Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Proposal

On

"Ocular Parking System v2"

[Code No: COMP 206]

(For the partial fulfillment of 2nd Year/ 1st Semester in Computer Science)

Submitted by:

Utsav Maskey (28)

Manish Bhatta (08)

Binod Sujakhu (76)

Submitted to:

Department of Computer Science and Engineering

Submission Date: 10th March, 2020

Bonafide Certificate

This project work on

"Ocular Parking System v2"

is the Bona-fide work of

Utsav Maskey, Manish Bhatta and & Binod Sujakhu

who carried out the project work under my supervision.

Pankaj Raj Dawadi

Assistant Professor

AKNOWLEDGEMENT

We would like to express our sincere appreciation to our supervisor, Mr. Pankaj Raj Dawadi sir for his constant guidance, encouragement, understanding and recommendations during the advancement of our project, without his valuable help this project would not have been possible. We were able to elevate our idea due to his guidance. We are highly indebted to him for believing in us and for being a constant source of motivation.

We would also like to thank our project coordinator Mr. Nabin Ghimire for instancing different ideas as a semester project, approving our project proposal, and assigning appropriate supervisor for the project.

We extend our gratitude to Kathmandu University Department of Computer Science and Engineering for providing us a platform where students can work on a topic as a semester long project and demonstrate their findings.

Thank You

Abstract

We present “Ocular Parking System v2”, a Computer Vision / Deep Learning focused project that implements image processing techniques to obtain information of parking lots and summarize it into a visual street map. We intend to provide an interface for vehicle drivers to locate and monitor nearby parking lots in a map and enable reservation of empty spaces. Our implementation is modeled into four sub-categories (a) calibration of parking regions (b) vehicle detection (c) obstruction analysis and (d) interactive map visualization. These modules comply with each other to analyze surveillance footage of parking lots and automate information summarization. In case the detection process fails, it is backed up by mouse-clickable User Interface that allows manual control of the detection process as well as a trackbar slider that controls the confidence of vehicle detection. An interactive street map interface with conventional icons and markers sum up the obtained results. Our approach leverages the use of Machine Learning techniques to model an automated solution to the problem of parking assistant systems.

Keywords: *Deep Learning, Computer Vision, Image Processing*

Code available at: https://github.com/Skelliger7/Ocular-Parking-System-v2

Contents

List of Figures

List of Tables

Table **Page No.**

Acronyms

CV	**Computer Vision**
CNN	**Convolutional Neural Network**
AI	**Artificial Intelligence**
FAIR	**Facebook AI Research**
NMS	**Non-Max Suppression**
CLI	**Command Line Interface**
UI	**User Interface**
API	**Application Programming Interface**
RAM	**Random Access Memory**
GPU	**Graphics Processing Unit**
VRAM	**Video RAM**

Chapter 1: Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. A quick glance at an image is sufficient for a human to point out and describe an immense amount of details about the visual scene [1]. However, this remarkable ability has proven to be a challenging task for our computational models and hence, research related to image recognition and machine learning has been growing rapidly.

The growth of number of vehicles in a city leads to issue of limited parking spots available. People spend on average 7.8 minutes in cruising for a parking spot which accounts for about 30% of the traffic flows in cities [2] and contributes to traffic congestion during the peak hours. With the help of CV, it has become easier and faster for creating a special monitoring system which searches for free parking space.

"Ocular Parking System v2" is a parking assistant system that addresses the task of gathering information from parking lots and summarizing its occupancy information. It loads visual information (video) from surveillance cameras which is processed by Machine Learning algorithm that detect the vehicles. We also specify the parking spots and hence the project mainly focuses on how the visible objects interact with parking spots.

1.1 Background

Deep Learning methods have dramatically influenced most sectors of Machine Learning including speech recognition, visual object recognition, object detection and many other domains [3]. Ever since the breakthrough by Krizhevsky et al. [4], different forms of CNN have primarily been a go-to algorithm for most Computer Vision related classification tasks. Even deeper architectures have been trained [5]–[8]. CNNs are now flexible enough to perform cutting-edge CV tasks like image Style Transfer [9]–[12],

Image Segmentation [13]–[19], Object Detection [20]–[25], Super Resolution [26]–[29], Image Restoration [30]–[34] Frame Interpolation [35], Image Synthesis [36]–[38].

In this report, we focus on object detection for detecting vehicles. In CV, Object detection is the process of detecting instances of semantic objects of a certain class (vehicles) in digital images and videos. On abstract level, object detection can be broken into 3 steps. First, objects of interest need to be identified in the image, which is referred as **localization**. Then, the objects of interest need to be identified as cars; this is referred to as **classification**. Finally, the detections with false triggers needs to be suppressed so that we obtain refined objects' regions.

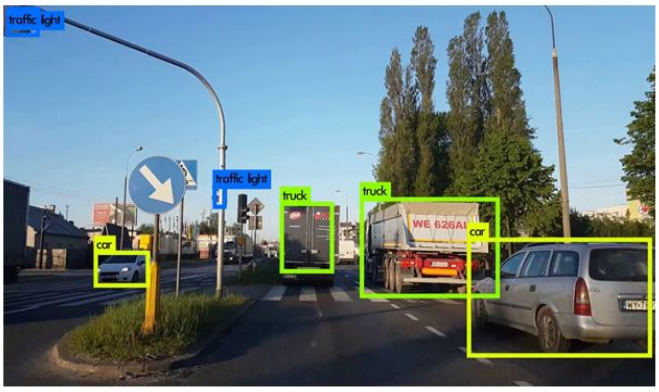

Figure 1.1 Object Detection

In deep learning, generally detection algorithms are categorized as **one-staged** and **two-staged** object detection. One-staged models like *YOLO* [20], *SSD* [21], *RetinaNet* [22], *FPN* [23] which are generally used for real-time detection, are faster as they try to combine the localization and classification steps. However, two-staged architecture like *Faster R-CNN* [25] produce much better results at the cost of computation.

After detection, the other factor is determining parking spots and how the vehicles interact with parking spots. This mostly involves image processing methods, from which we can compute occupancy information.

1.2 Objective

- To automate parking lot information distribution.
- To provide an interface for monitoring and reserving parking lots remotely.

1.3 Motivation and Significance

The field of Computer Vision is shifting from statistical methods to deep learning / neural network models. These methods have recently gained popularity in achieving state-of-the-art performance at most of the challenges related to computer vision. But still, the use of such models for practical applications is an active area of research. The motivation behind studying computer vision is to gain an insight into the integrated methodologies of combining machine learning and information processing in vision. The rise of image processing tools and Machine Learning libraries further motivates the automation of computationally processed vision tasks.

