OBJECT DETECTION AND COUNTING BY IMAGE PROCESSING

A PROJECT REPORT

Submitted by

DHINESHKUMAR.P (422516104008)

SIDDHARTHAN.G (42251610404)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING,VILLUPURAM

ANNA UNIVERSITY : CHENNAI 600 025

AUGUST 2020

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"OBJECT DETECTION AND COUNTING BY IMAGE PROCESSING"**

is the bonafide work of **"DHINESHKUMAR.P(422516104008) and SIDDHARTHAN.G(422516104044)"** who carried out the project under my supervision .

SIGNATURE

Dr. P. Arjun, M.E..,

HEAD OF THE DEPARTMENT

Assistant Professor Department Of Computer Science and Engineering University College Of Engineering Villupuram-605 103

SIGNATURE

Mr.R.Regan, M.E..,

SUPERVISOR

Teaching Fellow

Department Of Computer Science and Engineering

University College Of Engineering

Villupuram-605 103

Submitted for the University Examination held on _____

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We wish to express our sincere thanks and gratitude to our Dean In -charge **Dr.S.ARULCHELVAN, M.E, Ph.D.,** for offering us all the facilities to do the project.

We also express our sincere thanks to Head of the Department In-charge Mr.**P. ARJUN M.E.,** Department of Computer Science and Engineering for his support and guidance to do this project work.

We express our thanks to **Mr.R.REGAN M.E.,** our internal project guide, Department of Computer Science and Engineering for his effective support for the successful completion in implementing our valuable idea.

We would like to thank all the Faculty Members in our department for their guidance to finish this project successfully. We also like to thank all our friends for their willing assistance.

This project consumed huge amount of work, research and dedication. Still, implementation would not have been possible if we did not have a support of many individuals and organizations. We would like to extend our sincere gratitude to all of them

DHINESHKUMAR.P

SIDDHARTHAN.G

ABSTRACT

Object detection and counting by image processing have greater prominence in acquiring a clear image through the prevailing algorithm. The main objective of the paper is to get a flabbergasting clear vision of accuracy to reach its quality in image processing. The image classifier anatomy is organized by input image, pre-processing, deep learning-based feature algorithm and label assignment to obtain accurate vector plans. I created a vehicle detection and tracking pipeline with OpenCV, SKLearn, Histogram of Oriented Gradients (HOG), and Support Vector Machines (SVM). I then optimized and evaluated the model on video data from an automotive camera taken during highway driving. The python coding and 2D vector plans were explored to assist the available database of the image by output verification. Then, multiple Convolutional Neural Network (CNN) is evolved in 2015 on deep learning gave astounding computer vision with an accuracy of ninety-five percentage. Then, for image recognition, Deep learning through image processing becomes vital and inseparable in the modern world. For faster multiple object deduction R-CNN produce is employed. For image recognition a.k.a image classifier algorithm is progressed. It helps to recognize algorithm input and output contents of the image. a.k.a image classifier algorithm consists of thousands of images to recognize the objects and classes definitely.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

ABBREVATIONS	EXPLANATION
CNN	CONVOLUTION NEURAL NETWORK
TF	TENSORFLOW
SSD	SINGLE SHOT MULTIBOX DETECTOR
REMO	RELATIVE MOTION
GPU	GRAPHICS PROCESSING UNIT
API	APPLICATION PROGRAMMING INTERFACE
DNN	DEEP NEURAL NETWORK
JPEG	JOINT PHOTOGRAPHIC EXPERTS GROUP
ILSVRC	IMAGENET LARGE SCALE VISUAL RECOGNITION CHALLENGE
YOLO	YOU ONLY LOOK ONCE
UML	UNIFIED MODELLING LANGUAGE
XML	EXTENSIBLE MARKUP LANGUAGE
RELU	RECTIFIED LINEAR UNIT

PY	PYTHON
PB	PURE BASIC
RPN	REGION PREPOSAL NETWORK
R-CNN	REGION BASED CONVOLUTION NEURAL NETWORK

CHAPTER 1

INTRODUCTION

1.1 DOMAIN

1.1.1 OVERVIEW

Deep Learning is an advanced field of Machine Learning that uses the concepts of Neural Networks to solve highly-computational use cases that involve the analysis of multi-dimensional data. It automates the process of feature extraction, making sure that very minimal human intervention is needed. Modeled in accordance with the human brain, a Neural Network was built to mimic the functionality of a human brain. The human brain is a neural network made up of multiple neurons, similarly, an Artificial Neural Network (ANN) is made up of multiple perceptrons.

A neural network consists of three important layers:

- **Input Layer**: As the name suggests, this layer accepts all the inputs provided by the programmer.
- **Hidden Layer**: Between the input and the output layer is a set of layers known as Hidden layers. In this layer, computations are performed which result in the output.
- **Output Layer**: The inputs go through a series of transformations via the hidden layer which finally results in the output that is delivered via this layer.

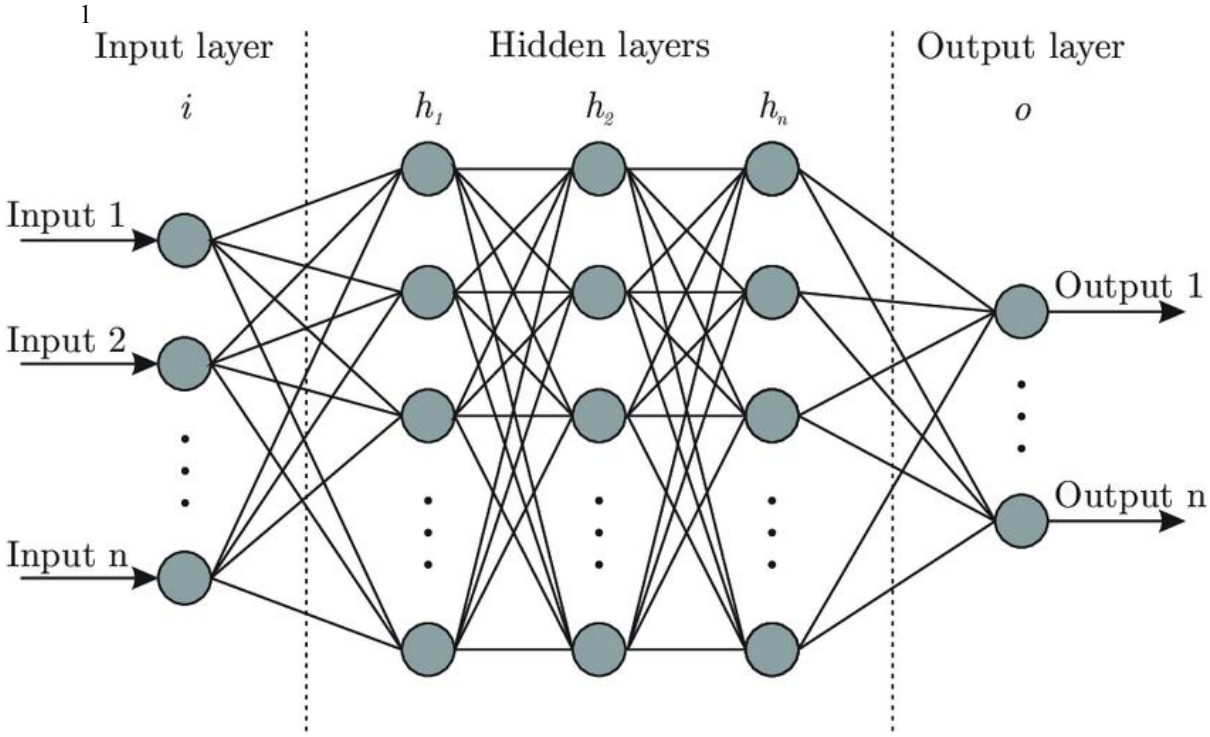

Figure 1.1.1.1 : Architecture of Neural Network

1.1.2 WORKS OF DEEP LEARNING

Deep learning has evolved hand-in-hand with the digital era, which has brought about an explosion of data in all forms and from every region of the world. This data, known simply as big data, is drawn from sources like social media, internet search engines, e-commerce platforms, and online cinemas, among others. This enormous amount of data is readily accessible and can be shared through fintech applications like cloud computing.

However, the data, which normally is unstructured, is so vast that it could take decades for humans to comprehend it and extract relevant information. Companies realize the incredible potential that can result from unraveling this wealth of information and are increasingly adapting to AI systems for automated support.

Deep learning learns from vast amounts of unstructured data that would normally take humans decades to understand and process.

1.1.3 DIFFERENCES ON AI, DL, ML

- Artificial Intelligence is the science of getting machines to mimic the behaviour of humans.
- Machine learning is a subset of Artificial Intelligence (AI) that focuses on getting machines to make decisions by feeding them data.
- Deep learning is a subset of Machine Learning that uses the concept of neural networks to solve complex problems.

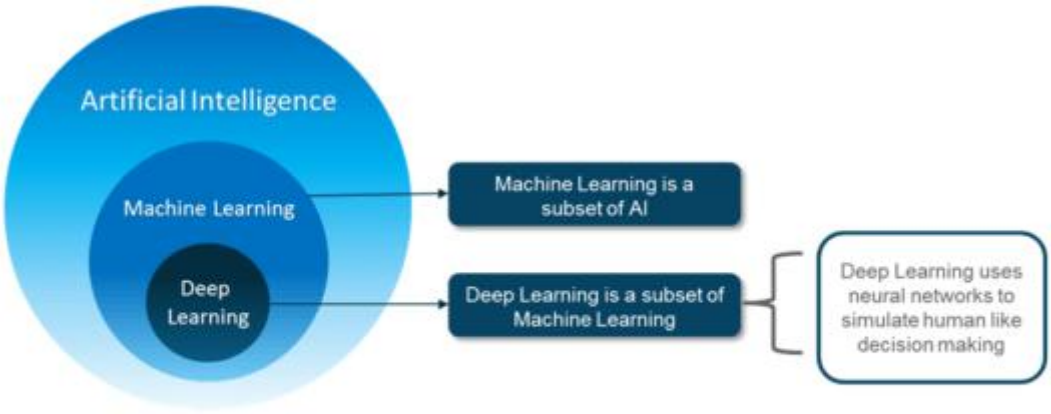

Figure 1.1.3.1 : Difference between AI, ML, DL

1.2 OBJECTIVES

Our project uses TensorFlow, a framework provided for deep learning, to model our neural network. This API is used to detect multiple objects in real-time video streams. SSD MobileNet, a predefined model offered by TensorFlow is used as the base and fine-tuned to improve the accuracy and the range of objects that can be detected. This model can be trained for any custom object that is required by the user to keep

a track of that object. Once the system is wellequipped to detect objects, it is trained to track the object as long as it is in the range of camera. It is interesting to observe how even objects themselves can be a source of information that can be used to detect their behavioural patterns by tracking their movement. This analysis of the objects finds applications in home automation and security. 2D trajectories of the objects captured by the camera are fed as input to tracking and pattern detecting algorithm. Research presented in regarding different patterns and REMO framework is used as a basis for developing a spacio-temporal pattern detecting algorithm. We aim to improve the productivity of the user by supporting his actions with memory and intelligence of machines.

1.2.1 TENSORFLOW

Released on the 15th of November 2015 by Google[1], TensorFlow [21] is the newest open source library written in Python for numerical computation.

It has immediately a great success in the Machine Learning community and in less than one year it also had a lot of support and development by Google [1] itself, more over by many community projects, developed in any area of Deep Learning.

The peculiarity of TensorFlow is its work flux, made by data flow graphs. Where Nodes represent mathematical operations, edges represent the multidimensional data arrays communicated between them; the latter can be

considered, as in elec- tronics, a Tensor, from here its name. In May 2016, Google [53] has revealed that it has used TensorFlow in AlphaGo project, with a special hardware dedicated to boost library's performances.

To reach rapidly this goals, Google dedicated a special attention to the user experience of TensorFlow[21], which arrives with a great basic support and a well- grown GitHub community [22], the key of this quick improvement.

All these are the peculiarities that pushed us to choose TensorFlow [21], over more developed and bigger communities. The irruption made in the Deep Learning environment, shows up its great future potentialities, that are rolling out day by day.

Normally, it is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

Advantages of TensorFlow over other frameworks:

1. Easy deployment (Python pip package manager deployed by TensorFlow facilitates easy installation).
2. Better support for GPUs as compared to other models.
3. It provides high level APIs for building models.
4. It is extremely easy to do unconventional and hard-core changes.

By analysing all the above characteristics and the requirements of our project, we used the TensorFlow framework to train our neural network.

