

**PREDICTING OPEN SPACE PARKING VACANCIES USING MACHINE
LEARNING**

By

Lee Wei Jun

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS)

INFORMATION SYSTEMS ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2023

REPORT STATUS DECLARATION FORM

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: Predicting Open Space Parking Vacancies using Machine Learning

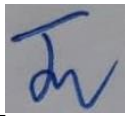
Academic Session: January 2023

I LEE WEI JUN
(**CAPITAL LETTER**)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



(Author's signature)



(Supervisor's signature)

Address:

11, DATARAN CEMPAKA

SARI 1, DESA CEMPAKA

31400, IPOH, PERAK

Ms Tseu Kwan Lee

Supervisor's name

Date: 26 APRIL 2023

Date: 26 APRIL 2023

FYP THESIS SUBMISSION FORM

Universiti Tunku Abdul Rahman			
Form Title : Sample of Submission Sheet for FYP/Dissertation/Thesis			
Form Number: FM-IAD-004	Rev No.: 0	Effective Date: 21 JUNE 2011	Page No.: 1 of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN

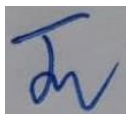
Date: 26 APRIL 2023

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that Lee Wei Jun (ID No: 19ACB03389) has completed this final year project/ dissertation/ thesis* entitled “Predicting Open Space Parking Vacancies using Machine Learning” under the supervision of Ms Tseu Kwan Lee (Supervisor) from the Department of Computer Science, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

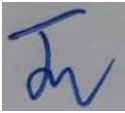
Yours truly,



(LEE WEI JUN)

DECLARATION OF ORIGINALITY

I declare that this report entitled “**PREDICTING OPEN SPACE PARKING VACANCIES USING MACHINE LEARNING**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  _____

Name : LEE WEI JUN _____

Date : 26-4-2023 _____

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Ms Tseu Kwan Lee who has given me this bright opportunity to engage in Green Software, Sustainability and Machine Learning field studies. When I was facing problems in this project, the advice from them always assists me in overcoming the problems. Again, a million thanks to my supervisor.

To a very special person in my life, my mother, Lim Saw Hoon, for her patience, unconditional support, and love, and for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

A finalised report paper from FYP1 named “Predicting Open Parking Space using Deep Learning and Support Vector Regression” had submitted to ISMSI 2023 (Paper ID: MS029). A screenshot of the acceptance notification of the full paper (ISMSI 2023 Acceptance Notification of Full Paper) is attached under the appendix.

ABSTRACT

Vehicle parking has become a significant issue in urban areas due to the imbalance between supply and demand for parking spaces, and increasing the number of parking spaces is no longer an effective solution. Predicting open parking vacancies using machine learning is a practical and effective solution to overcome parking issues. The ability to predict parking availability maximizes parking space utilization, ultimately alleviating traffic congestion. The reduction in idling vehicles results in a decrease in gas emissions, which reduces the burden on the environment. This study proposes a parking prediction model using support vector regression (SVR) to predict available parking spaces. A custom object detector developed using the YOLOv4 algorithm was used to collect the data for training the machine learning model. The results show that the custom YOLOv4 model accurately detects and identifies empty and occupied parking spaces, while the SVR prediction model can predict the number of empty parking spaces. Noise such as weather, lightning issue and obstacles is considered in YOLOv4 model. Next weather features is included in training the machine learning model. In this project, two additional machine learning algorithms, namely linear regression (LR) and decision tree regressor, were used to compare the performance of the support vector regression (SVR) prediction model. Additionally, four different hyperparameter tuning techniques were employed to obtain the most promising fine-tuned support vector regression (SVR) model, including grid search, random search, random search plus, and parameter optimization loop. Moreover, a PySimpleGUI was developed to provide an interactive parking vacancy prediction model graphic user interface (GUI).

TABLE OF CONTENTS

TITLE PAGE	1
REPORT STATUS DECLARATION FORM	ii
FYP THESIS SUBMISSION FORM	iii
DECLARATION OF ORIGINALITY	v
ACKNOWLEDGEMENTS	vi
ABSTRACT	vii
TABLE OF CONTENTS	viii
LIST OF FIGURES	xi
LIST OF TABLES	xv
LIST OF EQUATIONS	xvi
LIST OF SYMBOLS	xvii
LIST OF ABBREVIATIONS	xviii
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement and Motivation	2
1.2 Research Objectives	3
1.3 Project Scope and Direction	4
1.4 Contributions	4
1.5 Report Organization	5
CHAPTER 2 LITERATURE REVIEW	7
2.1 Previous works on Machine Learning	7
2.2 Previous works on Deep Learning	12
2.3 Limitations of Previous Studies	14
2.4 Proposed Solutions	16

2.5	Summary	17
CHAPTER 3 SYSTEM MODEL		18
3.1	Model Design Diagram	18
3.2	Pseudocode	19
3.3	Gantt chart	20
CHAPTER 4 SYSTEM DESIGN		25
4.1	Machine Learning Equation	25
4.2	Framework	31
4.3	Flowchart	32
CHAPTER 5 EXPERIMENT/SIMULATION		35
5.1	System Requirement	35
5.1.1	Hardware	35
5.1.2	Software	35
5.1.3	Software libraries	36
5.1.4	Datasets	38
5.2	Setting up software	43
5.3	YOLOv4 model	43
5.3.1	Configuration on YOLOv4 model	43
5.3.2	Train on YOLOv4 model	48
5.4	Prediction model	52
5.4.1	Dataset preprocessing (text label file to CSV file)	52
5.4.2	Data preprocessing	54
5.4.3	Train on prediction model	59
5.4.4	Fine tune the prediction model	65
5.5	Comment and highlight the feasibility of the proposed method	69
5.6	Concluding Remark	69
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSIN		71
6.1	Model Testing and Performance Metrics (YOLOv4 model)	71

6.2	Model Testing and Performance Metrics (Prediction model)	76
6.3	Project Challenges	88
6.4	Objective Evaluation	88
6.5	Concluding Remark	89
CHAPTER 7 CONCLUSION AND RECOMMENDATION		90
7.1	Conclusion	90
7.2	Recommendation	91
REFERENCES		93
APPENDIX A		1
	ISMSI 2023 Acceptance Notification of Full Paper	1
	YOLOv4 configuration files	3
	Prediction model preprocessing (text label file to CSV file)	4
	Prediction model preprocessing	7
	Prediction model training	8
	Fine tune the prediction model	10
	Predicting model performance result	13
FINAL YEAR PROJECT WEEKLY REPORT		15
POSTER		21
PLAGIARISM CHECK RESULT		1
FYP 2 CHECKLIST		1

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1	SVR diagram	7
Figure 2.2	LR diagram	8
Figure 2.3	General structure of BPNN	9
Figure 2.4	Forecasting results of the four parks with FM 1	11
Figure 2.5	Forecasting results of the four parks with FM 2	11
Figure 2.6	Current system in use to forecast when a parking space will become available	12
Figure 2.7	Result of YOLOv3 vehicle detection when the vehicle in-motion	14
Figure 3.1	YOLOv4 Architecture	18
Figure 3.2	Gantt Chart of FYP1 from June 13, 2022 (week 1) to September 16, 2022 (week 14)	21
Figure 3.3	Gantt Chart of FYP2 from January 30, 2023 (week 1) to March 5, 2023 (week 14)	23
Figure 4.1	k-fold Cross-validation method	26
Figure 4.2	Development Flowchart of Parking Vacancy Prediction Model	33
Figure 4.3	Development Flowchart of Custom YOLOv4 Object Detection Model	34
Figure 5.1	PKLot PUCPR sample image during sunny weather	38
Figure 5.2	PKLot UFPR04 sample image during rainy weather	38
Figure 5.3	PKLot UFPR05 sample image during cloudy weather	39
Figure 5.4	CARPK sample image	39
Figure 5.5	CNRPark+EXT Camera 1 sample image	40
Figure 5.6	Aerial View of Parking Lot sample image	40
Figure 5.7	Create a yolov4 folder in Google Drive	43
Figure 5.8	Create a training folder inside the yolov4 folder	43
Figure 5.9	Output of cloning the darknet git repository	44
Figure 5.10	The yolov4-custom file	44

Figure 5.11	Upload needed file to yolov4 folder	46
Figure 5.12	Copy the files and paste under data and cfg folder	47
Figure 5.13	Copy and paste process file	47
Figure 5.14	Download the weights file for the Yolov4 pre-trained model	48
Figure 5.15	Connect to Google Drive	48
Figure 5.16	Make changes in Makefile	48
Figure 5.17	Build darknet	49
Figure 5.18	Train the custom model using the pre-train yolov4 weights	49
Figure 5.19	Continue the training on last check point, last weights	50
Figure 5.20	Training output after 6000 iterations	50
Figure 5.21	Training output after 9000 iterations	51
Figure 5.22	Store parking features in data frame	52
Figure 5.23	Data frame of PKLot with weather condition	53
Figure 5.24	Read PKLot CSV file into data frame	54
Figure 5.25	Quick description of all attributes in the pk data frame	54
Figure 5.26	Check redundancy in DateTime	55
Figure 5.27	Statistics of all numerical attributes in pk data frame	55
Figure 5.28	Histogram of each numerical attribute in pk data frame	57
Figure 5.29	Correlation matrix of pk data frame	57
Figure 5.30	Count unique values of weather	58
Figure 5.31	Graph plot with Empty and DateTime in Cloudy	59
Figure 5.32	Graph plot with Empty and DateTime in Rainy	59
Figure 5.33	Graph plot with Empty and DateTime in Sunny	59
Figure 5.34	The shape of the original training set and the processed training set x and y	60
Figure 5.35	Performance output of the SVR model	61
Figure 5.36	Result of 5-fold cross validation with SVR using the training set	62
Figure 5.37	Performance output of the LR model	62
Figure 5.38	Result of 5-fold cross validation with LR using the training set	63

Figure 5.39	Performance output of the decision tree regression model	64
Figure 5.40	Result of 5-fold cross validation with decision tree regressor using the training set	64
Figure 5.41	Output after performed grid search on SVR model	66
Figure 5.42	Output after performed random search on SVR model	66
Figure 5.43	Output after performed random search plus on SVR model	67
Figure 5.44	Output after performed parameter optimization loop on SVR model	68
Figure 6.1	Training result	71
Figure 6.2	Training result with mAP@IoU=0.5	72
Figure 6.3	Training result with mAP@IoU=0.75	72
Figure 6.4	Loss and mAP chart	73
Figure 6.5	PKLot- PUCPR (i) test data, (ii) prediction result on the test data with (a) 0.2, (b) 0.7 thresh value	74
Figure 6.6	CARPK (i) test data, (ii) prediction result on the test data with (a) 0.2, (b) 0.7 thresh value	74
Figure 6.7	CNRPark+EXT (i) test data, (ii) prediction result on the test data with (a) 0.2, (b) 0.7 thresh value	75
Figure 6.8	Aerial View of Parking Lot (i) test data, (ii) prediction result on the test data with (a) 0.2, (b) 0.7 thresh value	75
Figure 6.9	The processed test set x	76
Figure 6.10	Mean of empty features after filer with total parking equal 100	76
Figure 6.11	Performance testing on the SVR model	77
Figure 6.12	Performance testing on the LR model	78
Figure 6.13	Performance testing on the decision tree regression model	78
Figure 6.14	Performance testing on the fine-tuned SVR model, obtained using grid search	79
Figure 6.15	Performance testing on the fine-tuned SVR model, obtained using random search	79

Figure 6.16	Performance testing on the fine-tuned SVR model, obtained using random search plus	80
Figure 6.17	Performance testing on the fine-tuned SVR model, obtained using parameter optimization loop	81
Figure 6.18	Save the fine-tuned SVR model using random search as sav file	86
Figure 6.19	Parking Vacancy Prediction Model-GUI	87

LIST OF TABLES

Table Number	Title	Page
Table 2.1	Limitation of previous studies on ML	16
Table 2.2	Limitation of previous studies on deep learning	16
Table 3.1	FYP1 Gantt Chart details	21
Table 3.2	FYP2 Gantt Chart details	23
Table 5.1	Specifications of laptop	35
Table 5.2	Summarize for the software libraries	37
Table 5.3	Comparison of the datasets	42
Table 6.1	Prediction result	82
Table 6.2	Prediction result (after fine-tuning)	83
Table 6.3	Parameter Grid	84

LIST OF EQUATIONS

- (1) Generic form of the multiple linear regression model
- (2) Pearson's correlation coefficient (PCC)
- (3) mean absolute error (MAE)
- (4) mean square error (MSE)
- (5) root mean square error (RMSE)
- (6) mean absolute percentage error (MAPE)
- (7) mean Average Precision (mAP)
- (8) loss function for YOLO
- (9) Precision
- (10) Recall
- (11) F_1 score

LIST OF SYMBOLS

Σ	Sigma
+	Plus
-	Minus
*	Asterisk (multiplication)
\times	Multiplication
\div	Division
=	Equals
()	Bracket
%	Percent

LIST OF ABBREVIATIONS

<i>SVM</i>	Support Vector Machine
<i>SVR</i>	Support Vector Regression
<i>LR</i>	Linear Regression
<i>BPNN</i>	Backpropagation Neural Network
<i>ARIMA</i>	Autoregressive Integrated Moving Average
<i>FM</i>	Forecasting Method
<i>PL1</i>	Parking Lots 1
<i>PL2</i>	Parking Lots 2
<i>PL3</i>	Parking Lots 3
<i>PL4</i>	Parking Lots 4
<i>RMSE</i>	Root Mean Square Error
<i>MSE</i>	Mean Square Error
<i>MAE</i>	Mean Absolute Error
<i>ML</i>	Machine Learning
<i>FOA</i>	Fruit Fly Optimization Algorithm
<i>FOA-SVR</i>	Fruit Fly Optimization Algorithm with Support Vector Machine
<i>NN</i>	Neural Network
<i>SRM</i>	Structural Risk Minimization
<i>RBF</i>	Radial Basis Function
<i>C</i>	Regularization Parameter
<i>ANN</i>	Artificial Neural Network
<i>CNN</i>	Convolutional Neural Networks
<i>GUI</i>	Graphical User Interface
<i>YOLO</i>	You Only Look Once
<i>YOLOv3</i>	You Only Look Once, Version 3
<i>YOLOv4</i>	You Only Look Once, Version 4
<i>YOLOv5</i>	You Only Look Once, Version 5
<i>PUCPR</i>	Pontical Catholic University of Parana
<i>UFPR</i>	Federal University of Parana
<i>CARPK</i>	Car Parking Lot Collection
<i>GPU</i>	Graphics Processing Unit

<i>PCC</i>	Pearson's Correlation Coefficient
<i>MAPE</i>	Mean Absolute Percentage Error
<i>mAP</i>	mean Average Precision
<i>AI</i>	Artificial Intelligence
<i>IoU</i>	Intersection over Union
<i>MA</i>	Moving Average
<i>AR</i>	Autoregression
<i>ARMA</i>	Autoregressive Moving Average

CHAPTER 1 INTRODUCTION

Vehicle parking is one of the global issues, especially in urban areas. With the increasing economic development and urbanization, the number of cars increases rapidly, which leads to an imbalance between the supply and demand for parking lot [1]. In urban areas, people typically spend more than 5 minutes searching for a car park and the time needed is longer during the weekend. When people start finding a car park, this is where idling starts. Idling is one of the most waste fuel actions done by every driver, especially Malaysians. Idling can see everywhere, even though within the educational centre too. One hour idling a day might spend individuals between RM 2.60 to RM 2.80 per day and that is around RM78 to RM84 per month [2].

