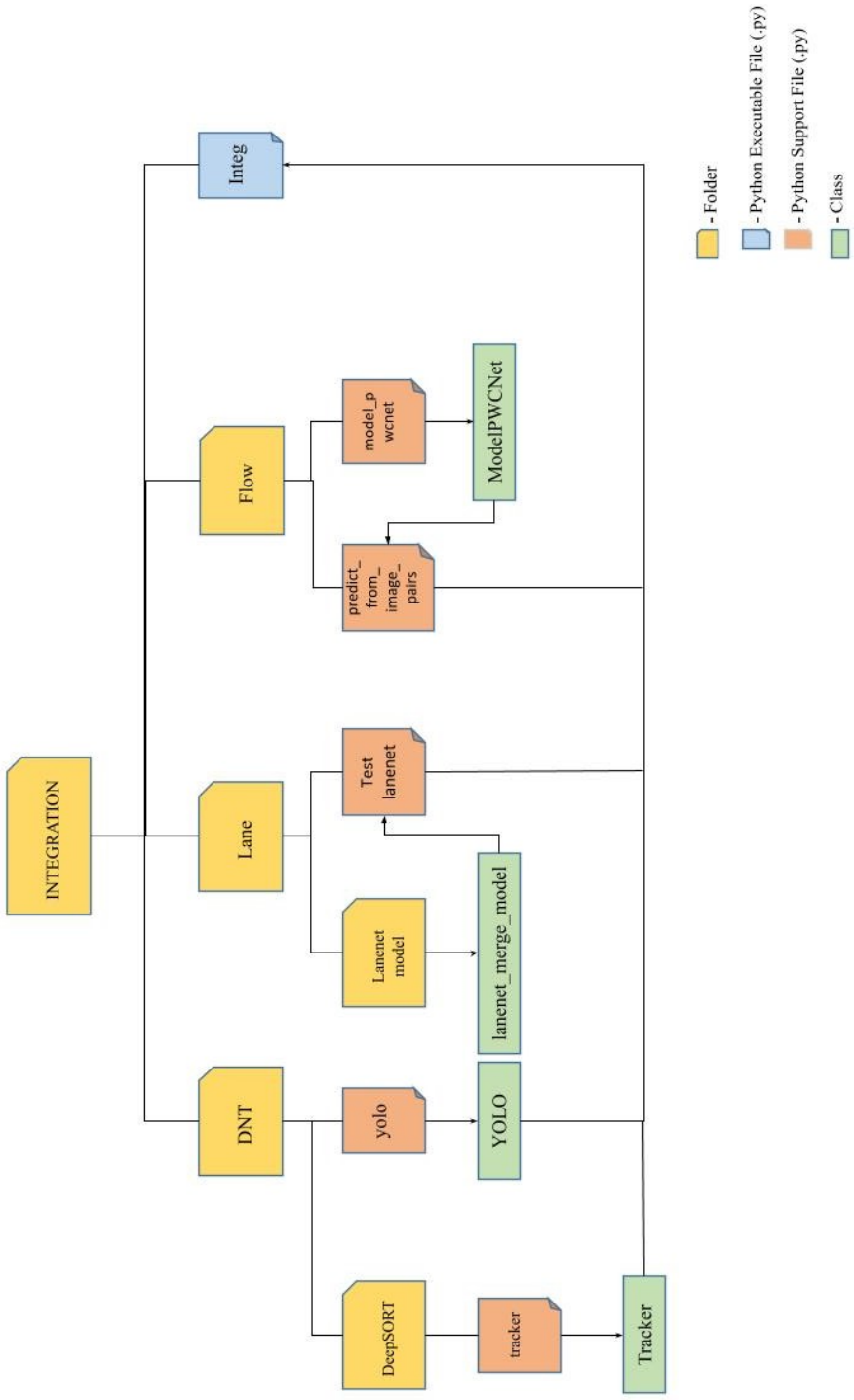
PROJECT FILES

The project files are uploaded on the followingGoogle Drive. They can be found here : **https://drive.google.com/open?id=14lMXFLPgS58JkF8KOUFI8YSge5K71R1J**

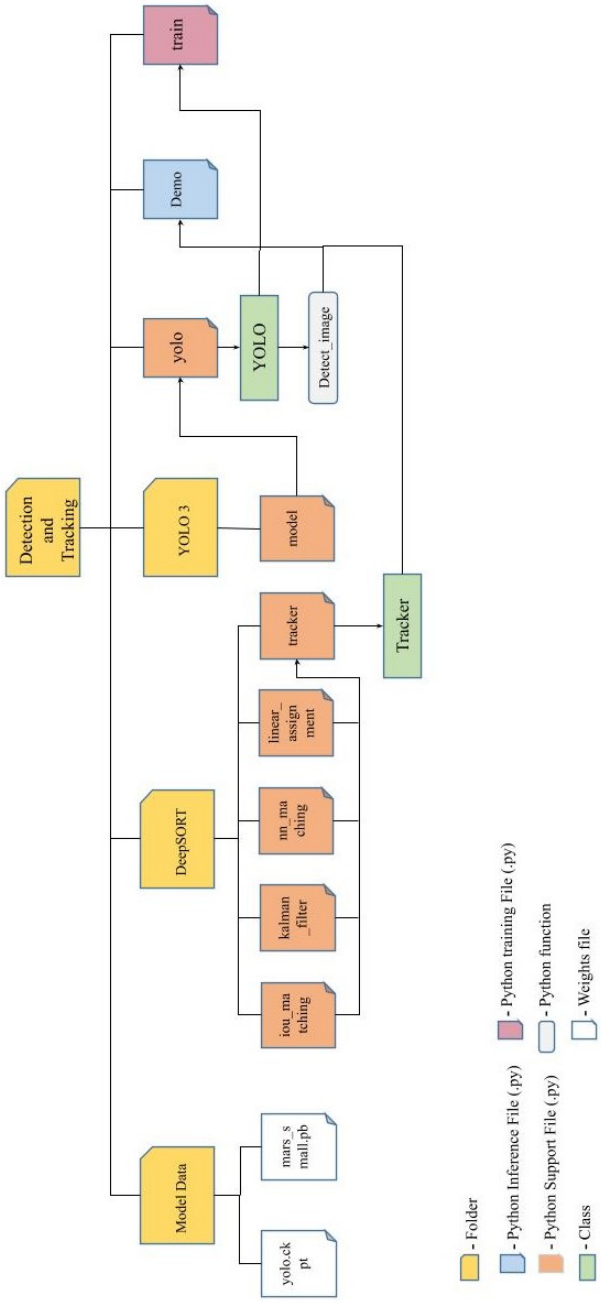
Appendix A.1 shows the schematic diagram of the organisation of files and folders in the Integration folder. The three networks are available as sub-folders along with the main integration script. The individual scripts, models and functions are accessed directly from the integration script. The test video should be available in the main folder.

Appendix A.2, Appendix A.3 and Appendix A.4 shows the schematic diagram of the files and scripts in Detection and Tracking, Lane and Flow folders. The relationship among the scripts are shown in the figures.

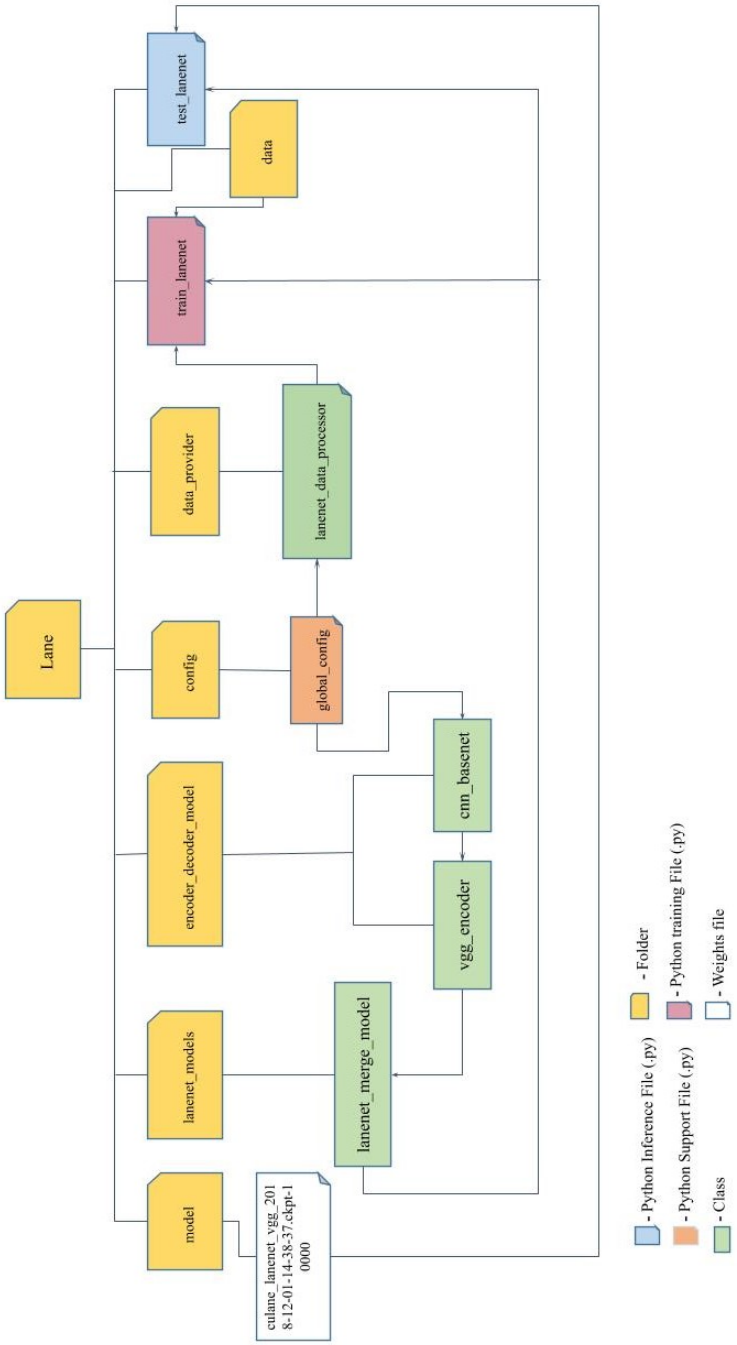
A.1 Integration File Directory



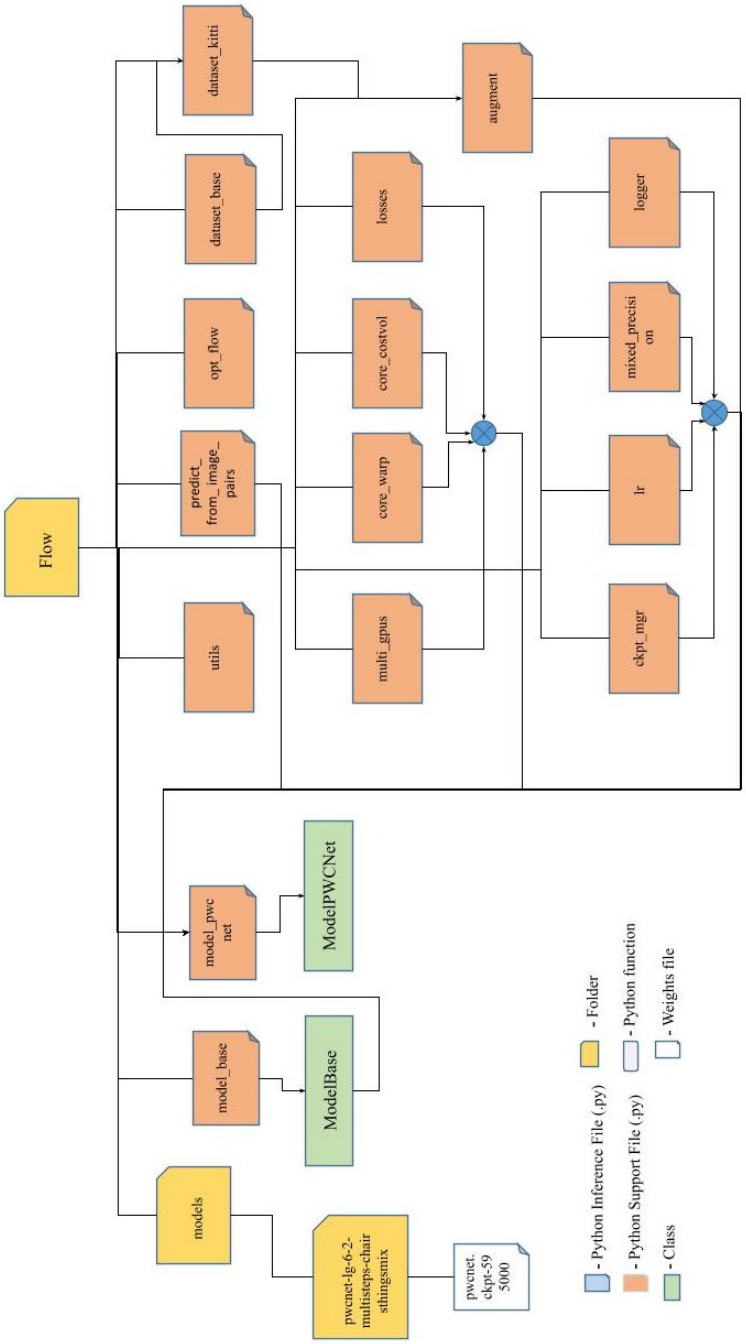
A.2 Detection and Tracking File Directory



A.3 Lane File Directory



A.4 Flow File Directory



APPENDIX B

INTEGRATION CODE

```
#%% IMPORTS
import numpy as np
from PIL import Image
import warnings
import cv2
import tensorflow as tf