1.2.2 MOBILENET

MobileNet is an architecture which is more suitable for mobile and embedded based vision applications where there is lack of compute power. This architecture was proposed by Google. A typical mobilenet architecture is shown below

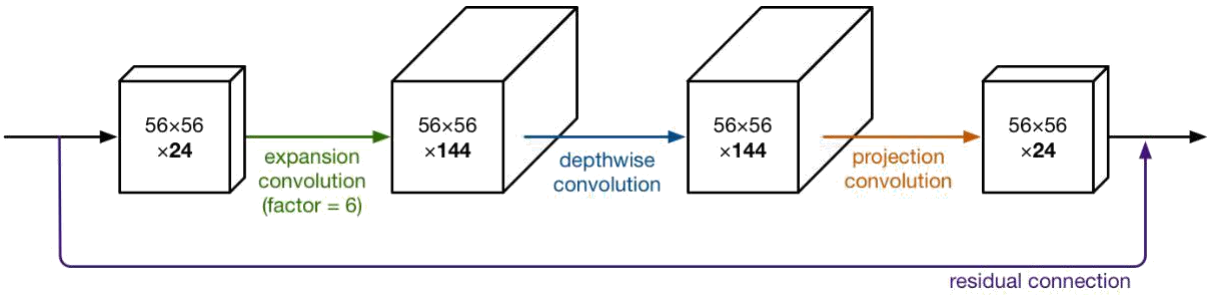

Figure 1.2.2.1: MobileNet Architecture.

- This architecture uses depthwise separable convolutions which significantly reduces the number of parameters when compared to the network with normal convolutions with the same depth in the networks. This results in light weight deep neural networks.
- The normal convolution is replaced by depthwise convolution followed by pointwise convolution which is called as depthwise separable convolution.
- In the normal convolution, if the input feature map is of *Hi*,*Wi*,*Ci* dimension and we want *Co* feature maps with convolution kernel size *K* then there are *Co* convolution kernels each with dimension *K*,*K*,*Ci*. This results in a feature map of *Ho*,*Wo*,*Co*

- dimension after convolution operation.

- In the depthwise separable convolution, if the input feature map is of *Hi*,*Wi*,*Ci* dimension and we want *Co* feature maps in the resulting feature map and the convolution kernel size is *K* then there are *Ci* convolution kernels, one for each input channel, with dimension *K*,*K*,1. This results in a feature map of *Ho*,*Wo*,*Ci* after depthwise convolution. This is followed by pointwise convolution [1x1 convolution]. This convolution kernel is of dimension 1,1,*Ci* and there are *Co* different kernels which results in the feature map of *Ho*,*Wo*,*Co* dimension.
- This results in the reduction of number of parameters significantly and thereby reduces the total number of floating point multiplication operations which is favorable in mobile and embedded vision applications with less compute power.

1.2.3 CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Networks, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, take a weighted sum over them, pass it through an activation and responds with an output.

There are **four** layered **concepts** are in Convolutional Neural Networks such as,

- Convolution,
- ReLu,
- Pooling and

Full Connectedness (Fully Connected Layer)

1.2.4 IMAGE SEGMENTATION

Segmentation algorithms generally are basedon one of two basis properties of intensityvalues,

Discontinuity: to partition an image basedon abrupt changes in intensity (such asedges)

Similarity: to partition an image into regionsthat are similar according to a set of predefined criteria.

1.3 DARKNET

Darknet is an open source neural network framework. It is a fast and highly accurate (accuracy for custom trained model depends on training data, epochs, batch size and some other factors) framework for real time object detection (also can be used for images)

Darknet in YOLO

Darknet is a framework to train neural networks, it is open source and written in C/CUDA and serves as the basis for YOLO. Darknet is used as the framework for training YOLO, meaning it sets the architecture of the network. ... Clone the repo locally and you have it.

Darknet uses Tensor Flow

Darknet is an open source custom neural network framework written in C and CUDA. ... Thanks to Trinh Hoang Trieu, Darknet models are converted to Tensorflow and can be installed on both Linux and Windows environments.

Darknet Framework needed for object detection

Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. ... Recurrent neural networks are powerful models for representing data that changes over time and Darknet can handle them without making use of CUDA or OpenCV.

CHAPTER 2

LITERATURE SURVEY

TITLE: REAL TIME OBJECT TRACKING ON A DRONE WITH MULTI-INERTIAL SENSING DATA

AUTHOR : Peng Chen, Yuanjie Dang

YEAR : 2018

Real-time object tracking on a drone under a dynamic environment has been a challenging issue for many years, with existing approaches using off-line calculation or powerful computation units on board. This paper presents a new lightweight real-time onboard object tracking approach with multi-inertial sensing data, wherein a highly energy-efficient drone is built based on the Snapdragon flight board of Qualcomm. The flight board uses a digital signal processor core of the Snapdragon 801 processor to realize PX4 autopilot, an open-source autopilot system oriented toward inexpensive autonomous aircraft. It also uses an ARM core to realize Linux, robot operating systems, open-source computer vision library, and related algorithms. A lightweight moving object using the oriented FAST and rotated binary robust independent elementary features algorithm and adapts a local difference binary algorithm to construct the image binary descriptors. The K-nearest neighbor method is then used to match the image descriptors. Finally, an object tracking method is proposed that fuses inertial measurement unit data, global positioning system data, and the moving object detection results to calculate the relative position between coordinate systems of the object and the drone. All the algorithms are run on the Qualcomm platform in real time. Experimental results demonstrate the superior performance of our method over the state-of-the-art visual tracking method.

TITLE : VEHICLE DETECTION AND COUNTING SYSTEM FOR REAL-TIME TRAFFIC SURVEILLANCE

AUTHOR : Boris A. Alpatov, Pavel V. Babayan, Maksim D. Ershov

YEAR : 2018

Vechicle detection and counting system for Real-Time Surveillance are considering road situation analysis tasks for traffic control and ensuring safety. The following image processing algorithms are proposed: vehicle detection and counting algorithm, road marking detection algorithm. The algorithms are designed to process images obtained from a stationary camera. The developed vehicle detection and counting algorithm was implemented and tested also on an embedded platform of smart cameras. The results of experimental research of proposed algorithms are presented.

TITLE : VEHICLE COUNTING SYSTEM IN REAL-TIME

AUTHOR: Salma Bouaich, Mohamed Adnane Mahraz, Jamal Riffi, Hamid Tairi

YEAR : 2018

To address the challenge of congestion, we propose a system that estimates the state of the road. To do that, we must be gone through several steps. In this work, we will present the first and the important step to estimate the vehicle flow; this later helps us to count the vehicles using the virtual line. Generally, we start with the background subtraction to isolate moving objects. To facilitate crossing of vehicles with the line, we apply the detection of objects. Our system uses the K-nearest neighbor (KNN) as a method to subtract the background, in order to apply our counting algorithm.

TITLE : MOVING OBJECT DETECTION AND TRACKING USING CONVOLUTIONAL NEURAL NETWORK

AUTHOR : Shraddha Mane, Prof.Supriya Mangale

YEAR : 2018

The object detection and tracking is the important steps of computer vision algorithm. The robust object detection is the challenge due to variations in the scenes. Another biggest challenge is to track the object in the occlusion conditions. Hence in this approach, the moving objects detection using TensorFlow object detection API. Further the location of the detected object is pass to the object tracking algorithm. A novel CNN based object tracking algorithm is used for robust object detection. The proposed approach is able to detect the object in different illumination and occlusion. The proposed approach achieved the accuracy of 90.88% on self generated image sequences.

TITLE : INTELLIGENT VEHICLE COUNTING AND CLASSIFICATION SENSOR FOR REAL-TIME TRAFFIC SURVEILLANCE

AUTHOR : Walid Balid, Member, Hasan Tafish, and Hazem H. Refai

YEAR : 2017

Real-time traffic surveillance is essential in today's intelligent transportation systems and will surely play a vital role in tomorrow's smart cities. The work detailed in this paper reports on the development and implementation of a novel smart wireless sensor for traffic monitoring. Computationally efficient and reliable algorithms for vehicle detection, speed and length estimation, classification, and time-synchronization were fully developed, integrated, and evaluated. Comprehensive system evaluation and extensive data analysis were performed to tune and validate the system for a reliable and robust operation. Several field studies conducted on highway and urban roads for different scenarios and under various traffic conditions resulted in 99.98% detection accuracy, 97.11% speed estimation accuracy, and 97% length-based vehicle classification accuracy. The developed system is portable, reliable, and cost-effective.

TITLE : VIDEO PROCESSING FROM ELECTRO-OPTICAL SENSORS FOR OBJECT DETECTION AND TRACKING IN A MARITIME ENVIRONMENT

AUTHOR : Dilip K. Prasad, Deepu Rajan, Lily Rachmawati, Eshan Rajabally, and Chai Quek

YEAR : 2017

We present a survey on maritime object detection and tracking approaches, which are essential for the development of a navigational system for autonomous ships. The electro-optical (EO) sensor considered here is a video camera that operates in the visible or the infrared spectra, which conventionally complements radar and sonar for situational awareness at sea and has demonstrated its effectiveness over the last few years. This paper provides a comprehensive overview of various approaches of video processing for object detection and tracking in the maritime environment. We follow an approach-based taxonomy wherein the advantages and limitations of each approach are compared. The object detection system consists of the following modules: horizon detection, static background subtraction, and foreground segmentation. Each of these has been studied extensively in maritime situations and has been shown to be challenging due to the presence of background motion especially due to waves and wakes. The key processes involved in object tracking include video frame registration, dynamic background subtraction, and the object tracking algorithm itself. The challenges for robust tracking arise due to camera motion, dynamic background, and low contrast of tracked object, possibly due to environmental degradation. The survey also discusses multisensor approaches and commercial maritime systems that use EO sensors. The survey also highlights methods from computer vision research, which hold promise to perform well in maritime EO data processing.

TITLE : MONO-VISION BASED MOVING OBJECT DETECTION IN COMPLEX TRAFFIC SCENES

AUTHOR : Vincent Frémont ; Sergio Alberto Rodríguez Florez ; Bihao Wang

YEAR : 2017

Vision-based dynamic objects motion segmentation can significantly help to understand the context around vehicles, and furthermore improve road traffic safety and autonomous navigation. Therefore, moving object detection in complex traffic scene becomes an inevitable issue for ADAS and autonomous vehicles. In this paper, we propose an approach that combines different multiple views geometry constraints to achieve moving objects detection using only a monocular camera. Self-assigned weights are estimated online moderating the contribution of each constraint. Such a combination enhances the detection performance in degenerated situations. According to the experimental results, the proposed approach provides accurate moving objects detections in dynamic traffic scenarios with large camera motions.

TITLE : OBJECT COUNTING ON LOW QUALITY IMAGES

AUTHOR : Jean-Francois Rajotte, Martin Sotir, Cedric Noiseux, Louis-Philippe Noel and Thomas Bertiere

YEAR : 2018

The installation and management of traffic monitoring devices can be costly from both a financial and human resource point of view. It is therefore important to take advantage of available infrastructures to maximize the information extraction for each technology. Here we show how low-quality urban road traffic images from cameras, already installed in many cities such as Montreal, Vancouver and Toronto can be used as a nonintrusive traffic monitoring. To this end, we use a pre-trained object detection neural network to count vehicles within images. We then compare the results with human annotations gathered through crowdsourcing campaigns. We use thiscomparison to assess performance and calibrate the neural network annotations. The performance of our system allows us to consider applications which can monitor the traffic conditions in near real-time, making the counting usable for traffic-related services. Furthermore, the resulting annotations pave the way for building a historical vehicle counting dataset to be used for analysing the impact of road traffic on many city-related issues such as urban planning, security, and Pollution.

TITLE : OBJECT DETECTION AND TRACKING APPROCHES FOR VIDEO SURVEILLANCE OVER CAMERA NETWORK

AUTHOR : Nitesh Funde ; Parnika Paranjape ; Kamal Ram ; Punit Magde

YEAR : 2018

Object detection and tracking are the most challenging part of any computer vision applications. In computer vision, video surveillance is a popular research area in a dynamic environment, particularly for security reasons. The video surveillance technology plays a crucial role to prevent crime, terrorism etc. The video outputs are filtered and processed by human operators and in case of a forensic, the high volume of data made it difficult to track any object. This work has been done with an aim to reduce the effort of human operators with an increase in the response time to forensic events. It involves designing of an efficient object tracking system for simple environments where the camera is static, background is simple and no similar object to the one being tracked is present. The system is provided with network configuration of the cameras and roads of the surveillance area, videodumps and an image of the object to be tracked. It tracks the objects through the videos and dumps the tracked portions of the videos where the object was present.