People nowadays are able to calculate the estimated driving time needed in order to reach the destination on time, sometimes however they could still face the challenges of being late due to obstacles in finding a car park. Increasing the parking area is not a sustainable solution when it is limited in space. However, applying effective parking management would be a realistic alternative [3]. A prediction of parking vacancies using ML would provide an estimation of how long it would take them to have their car parked. With the allotted time on distance and vehicle parking, there is less chance that people will be late for their school or work, etc.

Furthermore, people nowadays are less patient than before, especially doing things that they think are not worth it, for example, finding a parking space. Individuals' mood could spoil easily if they spend too much time finding a parking space. The longer the time an individual spends searching for parking, the less time will be for the next activity on their schedule. Additionally, people experience anxiety when trying to find a parking spot since they are unsure of when they will be able to park or where the next spot will be.

Parking vacancies prediction plays a part in protecting the earth as there will be less idling, which means reducing greenhouse gas emissions (CO₂) and pollutants that are harmful to human health. Moreover, parking vacancies prediction also significantly impacts people's planning and saves people's time. Such that, if the prediction result shows that finding parking vacancies requires more than 30 minutes, they will change their mind by calling a grab to their destination instead of driving by themselves.

Furthermore, parking vacancies prediction could avoid people's daily emotions from getting spoiled because of spending a long time finding a car park.

1.1 Problem Statement and Motivation

Nowadays, finding a parking space is challenging as the number of vehicles is growing faster than the number of parking spaces. Finding a parking space would be time-consuming, especially in a high population and car-density area. According to [4], individuals in Kuala Lumpur spend around 25 minutes daily searching for parking. A person who lives in Southeast Asia spends an average of 30 minutes seeking parking. Sometimes, the time estimate for individuals to find parking which according to the research, may not be accurate in real-time situations, which means individuals may spend 10 minutes, 30 minutes, or even more to get their vehicle parked.

There will be more petrol waste while the longer time individuals spend on searching parking. People usually have their car idling while waiting for an available parking space. Idling for 10 minutes costs between 1/10 and 4/10 of a litre of gasoline, depending on the individual's vehicle [2]. So, if more people put their car in idling, the more petrol will be wasted, and the money spent on refuelling their car.

Gasoline will continue to burst as long as the car's engine runs, indirectly causing environmental pollutants such as carbon dioxide, nitrogen oxide, and other hydrocarbons which escape through the tailpipe. These pollutants will cause climate change, air pollution, global warming and affect the health of all living creatures. According to [50], 11 hours of CO₂ emission while cruising for parking in a shopping center, where 4 h represent guest non-peak hours and the remaining 7 h represent guest peak hours, will result in 37kg of CO₂ emission, which is comparable to 122 km driven by an average normal car in an area of 320 m.

Furthermore, the weather condition is one of the parameters that would affect the prediction of the open space parking vacancies. Weather changes frequently and the changes in weather conditions like sunny to rainy would somehow restrict the movement of the parking, as during rainy days if people do not have an umbrella, it would restrict them from picking up their car or getting out of the car. Additionally, the weather also affects people's mood and thereby affects the number of parking vacancies,

for example during cloudy people will be unlikely to go out as they are afraid of getting themselves wet.

Next, this project aims to propose a prediction model for parking space vacancies to help individuals to identify the estimated time for the available parking space. Although the model cannot instantly create an available parking space for individuals, it does calculate the estimated time individuals will have to wait to get parking. With the estimated time given, individuals can save their time by doing other meaningful activities, such as replying to messages, catching up on social media and cleaning their car's cupboard instead of keeping their eyes on the parking lot all the time. Since people know how much time is needed to find their next open parking place, searching for a parking space won't ruin anyone's mood anymore. As a result, people won't feel anxious about finding parking as they always did.

Furthermore, parking prediction is a more environment-friendly solution and less petrol waste because there will be less idling. Through parking prediction, people spend lesser time finding a parking space, since available parking space is shown to them. Gradually, there will be less idling and lesser car emission, which less pollution to the earth.

1.2 Research Objectives

In this project, a review of existing studies has been conducted to identify the problems that occur in predicting open space parking vacancies. After reviewing the existing studies, an ML algorithm that is capable to fit well with the parking data and noise (etc, weather and nearby parking condition) is proposed for use in developing the parking prediction model and it is SVR.

The main objectives of this project are:

1. To investigate ML algorithms and techniques that have been used in the existing studies which are related to predicting open space parking vacancies. This allows us to study and identify the problems and other factors that occur in the existing study.
2. To propose ML algorithms which concerned with more noises to predict the parking space, like weather and nearby parking lot conditions. Dealing with environmental factors will increase the accuracy and performance of the model proposed.

3. To build the model using the proposed SVR algorithm to predict the available parking space. The proposed model is practical at handling time series and nonlinear problems, resulting in it being capable of predicting the parking space with a standard level of accuracy.
4. To test the proposed model's performance by using MSE, RMSE, MAE, and MAPE performance metrics. This allows us to identify how well the proposed model is in learning and predicting the available parking space.
5. To optimize the performance of an SVR model by employed multiple hyperparameter tuning techniques. By doing so, a higher likelihood of identifying the best set of parameters for SVR model with the selected dataset.

1.3 Project Scope and Direction

The scope of the project is to propose a parking prediction model using ML to predict available parking spaces in open parking spaces. The scale of open parking spaces includes small, medium and large parking lots. This model is considered with noise, such as weather of the day, the current capacity of the nearby parking lot and the road condition of the parking lot. So less chance that the prediction result is getting affected, thereby improving the accuracy of the prediction result.

To develop the parking vacancies prediction model, the software libraries that are going to use are the Scikit-learn library and the YOLO framework. Scikit-learn is the Python libraries use for developing ML algorithms. To issue these software libraries, the programming language will be python. Then YOLO is used to develop an object detecting model to collect the parking lot dataset and will be used to fit in the ML model.

One month later after the release of YOLOv4, YOLOv5 was released. A study [46] was conducted and showing that YOLOv5 is faster in inference time when compared to YOLOv4. However, they stated that YOLOv4 on Darknet continues to be the most accurate if the user is looking for the latest and not frightened of a little more specific configuration [46]. Hence, YOLOv4 algorithm was chosen to use in this project.

1.4 Contributions

This project will propose a parking vacancy prediction to make the parking lot more efficient. As parking space is predictable, utilization of the parking lot will be

maximized. Maximizing the utilization of parking lots will decrease the occurrence of street parking and double parking, which will increase the capacity of the road. As a result, the problem of traffic congestion will be alleviated as all the lanes on the road are smooth and unimpeded.

Individuals may save more money while also helping to keep the environment air cleaner. When idling is reduced, less petrol is consumed, which means less money is spent on refuelling their automobile. In addition, lesser idling means lesser gas emissions by the car and results in reducing the burden on the natural environment.

Furthermore, the object detection model built with YOLOv4 can classify empty and occupied parking spaces and later generate numerical data. These data will be used as a dataset to develop the parking prediction model. Thereby the data collection process in the parking lot shall be lightened as this process is automated. Moreover, the parking vacancy prediction model can be further used by researchers as a base for developing a parking prediction software or system.

1.5 Report Organization

In the first Chapter, an introduction and problem statement about parking vacancy are made. The objective, scope, and direction of this project are stated here as well as the contribution of this research. Then the details of this research are shown in the following chapters. In Chapter 2, previous studies related to predicting open space parking vacancy is being reviewed, and the technique used in these studies include ML, NN, and deep learning. strengths and weaknesses in each study are identified, and a proposed solution is founded at the end.

Chapter 3 details the system models proposed to address open space parking vacancies, including diagrams, pseudocode, and a timeline for the project. Chapter 4 focuses on the system design and presents the equations and techniques used in the object detection model and ML model, along with flowcharts. Additionally, a GUI framework is introduced for making predictions.

Chapter 5 lists the required resources for the experiments and simulations, including hardware, software, software libraries, and datasets. This chapter also includes the configuration and development of the model. Chapter 6 presents the system evaluation and discussion. This chapter covers the testing and analysis results for both the YOLOv4 and ML algorithms, which are thoroughly recorded and discussed.

CHAPTER 1

Lastly, in Chapter 7, the conclusion and recommendation section summarize the main findings and contributions of the study and suggest directions for future research.

CHAPTER 2 LITERATURE REVIEW

2.1 Previous works on Machine Learning

In the year 2020, a study was conducted by authors [3] in predicting the parking occupation where the study are focused on parking type and parking scale. The ML technique used in this study includes SVR, LR, BPNN, and ARIMA.

SVM is a statistical learning theory which able to handle linear and nonlinear problems and is useful for a wide range of tasks [6]. It is built on the concept of finding the optimal hyperplane for separating characteristics into distinct domains [7]. SVM can obtain decision-making rules and accomplish minimal error for independent tests based on the SRM principle to effectively handle learning problems [8].

SVM is a more complex model and is better fit when it has a greater number of parameters [9]. Even when the number of dimensions exceeds the number of samples, the method is still successful [10]. Last, a convex optimization problem is what SVM is. Finding the global optimum is simple because the locally optimal solution is also the optimal global solution. However, noise does affect the accuracy of SVM, which mean large data set is not suitable for SVM as the larger the data set, the more the noise [11].

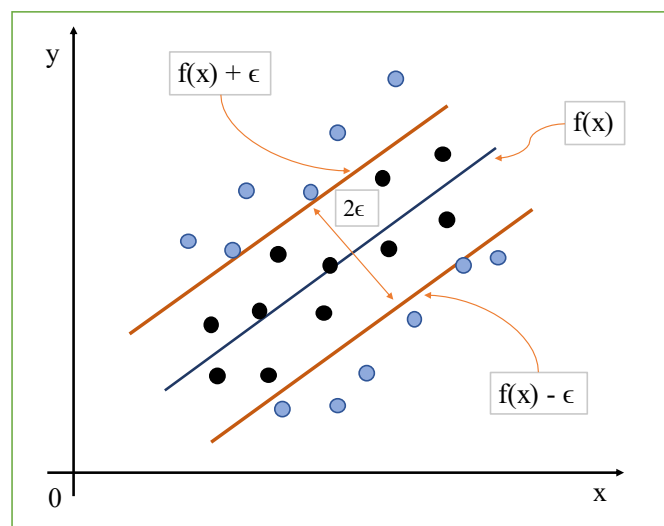


Figure 2.1: SVR diagram [5]

SVR is based on the same concept as SVM, but it is used to solve regression issues [12]. SVR is common to use in times series problems, and it has a strong ability to deal with nonlinear problems. SVR enables the determination of acceptable error in

the model and matches the data with a best-fit line, the hyperplane line with the greatest number of points [13]. Because it is based on the SRM concept, which is effectively a convex quadratic programming optimization problem with linear constraints, SVR decreases risk [5].

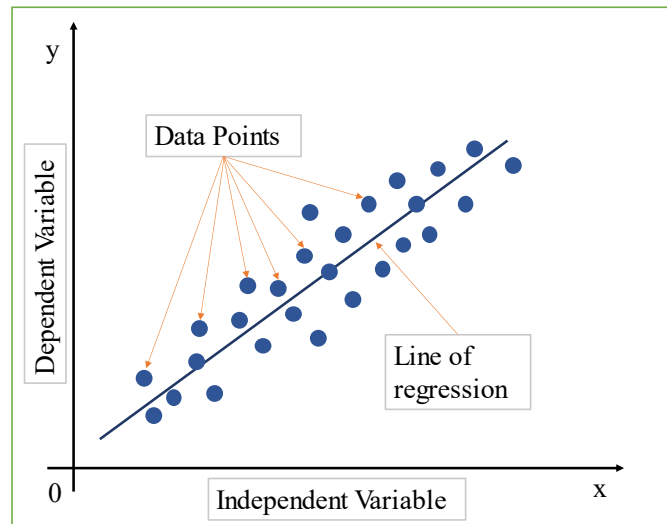


Figure 2.2: LR diagram [14]

LR is a supervised ML learning method. It carries out a regression function. The concept can be illustrated by diagram as Figure 2.2. The simplest concept of linear regression is to assign a point set, D , and create a function (line of regression) that fit the point set (datapoint) with the least number of errors possible. As a result, the term Linear Regression was being coined. Because linear regression indicates a linear relationship, it determines how the dependent variable's value changes as the independent variable's value changes [14].

The following is the generic form of the multiple linear regression model:

$$Y = B_0 + B_1X_1 + B_2X_2 + \dots + B_kX_k + u \quad (1)$$

Where

Y = the explained variable

X_i ($i=1, 2, \dots, K$) is the K explanatory variable

B_i ($i=1, 2, \dots, K$) is the $K+1$ unknown parameter

u = the random error term

LR is smooth in the calculation. There are no adjustment parameters, so it is simple and easy to understand and explain [15]. However, LR operates poorly when

there is a nonlinear relationship. The reason is they do not have the innate flexibility to record more complicated patterns [15].