from timeit import default_timer as timer
# VEHICLE DETECTION
from dnt.yolo import YOLO

# TRACKER
from dnt.deep_sort import preprocessing
from dnt.deep_sort import nn_matching
from dnt.deep_sort.detection import Detection
from dnt.deep_sort.tracker import Tracker
from dnt.tools import generate_detections as gdet
#from dnt.deep_sort.detection import Detection as ddet
warnings.filterwarnings('ignore')

# LANE DETECTION
from lane.lanenet_model import lanenet_merge_model
from lane.config import global_config
from lane.test_lanenet2_2 import resize_img, post_processing
#from lanenet_model.lanenet_merge_model import LaneNet

# OPTICAL FLOW
from pwc import pwcnet_predict_from_img_pairs


#%% PARAMETERS

# Input and Output Paths
video_path      = r'./opposite lane car changing lane.mp4'
output_path     = video_path.replace('.mp4','_output.mp4')

# deep_sort
model_filename  = 'dnt/model_data/mars-small128_1.pb'
encoder         = gdet.create_box_encoder(model_filename,batch_size=1)
max_cosine_distance = 0.3
```

```
nn_budget           = None
nms_max_overlap     = 1.0
metric              = nn_matching.NearestNeighborDistanceMetric("cosine", max_cosine_distance, nn_budget)