This study holds significant importance: Firstly, it attempts to understand complex structures of objects in an image, different methods of its detection. Secondly, it facilitates an interactive environment for drivers to know information about nearby parking lots and for parking lot owners / maintainers to monitor them. Its proper implementation helps in reducing traffic congestion as well. Finally, it makes a significant contribution to the existing literature as it relates to vision-based parking assistant systems.

Chapter 2: Related Works / Existing Works

We focus on studying the problem of in-vehicle vision-based parking-slot detection. Outside CV, PAS generally use network of sensors which includes weigh-in-motion sensors which are embedded into the parking space (intrusive sensors). Counter-based systems use sensors to count the number of vehicles entering and exit a car park area. Wireless Parking system using Bluetooth [39] is a short distance wireless communication technology which analyzes parking regions using RFID technology. Similarly, Wolff et al. used magnetic field sensors and wired-based concept to test on their simulation model [40]. Tang, Zheng and Cao [41] developed such a system using sensor board is equipped with the sensors of light, temperature, acoustic and a sounder. There are also other researches using wireless-based methods such as in [42].

Numerous classical imaged-based methodologies have also been deployed widely in areas such as security surveillance, motion tracking [43] and traffic control. Images can also be constructed by laser scan [42] but this usually comes with a moving autonomous vehicle. Existing studies also focused on the applications of car parking system using video sensor technologies.

Similarly, machine learning methods have also been implemented. Amato et al. [44] developed a solution for visual parking space occupancy detection using deep CNN and smart cameras using AlexNet [4], which can be improved by using contemporary better models. An article by A. Geitgey [45] used deep learning methods to monitor parking area by observing vehicle movements and labeling parking spots where the still vehicles reside. A recent article from Nvidia, cited a paper by Cai et al. [46], which uses Mask R-CNN [47] for detection and follows a parking spot labeling that is very similar to ours, i.e. manually marking out parking lot boundaries.

Chapter 3: Design

3.1 Problem Definition

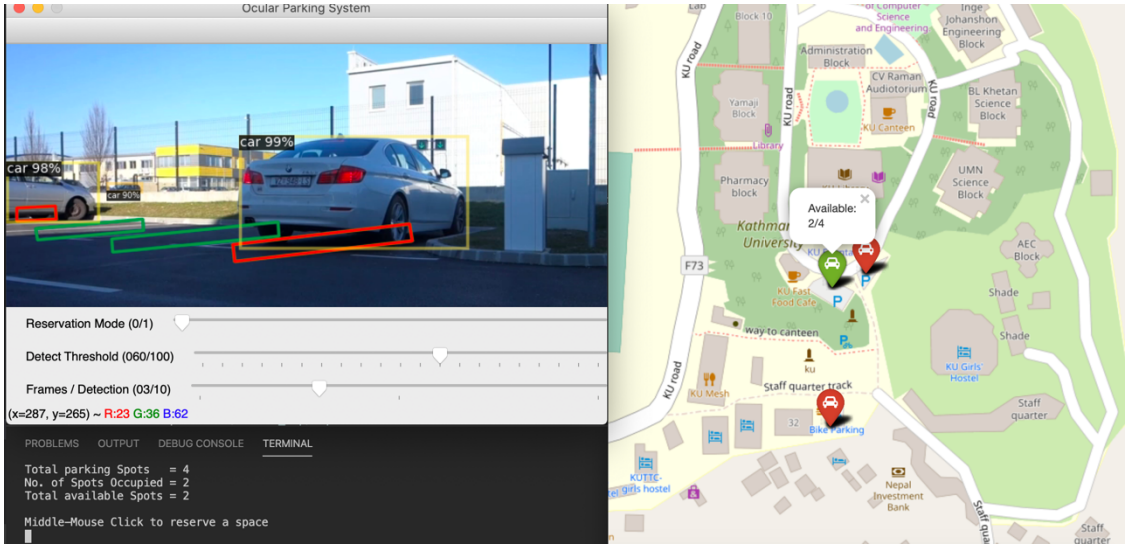

Figure 3.1 Final Output: Vacant Spot (Green), Detected Vehicle (Yellow), Occupied Spot (Red)

Overview: The ultimate goal of our project is to obtain car park occupancy or vacancy information (Figure. 3.1). We first present a model that detects park-able (Green) Regions. Then our vehicle detection module finds regions that contains vehicles (Yellow Regions). We combine these modules and declare that a park-able (Green) Region is occupied if any of the detected vehicles (Yellow) region interacts with it. The occupied regions are drawn as (Red). Hence occupancy information is obtained and sent to the map.

3.2 Solution Specification

Our procedure follows an iterated model as shown in the flow chart (Figure. 3.2). The input to our program:

1.) Video Frame / Camera Input
2.) Model Configuration Files + Weights (For vehicle detection module)

Flow Chart:

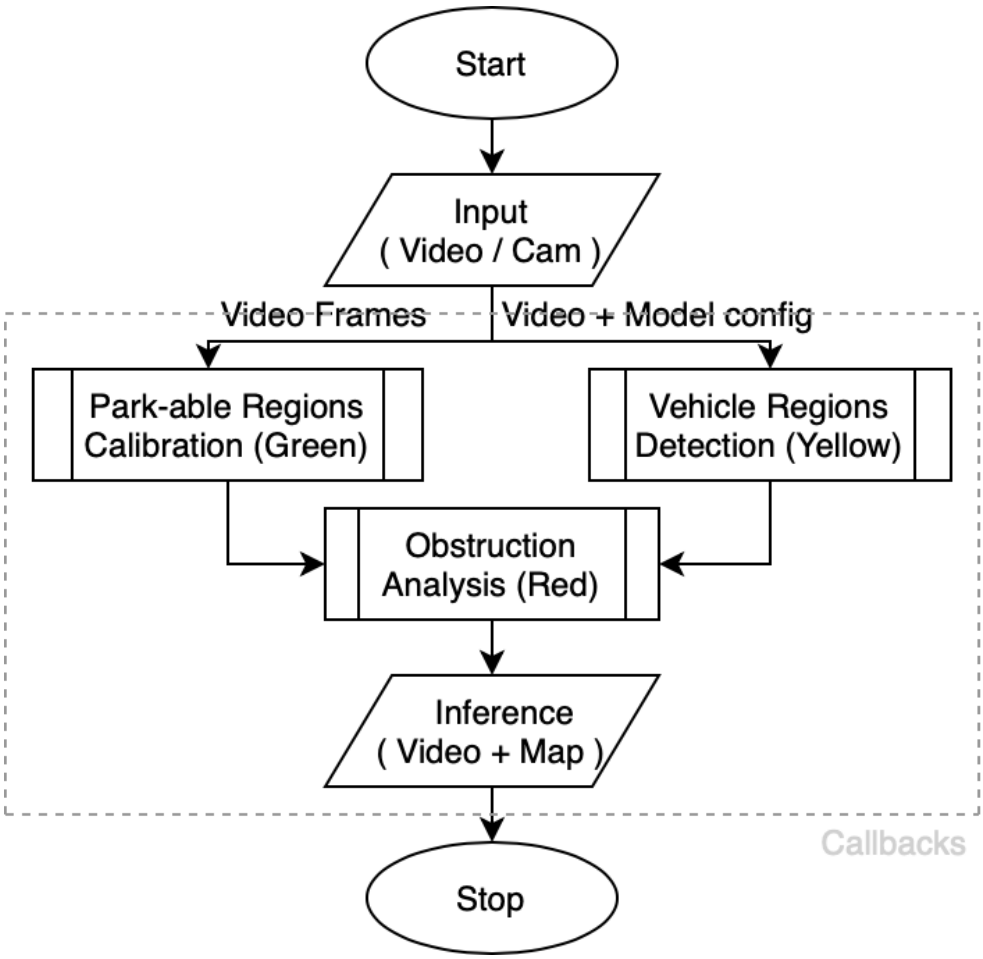

Figure 3.2 Flow Chart of Main Program

Our input (video + model) flows into two modules: (a) Calibration and (b) Detection Module. Outputs from these modules are then analyzed for obstruction analysis, which estimates occupancy information. The highlighted operations are iterated for every frame in the video sequence. As for inference, we output our findings to a client-side street map and a server-side video interface.

3.2.1 Park-able (Green) Region Calibration Module

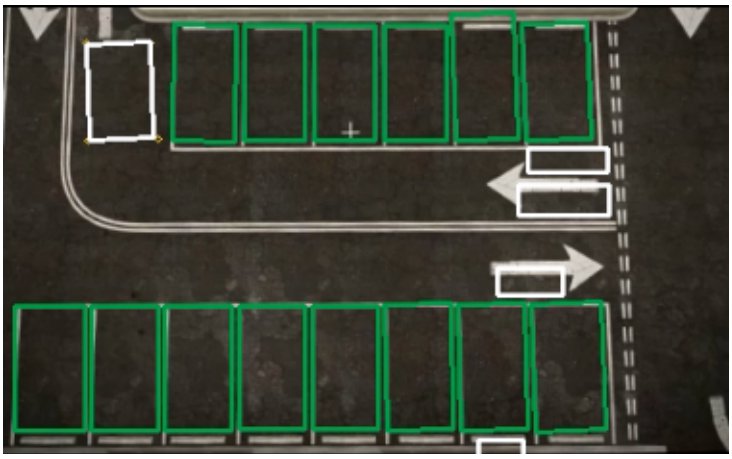

Figure 3.3 Calibration Implementation

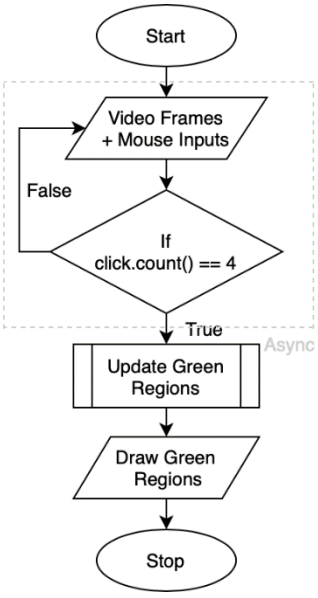

Figure 3.4 Flow Chart of Park-able Region Calibration Module

This module focuses on providing an interface for manual calibration of Park-able Regions. It asynchronously checks for mouse left-clicks, and expects four clicks for each corner of a rectangle. Then we structure a rectangle object that fits the minimum area of four co-ordinates and the rectangle is then appended to the list of *Green Objects*. Similarly, it also checks for right-clicks that removes mis-calibrated object, shown in white (Fig 3.3). This module mostly consists of mouse callbacks, so it runs multi-threaded for entire program asynchronously.

3.2.2 Vehicle (Yellow) Detection Module

This module detects and structures the vehicle as *Yellow Objects.* For detection, we actually implemented two models: (a) one-staged *Retina Net* [22] using FastAI[1] and

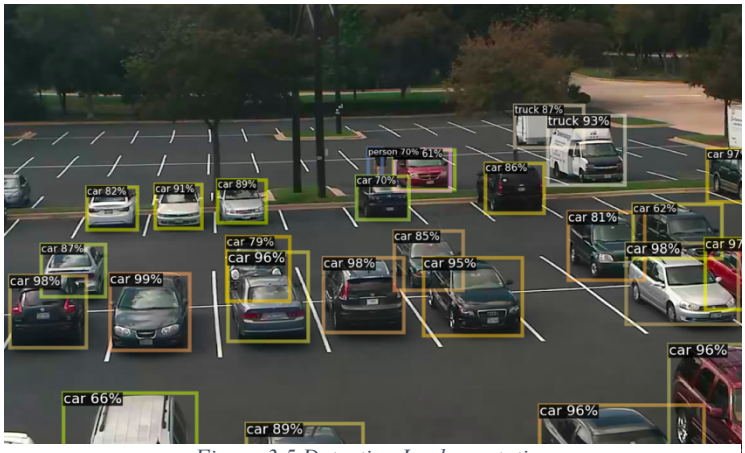

Figure 3.5 Detection Implementation

[1] Fast.ai is python based deep learning library built on top of Pytorch

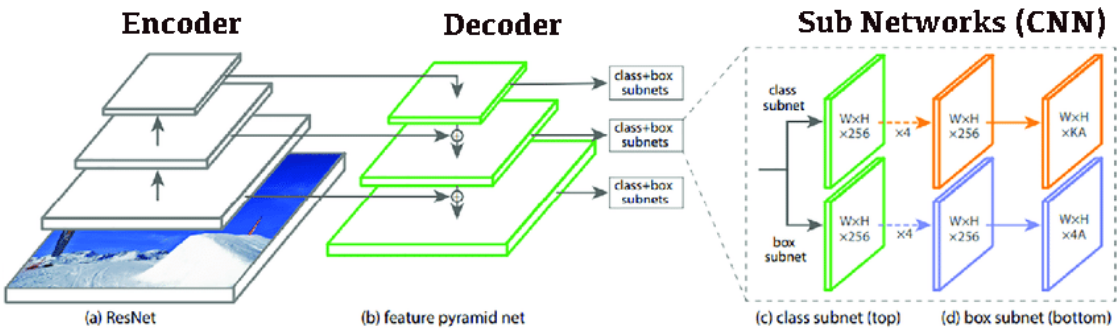

Figure 3.6 RetinaNet Architecture. Figure from [22]