CHAPTER 3

SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

3.1.1.OVERVIEW OF EXISTING SYSTEM

Traditional deep learning-based vehicle detectionmethods are often designed using a pyramid of filters with multiple scales and sizes; therefore, the processing time is slow due to the large number of scales used and because the classifier runs at all scales. Recently, a deep learning-based region proposal network was introduced to detect vehicles that only employ the network one time regardless of the size of the input image. In object detection, deep learning-based region proposal networks have achieved state-of-the-art performance in terms of accuracy. These systems achieve a very high accuracy under normal driving conditions; however, their performance decreases under difficult driving conditions such as in snow, rain, or fog. In addition, the current state-of-the-art system-based region proposal networks still fail to satisfy the real-time requirements of the driving assistant systems. More recently, the identification of local patterns has been shown to improve the performance of the traditional deep-learning systems; hence, this paper investigates local patterns in region proposal networks to improve their accuracy. Depth information is also investigated to improve the processing time of current region proposal networks. Our experimental results show that the proposed system obtains better performance than the state-of-the-art object region detection systems in terms of both accuracy and processing time.

EXPERIMENTAL RESULTS AND DISCUSSION

Our proposed vehicle detection is composed of four main modules. The first module is real-time stereo vision-based-FPGA .

Second is an adaptive region proposal estimation that accurately produces candidate regions using the disparity information, while the third module is the Fast R-CNN detector, which uses the proposed candidate regions to detect and

classify the object. Finally, we intro-duced a robust 3-channel pattern to provide a robust feature for training the proposed deep learning system.

The entire system is a unified network for vehicle detection. We train and test both region proposal and object detection networks on images of a single scale . We re-scale the images such that their shorter side is ‘s’ = 600 pixels. For training Fast R-CNN, ROI pooling layers with max pooling are used to convert the input features into a small feature map. Each ROI is defined by four positions at the top-left corner as well as height and width. The detection network is initialized by ImageNet-pre-trained .

To reduce the redundancy of region proposal candidates that overlap each other, we apply non-maximum suppression (NMS) to the proposal region. NMS with a threshold of 0.7 is used. For the ImageNet-pre-trained network, we use a fast version of the Zeiler and Fergus model (ZF) that has 5 convolutional layers and 3 fully-connected layers. In this research, we use the ZF network because it is suitable for real-time implementation.

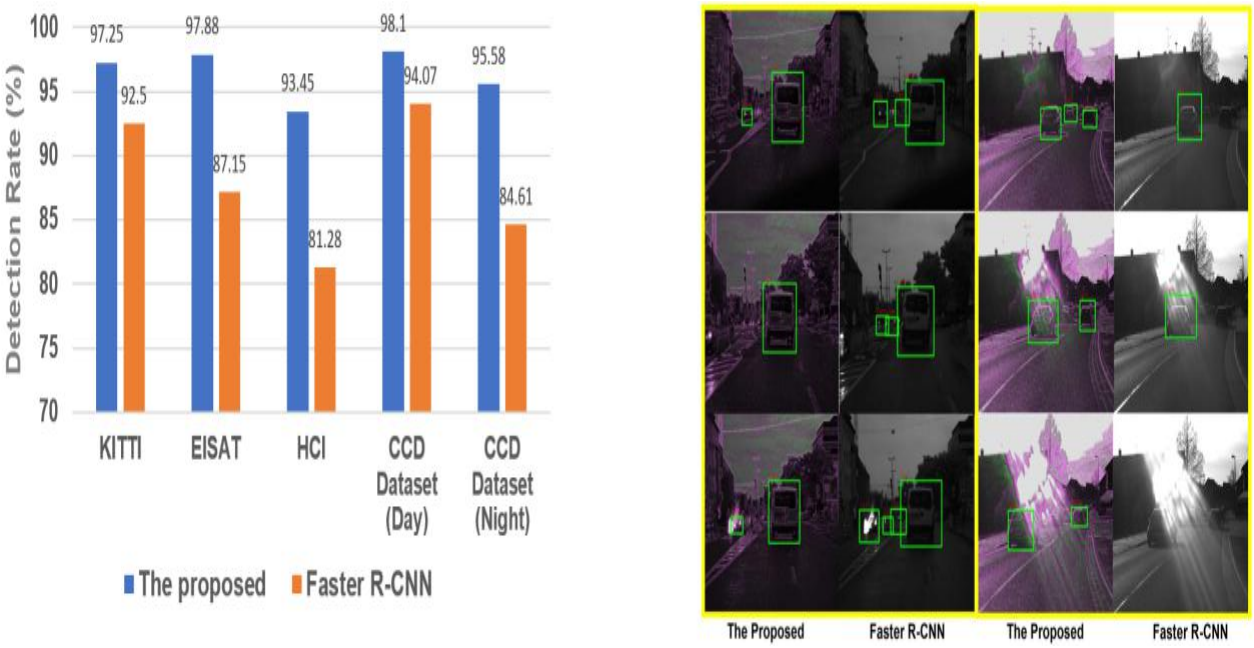

Figure 3.1.1.1: Experimental results of the proposed system and Faster R-CNN under rain-flare and sun-flare conditions.

Figure 3.1.1.2 : Experimental results of the proposed system-based ZF basenet (left) and Faster R-CNN-based ZF basenet (right) using the KITTI dataset.

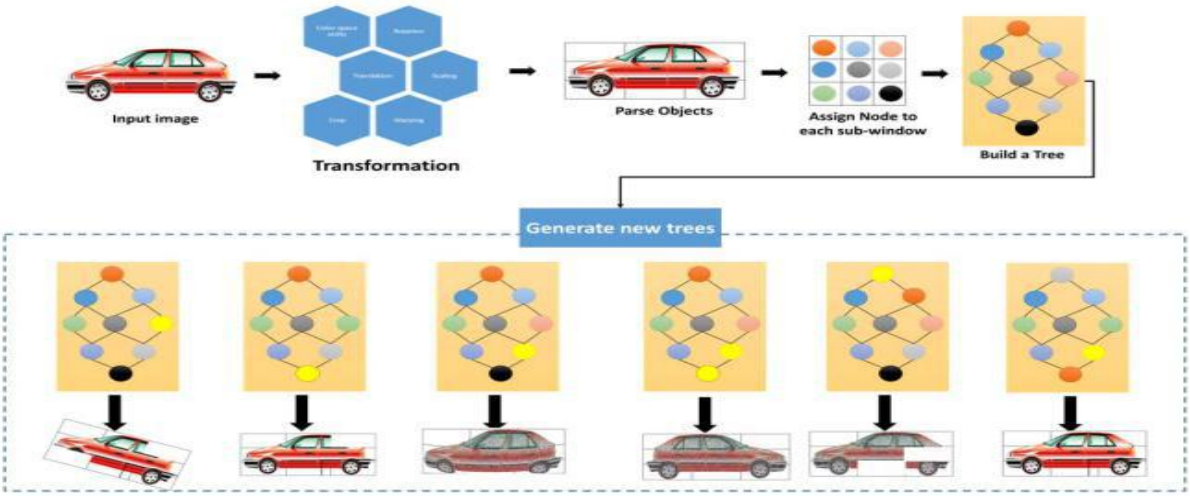

Figure 3.1.1.3: Work flow of the proposed data augmentation algorithm.

3.1.2 ARCHITECTURE

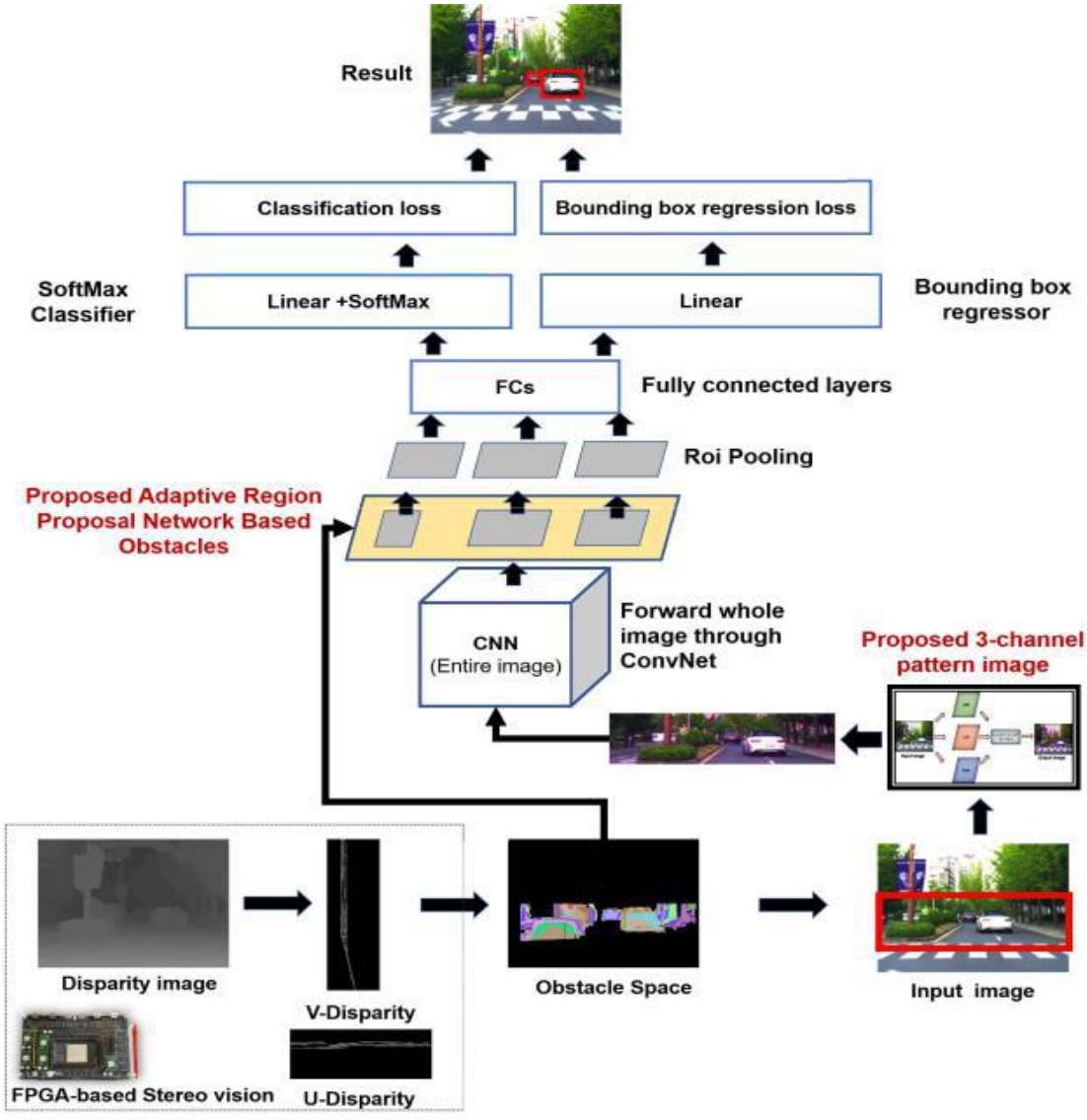

Figure 3.1.2.1: Existing System Architecture

3.1.3 EXISTING SYSTEM LIMITATIONS

- Else there is no counting of detected vehicles.
- Using two and more algorithms to detect the vehicles for accuracy.
- detects only vehicles in a framework.
- It results in poor accuracy on analyzing the data due to the lack of sentiments analyzing.
- The amount of training data will larger.
- Along with we can’t count the pre-trained objects from recorded video streaming.

3.2 PROPOSED SYSTEM

3.2.1 OVERVIEW OF PROPOSED SYSTEM

The proposed system presents an approach to count the moving objects or vehicles of real time scenes by static cameras. Different algorithms and sensors are used for object detection tracking and counting which increases the overall cost and gives less accurate result due to intensity of camera used for recording the video. Most of this method is used in tracking human, car or any type of moving object. Proposed system will work on YOLOv3-(You Only Look Once) algorithm. To represent object their shapes and appearances are important factor.