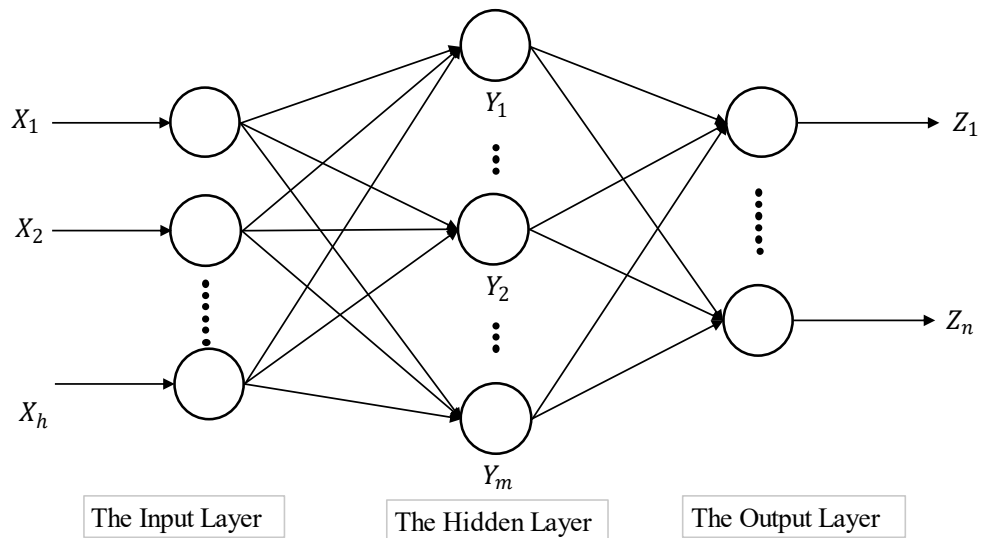


Figure 2.3: General structure of BPNN [16]

BPNN is a feed-forward hierarchical ANN made up of three or more completely linked layers of neurons, as shown in Figure 2.3 [17]. BP network can store and learn a lot of input-output mode-mapping relations without exposing the mathematical equation in advance.

Furthermore, the learning rule is to use the gradient descent approach using backpropagation to continually update the network's weight and threshold [3]. The heart of NN training is backpropagation. Backpropagation is the process of fine-tuning the weights of a NN depending on the preceding epoch's (i.e. iteration) error rate (i.e. loss). As a result, continually updating the network's weight and threshold do reduce error rates, boosting the model's generalization, thus making it more reliable [18].

In the training process of [3], the network's predictability was maximized by training the model with various combinations of these parameters. The predictability was being maximized because the learning process automatically adjusts the weights and thresholds in BPNN to reduce mistakes, allowing a single hidden layer BPNN to approximate any nonlinear function with arbitrary precision [19].

The fault tolerance and robustness of BPNN are excellent. The BP's convergence speed, on the other hand, is slow, and it's simple to slip into the local

minimum. Furthermore, because of its fundamental properties, BPNN's prediction outputs are unstable [5].

ARIMA is a ML algorithm. For prediction, ARIMA needs ongoing historical data. It is commonly use in demand forecasting, for example predicting future demand for the food production and the price of the stocks in the future based on previous prices. Based on the stability of the initial sequence and the variations in regression, ARIMA models may be separated into MA, AR, ARMA, and ARIMA. The advantages of the ARIMA model are it performs well for short-term predictions and models non-stationary time series. However, it is challenging to predict turning points, performs worse for long-term forecasting, and cannot be employed for seasonal time series [20].

In the study [3], the data collected in this study include total four parking lots from Shenzhen, Dongguan, and Shanghai with different scales. There is a total of 606,959 records of vehicles' in-time and out in the data set. A total of 7 weeks of data is collected. The data for the first 6 weeks are being used to train models, and the data for the last week are being utilized as test dataset.

There is two FM used in [3], which FM regards weekday and weekend as the same set and FM2 regards weekends and weekdays as two separate sets. The type of parking lot used to examine these ML techniques includes commercial, official and mixed functional and size includes large, medium and small. PL1 is large commercial parks. PL2 is medium office parking lots. PL3 is a small mixed-function parking lot. For the last parking lots, PL4 is small office parking lots.

The metric used to evaluate the model performance are RMSE and MAE, as shown in Figure2.4-2.5. The lesser the error rate represents, the better the algorithm.

Model	Parking lot	RMSE	MAE
LR	PL1	87.4	64.3
	PL2	12.2	8.9
	PL3	34.5	31.9
	PL4	32.1	23.0
ARIMA	PL1	136.0	97.0
	PL2	19.7	17.0
	PL3	27.3	21.7
	PL4	23.4	18.3
NN	PL1	89.0	63.9
	PL2	13.3	10.4
	PL3	35.7	32.8
	PL4	35.4	25.3
SVM	PL1	84.3	59.6
	PL2	8.4	6.2
	PL3	18.8	15.6
	PL4	23.3	17.9

Note. LR indicates linear regression. NN indicates backpropagation neural network.

Figure 2.4: Forecasting results of the four parks with FM 1 [3]

Model	Parking lot	RMSE	MAE
LR	PL1	88.2	60.7
	PL2	9.1	7.0
	PL3	36.2	28.9
	PL4	21.9	15.9
NN	PL1	99.5	66.3
	PL2	13.1	10.9
	PL3	36.7	30.6
	PL4	29.0	20.9
SVM	PL1	99.6	68.6
	PL2	8.2	6.1
	PL3	20.6	17.5
	PL4	21.4	15.7

Figure 2.5: Forecasting results of the four parks with FM 2 [3]

The ARIMA model was not applied in FM 2 because it is not suitable for this method as it is a time series model. In Figure 2.4, the ARIMA model overall has the worst performance in FM 1. In Figure 2.4 and Figure 2.5, BPNN which is stated as NN performs the worst for the mixed parking lots, PL3. As a result, the overview for the figures above shows that SVM has the best performance between ARIMA, LR and NN, except large commercial parking, PL1 for both forecasting methods. It concludes that SVM effectively solves practical problems using small samples, local minimum points, high-level pattern recognition, and nonlinearity.

Following in the year 2018, [5] has also conducted a similar study in predicting the number of vacant parking spaces. SVR is being applied and optimized using a FOA.

CHAPTER 2

The comparative models include BPNN extreme learning machine, and wavelet NN, which are all widely used prediction models.

The type of parking space includes the large parking lot of a local grand shopping mall and the small parking lot of a restaurant. The traffic is heavy for the local grand shopping mall, and the traffic flow trend is obvious. While for restaurants, the traffic is light.

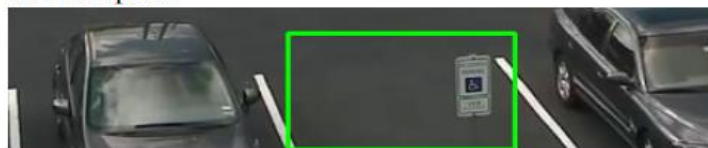
The data are collected in two days from 11.00 am to 8.00 pm and recorded every 10 minutes. A total of 108 data was collected. The 54 data collected on the first day is used as training data set, and the remaining data were used as the test data set.

The result shows that FOA-SVR exceeds all other models in terms of accuracy, while it is only second to SVR in terms of stability. This prediction model can also be used in any parking lot or garage if only information on car activities such as arrival and departure times, among other things, are provided.

2.2 Previous works on Deep Learning



(a) Prediction on some spots 3. The system does get triggered with a human passing through the classification area, which is classified as vacant. The spot 1 and 3 (first two rectangles to the left) are correctly classified as occupied.



(b) Prediction on some spots 2. The street sign does not trigger the classifier as it is too small to change the color distribution of the area covered by the green rectangle.

Figure 2.6: Current system in use to forecast when a parking space will become available [21]

This study which was conducted in 2018 by [21] shows how to use Caffe and the Nvidia DiGITS framework to create a real-time parking spot categorization system based on CNN. Some CNN configurations based on LeNet network with the Nesterov

Accelerated Gradient as solver and the AlexNet network with the Stochastic Gradient Descent were built in this study to utilise in the classification system.

CNN are complex feed forward NN in deep learning. The study in [21] said, the architecture of CNN is similar to a human NN, which is made up of synapses (weights) and neurons and deals with a complex task. It is a Deep Learning method that can take an image as input, assign importance (learnable weights and biases) to distinct aspects/objects in the image, and distinguish one from the other [23].

CNN has an accuracy greater than 90% inter and intra-datasets. Because of its great accuracy, CNN is utilized in picture categorization and recognition [24]. CNN was used in [21] as it makes the process of creating a classifier easier. It is because they extract and use information from the dataset automatically. CNN's performance is entirely dependent on the hardware employed. If the CNN contains several layers, the training process will take a longer time if the device does not have a powerful GPU [25].

There are 782 photos of parking lots in the collection, which were taken from two universities of the Witwatersrand parking sites. The data was collected during a working week (5 days), at a rate of one frame per minute, from 6 am until around 8 am. The camera's position and the buildings' placement in this study were set such that a car leaving a parking spot does not obscure more than 60% of the following automobile on the road.

The study result shows that computer vision employing a single camera and CNN has a success rate comparable to older approaches (using sensors). As illustrated in Figure 2.6, the system can produce an accurate output depending on the present state of the parking place the camera is facing, provided that the spot is accurately specified by the user when the system is activated.



Figure 2.7: Result of YOLOv3 vehicle detection when the vehicle is in-motion [22]

Next, in the year 2021, [22] conducted a study of parking availability prediction using YOLOv3 CNN on a university campus. YOLOv3 algorithm is used to train and predict whether the parking area is occupied or unoccupied. This study utilized three datasets: PKLot, COCO, and an in-house dataset. In this study, the proposed model is trained using 50% of the data, and the model is tested on the other 50%. In addition, the in-house dataset's video clip contains noise from various sources, such as rain and drivers adjusting their cars in the parking lot. This type of video clip could help to test the predicted performance of the suggested technique to find vacant parking spots.

Since YOLOv3 can maintain the complete segmented picture after segmentation, classification based on the full segmented image may very well be accomplished in this study. Additionally, YOLOv3 offers quick video rendering at 45 frames per second, which makes YOLOv3 algorithm a pleasant choice for real-time processing. YOLOv3 can identify vehicles in this study and calculate the available parking spaces, but it is still unable to detect moving vehicles, as shown in Figure 2.7.

The result of the study by [22] shows that the YOLOv3 algorithm can distinguish between vacant and occupied parking spaces in real-time. However, the algorithm may be improved in certain ways, particularly regarding the detection of incorrect parking and moving cars.

2.3 Limitations of Previous Studies

Studies from [3] and [5] did not take into account of nearby parking lots. For university, college, famous food court, and other parking spaces are severely lacking

CHAPTER 2

during peak hours. When the parking space is fully occupied, people will start parking their cars at the nearest parking space. As a result, a full parking lot affects the parking vacancy of the nearby parking lot, thereby affecting the accuracy of parking vacancy prediction.

Furthermore, the weather is one of the elements that affect the prediction result of the number of parking vacancies. The previous studies by [3] and [5] did not take into account weather as one of the features in their dataset. Weather does affect individuals' movement, as during rainy individuals may not pick up their car because of no umbrella or because the rain is too heavy to go outside.

In addition, weather was not considered in the previous study by [21]. Weather is an important element that can affect the accuracy of the parking vacancy prediction. Weather such as heavy rains can be affecting the image capture by the camera, and it might be difficult for the prediction model to learn the data and predict the available space. Therefore, countries near the equator and surrounded by the sea, such as Singapore as well as Malaysia, should consider the weather situation while predicting parking vacancies.

The parking space used to do training and testing for parking detection in the previous study are mostly captured in a good condition, which is the parking line is clear and no puddle because during and after rainy. The previous studies by [21] did not consider much on the noise that capture in the images, for example the parking spot that block by growing tree and streetlamp. Moreover, they did not consider parking space's road condition and do some special cases to handle the noises in the parking space. For example, the tree branching drop-down and blocked the parking space, the large vehicle which occupies more than one normal car park and puddle which it would cause reflection. In the absence of any special case assumption, the accuracy of parking space detection tends to decrease over time.

Table 2.1: Limitation of previous studies on ML

	[3]	[5]
Consider on the vacancy of the nearby parking lots	No	No

Consider on the weather condition (rainy, cloudy and sunny)	No	No
---	----	----

Table 2.2: Limitation of previous studies on deep learning

	[21]	[22]
Special cases consider on parking space's road condition	No	Yes
Noises consider on the parking spot that block by tree or streetlamp	No	Yes
Weather condition (rainy, cloudy and sunny)	No	Yes

2.4 Proposed Solutions

This project aims to improve the accuracy of the method used by considering the light of the environment (which cause by weather), weather situation, nearby parking situation, parking space's road conditions (double park and puddle), and the noise capture in the images that commonly appears such as parking spot that block by growing tree and streetlamp and do the correction on it. With the consideration of the element above, parking detection should be more accurate while performing the testing for the dataset.

South Asian countries experience unpredictable weather patterns, with frequent shifts between sunny and rainy weather due to their tropical rainforest climate and high annual rainfall. Given that the dataset collected in Malaysia, it is essential to consider the weather conditions in developing an accurate prediction model.

SVR was chosen to develop the prediction model due to its strong ability to handle nonlinear time series problems and reduce the risk of inaccurate predictions based on the SVM concept. The prediction accuracy of the SVR model should be improved through hyperparameter tuning. This process can aid in the identification of the best hyperparameters for the model. Moreover, a Python GUI framework name PySimpleGUI should be employee to develop a GUI for end user to perform prediction on the parking vacancy model.

This project utilizes CNN due to its high accuracy rate of 90% and ability to automatically extract and utilize information from datasets, making the creation of a classifier easier. Additionally, CNN was utilized in processing the images in the parking prediction dataset. The YOLO algorithm, which employs CNN and is known for its rapid video rendering capabilities, was employed for the image processing in this project's parking prediction.

2.5 Summary

Out of the four ML algorithms studied, SVM is effective for models with many parameters, and the convex optimization problem of SVM makes finding a global optimum simple. However, SVM is unsuitable for building a prediction model because it is a classifier. Therefore, it is replaced by SVR, which is based on SVM and is a regressor. LR's equation is easy to understand, but it is not flexible enough to record complicated patterns, and it performs worse when there is a nonlinear relationship. BPNN is great in fault tolerance and robustness; however, it is slow in convergence speed and easily falls into the local minimum. Therefore, BPNN may not perform well without a large training dataset. In conclusion, SVR is undoubtedly the ideal option to develop the parking vacancy prediction model.

To optimize the SVR model's prediction accuracy, four hyperparameter tuning techniques were utilized, namely, grid search, random search, random search plus and parameter optimization loop. These methods allow for a systematic search of hyperparameter values to identify the optimal combination for the specific problem at hand.