(b) two-staged FAIR's detectron2[2] API [48] that uses Faster R-CNN [25] model. **RetinaNet** model is shown in (Figure 3.6). This approach is based on feed-forward convolutional Encoder path followed by Decoder path that maps images to its differential sized spatial features. We implement our early encoder or down-sampling path as a transferred version of K. He et al.'s [5] ResNet34 model. Similarly, we link this prior model to a decoder or up-sampling model implemented in FastAI library. Skip connections from the down-sampling to the up-sampling path have been adopted to allow for a finer information recovery for image translation. FastAI uses pixel-shuffle [49] as deconvolution layers at up-sampling path for pixel-wise feature extraction.

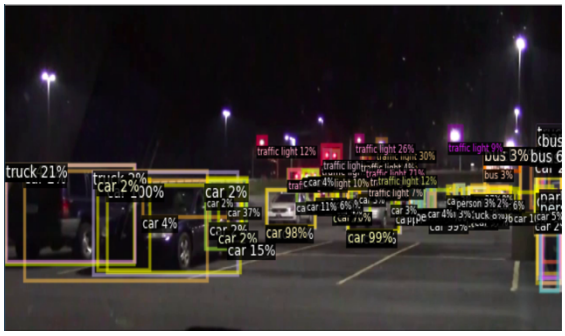

Figure 3.8 Detection without suppression

Figure 3.7 Detection with NMS of 90%

These extracted features are then mapped to actual labels (vehicles) using two sub-networks, each for localization (box subnet) and classification (class subnet). Finally, we suppress the detected boxes to keep only those with good confidence of prediction.

[2] Detectron2 is Facebook AI Research's next-generation platform for object detection and segmentation

Training: We trained RetinaNet model on Pascal VOC dataset [50] with following architecture details:

Architecture details	
No. of layers	50-layer ResNet [5] + 5 Up-sampling layers * (4 localization + 4 classification) Total: 90
Dataset:	Pascal VOC (2007 + 2009)
Batch Size:	64 images (256 x 256px)
Activation Function	ReLU [51]
Learning Rate	One-cycle learning rate [52]
Optimizers	• Adam [53] • BatchNorm [54] • Weight Decay
Loss function	RetinaNet Focal Loss [22]
No. of trainable parameters	57,505,189

Figure 3.9 RetinaNet Implementation Details

Our model has 50 encoding layers, out of which 5 Lateral decoding layers are taken for up-sampling path. Each up-sampling layer are then connected to 4 localization and 4 classification layers. This model trains on a total of 6000+ images of size 256 x 256 px. That maps to bounding boxes of 20 labels (including car, bus, bicycle, motorbike, etc.)

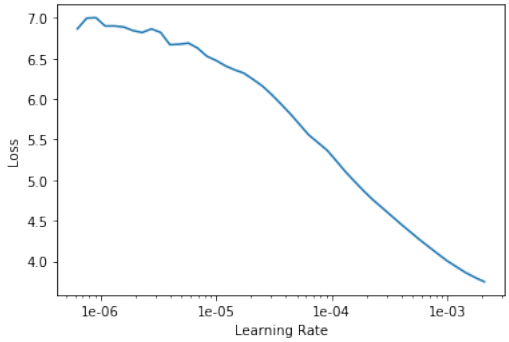

Figure 3.11 Learning Rate Finder Graph

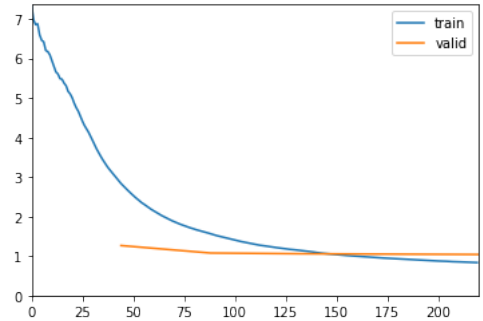

Figure 3.10 Loss vs Batch graph (at near end of training)

We use One-cycle learning rate finder and select '1e-4' as our initial rate which changes as per one-cycle policy [52]. We then train the model. We note decreasing loss function as plotted in following graphs.

3.2.3 Obstruction Analysis

This module mostly consists of image processing. The inputs to this module are: Park-able (Green) and Vehicle (Yellow) objects. To define how vehicle interacts with parking lots, we generate co-ordinates of points that are near the tire of each vehicle. If any of the generated tire co-ordinates lie under the region of any parking space, we assume this space to be occupied. The space remains occupied until the vehicle breaks the interaction.

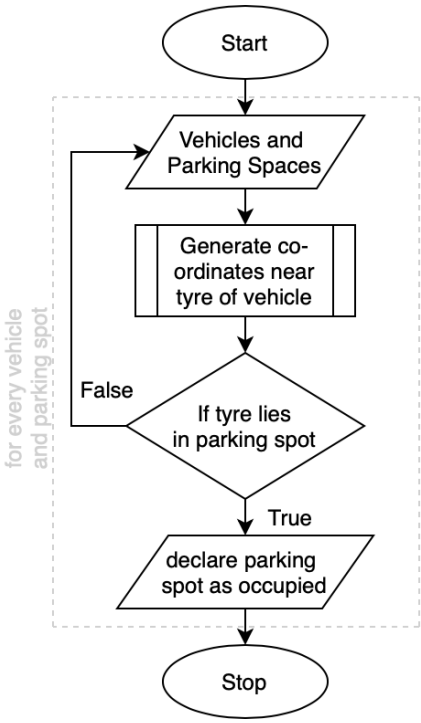

Figure 3.14 Obstruction Analysis

We approach this module by iteration of all green regions over that of every yellow region to check for any interaction. Occupied Regions are basically clones of Green Regions that satisfy the matching condition. We also check for obstruction's redundancy and removal.

Figure 3.13 Last parking spot before vehicle interaction

Figure 3.12 Vehicle interacts with parking spot

3.2.4 Output

In addition to video output, we also facilitate remote monitoring of parking lots through a map. We basically count the frequency of vehicles and available parking spots. We specify location co-ordinates of parking lots and update vacancy information periodically.