3.2.2 ARCHITECTURE

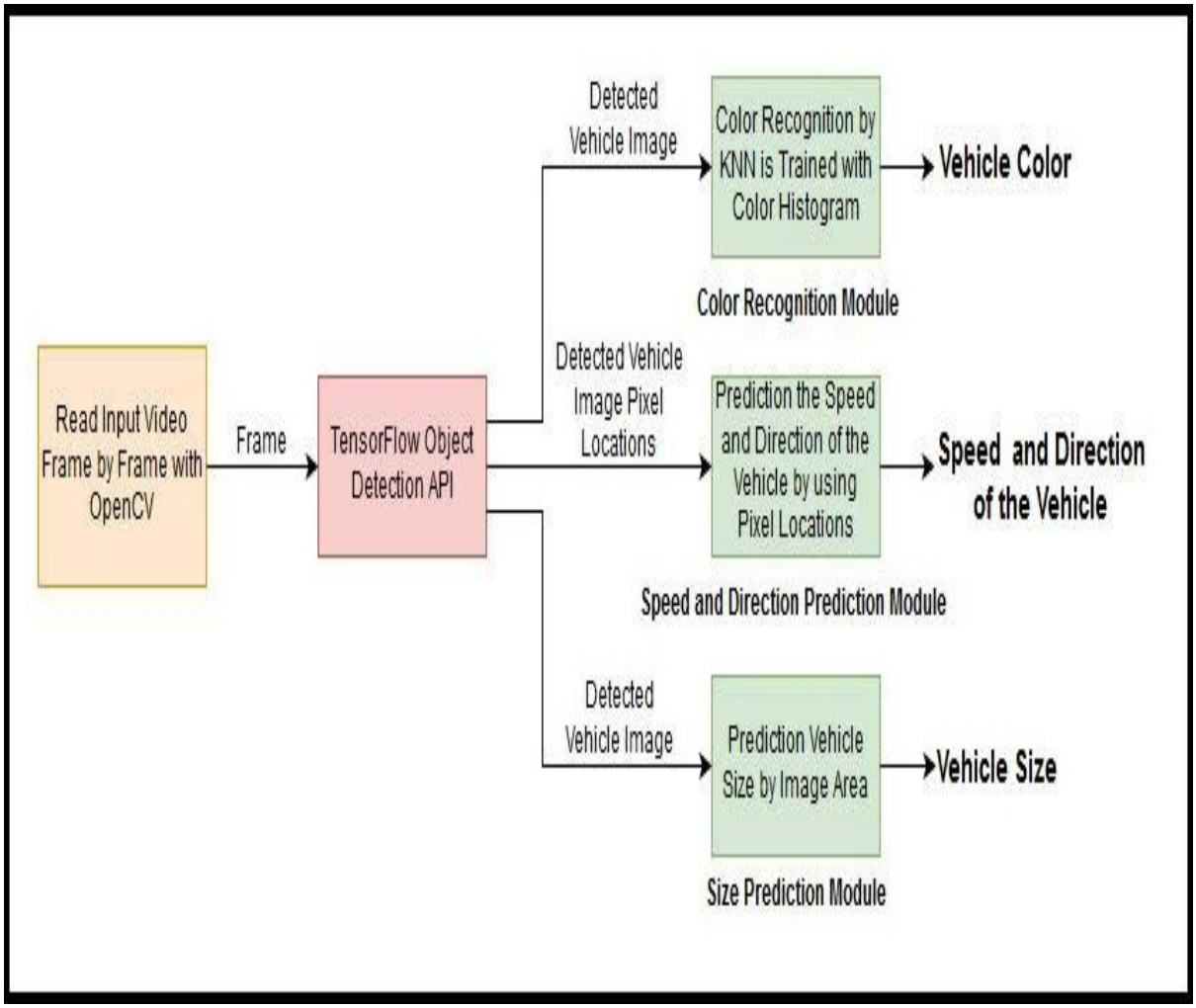

Figure 3.2.2.1: Proposed System Architecture

3.3 IMPLEMENTATION OF PROPOSED MODEL

3.3.1 ALGORITHMS OF PROPOSED WORK

- RCNN
- Fast RCNN
- Faster RCNN
- YOLO

3.3.1.1.YOLO OBJECT DETECTION

YOLO ("You Only Look Once") is an effective real-time object recognition algorithm, first described in the seminal 2015 paper by Joseph Redmon et al. In this article we introduce the concept of object detection, the YOLO algorithm itself, and one of the algorithm's open source implementations: Darknet.

Image classification is one of the many exciting applications of convolutional neural networks. Aside from simple image classification, there are plenty of fascinating problems in computer vision, with object detection being one of the most interesting. It is commonly associated with self-driving cars where systems blend computer vision, LIDAR and other technologies to generate a multidimensional representation of the road with all its participants. Object detection is also commonly used in video surveillance, especially in crowd monitoring to prevent terrorist attacks, count people for general statistics or analyze customer experience with walking paths within shopping centers.

3.3.1.2. YOLO V3 ALGORITHM

YOLO (You Only Look Once) is a solution which is much accurate and faster than the sliding window algorithm. There is a minor tweak in the algorithms. The image is divided into multiple grids. The

label of the data is changed so that the classification and localization algorithm can be used for each grid cell. The algorithm proceeds as follows:

1. The image is divided into multiple grids. 4x4 grids are drawn in the figure, but the actual implementation of YOLO has a different number of grids.

2. The training data is labelled. If the number of unique objects in the data is C, the number of grids into which the image is split would be S*S. The output length of the vector would be S*S*(C+5).

3. A deep CNN is made with loss function as the error between the label vector and output activations. The model predicts the output of all the grids in a forward pass of the input image through CNN.

4. The label for the object present in a grid cell is determined by the presence of the centroid of the object in the grid. This helps to ensure that the object is not counted multiple times in different grids.

Algorithms based on classification. They are implemented in two stages. First, they select regions of interest in an image. Second, they classify these regions using convolutional neural networks. This solution can be slow because we have to run predictions for every selected region. A widely known example of this type of algorithm is the Region-based convolutional neural network (RCNN) and its cousins Fast-RCNN, Faster-RCNN and the latest addition to the family: Mask-RCNN. Another example is RetinaNet.

Algorithms based on regression – instead of selecting interesting parts of an image, they predict classes and bounding boxes for the whole image in one run of the algorithm. The two best known examples from this group are the YOLO (You Only Look Once) family algorithms and SSD (Single Shot Multibox Detector). They are commonly used for real-time object detection as, in general, they trade a bit of accuracy for large improvements in speed.

To understand the YOLO algorithm, it is necessary to establish what is actually being predicted. Ultimately, we aim to predict a class of an object and the bounding box specifying object location.

Each bounding box can be described using four descriptors:

- center of a bounding box (bxby)
- width (bw)
- height (bh)
- value is corresponding to a class of an object (such as: car, traffic lights, etc.).

In addition, we have to predict the pc value, which is the probability that there is an object in the bounding box.

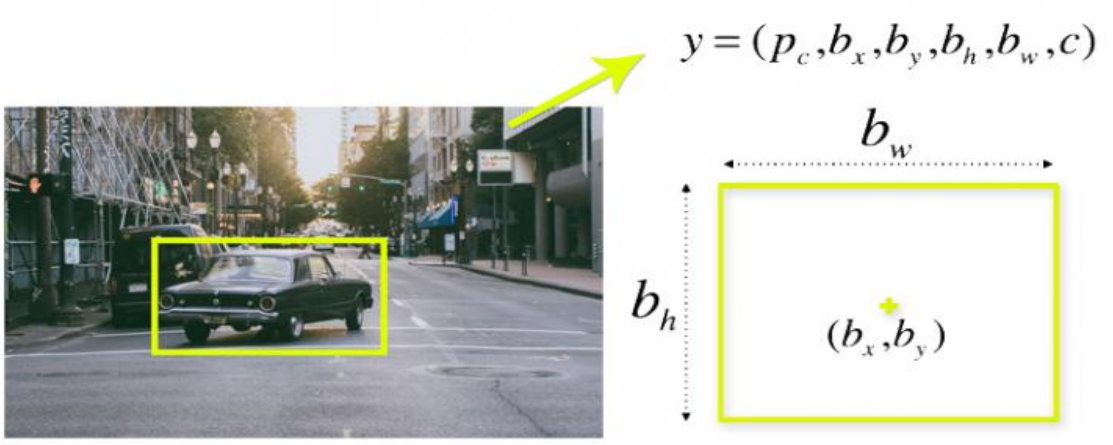

Figure 3.3.1.2.1: Probability of Object in Boundary Boxes.

As we mentioned above, when working with the YOLO algorithm we are not searching for interesting regions in our image that could potentially contain an object.

Instead, we are splitting our image into cells, typically using a 19×19 grid. Each cell is responsible for predicting 5 bounding boxes (in case there is more than one object in this cell). Therefore, we arrive at a large number of 1805 bounding boxes for one image.

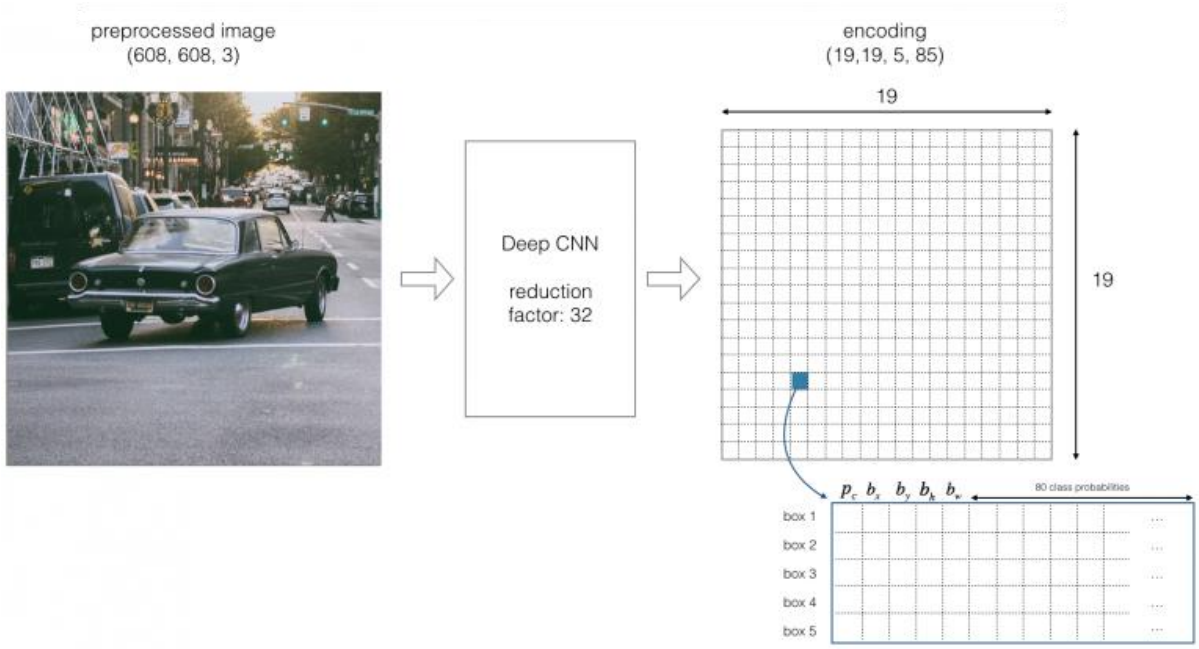

Figure 3.3.1.2.2: Data Augmentation of Object During Deep CNN.

Most of these cells and bounding boxes will not contain an object. Therefore, we predict the value pc, which serves to remove boxes with low object probability and bounding boxes with the highest shared area in a process called non-max suppression.

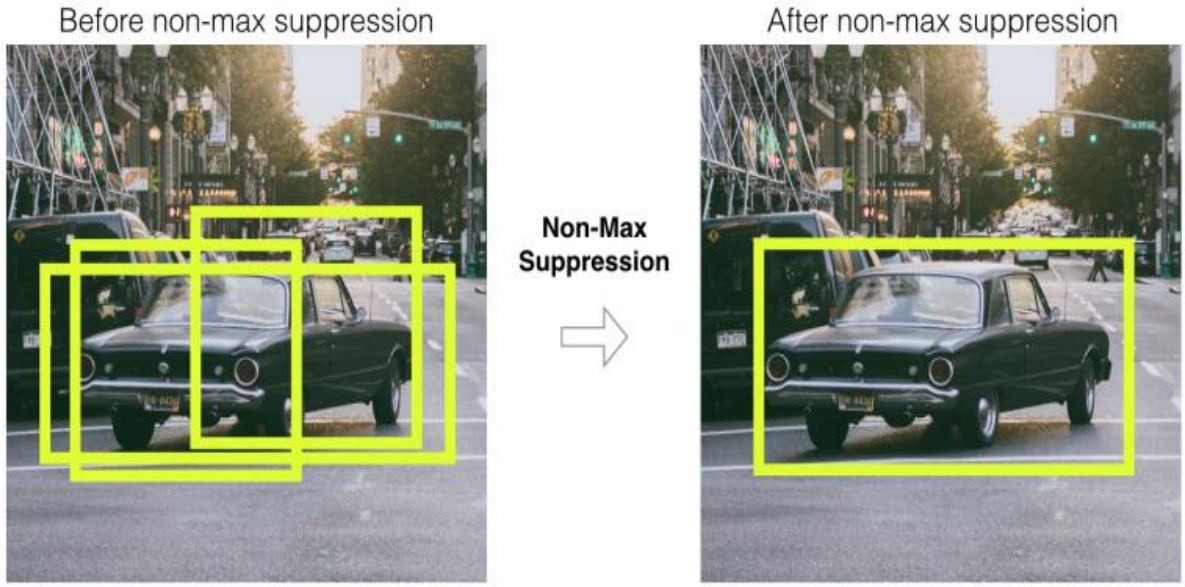

Figure 3.3.1.2.3: Suppersions of Real Time Objects

Darknet prints out the objects it detected, its confidence, and how long it took to find them. We didn't compile Darknet with OpenCV so it can't display the detections directly. Instead, it saves them in predictions.png. You can open it to see the detected objects. Since we are using Darknet on the CPU it takes around 6-12 seconds per image. If we use the GPU version it would be much faster.