tracker             = Tracker(metric)


# LANE DETECTION
CFG                 = global_config.cfg
VGG_MEAN            = [103.939, 116.779, 123.68]
array_size          = 300

weights_path        = r'./lane/model/culane_lanenet_vgg_2018-12-01-14-38-37.ckpt-10000'

input_tensor        = tf.placeholder(dtype=tf.float32, shape=[None,CFG.TRAIN.IMG_HEIGHT,CFG.TRAIN.IMG_WIDTH, 3], name='input_tensor')
phase_tensor        = tf.constant('test', tf.string)

net                 = lanenet_merge_model.LaneNet()

sess_config         = tf.ConfigProto(device_count={'GPU': 1})

sess_config.gpu_options.per_process_gpu_memory_fraction = CFG.TEST.GPU_MEMORY_FRACTION
sess_config.gpu_options.allow_growth = CFG.TRAIN.TF_ALLOW_GROWTH
sess_config.gpu_options.allocator_type = 'BFC'

sess = tf.Session(config=sess_config)

binary_seg_ret, instance_seg_ret = net.test_inference(input_tensor=input_tensor,phase=phase_tensor,name='lanenet_loss')
initial_var         = tf.global_variables()
final_var           = initial_var[:-1]
saver               = tf.train.Saver(final_var)

sess.run(tf.global_variables_initializer())