This marks the end of Design procedure.

Figure 3.15 Street Map Output of Fig 3.13

Chapter 4: Implementation

4.1 Project Directory Structure

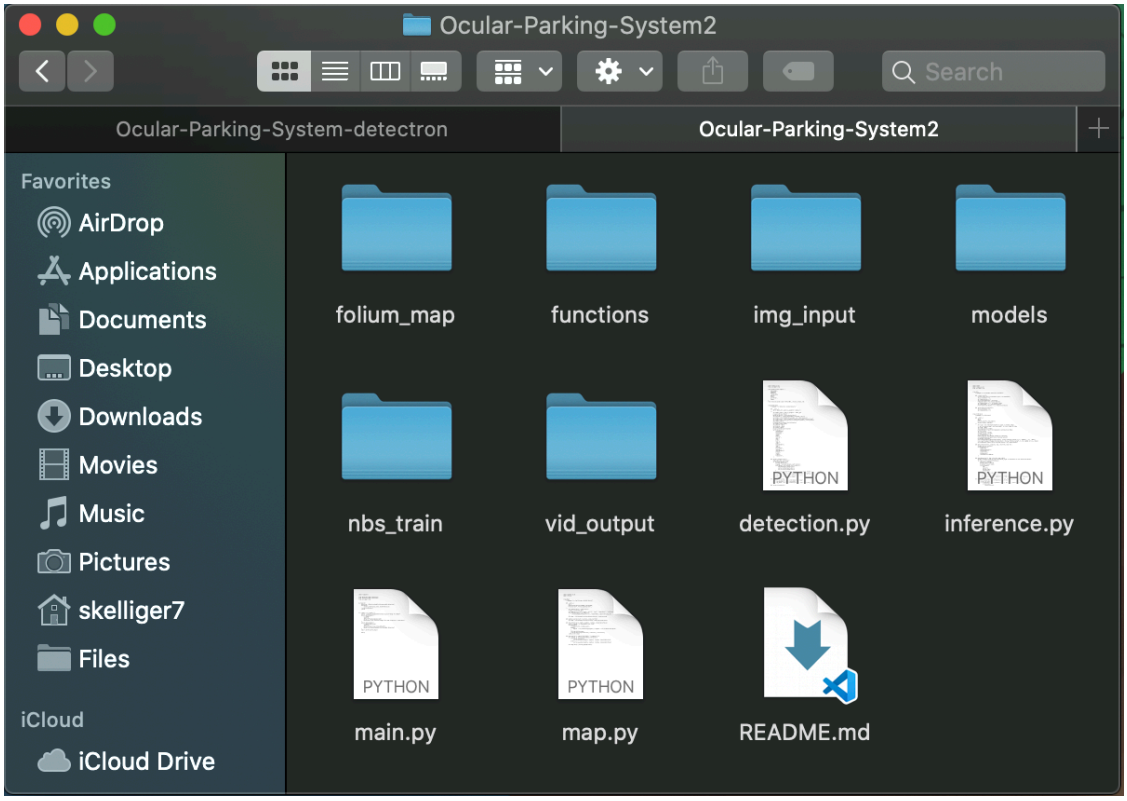

Figure 4.1 Project Directory Structure

We organized our project directory such that the main program lies in 'main.py' file. This executable file drives our detection and inference modules with corresponding CLI arguments. We structure our directory to store ML architecture and its functions in 'functions' directory and its Weights at 'models' directory. Similarly, we can also test our program on video / image files located at 'img_input'. Our training notebooks are located at 'nbs_train'. Street map data generated by 'map.py' are conveniently stored at 'folium_map' directory in javascript readable format. This directory also stores the location co-ordinates and other information about parking lots in csv format at './folium_map/parking_lots.csv' file.

4.2 User Interface

We implemented minimal User Interface for controlling the detection process. The occupancy information is displayed on the interactive street map. Manual edit of Detected Regions and some of its parameters are facilitated as:

4.2.1 Window UI Utilities

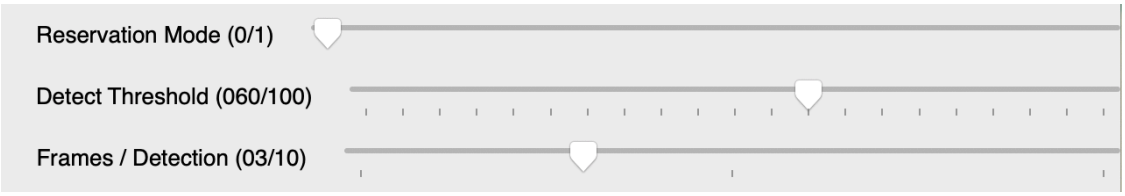

Figure 4.2 Detection Control UI

Name	Type	Description
Ocular Parking System	Window	Window to display the frames
Reservation Mode	Trackbar	Toggle manual control of detection
Detection Threshold	Trackbar	Suppress Detection rate (0 - 100%)
Frames / Detection	Trackbar	No. of Detections per second

Table 4.1 Window Element Properties

4.2.2 Mouse Operations (Callbacks)

Mouse Button	Usage	Usage Description
Left Click	Addition of Park-able Regions	Four left-clicks add a Rectangle (Green) Region.
Right Click	Deletion of Park-able Regions	Single right-click inside a (Green) region removes the region.
Middle Click	Reserve Parking Space	Single Middle-Click inside a Rectangle Region reserves or unreserve the Region. i.e. Convert Green Regions to Red and vice-versa. This only works with 'Reservation Mode' turned on.

Table 4.2 Mouse Callback Operations

4.2.3 Interactive Street Map

Figure 4.3 Folium Map UI

The occupancy information extracted from our ML model are summarized in a street map. We use folium map, which is a python API for creating inter-active maps, build on top of Leaflet[3]. It allows to zoom in, zoom out as well as hover and click parking-lot markers. Parking lots with available space are marked Green, whereas non-empty ones are marked Red. We specify location co-ordinates of parking lots and update its occupancy information.

4.3 Classes and functions

```
def main():
    detection = Detection(modelWeights=args.model_detection, cfgPath= args.cfg_path)
    Inference(
        content_video=args.video, model=detection
    ).start_inference(generateMap=True)
    return
```

Figure 4.4 Main program

The main function takes CLI arguments and initializes three major classes: 'Detection', 'Inference' and 'Map'. Some of its implementation are tabulated below.