3.3.2. WHAT'S NEW VERSION IN YOLO VERSION 3

YOLOv3 uses a few tricks to improve training and increase performance, including: multi-scale predictions, a better backbone classifier, and more. The full details are in our paper!

3.3.3.DARKNET – A YOLO IMPLEMENTATION

There are a few different implementations of the YOLO algorithm on the web. Darknet is one such open source neural network framework(a PyTorch implementation can be found here or with some extra fast.ai functionality here; a Keras implementation can be found here). Darknet was written in the C Language and CUDA technology, which makes it really fast and provides for making computations on a GPU, which is essential for real-time predictions.
Darknet computer vision.

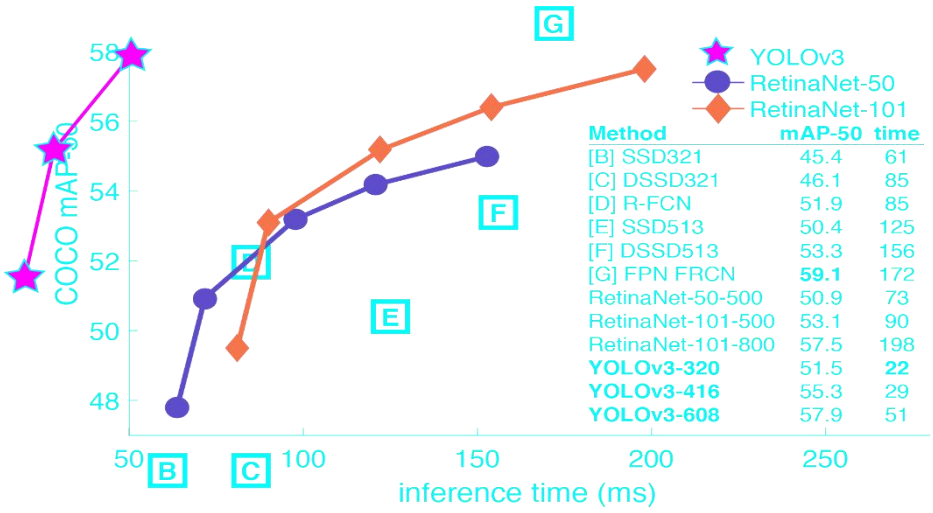

Figure 3.3.3.1 : Performance on the COCO Dataset

3.3.4. COCO Dataset on YOLO

Model	Train	Test	mAP	FLOPS	FPS	cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		Link
SSD500	COCO trainval	test-dev	46.5	-	19		Link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

Table 3.3.4.1 : COCO Dataset.

HOW IT WORKS?

Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections.

We use a totally different approach. We apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

Our model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN. See our paper for more details on the full system.

3.4 DESIGN OF PROPOSED MODEL

3.4.1. UML DIAGRAM

UML stands for Unified Modelling Language. UML is a standardized generalpurpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: A Meta-model and a notation. In the future, some form of method or process may also be added to; or associated With, UML. 38 The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artefacts of software system, as well as for business modelling and other nonsoftware.

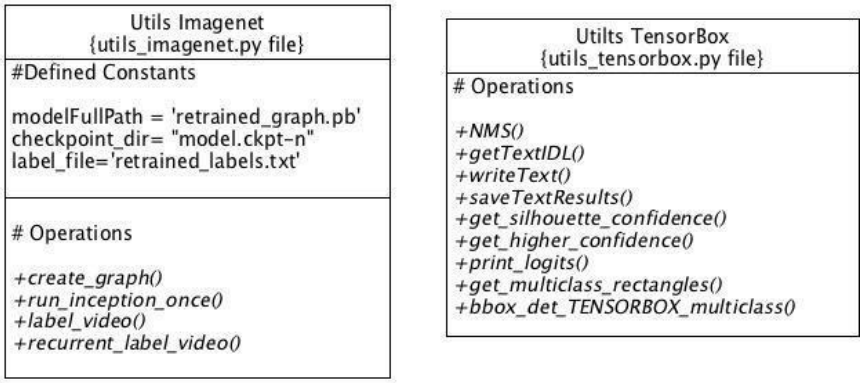

Figure 3.4.1.1 : UML diagram of the interface classes implemented into the project.

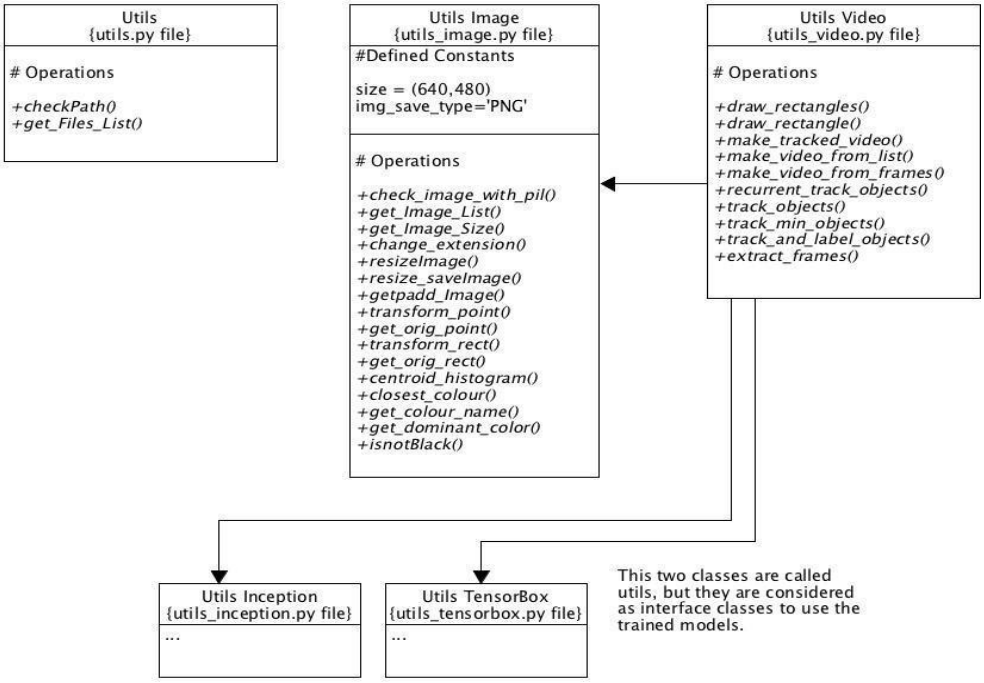

Figure 3.4.1.2 : UML diagram of the utils classes implemented into the project.

3.5 MERITS OF PROPOSED SYSTEM

- Creates easy platform to detect the ojects.
- Easily overcomes the limitations of existing work.
- Decreasing the usage of algorithms for consume less number of time to execute the counts of detected objects.
- Easy deployment (Python pip package manager deployed by TensorFlow facilitates easy installation).
- Better support for GPUs as compared to other models.
- It provides high level APIs for building models.
- It is extremely easy to do unconventional and hard-core changes.

CHAPTER 4

4.MODULES

4.1 IMAGE CLASSIFICATION

Aims at assigning an image to one of a number of different categories (e.g. car, dog, cat, human, etc.), essentially answering the question “What is in this picture?”. One image has only one category assigned to it.

4.2 OBJECT LOCALIZATION

Then allows us to locate our object in the image, so our question changes to “What is it and where it is?”.

In a real real-life scenario, we need to go beyond locating just one object but rather multiple objects in one image. For example, a self-driving car has to find the location of other cars, traffic lights, signs, humans and to take appropriate action based on this information

4.3 OBJECT DETECTION

Provides the tools for doing just that – finding all the objects in an image and drawing the so-called bounding boxes around them. There are also some situations where we want to find exact boundaries of our objects in the process called instance segmentation, but this is a topic for another post.

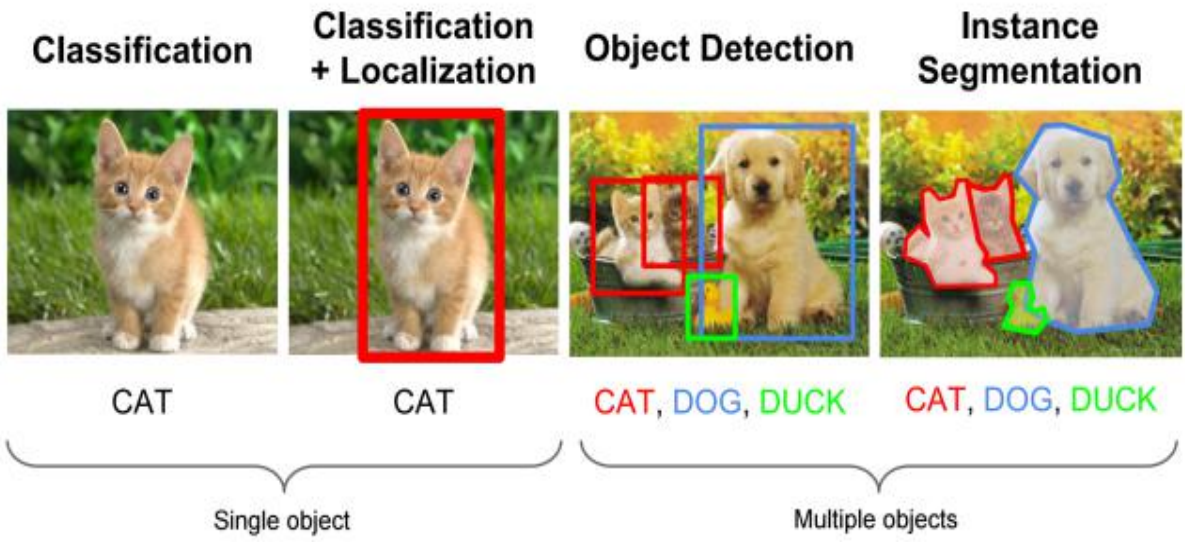

Figure 4.4 : Object Detection Overview

CHAPTER 5

5.SYSTEM REQUIREMENTS

The purpose of software requirements specification is to provide a detailed overview of the software project, its parameters and goals. This describes the project target audience and its user interface, hardware and software requirements. It defines how the client, team and audience see the project and its functionality.

5.1 HARDWARE REQUIREMENTS:

- System : Intel Dual core 2.4 GHz.
- Hard Disk : 70GB (min)
- RAM : 4GB (min)
- Monitor : Color Monitor
- Mouse : Wired or Wireless Mouse
- Keybord : Standard Windows Keybord

5.2 SOFTWARE REQUIREMENTS:

- Tools : Linux Terminal
- Front end : Python3
- Backend : Python Flask
- Operating system : Linux (or) Ubuntu

CHAPTER 6

6. IMPLEMENTATIONS

6.1 LANGUAGE USED

6.1.1 PYTHON 3 – FRONT END

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991.

The Python Programming Language

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Advantages of python

- Improved Productivity
- Interpreted Language
- Dynamically Typed
- Portable
- Free and Open-Source
- High performance
- Vast Libreries Support

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Python programming language is unusual in that a program is interpreted.

6.1.2 PYTHON FLASK – BACK END

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more frequently than the core Flask program.

Features

- Development server and debugger
- Integrated support for unit testing
- RESTful request dispatching
- Uses Jinja templating
- Support for secure cookies (client side sessions)
- 100% WSGI 1.0 compliant
- Unicode-based
- Extensive documentation
- Google App Engine compatibility
- Extensions available to enhance features desire

6.2 IMAGE PROCESSING LIBRARIES IN PYTHON

6.2.1 OPENCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision.Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

Applications

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks

Hard acceleration

- If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.
- A CUDA-based GPU interface has been in progress since September 2010.[13]
- An OpenCL-based GPU interface has been in progress since October 2012,[14] documentation for version 2.4.13.3 can be found at docs.opencv.org.