saver.restore(sess=sess, save_path=weights_path)

cnt = 0

#%% Main script

dnt_info, lane_info, track_info, track_vid_info, classes_video = [], [], [], [], []
```

```
track_ids = []

def main(yolo):
    # Capture the video and extract useful information
    vid = cv2.VideoCapture(video_path)

    if not vid.isOpened():
        raise IOError("Couldn't open webcam or video")

    video_FourCC    = int(vid.get(cv2.CAP_PROP_FOURCC))
    video_fps       = vid.get(cv2.CAP_PROP_FPS)
    video_size      = (int(vid.get(cv2.CAP_PROP_FRAME_WIDTH)), int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    isOutput        = True if output_path != "" else False

    if isOutput:
        #print(output_path,video_FourCC,video_fps,video_size)
        out = cv2.VideoWriter(output_path, cv2.VideoWriter_fourcc(*'mp4v'), video_fps, video_size)

    # Initialisation of 'previous' frame for 1st frame for Optical Flow
    frame_prev=255*(np.ones([video_size[1],video_size[0],3], dtype=np.uint8))

    # Initialize frame no
    frame_no = -1     # as array index in python usually starts with 0
    lc_diff = np.empty([1])

    accum_time = 0
    curr_fps = 0
    fps = "FPS: ??"