4.3.1 Detection Class

4.3.1.1 Variables

Name	Type	Description
model_path	PATH	Path to Weights (pkl file)
model	torch.nn.Module	Torch Model for RetinaNet

[3] Leaflet is an open source JavaScript library used to build web mapping applications

classes	list of strings	Labels of Detection (eg. vehicles)
bboxes (**Rectangles**)	Tensor / List	Rectangular bounding box co-ordinates of **Detected vehicles**
preds		Predicted label of detection
scores		Confidence of detection

Table 4.3 Variable Tables

4.3.1.2 Functions

Name	Description
suppress_outputs	Apply NMS (i.e. suppress less confident outputs)
np_to_tensor	Convert np.ndarray to Pytorch Compatible Image
tensor_to_np	Convert Tensors to OpenCV Compatible Types
show	Displays content input image and its corresponding detected objects
start_detection	Detects, Updates and Returns bboxes (**Vehicles**), preds, scores and classes for next video frame

Table 4.4 Function Tables

4.3.2 UI Class

UI class is a wrapper for In-Window Interface Variables

4.3.2.1 Variables

Name	Type	Description
clickCount	int	Counts four mouse-clicks for corners of rectangles and then resets itself
mouseCoords	List of co-ordinates	Temporary storage for four co-ordinates of rectangle
selectedContour	list of co-ordinates	cv2.minAreaRect(mouseCoords)
bboxesGreen	List of rectangles	List of **calibrated parking spaces**
bboxesRed	List of rectangles	List of **Occupied spaces**

4.3.2.2 Functions

Name	Description
reset_mouse_coords	Resets clickCount and mouseCoords to null

4.3.3 Inference Class

Inference class is a wrapper for In-Window Interface Interaction

4.3.3.1 Variables

Name	Type	Description
img	Numpy Image	A frame of video
img_display	Numpy Image	Copy of img, boxes are drawn on this image
frameCounter	int	Counter to control Detection Rate

4.3.3.2 Functions

Name	Description
draw_contours	draws list of contours; vehicle / parking spot rectangles
draw_bboxes	draws Rectangle Objects
contour_containing_point	Returns contour under which the input co-ordinate lies
remove_contour_containing_point	Removes a detected 'contour' region from input co-ordinate (x, y)
generate_points_to_check	Generates list of co-ordinates from bottom-center to center of input bbox/rectangle
remove_np_array_from_list	Custom List Item Removal Function
occupied_space_detection	Updates List of Red/Occupied/bboxRed Regions on Obstruction Detection

4.3.3.3 Functions drivers

These functions utilize the above functions and provide interactive I/O interface:

Name	Description
parkable_region_inference	Inference for manually calibrating Green regions
video_inference	Inference for Main program
mouse_to_region	Adds Green Regions from mouse-clicked 'UI.mouseCoords'
start_inference	Mouse-Events ready User Interface. This function runs **parkable_region_inference** for the 1st frame of video and **video_inference** on all frames

4.4 Software Specification

4.4.1 Operating System: Linux, Mac, Windows (64-bit)

4.4.1 Python 3.7 Library Dependencies:

Python Library	Version
PyTorch	1.3.0
FastAI or Detectron2	1.0.58 or 1.1
Pillow	5.1.0
OpenCV	4.1.1.26
NumPy	1.17.2

Table 4.5 Python Library Dependencies Table

4.5 Hardware Specification

4.5.1 Training:

- **GPU:** At least 12GB[4] of available VRAM
- **System Memory:** Depends on dataset[5]

4.5.2 Inference:

- **CPU:** Dual-core 64-bit based processor (Modern CPU recommended)
- **RAM:** About 6 GB of RAM Recommended
- **System Memory:** 2 GB of free space

4.6 Optional Requirements

4.6.1 Web-Cam: Any camera supported by OS (only for external video input)

[4] We train our models on Tesla P100 16 GB GPU provided in Google Colab
[5] 1GB space for Pascal VOC Dataset [55] + 1 GB space for learned parameters

4.7 Execution

4.7.1 Dependencies installation

In this section, we simplify the installation of python dependencies and other requirements. These executable commands apply to pip package installer under bash (Linux / Unix) shell.

- [python3.7]
  - sudo apt-get install python3.7-dev
- [OpenCV >= 4.1.1.26]
  - python3.7 -m pip install --user opencv-python
- [FastAI >= 1.0.58]
  - python3.7 -m pip install --user fastai
- [Folium >= 0.10.1]
  - python3.7 -m pip install --user folium

Similarly, for detectron2 implementation, we need to build detectron2 and Pytorch from source. Links to Weights, Model / Config files and more instructions are available at our repo and its branches: https://github.com/Skelliger7/Ocular-Parking-System-v2

4.7.2 Usage

- **Execution:**
  - python3.7 main.py --video videoplayback.mp4

  Optional command line arguments:

  - ☐ --video: Path to video file relative to " ./img_input "
  - ☐ --model_detection: Path to Weights relative to " ./models "
  - ☐ --cfg_path: Path to model cfg file [Detectron2 only]

Chapter 5: Discussion on the Achievements

5.1 Challenges

5.1.1 Detection

We faced the problem of vision-based vehicle detection. Initially, our detection module didn't properly learn to predict accurate boxes. Bounding boxes were over-sized with many false-positives that would render our model useless. We therefore considered switching CNN model to use detectron2 [48]. Detectron's pre-trained model for Faster R-CNN improved our detection model significantly.

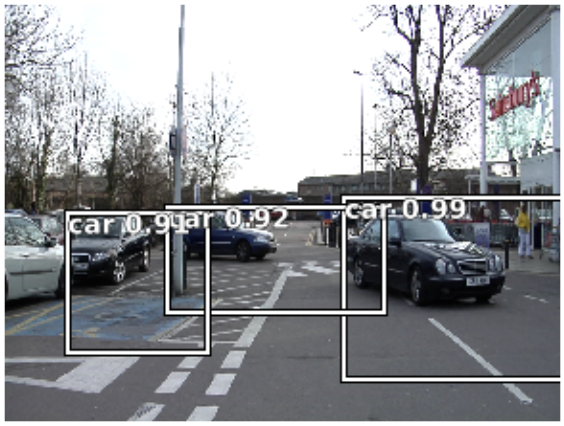

Figure 5.1 Detection Challenges

5.1.2 Time Complexity

Our ML model is computationally expensive. For relative comparison, a modern quad-core processor takes about 4-5 seconds for a single detection. So, it is not convenient to run our model in real-time. Detection performance can be improved by using GPUs, but we approached this issue by suppressing the rate of detection. Our 'detection / frame' UI component facilitates a trackbar slider that can lower the rate of detection (per seconds) and improve efficiency.