6.2.2 DARKFLOW

DarkFlow is a node based image processor for astronomers.
This software is dedicated to astronomical images processing. It gives you the opportunity to track the processing of your images using a non destructive approach. So you no longer need to use your memory to replay treatments you have taken some much time to implement.

6.2.3 SCIKIT-LEARN

Scikit-learn is largely written in Python, and uses numpy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around

LIBLINEAR. In such cases, extending these methods with Python may not be possible.

6.2.4 CYTHON

Cython is a programming language that aims to be a superset of the Python programming language, designed to give C-like performance with code that is written mostly in Python with optional additional C-inspired syntax.

Cython is a compiled language that is typically used to generate CPython extension modules. Annotated Python-like code is compiled to C or C++ then automatically wrapped in interface code, producing extension modules that can

be loaded and used by regular Python code using the import statement, but with significantly less computational overhead at run time. Cython also facilitates wrapping independent C or C++ code into python-importable modules.
Cython is written in Python and C and works on Windows, macOS, and Linux, producing source files compatible with CPython 2.6, 2.7, and 3.3 through 3.8.

Cython works by producing a standard Python module. However, the behavior differs from standard Python in that the module code, originally written in Python, is translated into C. While the resulting code is fast, it makes many calls into the CPython interpreter and CPython standard libraries to perform actual work. Choosing this arrangement saved considerably on Cython's development time, but modules have a dependency on the Python interpreter and standard library.

6.2.5 NUMPY

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

NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents.

NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops, using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[17] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

CHAPTER 7

7.CONCLUSION

Our project is totally about improving the accuracy of object detection and objects counting(Real-Time), since it is a major technical that is very much needed for various applications. Some of the applications that can enhanced by our project are as follows, human-computer interaction (HCI), robotics (e.g., service robots), consumer electronics like smart-phones, security (e.g., recognition, tracking), retrieval (e.g., search engines, photo management), and transportation (e.g., autonomous and assisted driving). Our project is an extension of Google's object detection API. The improvement of this technology may drastically change the live if millions. The above said applications are most popular current applications of our project.

The futuristic scope of this project is practically limitless. The object detection and Counting is a major part of futuristic robots as robots could use this tech to learn their environment. This tech can be used in surveillance drones to detect hostilities during battles. Object counting and detection is a key ability for most computer and robot vision system. Although great progress has been observed in the last years, we are still far from achieving human-level performance, in particular in terms of open-world learning.It should be noted that object detection has not been used much in many areas where it could be of great help. As mobile robots, and in general autonomous machines, are starting to be more widely deployed, the need of object detection systems is gaining more importance. The object detection system could be usedfor robots that will explore areas that have not been seen by humans, such as depth parts of the sea or other planets, and the counting systems(Real-time) will have to learn to new object classes as they are encountered.

CHAPTER 8

8. APPENDIX

8.1 SCREENSHOTS

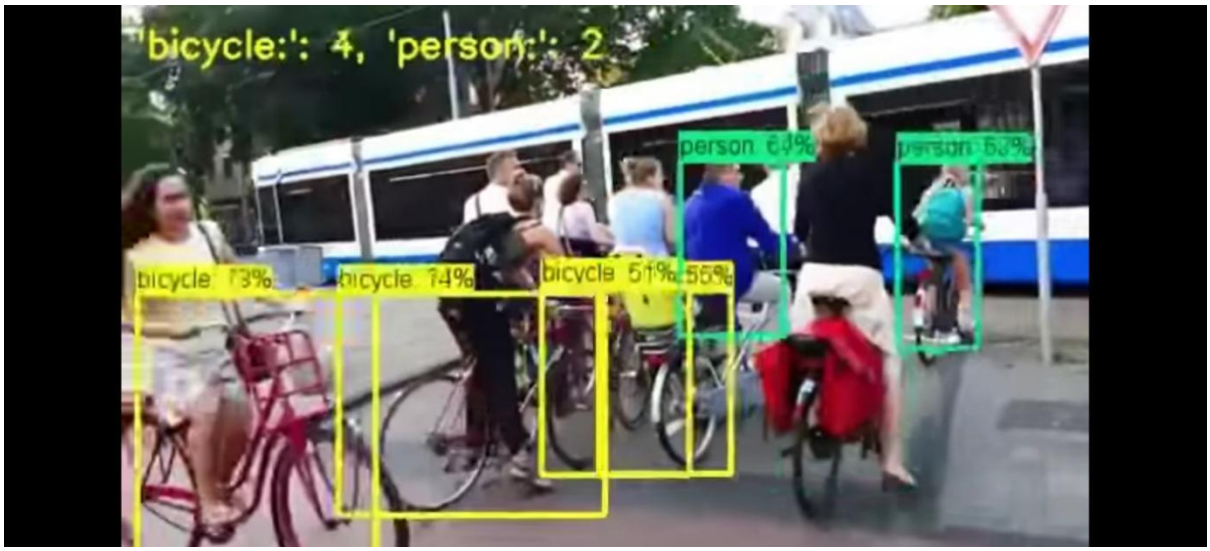

Figure 8.1.1: Object Detection Result 1

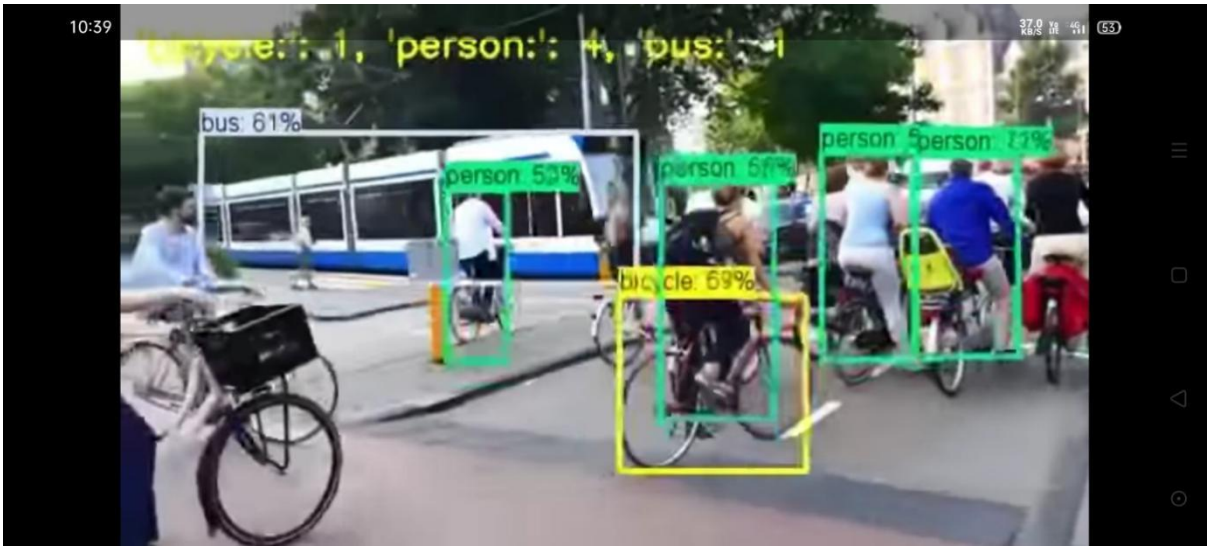

Figure 8.1.2 : Object Detection and Counting Result 2.

Figure 8.1.3 : Object Detection and Counting Result 3.

Figure 8.1.4 : Object Detection and Counting Result 4.

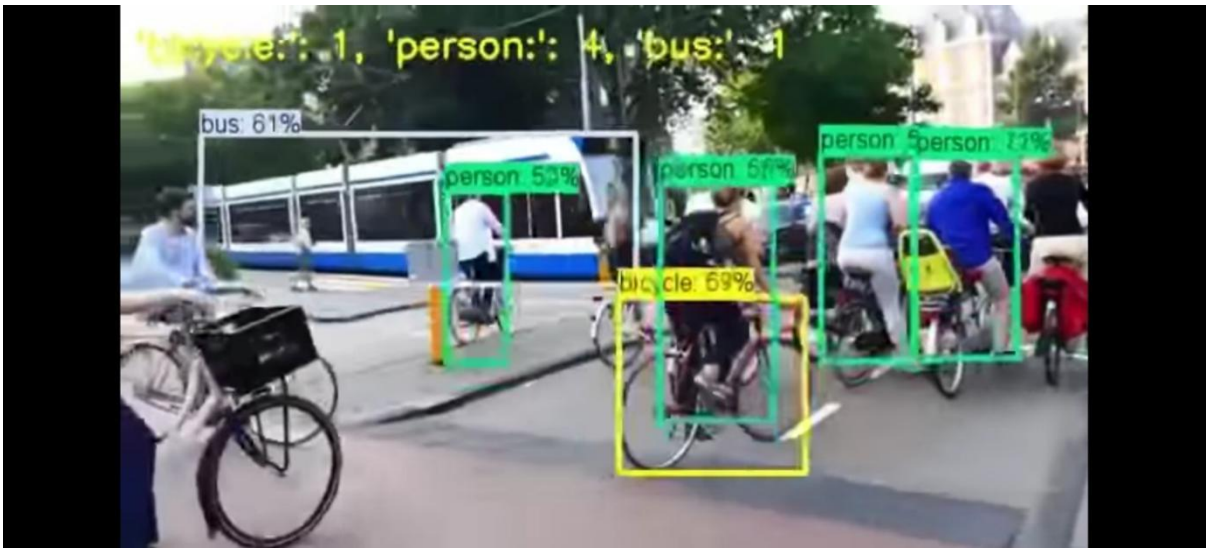

Figure 8.1.5 : Object Detection and Counting Result 5.

Figure 8.1.6 : Object Detection and Counting Result 6.

8.2 REAL – TIME SCREENSHOTS

Figure 8.2.1 : Object Detection and Counting Result 7.

Figure 8.2.2 : Object Detection and Counting Result 8.

8.3 SOURCE CODE

count_cars_crosing_virtual_line .py

```
import cv2
from object_counting_api import ObjectCountingAPI
options = {"model": "cfg/yolov2.cfg", "load": "cfg/yolov2.weights", "threshold":
0.6, "gpu": 1.0}
VIDEO_PATH = "inputs/The Dancing Traffic Light Manikin by smart.mp4"
cap = cv2.VideoCapture(VIDEO_PATH)
counter = ObjectCountingAPI(options)
counter.count_objects_on_video(cap, show=True)
```

count_objects_from_camera.py

```
import cv2
from object_counting_api import ObjectCountingAPI
options = {"model": "cfg/yolov2.cfg", "load": "cfg/yolov2.weights", "threshold":
0.5, "gpu": 1.0}
cap = cv2.VideoCapture(0)
counter = ObjectCountingAPI(options)
counter.count_objects_on_video(cap, show=True)
```

count_people_on_image.py

```
import cv2
from object_counting_api import ObjectCountingAPI
options = {"model": "cfg/yolov2.cfg", "load": "cfg/yolov2.weights", "threshold":
0.5, "gpu": 1.0}
IMG_PATH = "inputs/3.jpg"
```

```
img = cv2.imread(IMG_PATH)
counter = ObjectCountingAPI(options)
counter.count_objects_on_image(img, targeted_classes=[], show=True)
```

count_person.py

```
import cv2
from object_counting_api import ObjectCountingAPI
options = {"model": "cfg/yolov2.cfg", "load": "cfg/yolov2.weights", "threshold":
0.6, "gpu": 1.0}
VIDEO_PATH = "inputs/pedestrians.mp4"
cap = cv2.VideoCapture(VIDEO_PATH)
counter = ObjectCountingAPI(options)
counter.count_objects_on_video(cap, show=True)
```