    prev_time = timer()

    while (vid.isOpened()):
        return_value, frame = vid.read()
        tr = 0
        final_label = []
        if return_value == True :
            frame_no += 1
            #print('frame no',frame_no)
            frame_info = []
            track_ids_curr_frame =[]
```

```
############################################################
# OBJECT DETECTION AND TRACKING
############################################################

image = Image.fromarray(frame)
boxs, out_class = yolo.detect_image(image)

features = encoder(frame,boxs)

# score to 1.0 here.
detections = [Detection(bbox, 1.0, feature) for bbox, feature in zip(boxs, features)]

#classes = [Detection(out_class, 1.0, feature) for out_class, feature in zip(out_class, features)]

# Run non-maxima suppression.
boxes = np.array([d.tlwh for d in detections])
scores = np.array([d.confidence for d in detections])
indices = preprocessing.non_max_suppression(boxes, nms_max_overlap, scores)
detections = [detections[i] for i in indices]

# Call the tracker
tracker.predict()
tracker.update(detections)
classes, bboxs = [], []

out_class.append('dummy')
classes_video.append(out_class)

for track in tracker.tracks:

    if not track.is_confirmed() or track.time_since_update > 1:

        continue
    bbox =  track.to_tlbr()

    track_id = str(track.track_id)

    tr += 1
    track_vid_info.append(frame_no)
```

```
        track_ids.append(track_id)
        track_ids_curr_frame.append(track_id)
        ## get the Frame index when
        frame_index = track_ids.index(track_id)

        if track.is_confirmed() or track.time_since_update > 1 and tr==1:

            first_tracked_frame = track_vid_info[frame_index]


        bboxs.append(bbox)

    ## get the class names from yolo
    if len(bboxs) != 0:
        for i in range(len(bboxs)):
            class_name = classes_video[first_tracked_frame][i]
            obj = class_name + ' object no '+ track_ids_curr_frame[i]
            classes.append(obj)

    frame_info.append(bboxs)
    frame_info.append(classes)
    dnt_info.append(frame_info)

    ############################################################
    # LANE DETECTION
    ############################################################

    imgs = resize_img(frame, VGG_MEAN)
    instance_seg_image, existence_output = sess.run([binary_seg_ret, instance_seg_ret],feed_dict={input_tensor: [imgs]})
    lane_coordinates,lines_col,lines_row = post_processing(instance_seg_image, existence_output, cnt, video_size)

    # lane_coordinates gives [col_ind,row_ind]
    lane_info.append(lane_coordinates)

    l2 = lane_coordinates[1]
    l3 = lane_coordinates[2]

    lane_mark_pixels = l2+l3
```

```
lane_mark_pixels = np.array(lane_mark_pixels)
#print('no of ego lane pixels: ',np.shape(lane_mark_pixels))

lane_mark_pixels = lane_mark_pixels-1  # array indices are from 0 to n-1


############################################################
# OPTICAL FLOW
############################################################

frame_curr = frame*np.ones([video_size[1],video_size[0],3], dtype=np.uint8)

flow_frame = pwcnet_predict_from_img_pairs.pwc_prediction(frame_prev, frame_curr)

#print('flow frame shape : ',np.shape(flow_frame))

############################################################
# INTEGRATION
############################################################

# Flow vectors of Lane pixels
_, lane_flow_vec = flow_norm_obj(lane_mark_pixels, flow_frame)

# Decision Tree branch level 1
if lane_flow_vec[1] < 5 :
    ego_motion = 'EGO MOVING'            # ego vehicle moving
else:
    ego_motion = 'EGO REST'            # ego vehicle at rest


# Normalization of flow for each object ID
obj_pixels=np.empty((0,2),int)

obj_flow_vec, rel_flow_vec= [], []

for i in range(len(frame_info[0])):

    row_min = int(frame_info[0][0][1])
```

```
row_max = int(frame_info[0][0][3])
col_min = int(frame_info[0][0][0])
col_max = int(frame_info[0][0][2])

# Making the B_Box smaller (80% along each axis) to get accurate flow output
new_row_min = row_min + int(0.2*(row_max-row_min))
new_row_max = row_max - int(0.2*(row_max-row_min))