5.1.3 Working with video frames

We process videos / camera feed by iterating over every frame and performing detection tasks. However, this loop introduces many challenges. Debugging runtime errors is tedious because it is difficult to find the iteration that caused the problem if previous iterations were running correctly. Also, sometimes the callback functions conflict each other for a single variable on some circumstances raising semantic errors.

5.2 Features

1. **Object Detection:**
   This feature detects the object and structures it as rectangular bounded regions. The detected vehicles are drawn as yellow boundary boxes.

2. **Parking Region Calibration:**
   This feature facilitates a video-in-window interface that can be interacted by mouse operations. It enables addition / removal of parking spaces. Once calibrated, it works as long as the camera position remains intact.

3. **Parked Space Locator:**
   This feature marks the parking spaces that are occupied.

4. **Detection Rate Control:**
   This UI component controls the detection rate. Detecting the objects in real-time is computationally expensive. So, it is preferred that detection process be delayed by some time interval. This trackbar slider ranges from 0 to 10 seconds.

5. **Detection Threshold:**
   Threshold trackbar ranging from 0 to 100%, it specifies the confidence interval of detection. For example, if we select a threshold of 90%, only those detection with 90% confidence are taken into account (See figure 3.7 & 3.8). Selecting threshold of 0% means to consider all the detections.

6. **Reservation Mode:**
   This UI component toggles the manual control of obstruction module on or off. When turned on, it is possible to declare an empty parking spot as occupied, even without any vehicle interaction.

7. **Interactive Map:** Occupancy information are sent to map. This interface shows parking lots along with its information on a map using conventional markers.

Chapter 6: Conclusion and Recommendation

We implemented a ML model that summarizes parking lot occupancy information using computational resources. Our model produced exceptional results in detection, even on extreme environments. It indicates that the ease of utilizing camera or vision-based system to monitor car parks is feasible.

In order to overcome some of the weakness that was observed in the development of vision-based system, we facilitated trackbar interface to control various aspects of the program. We tested many ML and image processing tools, became familiar with mechanisms that govern Computer Vision and grabbed problem solving skills. This project inspired us to deep dive into practical ML implementations and improve computational thinking habits.

6.1 Limitations

- Our model needs a static camera feed. Regions need to be re-calibrated for any undesired movement of camera.
- Camera needs to be properly placed such that no vehicle visually overlaps other vehicle.

6.2 Future enhancements

- Integration of street-map into a mobile application.
- Use of stereo camera feed for large-scale parking lots.
- Real-time server powered information system.

References

[1] F. F. Li, A. Iyer, C. Koch, and P. Perona, "What do we perceive in a glance of a real-world scene?," *J. Vis.*, 2007, doi: 10.1167/7.1.10.

[2] R. Arnott and E. Inci, "An integrated model of downtown parking and traffic congestion," *J. Urban Econ.*, 2006, doi: 10.1016/j.jue.2006.04.004.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, Oct. 2015, doi: 10.1038/nature14539.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, vol. abs/1512.03385, 2015.

[6] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, doi: 10.1109/CVPR.2015.7298594.

[7] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, doi: 10.1109/CVPR.2017.243.

[8] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," in *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, 2017.

[9] L. Gatys, A. Ecker, and M. Bethge, "A Neural Algorithm of Artistic Style," *J. Vis.*, 2016, doi: 10.1167/16.12.326.

[10] M.-Y. and L. X. and Y. M.-H. and K. J. Li Yijunand Liu, "A Closed-Form Solution to Photorealistic Image Stylization," in *Computer Vision – ECCV 2018*, 2018, pp. 468–483.

[11] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images," in *33rd International Conference on Machine Learning, ICML 2016*, 2016.

[12] A. and F.-F. L. Johnson Justinand Alahi, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," in *Computer Vision – ECCV 2016*, 2016, pp. 694–711.

[13] S. Jégou, M. Drozdzal, D. Vázquez, A. Romero, and Y. Bengio, "The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation," *CoRR*, vol. abs/1611.0, 2016.

[14] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 2015, pp. 234–241.

[15] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: A Deep Convolutional Encoder-Decoder Architecture for ImageSegmentation,” *CoRR*, vol. abs/1511.00561, 2015.

[16] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “PSPNet,” *CVPR*, 2017, doi: 10.1109/CVPR.2017.660.

[17] G. Lin, A. Milan, C. Shen, and I. Reid, “RefineNet,” *CVPR*, 2017, doi: 10.1109/CVPR.2017.549.

[18] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Rethinking Atrous Convolution for Semantic Image Segmentation Liang-Chieh,” *arXiv.org*, 2018, doi: 10.1159/000018039.

[19] E. Shelhamer, J. Long, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017, doi: 10.1109/TPAMI.2016.2572683.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, doi: 10.1109/CVPR.2016.91.

[21] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *CoRR*, vol. abs/1512.02325, 2015.

[22] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, doi: 10.1109/ICCV.2017.324.

[23] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, “Feature Pyramid Networks for Object Detection,” *CoRR*, vol. abs/1612.03144, 2016.

[24] J. Li, X. Liang, Y. Wei, T. Xu, J. Feng, and S. Yan, “Perceptual generative adversarial networks for small object detection,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, doi: 10.1109/CVPR.2017.211.

[25] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017, doi: 10.1109/TPAMI.2016.2577031.

[26] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, “Enhanced Deep Residual Networks for Single Image Super-Resolution,” *CoRR*, vol. abs/1707.02921, 2017.

[27] W. Shi *et al.*, “Real-Time Single Image and Video Super-Resolution Using an EfficientSub-Pixel Convolutional Neural Network,” *CoRR*, vol. abs/1609.05158, 2016.

[28] C. Ledig *et al.*, “Photo-realistic single image super-resolution using a generative adversarial network,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, doi: 10.1109/CVPR.2017.19.

[29] W. Yang, X. Zhang, Y. Tian, W. Wang, and J.-H. Xue, "Deep Learning for Single Image Super-Resolution: A Brief Review," *CoRR*, vol. abs/1808.03344, 2018.

[30] J. Lehtinen *et al.*, "Noise2Noise: Learning image restoration without clean data," in *35th International Conference on Machine Learning, ICML 2018*, 2018, vol. 7, pp. 4620–4631.

[31] J. Howard, U. Manor, and J. Antic, "Decrappification, DeOldification, and Super Resolution," in *Facebook F8 conference*, 2019.