object_counting_api.py

```
from darkflow.net.build import TFNet
from sort import Sort
from utils import COLORS, intersect, get_output_fps_height_and_width
import cv2
import numpy as np
DETECTION_FRAME_THICKNESS = 1
OBJECTS_ON_FRAME_COUNTER_FONT =
cv2.FONT_HERSHEY_SIMPLEX
OBJECTS_ON_FRAME_COUNTER_FONT_SIZE = 0.5
LINE_COLOR = (0, 0, 255)
LINE_THICKNESS = 3
```

```
LINE_COUNTER_FONT = cv2.FONT_HERSHEY_DUPLEX
LINE_COUNTER_FONT_SIZE = 2.0
LINE_COUNTER_POSITION = (20, 45)
class ObjectCountingAPI:
    def __init__(self, options):
        self.options = options
        self.tfnet = TFNet(options)
    def _write_quantities(self, frame, labels_quantities_dic):
        for i, (label, quantity) in enumerate(labels_quantities_dic.items()):
            class_id = [i for i, x in enumerate(labels_quantities_dic.keys()) if x ==
label][0]
            color = [int(c) for c in COLORS[class_id % len(COLORS)]]
            cv2.putText(
                frame,
                f"{label}: {quantity}",
                (10, (i + 1) * 35),
                OBJECTS_ON_FRAME_COUNTER_FONT,
                OBJECTS_ON_FRAME_COUNTER_FONT_SIZE,
                color,
                2,
                cv2.FONT_HERSHEY_SIMPLEX,
            )
    def _draw_detection_results(self, frame, results, labels_quantities_dic):
        for start_point, end_point, label, confidence in results:
            x1, y1 = start_point
            class_id = [i for i, x in enumerate(labels_quantities_dic.keys()) if x ==
label][0]
            color = [int(c) for c in COLORS[class_id % len(COLORS)]]
            cv2.rectangle(frame, start_point, end_point, color,
```

```
DETECTION_FRAME_THICKNESS)
            cv2.putText(frame, label, (x1, y1 - 5),
OBJECTS_ON_FRAME_COUNTER_FONT,
OBJECTS_ON_FRAME_COUNTER_FONT_SIZE, color, 2)

def_convert_detections_into_list_of_tuples_and_count_quantity_of_each_label(
          self, objects):
        labels_quantities_dic = {}
        results = []
        for object in objects:
            x1, y1 = object["topleft"]["x"], object["topleft"]["y"]
            x2, y2 = object["bottomright"]["x"], object["bottomright"]["y"]
            confidence = object["confidence"]
            label = object["label"]
            try:
                labels_quantities_dic[label] += 1
            except KeyError:
                labels_quantities_dic[label] = 1
            start_point = (x1, y1)
            end_point = (x2, y2)
            results.append((start_point, end_point, label, confidence))
        return results, labels_quantities_dic
    def count_objects_on_image(self, frame, targeted_classes=[],
output_path="count_people_output.jpg", show=True):
        objects = self.tfnet.return_predict(frame)
        if targeted_classes:
            objects = list(filter(lambda res: res["label"] in targeted_classes, objects))
```

```
results, labels_quantities_dic =
self._convert_detections_into_list_of_tuples_and_count_quantity_of_each_label(objects)

      self._draw_detection_results(frame, results, labels_quantities_dic)
      self._write_quantities(frame, labels_quantities_dic)
       if show:
           cv2.imshow("frame", frame)
           cv2.waitKey()
           cv2.destroyAllWindows()
       cv2.imwrite(output_path, frame)
       # return frame, objects
   def count_objects_on_video(self, cap, targeted_classes=[],
output_path="the_output.avi", show=True):
       ret, frame = cap.read()
       fps, height, width = get_output_fps_height_and_width(cap)
       fourcc = cv2.VideoWriter_fourcc(*'XVID')
       output_movie = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
       while ret:
           objects = self.tfnet.return_predict(frame)
           if targeted_classes:
               objects = list(filter(lambda res: res["label"] in targeted_classes,
objects))
           results, labels_quantities_dic =
self._convert_detections_into_list_of_tuples_and_count_quantity_of_each_label(objects)
           self._draw_detection_results(frame, results, labels_quantities_dic)
           self._write_quantities(frame, labels_quantities_dic)
```

```
output_movie.write(frame)
            if show:
                cv2.imshow('frame', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break


            ret, frame = cap.read()
        cap.release()
        cv2.destroyAllWindows()
    def count_objects_crossing_the_virtual_line(self, cap, line_begin, line_end,
targeted_classes=[],output_path="the_output.avi", show=True):
        ret, frame = cap.read()
        fps, height, width = get_output_fps_height_and_width(cap)
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        output_movie = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
        tracker = Sort()
        memory = {}
        line = [line_begin, line_end]
        counter = 0
        while ret:
            objects = self.tfnet.return_predict(frame)
            if targeted_classes:
                objects = list(filter(lambda res: res["label"] in targeted_classes,
objects))
            results, _ =
self._convert_detections_into_list_of_tuples_and_count_quantity_of_each_labe
l(objects)
            # convert to format required for dets [x1, y1, x2, y2, confidence]
```

```
 dets = [[*start_point, *end_point] for (start_point, end_point, label, confidence)
in results]
        np.set_printoptions(formatter={'float': lambda x:
"{0:0.3f}".format(100)})
        dets = np.asarray(dets)
        tracks = tracker.update(dets)
        boxes = []

        indexIDs = []
        previous = memory.copy()
        memory = {}
        for track in tracks:
            boxes.append([track[0], track[1], track[2], track[3]])
            indexIDs.append(int(track[4]))
            memory[indexIDs[-1]] = boxes[-1]
        if len(boxes) > 0:
            i = int(0)
            for box in boxes:
                (x, y) = (int(box[0]), int(box[1]))
                (w, h) = (int(box[2]), int(box[3]))
                color = [int(c) for c in COLORS[indexIDs[i] % len(COLORS)]]
                cv2.rectangle(frame, (x, y), (w, h), color,
DETECTION_FRAME_THICKNESS)
                if indexIDs[i] in previous:
                    previous_box = previous[indexIDs[i]]
                    (x2, y2) = (int(previous_box[0]), int(previous_box[1]))
                    (w2, h2) = (int(previous_box[2]), int(previous_box[3]))
                    p0 = (int(x + (w - x) / 2), int(y + (h - y) / 2))
```

```
                    p1 = (int(x2 + (w2 - x2) / 2), int(y2 + (h2 - y2) / 2))
                    cv2.line(frame, p0, p1, color, 3)
                    if intersect(p0, p1, line[0], line[1]):
                        counter += 1
                text = "{}".format(indexIDs[i])
                cv2.putText(frame, text, (x, y - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
                i += 1
        cv2.line(frame, line[0], line[1], LINE_COLOR, LINE_THICKNESS)

        cv2.putText(frame, str(counter), LINE_COUNTER_POSITION,
LINE_COUNTER_FONT, LINE_COUNTER_FONT_SIZE,
                    LINE_COLOR, 2)
        output_movie.write(frame)
        if show:
            cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        ret, frame = cap.read()
    cap.release()
    cv2.destroyAllWindows()
if __name__ == '__main__':
    options = {"model": "cfg/yolov2.cfg", "load": "bin/yolov2.weights",
"threshold": 0.5, "gpu": 1.0}
    img = cv2.imread("sample_inputs/united_nations.jpg")
    VIDEO_PATH = "sample_inputs/highway_traffic.mp4"
    cap = cv2.VideoCapture(VIDEO_PATH)
    counter = ObjectCountingAPI(options)
```

```
counter.count_objects_crossing_the_virtual_line(cap, line_begin=(100, 300),
line_end=(320, 250), show=True)
   # counter.count_objects_on_image(img, targeted_classes=["person"],
show=True)
```

darknet.py

```
from ..utils.process import cfg_yielder
from .darkop import create_darkop
from ..utils import loader
import warnings

import time
import os
class Darknet(object):
   _EXT = '.weights'
   def __init__(self, FLAGS):
      self.get_weight_src(FLAGS)
      self.modify = False
      print('Parsing {}'.format(self.src_cfg))
      src_parsed = self.parse_cfg(self.src_cfg, FLAGS)
      self.src_meta, self.src_layers = src_parsed
      if self.src_cfg == FLAGS.model:
         self.meta, self.layers = src_parsed
      else:
        print('Parsing {}'.format(FLAGS.model))
        des_parsed = self.parse_cfg(FLAGS.model, FLAGS)
        self.meta, self.layers = des_parsed
```

```
        self.load_weights()
    def get_weight_src(self, FLAGS):
        self.src_bin = FLAGS.model + self._EXT
        self.src_bin = FLAGS.binary + self.src_bin
        self.src_bin = os.path.abspath(self.src_bin)
        exist = os.path.isfile(self.src_bin)
        if FLAGS.load == str(): FLAGS.load = int()
        if type(FLAGS.load) is int:
            self.src_cfg = FLAGS.model
            if FLAGS.load: self.src_bin = None
            elif not exist: self.src_bin = None
        else:
            assert os.path.isfile(FLAGS.load), '{} not found'.format(FLAGS.load)
            self.src_bin = FLAGS.load
            name = loader.model_name(FLAGS.load)
            cfg_path = os.path.join(FLAGS.config, name + '.cfg')
            if not os.path.isfile(cfg_path):
                warnings.warn(
                    '{} not found, use {} instead'.format(
                    cfg_path, FLAGS.model))
                cfg_path = FLAGS.model
            self.src_cfg = cfg_path
            FLAGS.load = int()
    def parse_cfg(self, model, FLAGS):
        args = [model, FLAGS.binary]
        cfg_layers = cfg_yielder(*args)
        meta = dict(); layers = list()
        for i, info in enumerate(cfg_layers):
```

```
            if i == 0: meta = info; continue
            else: new = create_darkop(*info)
            layers.append(new)
        return meta, layers
    def load_weights(self):
        print('Loading {} ...'.format(self.src_bin))
        start = time.time()
        args = [self.src_bin, self.src_layers]
        wgts_loader = loader.create_loader(*args)
        for layer in self.layers: layer.load(wgts_loader)
        stop = time.time()
        print('Finished in {}s'.format(stop - start))
```

train.py

```
import tensorflow.contrib.slim as slim
import pickle
import tensorflow as tf
from .misc import show
import numpy as np
import os
def loss(self, net_out):
    # meta
    m = self.meta
    sprob = float(m['class_scale'])
    sconf = float(m['object_scale'])
    snoob = float(m['noobject_scale'])
    scoor = float(m['coord_scale'])
    S, B, C = m['side'], m['num'], m['classes']
```

```
SS = S * S # number of grid cells
print('{} loss hyper-parameters:'.format(m['model']))
print('\tside    = {}'.format(m['side']))
print('\tbox     = {}'.format(m['num']))
print('\tclasses = {}'.format(m['classes']))
print('\tscales  = {}'.format([sprob, sconf, snoob, scoor]))
size1 = [None, SS, C]
size2 = [None, SS, B]
# return the below placeholders
_probs = tf.placeholder(tf.float32, size1)
_confs = tf.placeholder(tf.float32, size2)
_coord = tf.placeholder(tf.float32, size2 + [4])
# weights term for L2 loss
_proid = tf.placeholder(tf.float32, size1)
# material calculating IOU
_areas = tf.placeholder(tf.float32, size2)
_upleft = tf.placeholder(tf.float32, size2 + [2])
_botright = tf.placeholder(tf.float32, size2 + [2])
self.placeholders = {
   'probs':_probs, 'confs':_confs, 'coord':_coord, 'proid':_proid,
   'areas':_areas, 'upleft':_upleft, 'botright':_botright
}
# Extract the coordinate prediction from net.out
coords = net_out[:, SS * (C + B):]
coords = tf.reshape(coords, [-1, SS, B, 4])
wh = tf.pow(coords[:,:,:,2:4], 2) * S # unit: grid cell
area_pred = wh[:,:,:,0] * wh[:,:,:,1] # unit: grid cell^2
centers = coords[:,:,:,0:2] # [batch, SS, B, 2]
```

```
floor = centers - (wh * .5) # [batch, SS, B, 2]
    ceil  = centers + (wh * .5) # [batch, SS, B, 2]
    # calculate the intersection areas
    intersect_upleft   = tf.maximum(floor, _upleft)
    intersect_botright = tf.minimum(ceil , _botright)
    intersect_wh = intersect_botright - intersect_upleft
    intersect_wh = tf.maximum(intersect_wh, 0.0)
    intersect = tf.multiply(intersect_wh[:,:,:,0], intersect_wh[:,:,:,1])
    # calculate the best IOU, set 0.0 confidence for worse boxes
    iou = tf.truediv(intersect, _areas + area_pred - intersect)
    best_box = tf.equal(iou, tf.reduce_max(iou, [2], True))
    best_box = tf.to_float(best_box)
    confs = tf.multiply(best_box, _confs)

    # take care of the weight terms
    conid = snoob * (1. - confs) + sconf * confs
    weight_coo = tf.concat(4 * [tf.expand_dims(confs, -1)], 3)
    cooid = scoor * weight_coo
    proid = sprob * _proid
    # flatten 'em all
    probs = slim.flatten(_probs)
    proid = slim.flatten(proid)
    confs = slim.flatten(confs)
    conid = slim.flatten(conid)
    coord = slim.flatten(_coord)
    cooid = slim.flatten(cooid)
    self.fetch += [probs, confs, conid, cooid, proid]
    true = tf.concat([probs, confs, coord], 1)
```

```
    wght = tf.concat([proid, conid, cooid], 1)
    print('Building {} loss'.format(m['model']))
    loss = tf.pow(net_out - true, 2)
    loss = tf.multiply(loss, wght)
    loss = tf.reduce_sum(loss, 1)
    self.loss = .5 * tf.reduce_mean(loss)
    tf.summary.scalar('{} loss'.format(m['model']), self.loss)
```