After reviewing previous studies, CNN was found to have a great accuracy result. Creating a classifier is easier because it automatically extracts and uses information from the data set. In addition, CNN needs modification to deal with all probable noise, so it requires a lot of input data and training. As a result, YOLO is a realistic option to use to develop the object detection model since it applies CNN and has CNN characteristics like automatically adjusting the image size during training and excellent accuracy.

CHAPTER 3 SYSTEM MODEL

3.1 Model Design Diagram

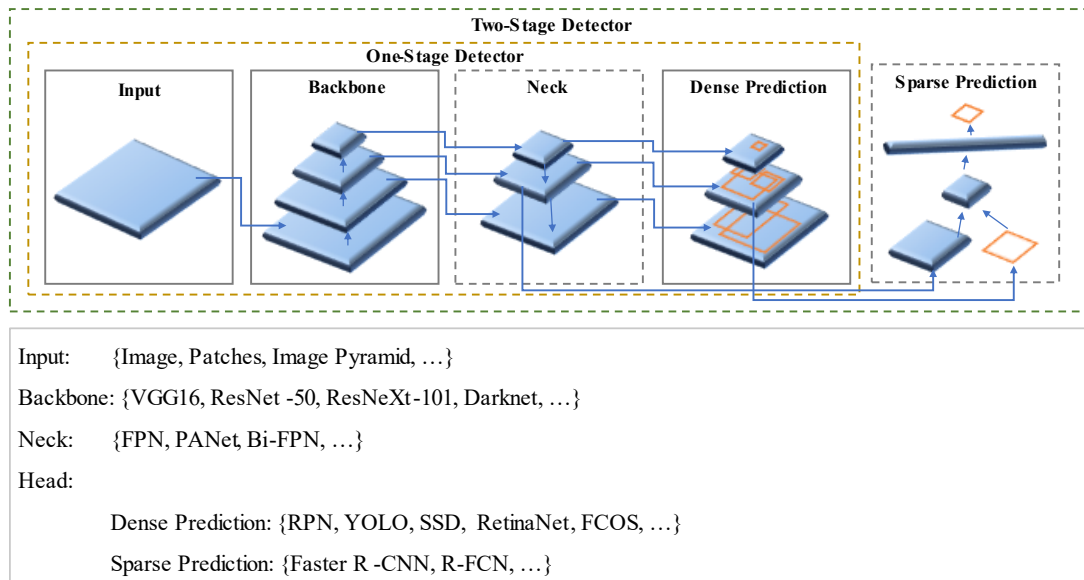


Figure 3.1: YOLOv4 Architecture [38]

The YOLOv4 architecture (in Figure 3.1) is made up of several components [38]. The input is the collection of training images fed into the network, processed in parallel batches by the GPU. The Backbone and Neck modules are responsible for extracting and combining features, respectively. The detection neck and detection head work together as the object detector, while the head performs the detection and prediction of objects.

The SVR diagram (as in Figure 2.1) features an interval with $f(x)$ as its central value and a width of 2ϵ . If the prediction for a training sample that falls within the range, as defined by the lines $f(x) + \epsilon$ and $f(x) - \epsilon$, is deemed accurate, and thus no loss is incurred. In contrast, a loss is computed for predictions that fall outside of this range. The loss is only calculated when the absolute difference between the predicted and actual values exceeds the insensitivity threshold (ϵ).

3.2 Pseudocode

The table below shows the pseudocode for developing the parking vacancy prediction model, which start from data loading, data pre-processing, training with ML algorithms process, fine-tuning model process till model testing process.

Parking vacancy prediction model pseudocode:
Read the labelled file and transform the data into a data frame.
Save the data frame table as a CSV file.
Load the table in the CSV file into Jupyter Notebook as a data frame.
Convert the features in the data frame into integer and datetime date type (time series).
Convert categorical data to numerical data.
Standardize the numerical data.
Split the dataset into input matrix and output vector.
Split the dataset into test and train sets.
Train and validate the model.
Evaluate the model using the train, validation and test dataset.
Fine-tuning model using test dataset.
Make prediction using the model.

The table below shows the pseudocode for developing the custom YOLOv4 object detection from create storage, download needed library, data pre-processing, training process till the testing process (start prediction).

Custom YOLOv4 object detection pseudocode:
Connect Google Colab to Google Drive.
Download the darknet library from GitHub into Google Drive.
Customize the configuration file and set only two classes for the training later.
Paste the dataset and all the configuration files into Google Drive.
Split the dataset into train and test sets.
Download pre-train YOLOv4 weight.
Enable the GPU and OPENCV before building the darknet.
Build the darknet and start the training process using the pre-train YOLOv4 weight.
Test the model performance on unseen video and pictures using the best weight created.

Predict images and generate the labelled text file for later use in the prediction model.

3.3 Gantt chart

FYP 1 Gantt chart.

CHAPTER 3

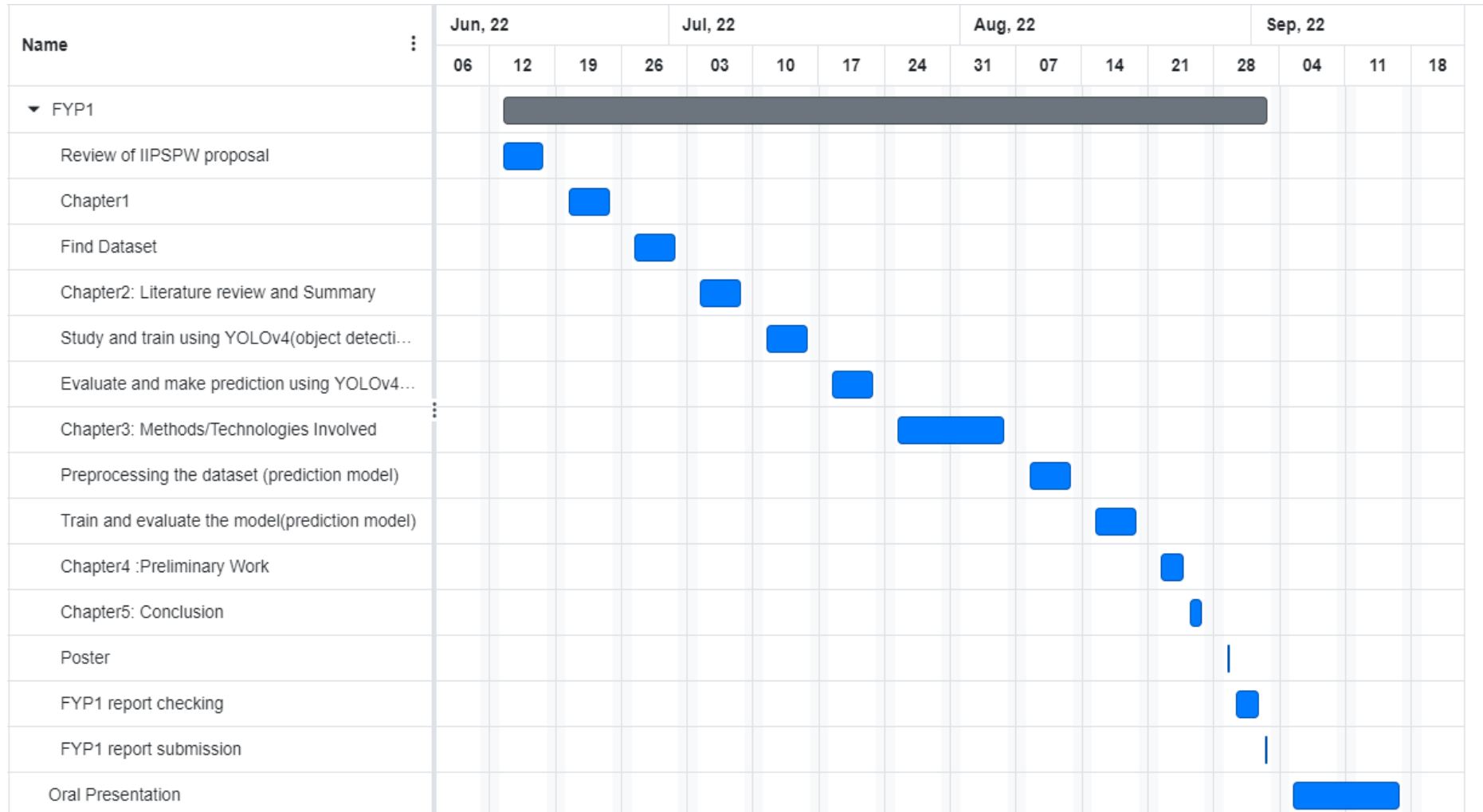


Figure 3.2: Gantt Chart of FYP1 from June 13, 2022 (week 1) to September 16, 2022 (week 14)

Table 3.1 FYP1 Gantt Chart details

ID	Task Name	Duration	12-06-22	02-09-22
1	FYP1	60 days	13-06-22	17-06-22
2	Review of IIPSPW proposal	5 days	20-06-22	24-06-22
3	Chapter1	5 days	27-06-22	01-07-22
4	Find Dataset	5 days	04-07-22	08-07-22
5	Chapter 2: Literature review and Summary	5 days	11-07-22	15-07-22
6	Study and train using YOLOv4(object detection model)	5 days	18-07-22	22-07-22
7	Evaluate and make prediction using YOLOv4 (object detection model)	5 days	25-07-22	05-08-22
8	Chapter 3: Methods/Technologies Involved	10 days	08-08-22	12-08-22
9	Pre-processing the dataset (prediction model)	5 days	15-08-22	19-08-22
10	Train and evaluate the model (prediction model)	5 days	22-08-22	24-08-22
11	Chapter 4: Preliminary Work	3 days	25-08-22	26-08-22
12	Chapter 5: Conclusion	2 days	29-08-22	29-08-22
13	Poster	1 day	30-08-22	01-09-22
14	FYP1 report checking	3 days	02-09-22	02-09-22
15	FYP1 report submission	1 day	05-09-22	16-09-22
16	Oral Presentation	10 days	12-06-22	02-09-22

FYP 2 Gantt chart.

CHAPTER 3

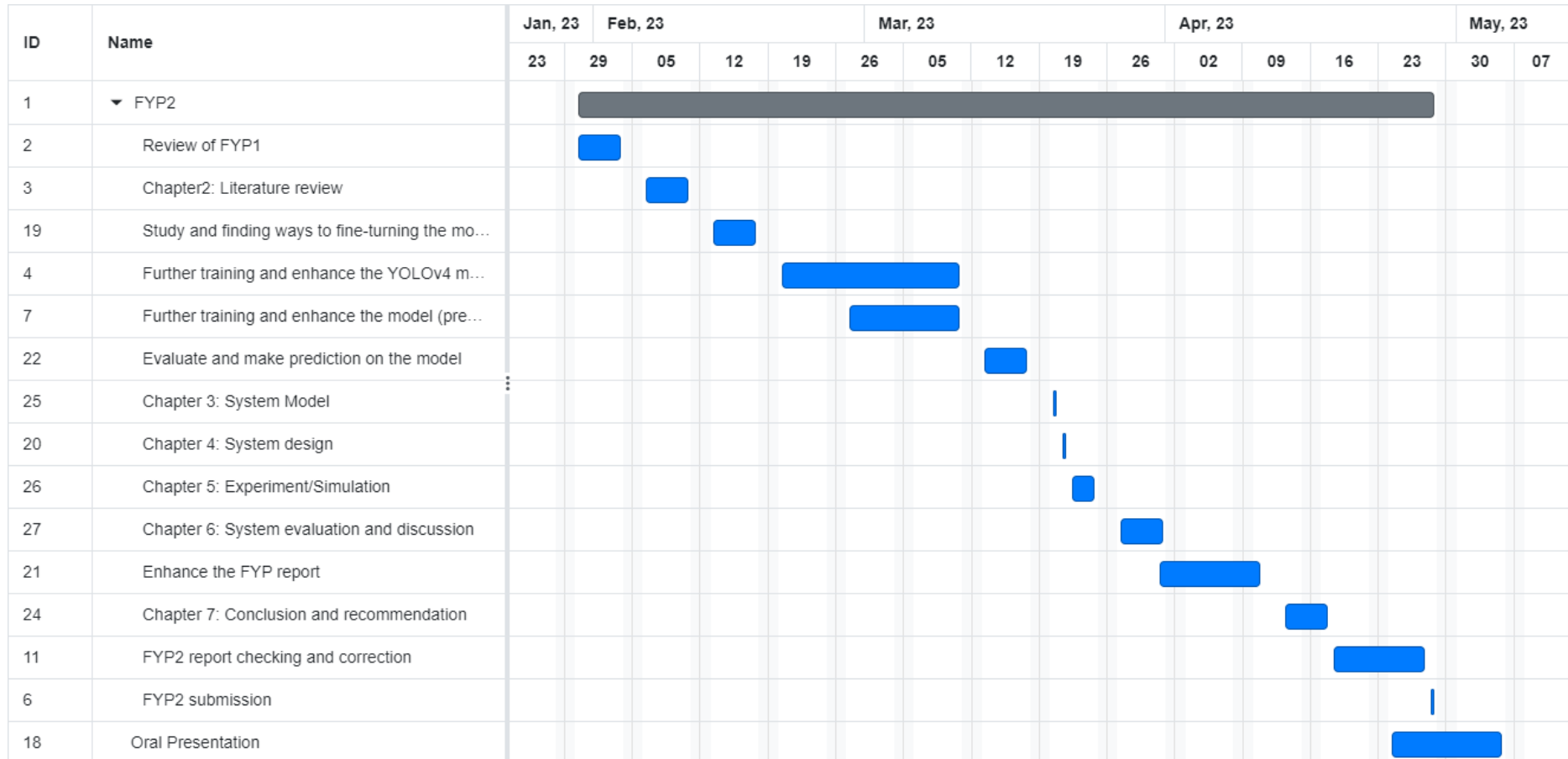


Figure 3.3: Gantt Chart of FYP2 from January 30, 2023 (week 1) to March 5, 2023 (week 14)

Table 3.2 FYP2 Gantt Chart details

ID	Name	Duration	Start Date	End Date
1	FYP2	65 days	30-01-23	28-04-23
2	Review of FYP1	5 days	30-01-23	03-02-23
3	Chapter2: Literature review	5 days	06-02-23	10-02-23
4	Study and finding ways to fine turning the model	5 days	13-02-23	17-02-23
5	Further training and enhance the YOLOv4 model (object detection model)	15 days	20-02-23	10-03-23
6	Further training and enhance the model (prediction model)	10 days	27-02-23	10-03-23
7	Evaluate and make prediction on the model	5 days	13-03-23	17-03-23
8	Chapter 3: System Model	1 day	20-03-23	20-03-23
9	Chapter 4: System design	1 day	21-03-23	21-03-23
10	Chapter 5: Experiment/Simulation	3 days	22-03-23	24-03-23
11	Chapter 6: System evaluation and discussion	5 days	27-03-23	31-03-23
12	Enhance the FYP report	7 days	31-03-23	10-04-23
13	Chapter 7: Conclusion and recommendation	3 days	13-04-23	17-04-23
14	FYP2 report checking and correction	8 days	18-04-23	27-04-23
15	FYP2 submission	1 day	28-04-23	28-04-23
16	Oral Presentation	10 days	24-04-23	05-05-23

CHAPTER 4 SYSTEM DESIGN

4.1 Machine Learning Equation

To develop a parking vacancy prediction model, two models are necessary. The first model is an object detection model that employs YOLOv4 to detect the number of empty and occupied parking spaces in input images. It produces labelled text files for each input image. The second model employs SVR to create the parking vacancy prediction model. The output text files from the object detection model serve as the input dataset for the SVR model [1].