[32] S. Lefkimmiatis, "Non-local color image denoising with convolutional neural networks," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, doi: 10.1109/CVPR.2017.623.

[33] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: Can plain neural networks compete with BM3D?," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2012, doi: 10.1109/CVPR.2012.6247952.

[34] G. Liu, F. A. Reda, K. J. Shih, T. C. Wang, A. Tao, and B. Catanzaro, "Image Inpainting for Irregular Holes Using Partial Convolutions," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, doi: 10.1007/978-3-030-01252-6_6.

[35] H. Jiang, D. Sun, V. Jampani, M. H. Yang, E. Learned-Miller, and J. Kautz, "Super SloMo: High Quality Estimation of Multiple Intermediate Frames for Video Interpolation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, doi: 10.1109/CVPR.2018.00938.

[36] Q. Chen and V. Koltun, "Photographic Image Synthesis with Cascaded Refinement Networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, doi: 10.1109/ICCV.2017.168.

[37] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu, "Semantic Image Synthesis with Spatially-Adaptive Normalization," *CoRR*, vol. abs/1903.07291, 2019.

[38] P. Isola, J. Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, doi: 10.1109/CVPR.2017.632.

[39] H. A. Bin Sulaiman, M. F. B. M. Afif, M. A. Bin Othman, M. H. Bin Misran, and M. A. B. M. Said, "Wireless based smart parking system using zigbee," *Int. J. Eng. Technol.*, 2013.

[40] J. Wolff, T. Heuer, H. Gao, M. Weinmann, S. Voit, and U. Hartmann, "Parking monitor system based on magnetic field sensors," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2006, doi: 10.1109/itsc.2006.1707398.

[41] V. W. S. Tang, Y. Zheng, and J. Cao, "An intelligent car park management system based on wireless sensor networks," in *SPCA 2006: 2006 First International Symposium on Pervasive Computing and Applications, Proceedings*, 2006, doi:

10.1109/SPCA.2006.297498.

[42] C. T. M. Keat, C. Pradalier, and C. Laugier, “Vehicle detection and car park mapping using laser scanner,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005, doi: 10.1109/IROS.2005.1545233.

[43] J. Cotter and G. A. Jones, “Intelligent retrieval and storage of outdoor events dertved from video,” *IEE Colloq.*, 1997, doi: 10.1049/ic:19970384.

[44] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo, “Car parking occupancy detection using smart camera networks and Deep Learning,” in *Proceedings - IEEE Symposium on Computers and Communications*, 2016, doi: 10.1109/ISCC.2016.7543901.

[45] A. Geitgey, “Snagging Parking Spaces with R-CNN and Python,” 2019.

[46] B. Y. Cai, R. Alvarez, F. D. Michelle Sit, and C. Ratti, “Deep Learning Based Video System for Accurate and Real-Time Parking Measurement,” 2020. .

[47] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2020, doi: 10.1109/TPAMI.2018.2844175.

[48] Y. Wu, A. Kirillov, F. Massa, R. Girshick, and W.-Y. Lo, “Detectron2,” 2019.

[49] A. P. Aitken, C. Ledig, L. Theis, J. Caballero, Z. Wang, and W. Shi, “Checkerboard artifact free sub-pixel convolution: A note on sub-pixelconvolution, resize convolution and convolution resize,” *CoRR*, vol. abs/1707.02937, 2017.

[50] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (VOC) challenge,” *Int. J. Comput. Vis.*, 2010, doi: 10.1007/s11263-009-0275-4.

[51] A. F. Agarap, “Deep Learning using Rectified Linear Units (ReLU),” *CoRR*, vol. abs/1803.08375, 2018.

[52] L. N. Smith and N. Topin, “Super-Convergence: Very Fast Training of Residual Networks Using LargeLearning Rates,” *CoRR*, vol. abs/1708.07120, 2017.

[53] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” 2014.

[54] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by ReducingInternal Covariate Shift,” *CoRR*, vol. abs/1502.03167, 2015.

[55] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, “Segmentation and recognition using structure from motion point clouds,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, doi: 10.1007/978-3-540-88682-2-5.

Appendix

Appendix 1 - Project Account and Journal

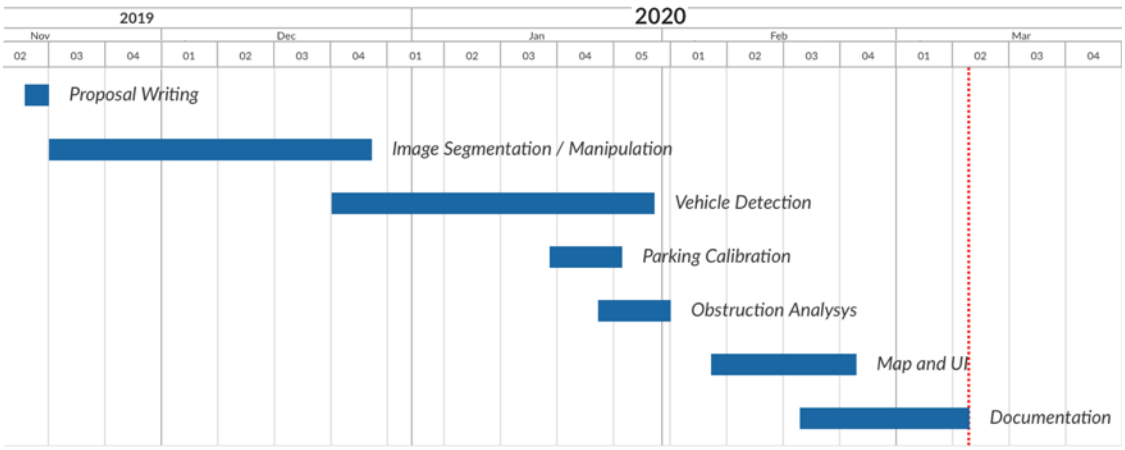

Table 8.1 GANTT Chart

Legend	
Tasks	**Sub-categories**
Proposal Writing	-
Vehicle Detection	Study
	Training
	Inference
Parking Region Calibration	Implementation
Obstruction Analysis	Implementation
	Improvement
Map and UI	Implement
Documentation	Report Writing
Further improvements	Parameter Tuning

Table 8.2 GANTT Chart Legend

Appendix 2 – Work Division

Member	Vehicle Detection	Parking Calibration	Obstruction Analysis	Map and UI
Utsav Maskey				
Manish Bhatta				
Binod Sujakhu				