new_col_min = col_min + int(0.2*(col_max-col_min))
new_col_max = col_max - int(0.2*(col_max-col_min))

for p in range(new_row_min, new_row_max):
    for q in range(new_col_min, new_col_max):
        temp = np.array([q,p],dtype='int')   # [col_ind,row_ind]
        #print(temp)
        obj_pixels = np.vstack((obj_pixels, temp))
        #obj_pixels = np.append(obj_pixels, temp, axis=0)

obj_pixels = obj_pixels-1   # array indices are from 0 to n-1
#print('no. of obj pixels', np.shape(obj_pixels))
#print('obj_pixels:',obj_pixels)
ff, obj_flow_vec = flow_norm_obj(obj_pixels, flow_frame)


# Find relative flow vector
#print('Lane Flow Vector = {}'.format(lane_flow_vec))
#print('Object Flow Vector = {}'.format(obj_flow_vec))

rel_flow_vec = obj_flow_vec-lane_flow_vec
#print('Relative Flow Vector = {}'.format(rel_flow_vec))

obj_state = " "
# Object motion detection
if ego_motion == 'EGO REST':
    if abs(rel_flow_vec[1]) <= 1 :
        obj_state = "rest"            # 0 - at rest
    elif rel_flow_vec[1] > 1 :
        obj_state = "forward"          # 1 - forward
    else :
        obj_state = "oncoming"            # 2 - oncoming
```

```
elif ego_motion == 'EGO MOVING':
    if abs(rel_flow_vec[1]) <= 1 :
        obj_state = "rest"
    elif rel_flow_vec[1] > abs(lane_flow_vec[1]) :
        obj_state = "forward"            # for now, all 3 cases are same
    else :
        obj_state = "oncoming"

# Lane Change Detection

if frame_no > 1:
    lc_cond = False
    for i in range(len(lane_info[frame_no])):
        for j in range(len(lane_info[frame_no][i])):
            if lane_info[frame_no][i][j][1] >= row_min :
                if lane_info[frame_no][i][j][0] > col_min and lane_info[frame_no][i][j][0] <= col_max :
                    lc_cond = True
                    lc_col = lane_info[frame_no][i][j][0]
                    #lc_row = lane_info[frame_no][i][j][1]
                    break
                else:
                    lc_cond = False
            else:
                lc_cond = False
        if lc_cond :
            break

    if lc_cond :
        diff_right = col_max - lc_col
        diff_left = lc_col - col_min
        lc_diff_temp = np.array([diff_right],int)

    else :
        lc_diff_temp = np.array([np.nan])
```

```
            else:
                lc_diff_temp = np.array([np.nan])


            lc_diff = np.append(lc_diff,lc_diff_temp)

            if (lc_diff[-1] - lc_diff[-2])==np.nan:
                lc_label = "-"
            elif (lc_diff[-1] - lc_diff[-2]):
                lc_label = "left to right"
            elif (lc_diff[-2] - lc_diff[-1]):
                lc_label = "right to left"
            else:
                lc_label = "-"

            #cv2.putText(frame,ego_motion, (10,video_size[1]-30),
            #            cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)
            #for box in bbox:
               # cv2.putText(frame, str(obj_state),(int(box[0]),
               #             int(box[1])),0, 5e-3 * 200, (0,255,0),2)
            #cv2.namedWindow("result", cv2.WINDOW_NORMAL)
            #cv2.imshow("result", frame)

        # Press Q to stop!
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        if isOutput:
            out.write(frame)
    else:
        out.release()
        cv2.destroyAllWindows()
        break

    curr_time = timer()
    exec_time = curr_time - prev_time
    prev_time = curr_time
    accum_time = accum_time + exec_time
    curr_fps = curr_fps + 1
    if accum_time > 1:
```

```
            accum_time = accum_time - 1
            fps = "FPS: " + str(curr_fps)
            curr_fps = 0
        print('FPS:',fps)

    sess.close()


def flow_norm_obj(obj_pixels, flow_frame): # obj_pixels is array of [row_index,col_index]


    flow_sum = np.array([0,0],dtype='float64')
    for i in range(np.shape(obj_pixels)[0]):

        flow_sum += flow_frame[obj_pixels[i][1]][obj_pixels[i][0]]

    flow_mean = flow_sum / np.shape(obj_pixels)[0]


    for i in range(np.shape(obj_pixels)[0]):

        flow_frame[obj_pixels[i][1]][obj_pixels[i][0]] = flow_mean

    return flow_frame, flow_mean


if __name__ == '__main__':
    main(YOLO())
```