The cost function includes PCC, and the k-fold Cross-validation method. Then, MAE, MSE, RMSE, and MAPE are used to evaluate to performance of the SVR model. Later, mAP is used in YOLOv4, and the loss function is used in YOLOv4. Precision and recall are then used to evaluate the classification model, and the F₁ score, which is a harmonic mean of precision and recall, is used as the final evaluation metric.

Four types of hyperparameter tuning methods are utilized to determine the optimal combination of hyperparameters for the SVR model, which include grid search, random search, random search plus and parameter optimization loop.

To identify the most useful features in the dataset before training, it is necessary to use a method such as PCC [39]. By applying this equation, it is possible to determine which features exhibit a strong positive correlation with the output feature. The feature with the highest correlation can then be identified as the most useful feature in the dataset.

Here is the equation for Pearson's correlation coefficient (PCC):

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \quad (2)$$

Where

r = Pearson Coefficient

n = number of attributes

$\sum xy$ = sum of products of the x and y values

$\sum x$ = sum of the x values

$\sum y$ = sum of the y values
 $\sum x^2$ = sum of the squared x scores
 $\sum y^2$ = sum of the squared y scores

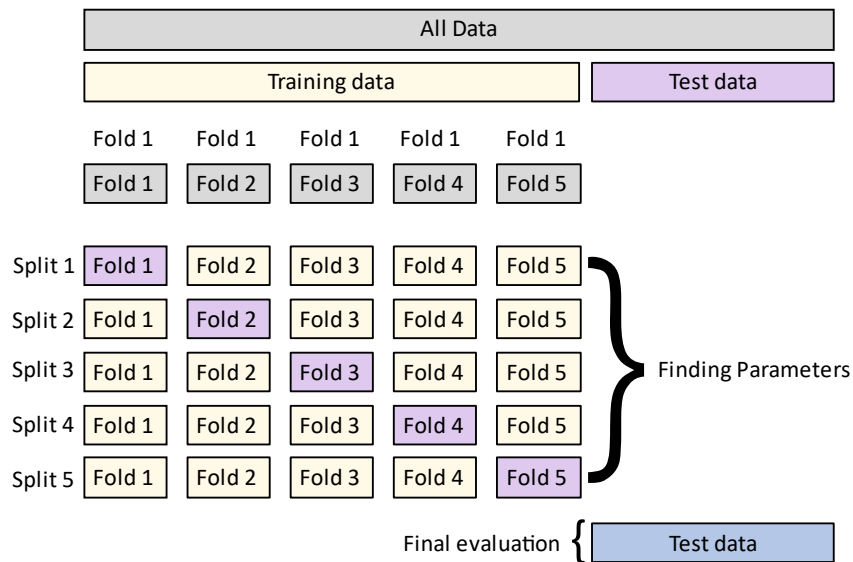


Figure 4.1: k-fold Cross-validation method [39]

Subsequently, to evaluate whether the algorithm is overfitting or underfitting, the k-fold Cross-validation method is used [39]. Cross-validation is used when a piece of the training set is put aside expressly for analysis and optimization. Parameters are learned by this method on the training set and then evaluated the performance on the validation set. In Figure 4.1, other folds are used as a training set and saved one-fold as a validation set. It repeats using the validation set as each fold in turn.

To measure the performance of the model, performance metrics are used. The first equation is MAE [3]. The size of errors for the entire group is determined by MAE using the average of absolute errors for a set of predicted and real values. Smaller MAE indicates it is better.

Here is the equation for mean absolute error (MAE):

$$MAE = \frac{1}{N} \sum_{i=1}^N |\hat{y}_i - y_i| \quad (3)$$

Where
 MAE = mean absolute error

N = Total number of data points
 y_i = true value/observed value
 \hat{y}_i = predicted value

Moreover, the equation used to evaluate the performance result of the prediction model is MSE. It is certainly the most straightforward and typical loss function. MSE is expressed in units of the target variable's square. It is used to calculate the difference between predicted and observed values.

Here is the equation for mean square error (MSE):

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (4)$$

Where

MSE = mean square error
 N = Total number of data points
 y_i = observed value
 \hat{y}_i = predicted value

The next equation is the arithmetic square root of MSE, or RMSE is used to calculate the difference between predicted and observed values too [3]. Larger errors are effectively penalised more harshly by RMSE. The accuracy of the prediction model can be evaluated by calculating the RMSE. A smaller value of RMSE indicates a better fit between the data and the model, and therefore a more accurate prediction model.

Here is the equation for root mean square error (RMSE):

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2} \quad (5)$$

Where

$RMSE$ = root mean square error
 N = Total number of data points
 y_i = observed value
 \hat{y}_i = predicted value

Furthermore, a statistical metric MAPE is used to examine how accurately a ML algorithm performs on a given dataset. The error referred to by the model

evaluation can be defined by MAPE as a loss function. A smaller number of MAPE indicate the model is better and with lesser error. Compared to RMSE, MAPE is a better option since it is expressed as a percentage, which is simple for both developers and end users to understand [48].

Here is the equation for mean absolute percentage error (MAPE):

$$MAPE = \frac{1}{N} \sum_{i=1}^N \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (6)$$

Where

$MAPE$ = mean absolute percentage error

N = Total number of data points

y_i = observed value

\hat{y}_i = predicted value

Next, mAP is used to evaluate the object detection model using YOLOv4. It began by going through the process of turning a prediction score into a class label. Then, a precision-recall curve is produced using various thresholds. Average precision is measured from the curve. In the end, sum up the average precision of all the classes and divide by n number of classes [41]. A higher score results in greater precision of the model's detections.

Here is the equation for mean Average Precision (mAP):

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k \quad (7)$$

Where

mAP = mean Average Precision

n = number of classes

AP_k = the average precision (AP) of class k

Next, YOLO calculated loss using the sum-squared error between the predictions and the actual data [40]. The classification loss, the localization loss, and the confidence loss make up the loss function. If an object is spotted, the squared error of the class conditional probabilities for each class represents the classification loss for each cell. Next, localization loss measures the size and position errors of the predicted border boxes. Moreover, the confidence loss is the measurement of the objectness of

the box, and it includes 2 equations where the first equation is used when an object is detected in the box, and the second equation is used when an object is not detected in the box. Eventually, all these three loss equations form a loss function for YOLO.

Here is the loss function for YOLO [40]:

$$\begin{aligned}
& \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{8}$$

Where

Line 1&2: Localization loss

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0

λ_{coord} = increase the weight for the loss in the boundary box coordinates

Line 3: Confidence loss (front & back)

Front:

\hat{C}_i = the box confidence score of the box j in cell i

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0

Back:

$\mathbb{1}_{ij}^{noobj}$ = the complement of $\mathbb{1}_{ij}^{obj}$

\hat{C}_i = the box confidence score of the box j in cell i

λ_{noobj} = weights down the loss when detecting background

Line 4: Classification loss

$\mathbb{1}_i^{obj} = 1$ if the area i contains an item; otherwise, 0.

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i

Next, precision and recall are used to evaluate the classification model, which the object detection model. Precision measures how accurately a positive prediction was made. Recall measures the proportion of accurately recognised positive samples [39].

Here is the equation for precision:

$$Precision = \frac{TP}{(FP+TP)} \quad (9)$$

Where

TP = True positive

FP = False positive

Here is the equation for recall:

$$Recall = \frac{TP}{(FN+TP)} \quad (10)$$

Where

TP = True positive

FN = False negative

Next, F_1 score is applied to integrate recall and precision into a single score. When precision and recall are large, the value of F_1 score is high.

Here is the equation for F_1 score:

$$F_1 = \frac{TP}{TP + \frac{FN+FP}{2}} \quad (11)$$

Where

TP = True positive

FN = False negative

FP = False positive

Four types of hyperparameter tuning methods are utilized to determine the optimal combination of hyperparameters for the SVR model, which include grid search, random search, random search plus and parameter optimization loop.

The grid search technique is used to find the best hyperparameter combination for the SVR model by exhaustively testing every combination on a grid [39]. However, this method can take a long time and require a lot of computing power when the model has many hyperparameters or large grids.

The next hyperparameter tuning method is random search. This method has an advantage over grid search in terms of time and processing power because it only evaluates a limited range of hyperparameter combinations. The characteristic of random search allows for a wider range of hyperparameters to be explored without significant increases in computation time. However, this approach may result in higher variance during computation.

Another hyperparameter tuning method is random search plus, which can yield results comparable to random search with fewer samples by dividing the hyperparameter space into cells and exploring each one systematically. This ensures that no cell is overlooked and reduces the likelihood of missing a promising solution. Study from [52] have shown that random search plus outperforms scikit-learn's random search by 10-50% in SVM models. Furthermore, random search plus can sample more efficiently than random search, with shorter runtimes for each run. The optimal method for dividing the space for random search plus is to use a value of $k = 3$.

The final hyperparameter tuning method is the parameter optimization loop, which utilises a defined search strategy to iteratively adjust the parameters [53]. For each parameter, an output flow variable is generated and applied to the model. This method uses a random search strategy whereby the parameter combinations are picked at random and assessed. The loop comes to an end after a pre-set number of iterations.

4.2 Framework

PySimpleGUI is the GUI framework employed in this project to create an interactive UI display for predicting parking vacancy. It is open source and cross-

platform, with a lot of documentation and tutorials available. PySimpleGUI integrates with tkinter, Qt (pyside2), wxPython, and Remi (for browser support), making GUI development incredibly quick and easy to learn. It only requires a Python3 environment and can be installed with a single line of code (`pip install pysimplegui`). Users only need to call the PySimpleGUI library before using it [51].

Compared to other popular Python GUI frameworks in the market, PySimpleGUI is much easier to use and learn. In conclusion, PySimpleGUI is a great choice for Python developers who want to create graphical user interfaces quickly and easily without sacrificing functionality or control.

4.3 Flowchart

Figure 4.2 shows the flowchart of developing the parking vacancy prediction model using ML algorithm, SVR and Figure 4.3 shows the development flowchart of the custom YOLOv4 object detection model.