predict.py

```
from ...utils.im_transform import imcv2_recolor, imcv2_affine_trans
from ...utils.box import BoundBox, box_iou, prob_compare
import numpy as np
import cv2
import os
import json

from ...cython_utils.cy_yolo_findboxes import yolo_box_constructor
def _fix(obj, dims, scale, offs):
        for i in range(1, 5):
                dim = dims[(i + 1) % 2]
                off = offs[(i + 1) % 2]
                obj[i] = int(obj[i] * scale - off)
                obj[i] = max(min(obj[i], dim), 0)
def resize_input(self, im):
        h, w, c = self.meta['inp_size']
        imsz = cv2.resize(im, (w, h))
        imsz = imsz / 255.
        imsz = imsz[:,:,::-1]
```

```
        return imsz
def process_box(self, b, h, w, threshold):
        max_indx = np.argmax(b.probs)
        max_prob = b.probs[max_indx]
        label = self.meta['labels'][max_indx]
        if max_prob > threshold:
                left  = int ((b.x - b.w/2.) * w)
                right = int ((b.x + b.w/2.) * w)
                top   = int ((b.y - b.h/2.) * h)
                bot   = int ((b.y + b.h/2.) * h)
                if left  < 0    :  left = 0
                if right > w - 1: right = w - 1
                if top   < 0    :   top = 0
                if bot   > h - 1:   bot = h - 1
                mess = '{}'.format(label)
                return (left, right, top, bot, mess, max_indx, max_prob)
        return None

def findboxes(self, net_out):
        meta, FLAGS = self.meta, self.FLAGS
        threshold = FLAGS.threshold
        boxes = []
        boxes = yolo_box_constructor(meta, net_out, threshold)
        return boxes
def preprocess(self, im, allobj = None):
        if type(im) is not np.ndarray:
                im = cv2.imread(im)
        if allobj is not None: # in training mode
```

```
                result = imcv2_affine_trans(im)
                im, dims, trans_param = result
                scale, offs, flip = trans_param
                for obj in allobj:
                        _fix(obj, dims, scale, offs)
                        if not flip: continue
                        obj_1_ =  obj[1]
                        obj[1] = dims[0] - obj[3]
                        obj[3] = dims[0] - obj_1_
                im = imcv2_recolor(im)
        im = self.resize_input(im)
        if allobj is None: return im
        return im#, np.array(im) # for unit testing
def postprocess(self, net_out, im, save = True):
        meta, FLAGS = self.meta, self.FLAGS
        threshold = FLAGS.threshold
        colors, labels = meta['colors'], meta['labels']
        boxes = self.findboxes(net_out)
        if type(im) is not np.ndarray:
                imgcv = cv2.imread(im)
        else: imgcv = im
        h, w, _ = imgcv.shape
        resultsForJSON = []
        for b in boxes:
                boxResults = self.process_box(b, h, w, threshold)
                if boxResults is None:
                        continue
                left, right, top, bot, mess, max_indx, confidence = boxResults
```

```
            thick = int((h + w) // 300)
            if self.FLAGS.json:
                resultsForJSON.append({"label": mess, "confidence":
float('%.2f' % confidence), "topleft": {"x": left, "y": top}, "bottomright": {"x":
right, "y": bot}})
                continue
            cv2.rectangle(imgcv,
                (left, top), (right, bot),
                self.meta['colors'][max_indx], thick)
            cv2.putText(
                imgcv, mess, (left, top - 12),
                0, 1e-3 * h, self.meta['colors'][max_indx],
                thick // 3)
        if not save: return imgcv
        outfolder = os.path.join(self.FLAGS.imgdir, 'out')
        img_name = os.path.join(outfolder, os.path.basename(im))
        if self.FLAGS.json:
            textJSON = json.dumps(resultsForJSON)



            textFile = os.path.splitext(img_name)[0] + ".json"
            with open(textFile, 'w') as f:
                f.write(textJSON)
            return
        cv2.imwrite(img_name, imgcv)
```

CHAPTER 11

REFERENCES

[1] Chung-Wei Liang and Chia-Feng Juang, ―Moving Object classification using a Combination of Static Appearance Features and Spatial and Temporal Entropy Values of Optical Flows, pp. 01–12, 2015.

[2] H. Mao, S. Yao, T. Tang, B. Li, and Y. Wang, ―Towards Real-Time Object Detection on Embedded System, pp. 417–431, 2018.

[3] Yoginee B. Bramhe(Pethe), P.S. Kulkarni, ―An Implementation of Moving Object Detection, Tracking and Counting Objects for Traffic Surveillance System , pp. 143–148, 2011.

[4] P. S. Khude1, S. S. Pawar2, ―Object Detection, Tracking and Counting using enhanced BMA on Static Background Videos, 2013.

[5] Jean-Francois Rajotte∗‡, Martin Sotir∗†, Cedric Noiseux∗§, Louis-Philippe Noel∗ and Thomas Bertiere_, Object Counting on Low Quality Image: A Case Study for Near Real-Time Traffic Monitoring.

[6] Dr. Jangala. Sasi Kiran1, N. Vijaya Kumar 2, N. Sashi Prabha 3, M. Kavya4, ―A Literature Survey on Digital Image Processing Techniques in Character Recognition on Indian Language, Vol. 6, pp. 2065–2069, 2015.

[7] Yuewei Lin, Yan Tong, Yu Cao, Youjie Zhou, Song Wang, ―Visual Attention Based Background Modelling for Detection Infrequently Moving Objects,pp.01-13.

[8] Shraddha Mane,Prof.Supriya Mangale, ―Moving Object Detection and Tracking using Convolutional Neutural Network,pp.1809-1813, 2018.

[9] S. R. Balaji, Dr. S. Karthikeyan,―A Survey On Moving Object Tracking Using Image Processing, pp.469-474, 2017.

[10] Desiree Blizzard, Somayeh Davar, and Arash Mohammadi― Real_Time and Event-Triggered Object Detection, Recognition, and Tracking, pp.1589-1592, 2017.

[11] Yunus Çelik, Mahmut Altun, Mahit Güneş, ―Color Based Moving Object Tracking with An Active Camera Using Motion Information, 2017.

[12] Mejda Chihaoui, Akram Elkefi, , Wajdi Bellil and Chokri Ben Amar, ― Detection and tracking of the moving objects in a video sequence by geodesic active contour, pp. 212–4215, 2016.

[13] Juraj Ciberlin, Ratko Grbić, Nikola Teslić, Miloš Pilipović, ―Object Detection and Object Tracking in Front of the Vehicle using Front Camera View, pp. 27–32, 2019.

[14] Xurshedjon Farhodov, Oh-Heum Kwon, Kyung Won Kang, Suk-Hwan Lee, Ki-Ryong Kwon, ― Faster RCNN Detection Based OpenCV CSRT Tracker Using Drone Data, 2019.

[15] Qingchang Guo and Bing Qiao, ― Research on the Detection and Tracking Technology of Moving Object in Video Images, pp. 469–473, 2015.

[16] Yi-You Hou, Sz-Yu Chiou, Ming-Hung Lin, ― Real-time Detection and Tracking for Moving Objects Based on Computer Vision Method, pp. 213–217, 2017.

[17] Arghavan Keivani , Jules-Raymond Tapamo , Farzad Ghayoor, ― Motion-based Moving Object Detection and Tracking using Automatic K-means, pp. 32–37, 2017.

[18] KENSHI SAHO AND MASAO MASUGI, ―Automatic Parameter Setting Method for an Accurate Kalman Filter Tracker Using an Analytical Steady-State Performance Index, pp. 1919–1930, 2015.

[19] Bradley J. Koskowich, Maryam Rahnemoonfar, Michael Starek, ―Virtualot – A Framework Enabling Real – Time Coordinate Transformation and Occlusion Sensitive Tracking using UAS Products, Deep Learning Object Detection and Traditional Object Tracking Techniques, pp. 6416–6419, 2018.

[20] Igor I. Lychkov, Alexander N. Alfimtsev, Sergey A. Sakulin, ―Tracking of Moving Objects With Regeneration of Object Feature Points, 2018.

[21] Shraddha Mane, Prof.Supriya Mangale, ―Moving object detection and tracking Using Convolutional Neural Networks, pp. 1809–1813, 2018.

[22] Garima Mathur, Devendra Somwanshi, Dr. Mahesh M. Bundele, ―Intelligent Video Surveillance based on Object Tracking, 2018.

[23] Olarik Surinta, Sanya Khruahong, ―Tracking People and Objects with an Autonomous Unmanned Aerial Vehicle using Face and Color Detection, pp. 206–210, 2019.

[24] H. Mao, S. Yao, T. Tang, B. Li, and Y. Wang, ―Towards Real-Time Object Detection on Embedded System, pp. 417–431, 2018.

JASC
JOURNAL OF APPLIED SCIENCE AND COMPUTATIONS
An ISO : 7021 - 2008 Certified Journal
ISSN NO: 1076-5131 / web : http://j-asc.com/ / e-mail : submitjasc@gmail.com
Address : H.NO: C-72, Gali No: 3, Hardev Nagar, Jharoda, Burari, New Delhi - 110084

CERTIFICATE OF PUBLICATION

This is to certify that the paper entitled Certificate ID: JASC/3482

"SURVEY ON OBJECT DETECTION AND COUNTING BY IMAGE PROCESSING"

Authored by

R Regan

From

University College Of Engineering Villupuram, Villupuram Tamilnadu

Has been published in

JASC JOURNAL, VOLUME VII, ISSUE III, MARCH- 2020

DOI : 16.10089/JASC
CROSSREF.ORG THE CITATION LINKING BACKBONE
Dr. N. BALASUBRAMANIAN
Editor-In-Chief
JASC
http://j-asc.com/
5.8 IMPACT FACTOR
ISO International Organization for Standardization 7021-2008

JASC
JOURNAL OF APPLIED SCIENCE AND COMPUTATIONS
An ISO : 7021 - 2008 Certified Journal
ISSN NO: 1076-5131 / web : http://j-asc.com/ / e-mail : submitjasc@gmail.com
Address : H.NO: C-72, Gali No: 3, Hardev Nagar, Jharoda, Burari, New Delhi - 110084

CERTIFICATE OF PUBLICATION

This is to certify that the paper entitled Certificate ID: JASC/3482

"SURVEY ON OBJECT DETECTION AND COUNTING BY IMAGE PROCESSING"

Authored by

P Dhineshkumar

From

University College Of Engineering Villupuram, Villupuram Tamilnadu

Has been published in

JASC JOURNAL, VOLUME VII, ISSUE III, MARCH- 2020

DOI : 16.10089/JASC
CROSSREF.ORG THE CITATION LINKING BACKBONE
Dr. N. BALASUBRAMANIAN
Editor-In-Chief
JASC
http://j-asc.com/
5.8 IMPACT FACTOR
ISO International Organization for Standardization 7021-2008

JASC
JOURNAL OF APPLIED SCIENCE AND COMPUTATIONS
An ISO : 7021 - 2008 Certified Journal
ISSN NO: 1076-5131 / web : http://j-asc.com/ / e-mail : submitjasc@gmail.com
Address : H.NO: C-72, Gali No: 3, Hardev Nagar, Jharoda, Burari, New Delhi - 110084

CERTIFICATE OF PUBLICATION

This is to certify that the paper entitled Certificate ID: JASC/3482

"SURVEY ON OBJECT DETECTION AND COUNTING BY IMAGE PROCESSING"

Authored by

G Siddharthan

From

University College Of Engineering Villupuram, Villupuram Tamilnadu

Has been published in

JASC JOURNAL, VOLUME VII, ISSUE III, MARCH- 2020

DOI : 16.10089/JASC
CROSSREF.ORG
THE CITATION LINKING BACKBONE

Dr. N. BALASUBRAMANIAN
Editor-In-Chief
JASC
http://j-asc.com/

5.8 IMPACT FACTOR

International Organization for Standardization
7021-2008