Prediction Model Flowchart

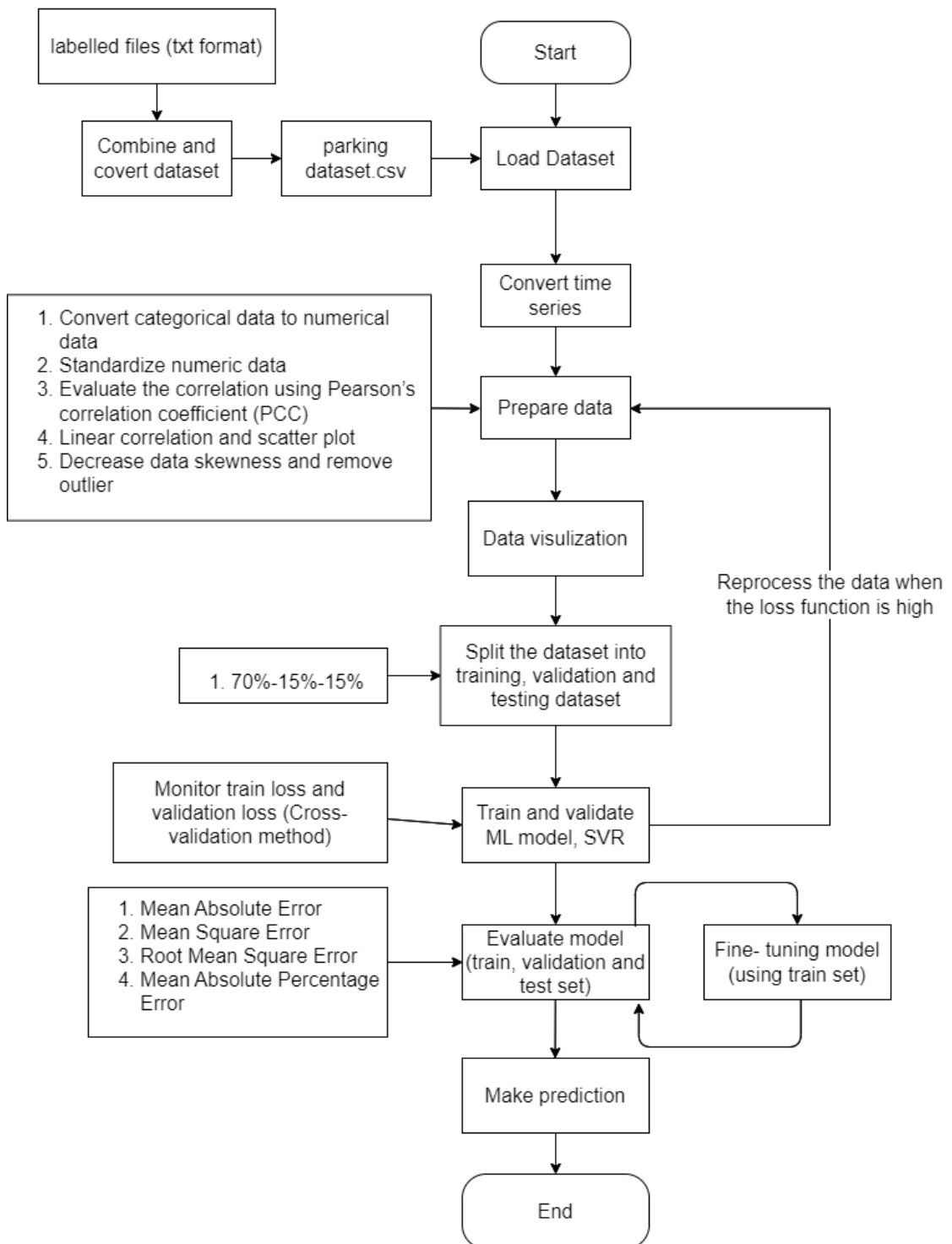


Figure 4.2: Development Flowchart of Parking Vacancy Prediction Model

YOLOv4 object detection flowchart

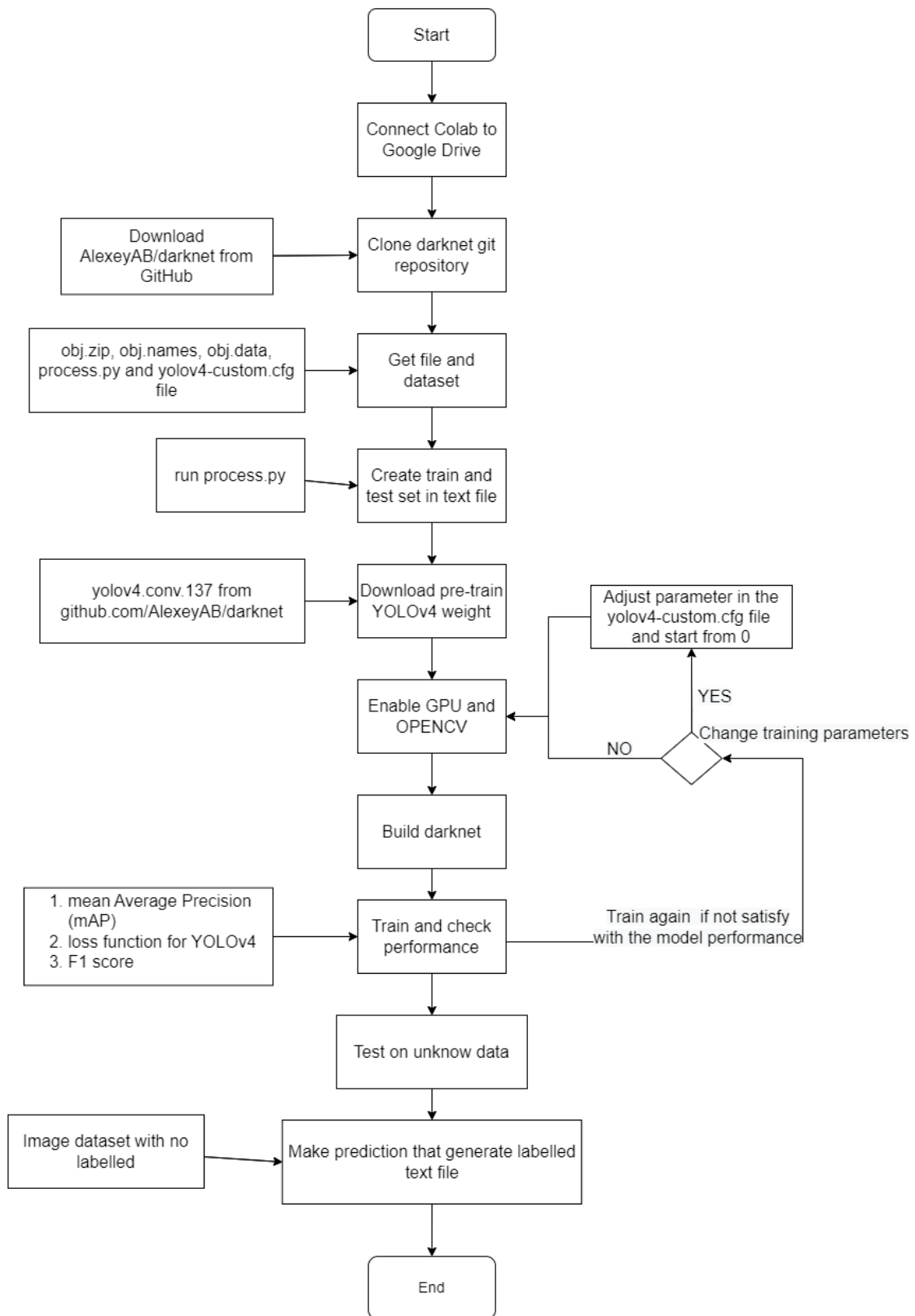


Figure 4.3: Development Flowchart of Custom YOLOv4 Object Detection Model

CHAPTER 5 EXPERIMENT/SIMULATION

5.1 System Requirement

5.1.1 Hardware

The hardware involved in this project is a laptop. A laptop was used to build the parking prediction model. Table 5.1 show the specification of the hardware used in conducting the research.

Table 5.1 Specifications of laptop

Description	Specifications
Model	Asus A510U series
Processor	Intel Core i5-8250U
Operating System	Windows 10
Graphic	Intel® UHD Graphics 620 (FT2) DDR3 & NVIDIA GeForce MX150 DDR5
Memory	4GB DDR4 + 8GB DDR4 RAM
Storage	TOSHIBA MQ04ABF100 1TB SATA HDD

5.1.2 Software

The software used in this project is Google Colab, and Jupyter Notebook runs in an anaconda environment.

Google Colab is a free cloud service for Python programming offered by Google. It allows users to access and work on Jupyter notebooks without needing to download or install any software. Additionally, Colab provides free access to GPUs, which is essential for training a customized YOLOv4 object detector. As the laptop used for this project lacked a strong GPU, Colab was utilized for the YOLOv4 model development. The notebooks created in Colab are saved in Google Drive and can be easily shared, similar to Google Docs or Sheets.

Jupyter Notebook is a web-based Python IDE designed for creating and sharing documents with text, live code, equations, and visualisations [27]. It is open-source software. Data scientists and researchers often prioritize data analysis over development, and Jupyter Notebook is an excellent tool for this purpose. It provides an interactive computational environment for creating data science apps, enabling users to experiment with data and see the results of the code for each command they execute [28]. Therefore, Jupyter Notebook was chosen for developing the parking vacancy prediction model due to its interactivity and ease of sharing. However, it was not used to develop the YOLOv4 object detection model.

Anaconda is an open-source distribution of the Python and R programming languages for data research that tries to streamline package management and deployment in Python version 3.9 [29]. A GUI programme called Anaconda Navigator is part of the Anaconda distribution and makes it simple to install, run, and configure applications like Jupyter Notebook. An isolated environment is one created using Conda Python. It enables you to install packages without changing the Python setup on your machine [30]. So, Anaconda has chosen to act as a virtual environment for running Jupyter Notebook.

5.1.3 Software libraries

The software libraries adopted in this project are Scikit-learn, YOLOv4 algorithm and PySimpleGUI.

Scikit-learn is one of the popular Python libraries for developing ML algorithms [31]. Scikit-learn includes libraries in solving classification, regression, clustering, pre-processing, model selection, and dimensionality reduction problem. SVM libraries is use in this project, the SVR algorithm is applying in this project especially during the model development.

Next, the YOLOv4 algorithm was a real-time object detection model created by Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao [43]. At the time of its release in April 2020, YOLOv4 was declared to be one of the most advanced real-time object detectors. It operates by splitting the object detection problem into two components: regression, which uses bounding boxes to identify object location, and

classification, which identifies the object's class. YOLOv4 is built using the Darknet framework [33]. Compared to YOLOv3, YOLOv4 performs 12 percent quicker and 10 percent more accurately [32]. Then in study [46] was conducted and showing that YOLOv5 is faster in inference time when compared to YOLOv4 but they also mention YOLOv4 on Darknet continues to be the most accurate if user is looking for the latest and do with more specific configuration. Hence, YOLOv4 algorithm was chosen to use in this project.

PySimpleGUI is a Python library that offers a robust and user-friendly GUI framework, built on top of the tkinter library. It builds on top of tkinter library, makes it easier to create GUIs by providing a wide range of capabilities and a simple API. It utilizes the Python programming language to develop a GUI that simplifies the visualization and interaction of data for users. PySimpleGUI can be easily installed using the "*pip install pysimplegui*" command. The use of PySimpleGUI can greatly expedite the prediction of parking vacancy models and create a simpler and clearer user experience.

Table 5.2: Summarize for the software libraries

Scikit-learn	<ul style="list-style-type: none"> • Python libraries for developing ML algorithms. • Libraries for solving ML problem, classification, regression and etc. • Include SVR algorithm
YOLOv4 algorithm	<ul style="list-style-type: none"> • Real-time object detection model • Faster and more accuracy than YOLOv3 • Slower than YOLOv5 in inference time • Continues to be the most accurate, without afraid of having more specific configuration
PySimpleGUI	<ul style="list-style-type: none"> • Python GUI framework • Easy install • Quick to learn and easy apply

5.1.4 Datasets

Total of four datasets is used in this project. The datasets included PKLot dataset [34], CARPK dataset [35], CNRPark+EXT dataset [36], and Aerial View of Parking Lot [37].



Figure 5.1: PKLot PUCPR sample image during sunny weather [34]



Figure 5.2: PKLot UFPR04 sample image during rainy weather [34]



Figure 5.3: PKLot UFPR05 sample image during cloudy weather [34]

The PKLot dataset [34] contains 12,417 labeled images of parking lots. All images were snapped at the parking lots of PUCPR and UFPR, which located in Curitiba, Brazil, classified as vacant or occupied. The dataset is segmented based on weather conditions, including sunny, overcast, and rainy, as shown in Figure 5.1-5.3. The images display various challenges such as shadows from trees and buildings, excessive sunshine exposure, poor lighting on rainy days, and differences in perspective.



Figure 5.4: CARPK sample image [35]

The second dataset utilized in this study is the CARPK dataset [35], which includes almost 90,000 vehicles photographed by drones in four parking lots. The images were captured from a height of approximately 40 meters, and each automobile's bounding box is indicated on the image set. Additionally, a portion of the dataset from PUCPR, which represents scenes that are obscured from the aerial view in the PKLot dataset, was incorporated. The training set of the CARPK dataset comprises 989, and the testing set consists of 459 images. This study was partially funded by Taiwan's Ministry of Science and Technology, thus justifying that part of the CARPK dataset (in Figure 5.4) was collected in Taiwan, although the study does not explicitly specify the dataset collection location.



Figure 5.5: CNRPark+EXT Camera 1 sample image [36]

The third dataset used in this project is CNRPark+EXT dataset [36]. The dataset CNRPark+EXT was created on a parking lot with 164 parking spots and contains around 150,000 labelled pictures. CNR-EXT is a subset of CNRPark+EXT. It is made up of photographs taken between November 2015 and February 2016 by 9 cameras using diverse views and angles of view and in varying weather situations (sunny, overcast and rainy) in the CNR Research Area in Pisa, Italy. CNR-EXT records a variety of lighting circumstances, including partial occlusion patterns imposed on barriers like trees, lampposts, and other vehicles, as well as completely or partially shadowing of the vehicles (Figure 5.5).

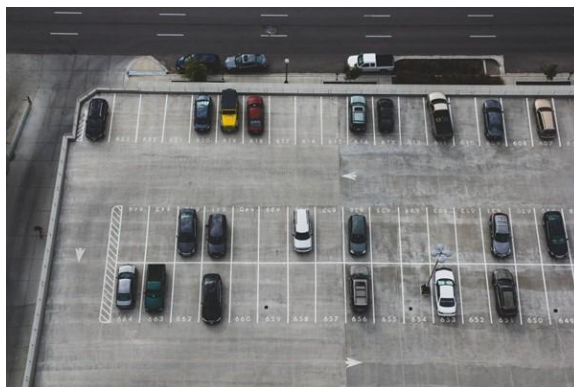


Figure 5.6: Aerial View of Parking Lot sample image [37]

The last dataset used in this project was named Aerial View of Parking Lot [37]. The dataset consists of pictures of several parking lots that were captured using a drone. To manage the large images, each raw image was divided into 6 smaller images, which were then compressed as shown in Figure 5.6. In total, 299 images were collected and

CHAPTER 5

divided into two segments: 280 images for training and 19 images for testing. The images were labelled using the Make-Sense.ai labelling tool. The dataset was collected by students at Tiangong University, and it can be reasonably inferred that the dataset was collected in Tianjin, China, although the study does not explicitly state the location of the dataset collection.

Table 5.3: Comparison of the datasets

Data set	Size	Location taken	Noise (consider)	Testing set size	Training set size
PKLot	12,417	Pontical Catholic University of Parana (PUCPR), and Federal University of Parana (UFPR), located in Curitiba, Brazil	Climate condition, buildings shadows, presence of the trees, lampposts, and other vehicles, excessive sunlight exposure	2483	9934
CARPK	1448	Pontical Catholic University of Parana (PUCPR) and Taiwan	Buildings shadows	459	989
CNRPark+EXT	144,965	CNR Research Area in Pisa, Italy	Climate condition, buildings shadows, presence of the trees, lampposts, and other vehicles, excessive sunlight exposure	31825	113,140
Aerial View	299	Tianjin, China	-	19	280

5.2 Setting up software

Before begin developing the ML model, Anaconda needs to be downloaded and installed on the laptop. Since Google Colab is a web IDE, no download or installation is required. Anaconda can be easily installed by downloading Anaconda Python 3.9 64-Bit Graphical Installer for Windows. After successfully installing Anaconda, Jupyter Notebook is ready to use.

5.3 YOLOv4 model

5.3.1 Configuration on YOLOv4 model

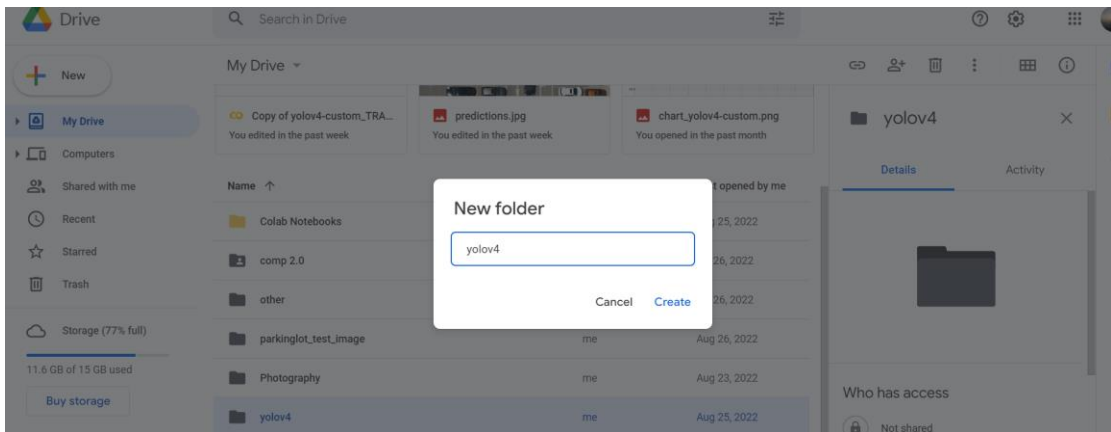


Figure 5.7: Create a yolov4 folder in Google Drive

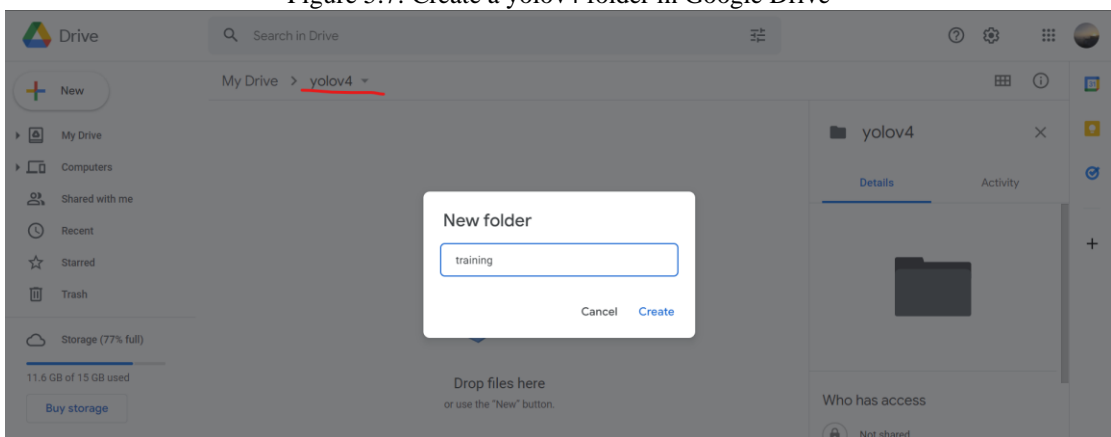


Figure 5.8: Create a training folder inside the yolov4 folder

Step 1, create a yolov4 folder in Google Drive (Figure 5.7) and a training folder inside the yolov4 folder (Figure 5.8) to store training weight. Download the yolov4-custom config file (*yolov4-custom.cfg*) from *darknet/cfg* directory.

```
!git clone https://github.com/AlexeyAB/darknet
```

```

Cloning into 'darknet'...
remote: Enumerating objects: 14691, done.
remote: Total 14691 (delta 0), reused 0 (delta 0), pack-reused 14691
Receiving objects: 100% (14691/14691), 13.24 MiB | 5.24 MiB/s, done.
Resolving deltas: 100% (9994/9994), done.
Checking out files: 100% (2023/2023), done.

```

Figure 5.9: Output of cloning the darknet git repository

Step 2, download the darknet to the yolov4 folder by cloning the darknet git repository (as in Figure 5.9).

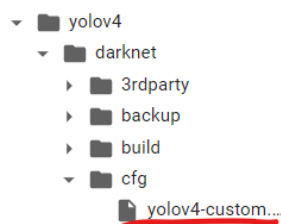


Figure 5.10: The yolov4-custom file

In step 3, make changes in the yolov4-custom file which can be found in the *cfg* folder under the yolov4 folder (in Figure 5.10).

Line batch size is the number of segments per iteration. If the line batch size is set as 64, then there will be 64 images in every line batch. Next, the subdivision number determines the number of “mini batches” in one batch. If the number of the subdivision is 16, a batch has broken into 16 pieces. The GPU process four images at a time ($64/16=4$), and it is repeated 16 times to complete a batch of 64 images. Once the batch is complete, a new batch of 64 images begin processing. The max batches are the number of iterations for the training, and the standard is the number of classes*2000 or not less than 6000 iterations.

Furthermore, the line steps are set according to 80% and 90% of the max batches. Moreover, the network size is the network resolution. The width and height must be multiples of 32 and increasing width and height can increase precision; however, increasing it does decrease the training speed. In addition, the line classes change according to the number of classes that need to be trained. Lastly, the filter size is equal to $(\text{number of classes}+5) * 3$ [42]. Moreover, the learning rate used to train the model is 0.001, the default learning rate set by the YOLOv4 model. After 1000 iterations, the learning rate will be manually updated to 0.0001 and used until 9000 iterations.

Changes made in the yolov4-custom configuration file are as below:

- change line batch to batch=64
- change line subdivisions to subdivisions=16
- change line max batches to 9000 (iterations)
- change line steps to 80% and 90% of max batches, which is steps= 7200, 8100
- set network size width=416 height=416
- change line classes to classes=2 in each of 3 [yolo]-layers
- change [filters=255] to filters= 21 in the 3 [convolutional]

The default value for the parameter "*ignore_thresh*" in YOLOv4 is 0.7, which means that duplicated detections are kept only if their IoU with the ground truth bounding box is greater than 0.7 [56]. Increasing this value can improve the accuracy of the detector. However, the parking lot image dataset used in this project contains around 100 objects with small bounding boxes per image, and a lot of noise due to obstructions, road conditions, and weather conditions. This is different from typical datasets that contain only a single or a few objects per image. Therefore, setting the "*ignore_thresh*" value to 0.9 as suggested in [43] may not be suitable for this dataset, as it is likely to cause overfitting issues.

Next, the parameter "*iou_normalizer*" serves as the normalizer for delta-IoU and it normalizes the delta-IoU for every object's bounding box [56]. The default value for this parameter in YOLOv4 is 0.07, while the value suggested in [43] is 0.5. However, using an "*iou_normalizer*" with a value of 0.5 is not suitable for the dataset used, as it may affect the calculation of the normalizer.

After several test on different value for the parameters network size width and height, "*ignore_thresh*" and "*iou_normalizer*" it was observed that adjusting the network size from 416 to 512 did not improve the object detector's accuracy. This lack of improvement could be attributed to the image dataset's resolution, which is 640x640. Another test was conducted using an "*ignore_thresh*" value of 0.9 and an "*iou_normalizer*" value of 0.5, which resulted in poor performance even after 6000 iterations (as shown in the appendix). Both the mAp50% and average IoU results were not more than 27%. Other tests were also conducted, such as adjusting the network size width and height with only "*ignore_thresh*" or only "*iou_normalizer*" and they showed

optimized results. However, after evaluating all the tested models, the default parameter values yielded the most promising result.

Step 4a, create a data file and a names file and named both of the file as *obj* (*obj.data* and *obj.names*). The *obj.data* file includes the number of classes, the directory for the training file, and the test file, which includes images and labelled file names. Next, the names file includes the names for the two classes, which are space-empty and space-occupied. Then, the directory for the names file saves classes' names, and the directory for the backup saves the training weight. Both files can be found in appendix.

Step 4b, create a python file name process (*process.py*). This *process.py* script generates two files, *train.txt* and *test.txt*, each containing the paths to 90% of the images and 10% of the images, respectively [43]. This python file (*process.py*) can be found in appendix.

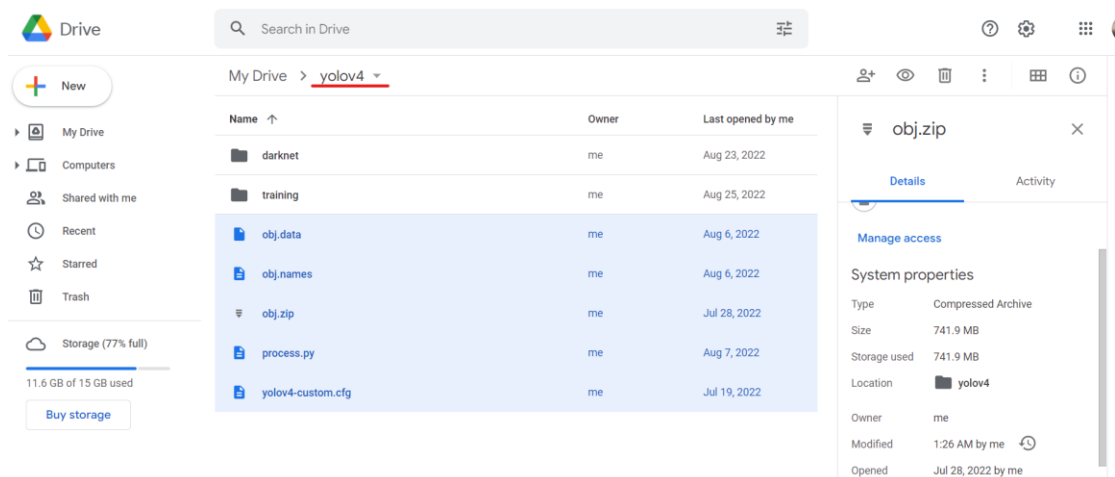


Figure 5.11: Upload needed file to yolov4 folder

Step 4c, upload the files (*obj.zip*, *yolov4-custom.cfg*, *obj.data*, *obj.names* and *process.py*) to Google Drive inside the *yolov4* folder (in Figure 5.11).

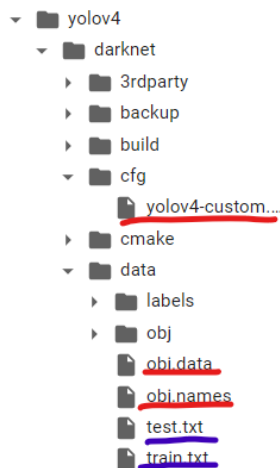


Figure 5.12: Copy the files and paste under data and *cfg* folder

During data cleaning of the *obj.zip* dataset (which contained 12,412 labelled *txt* files and images), it was discovered that some *txt* files were empty. After removing these files, the dataset was left with 10,926 labelled *txt* files and images.

Step 5, remove all files from the data and *cfg* folders except for the labels folder in the needed data. Then, move the files (*obj.names* and *obj.data*) from the yolov4 folder into the darknet directory. Afterward, unzip the *obj.zip* file to the data folder located inside the darknet folder (as in Figure 5.12).

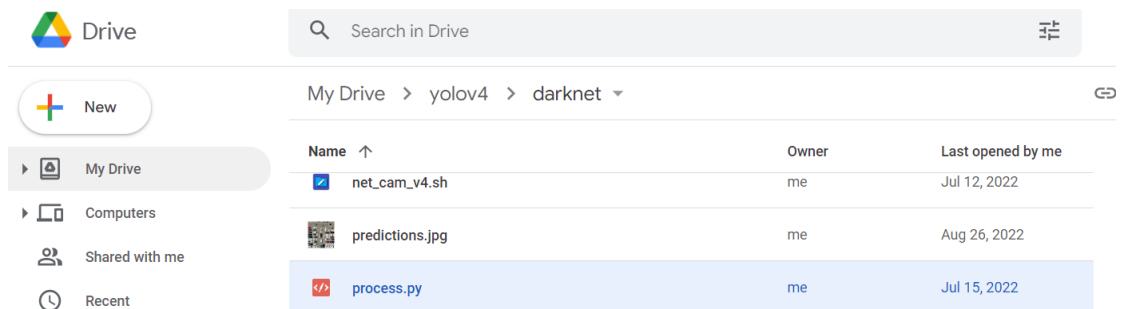


Figure 5.13: Copy and paste process file

Step 6, paste process file (*process.py*) to the darknet directory (in Figure 5.13) and run the file to produce the *train.txt* and *test.txt* files in our *darknet/data* folder (in Figure 5.12).

```

--2022-07-15 10:59:30-- https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
Resolving github.com (github.com)... 20.205.243.166
Connecting to github.com (github.com)|20.205.243.166|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/48bfe500-889d-11ea-819e-c4d182fcf0db?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAVC44AYSLPHBYQGN3T&X-Amz-Date=20220715T105930Z&X-Amz-Security-Token=eyJkZW5pdHA0eXB0ej0iOmE6ODQxeXBlc1Q7bnVudC51a2IomQ==&X-Amz-SignedHeaders=host&X-Amz-Target=aws-portal-2016-08-15/aws-portal-2016-08-15-programmatic-access-selector&X-Amz-Version=AWS3-HMAC-SHA256
--2022-07-15 10:59:30-- https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/48bfe500-889d-11ea-819e-c4d182fcf0db?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAVC44AYSLPHBYQGN3T&X-Amz-Date=20220715T105930Z&X-Amz-Security-Token=eyJkZW5pdHA0eXB0ej0iOmE6ODQxeXBlc1Q7bnVudC51a2IomQ==&X-Amz-SignedHeaders=host&X-Amz-Target=aws-portal-2016-08-15/aws-portal-2016-08-15-programmatic-access-selector&X-Amz-Version=AWS3-HMAC-SHA256
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 170038676 (162M) [application/octet-stream]
Saving to: 'yolov4.conv.137'

yolov4.conv.137  100%[=====] 162.16M  8.99MB/s   in 18s

2022-07-15 10:59:49 (8.93 MB/s) - 'yolov4.conv.137' saved [170038676/170038676]

```

Figure 5.14: Download the weights file for the YOLOv4 pre-trained model

Step 7, download the weights file for the YOLOv4 pre-trained model. Figure 5.14 show the process and result of downloading the weights file. Transfer learning is employed, whereby instead of training a model from scratch, YOLOv4 weights that have already been trained up to 137 convolutional layers are utilized. By leveraging transfer learning, the learning process can be accelerated and the accuracy can be improved.

5.3.2 Train on YOLOv4 model

```

/
Mounted at /content/gdrive
/content/gdrive/My Drive/yolov4

```

Figure 5.15: Connect to Google Drive

Before training the custom model, Google Colab must connect to Google Drive (Figure 5.15) to access the download darknet, custom file and the dataset used for the training.

```

%cd darknet/
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile

```

Figure 5.16: Make changes in Makefile

Then, modify the *Makefile* to enable *OPENCV* and *GPU* by turning *CUDNN*, *CUDNN_HALF* and *LIBSO* value to 1 (Figure 5.16). *OPENCV* is an open-source computer vision library designed to solve computer vision problems [44]. *CUDNN* is a

GPU-accelerated library of primitives for deep neural networks [45]. After making changes in *Makefile*, the darknet is ready to be built.

```

chmod +x *.sh
g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -DOPENCV `pkg-config --cflags opencv4 2> /dev/null` | pkg-config --cflags opencv -DGPU -I/
./src/image_opencv.cpp: In function 'void draw_detections_cv_v3(void**, detection*, int, float, char**, image**, int, int)':
./src/image_opencv.cpp:946:23: warning: variable 'rgb' set but not used [-Wunused-but-set-variable]
    float rgb[3];
    ~~~~~
./src/image_opencv.cpp: In function 'void draw_train_loss(char*, void**, int, float, float, int, int, float, int, char*, float, int, int, double)':
./src/image_opencv.cpp:1147:13: warning: this 'if' clause does not guard... [-Wmisleading-indentation]
    if (iteration_old == 0)
    ~
./src/image_opencv.cpp:1150:10: note: ...this statement, but the latter is misleadingly indented as if it were guarded by the 'if'
    if (iteration_old != 0){
    ~
./src/image_opencv.cpp: In function 'void cv_draw_object(image, float*, int, int, int*, float*, int*, int, char**)':
./src/image_opencv.cpp:1444:14: warning: unused variable 'buff' [-Wunused-variable]
    char buff[100];
    ~~~~~
./src/image_opencv.cpp:1420:9: warning: unused variable 'it_tb_res' [-Wunused-variable]
    int it_tb_res = cv::createTrackbar(it_trackbar_name, window_name, &it_trackbar_value, 1000);
    ~~~~~
./src/image_opencv.cpp:1424:9: warning: unused variable 'lr tb res' [-Wunused-variable]

```

Figure 5.17: Build darknet

```

Streaming output truncated to the last 5000 lines.
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.785164), count: 542, class_loss = 37.486061, iou_loss = 1454.8
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.777299), count: 314, class_loss = 19.182571, iou_loss = 261.80
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.706343), count: 19, class_loss = 1.392975, iou_loss = 2.075638
total_bbox = 15267566, rewritten_bbox = 0.452849 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.765984), count: 746, class_loss = 30.789055, iou_loss = 2476.9
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.787394), count: 141, class_loss = 6.781021, iou_loss = 145.28
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.667940), count: 2, class_loss = 0.289565, iou_loss = 0.359493
total_bbox = 15268455, rewritten_bbox = 0.452836 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.734681), count: 212, class_loss = 19.663879, iou_loss = 439.3
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.765305), count: 209, class_loss = 12.981966, iou_loss = 133.3
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.753150), count: 15, class_loss = 1.421540, iou_loss = 2.56104
total_bbox = 15268891, rewritten_bbox = 0.452823 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.782139), count: 502, class_loss = 43.971054, iou_loss = 1345.8
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.803378), count: 289, class_loss = 17.558838, iou_loss = 238.0
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.738978), count: 10, class_loss = 1.109795, iou_loss = 1.72407
total_bbox = 15269692, rewritten_bbox = 0.452812 %

```

Figure 5.18: Train the custom model using the pre-train yolov4 weights output

Code that trains the custom model using the pre-train yolov4 weights:

```

!./darknet detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont_show -
map

```

A command “*!make*” is executed to build the darknet (Figure 5.17), the training for the custom model starts by using the pre-train yolov4 weights using the code (trains the custom model using the pre-train yolov4 weights). The output after executed the code is shown in Figure 5.18. However, the output slot only can display up to 5000 lines, so the output since the start of the training cannot be previewed and screenshot here. Total iterations for this training are 6000.

Code that continues training yolov4 based on its last weights:

```

!./darknet detector train data/obj.data cfg/yolov4-custom.cfg /mydrive/yolov4/training/yolov4-
custom_last.weights -dont_show -map

```

```
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.901142), count: 458, class_loss = 14.549611, iou_loss = 2006.64)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.919537), count: 420, class_loss = 8.495869, iou_loss = 657.264)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.926224), count: 27, class_loss = 0.250400, iou_loss = 7.419254)
total_bbox = 18130233, rewritten_bbox = 0.414904 %

(next mAP calculation at 6264 iterations)

Tensor Cores are used.
Last accuracy mAP@0.50 = 99.98 %, best = 99.98 %
5931: 1.114398, 3.409323 avg loss, 0.000010 rate, 13.430198 seconds, 379584 images, 0.442981 hours left
Loaded: 0.000047 seconds
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.930141), count: 176, class_loss = 0.856180, iou_loss = 576.36)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.947173), count: 273, class_loss = 0.006016, iou_loss = 313.89)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.940689), count: 32, class_loss = 0.000059, iou_loss = 11.3117)
total_bbox = 18130714, rewritten_bbox = 0.414893 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.946800), count: 409, class_loss = 0.044203, iou_loss = 2035.7)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.949687), count: 300, class_loss = 0.001073, iou_loss = 511.02)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.948613), count: 18, class_loss = 0.011968, iou_loss = 8.13941)
total_bbox = 18131441, rewritten_bbox = 0.414876 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.912860), count: 246, class_loss = 2.920440, iou_loss = 846.36)
```

Figure 5.19: Continue the training on last check point, last weights

Training stopped after exceeding the usage limit of Google Colab. The previous code (*trains the custom model using the pre-train yolov4 weights*) needs to be modified to continue training the custom YOLOv4 model. Else, the training will be restarted from the first iterations. Thus, the last weight is used to continue the training from the last checkpoint using the code (*continues training yolov4 based on its last weights*) and with the continuous output in Figure 5.19.

```
(next mAP calculation at 6264 iterations)

Tensor Cores are used.
Last accuracy mAP@0.50 = 99.98 %, best = 99.98 %
6000: 2.817498, 2.947382 avg loss, 0.000010 rate, 14.802139 seconds, 384000 images, 0.283664 hours left
Resizing to initial size: 416 x 416 try to allocate additional workspace_size = 111.05 MB
CUDA allocate done!

calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
872
detections_count = 53019, unique_truth_count = 49732
class_id = 0, name = space-empty, ap = 99.98% (TP = 26622, FP = 123)
class_id = 1, name = space-occupied, ap = 99.98% (TP = 23088, FP = 140)

for conf_thresh = 0.25, precision = 0.99, recall = 1.00, F1-score = 1.00
for conf_thresh = 0.25, TP = 49710, FP = 263, FN = 22, average IoU = 94.21 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.999804, or 99.98 %
Total Detection Time: 33 Seconds

Set -points flag:
`-points 101` for MS COCO
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

mean_average_precision (mAP@0.50) = 0.999804
New best mAP!
Saving weights to /mydrive/yolov4/training/yolov4-custom_best.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_6000.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_last.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_final.weights
If you want to train from the beginning, then use flag in the end of training command: -clear
```

Figure 5.20: Training output after 6000 iterations

After a total of 6000 iterations of the training, the output is shown in Figure 5.20.

A custom YOLOv4 model was trained with the initial dataset of 12412 labelled files, including empty files. After performing data cleaning to remove the empty files, the dataset was reduced to 10926 labelled files. To improve the performance of the model, an additional 3000 iterations were performed using only the cleaned dataset of 10926 labelled files.

```

v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.944419)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Reg: ↑ ↓ ↻ ⌨ ⚙ 📄 🗑 ⋮
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.920443)
total_bbox = 11279364, rewritten_bbox = 0.404713 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.952081)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.932127)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.939809)
total_bbox = 11280353, rewritten_bbox = 0.404739 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.924554)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.932955)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.892959)
total_bbox = 11281589, rewritten_bbox = 0.404695 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.910029)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.916967)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.918151)
total_bbox = 11283371, rewritten_bbox = 0.404675 %
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.926821)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.925928)
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.912082)
total_bbox = 11285155, rewritten_bbox = 0.404638 %

Tensor Cores are used.

9000: 2.402813, 3.582072 avg loss, 0.000001 rate, 12.967691 seconds, 576000 images, 0.257
Saving weights to /mydrive/yolov4/training/yolov4-custom_9000.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_last.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_final.weights
If you want to train from the beginning, then use flag in the end of training command: -cl

```

Figure 5.21: Training output after 9000 iterations

Code that continues training yolov4 based on its last weights (modified):

```

!./darknet detector train data/obj.data cfg/yolov4-
custom.cfg /mydrive/yolov4/training/yolov4-custom_last.weights -dont_show

```

A modification was needed for the code (*continues training yolov4 based on its last weights*), as it would otherwise output an error. This was due to an incompatibility issue with the OpenCV version, as the training was being continued after a 6-month break [57]. The modification involved removing the “-map” command, and the resulting code (shown above) now outputs the iterations and saving process only, without performing or outputting any mAP calculation during training. After a total of 9000 iterations of training, the output is shown in Figure 5.21.

5.4 Prediction model

5.4.1 Dataset preprocessing (text label file to CSV file)

```
Empty DataFrame
Columns: [Location, Occupied, Total parking, Date, Time, Weather]
Index: []
```

Figure 5.22: Store parking features in data frame

The label text file generated from the YOLOv4 model is used to develop the prediction model. Unfortunately, the YOLOv4 model's accuracy is still not good enough, so the PKLot label text file is used to train this prediction model. In order to extract information from these text files, pre-processing is required.

First, execute the code (*creating a new data frame to store the data (parking features) extracted from all the files*) in appendix. The format of the data frame is shown as Figure 5.22.

Second, the code (*read the text files*) in appendix processes the text files (cloudy, rainy and sunny) that store the date and time and store the data (*label file name*) accordingly into the list (cloudy, rainy and sunny).

Third, the code (*reading label XML file*) in appendix indicates the process of reading label XML file (*second type of label file for the PKLot*) and store it into dictionary which to use later.

Some of the label text files are empty. Other than excluding these data in the dataset, the empty labelled text files are recorded and replaced by labelled XML files (*another type of label file for PKLot dataset*) that are downloaded from different sources.

All the data, such as DateTime, occupancy, and total parking, that is read in from the XML files is stored in a dictionary and later utilized. The dictionary (*my_dict*) key is the file name and include two values, numbers of occupied parking and the number of total parking spaces in the parking area.

Fourth, the code (*appends data into data frame*) in appendix demonstrates how to obtain data (*occupied parking, total parking, date, time, and weather*) from all label

files, apply an if else segment (*to fill in the different types of weather and the missing value*), and then save the data in accordance with the created data frame.

In this process, the date and time extracted from the label text file are compared with the weather lists to add one more feature, weather, and later stored in the data frame. The weather condition included cloudy (equal to 0), rainy (equal to 1) and sunny (equal to 2). Furthermore, if any text file matches the key in an empty label file dictionary, the dictionary value should be utilised (replace the total parking and occupancy value in the text file) and stored in the data frame.

	Location	Occupied	Total parking	Date	Time	Weather
0	PKlots	67.0	100.0	11/09/2012	15:16:58	2.0
1	PKlots	67.0	100.0	11/09/2012	15:27:08	2.0
2	PKlots	69.0	100.0	11/09/2012	15:29:29	2.0
3	PKlots	69.0	100.0	11/09/2012	15:31:50	2.0
4	PKlots	69.0	100.0	11/09/2012	15:36:32	2.0
...
12411	PKlots	38.0	40.0	16/04/2013	10:30:05	2.0
12412	PKlots	39.0	40.0	16/04/2013	10:35:05	2.0
12413	PKlots	39.0	40.0	16/04/2013	10:40:05	2.0
12414	PKlots	38.0	40.0	16/04/2013	10:45:05	2.0
12415	PKlots	36.0	40.0	16/04/2013	10:50:05	2.0

[12416 rows x 6 columns]

Figure 5.23: Data frame of PKLot with weather condition

Figure 5.23 show that all the data is successfully stored in the “df” data frame.

Fifth, the code (*saves data frame as csv*) in appendix shows saving the data frame into a CSV file and named “pklot_weatherNcomplete”. This dataset was used to develop the parking vacancy prediction model.

5.4.2 Data preprocessing

Out[24]:

	Location	Occupied	Total parking	Date	Time	Weather
0	PKlots	67.0	100.0	11/09/2012	15:16:58	2.0
1	PKlots	67.0	100.0	11/09/2012	15:27:08	2.0
2	PKlots	69.0	100.0	11/09/2012	15:29:29	2.0
3	PKlots	69.0	100.0	11/09/2012	15:31:50	2.0
4	PKlots	69.0	100.0	11/09/2012	15:36:32	2.0
...
12411	PKlots	38.0	40.0	16/04/2013	10:30:05	2.0
12412	PKlots	39.0	40.0	16/04/2013	10:35:05	2.0
12413	PKlots	39.0	40.0	16/04/2013	10:40:05	2.0
12414	PKlots	38.0	40.0	16/04/2013	10:45:05	2.0
12415	PKlots	36.0	40.0	16/04/2013	10:50:05	2.0

12416 rows × 6 columns

Figure 5.24: Read PKLot CSV file into data frame

The code (*import the needed libraries*) in appendix is applied to import the needed python libraries and the output after executed the code (*load the dataset into a data frame*) in appendix.

The code (*modify and create features*) in appendix shows create new features (DateTime and empty), drop unnecessary features (location, occupied, date and time) and convert “*DateTime*” to *DateTime* data type and the numerical data (empty, total parking and weather) to integer data type.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12416 entries, 0 to 12415
Data columns (total 4 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   Total parking   12416 non-null  int64
1   Weather         12416 non-null  int64
2   DateTime        12416 non-null  datetime64[ns]
3   Empty           12416 non-null  int64
dtypes: datetime64[ns](1), int64(3)
memory usage: 388.1 KB
```

Figure 5.25: Quick description of all attributes in the pk data frame

The output for the code (*displays description of all attributes*) in appendix is show in Figure 5.25. In Figure 5.25 a total of 12416 columns for four features and indicates no single cell in the column is a null value.

```
Out[19]: False    12416
         Name: DateTime, dtype: int64
```

Figure 5.26: Check redundancy in DateTime

The output for the code (*checks DateTime redundancy*) in appendix is display in Figure 5.26 and in this figure displays the redundancy happened in *DateTime* is False for 12416 over 12416 data. As a result, no redundancy occurred.

```
Out[12]:
```

	Total parking	Weather	Empty
count	12416.000000	12416.000000	12416.000000
mean	57.951675	1.221488	30.907780
std	31.917497	0.918109	31.804152
min	28.000000	0.000000	0.000000
25%	28.000000	0.000000	4.000000
50%	40.000000	2.000000	27.000000
75%	100.000000	2.000000	40.000000
max	100.000000	2.000000	100.000000

Figure 5.27: Statistics of all numerical attributes in pk data frame

The output for the code (*check statistics of all numerical attribute*) in appendix is display in Figure 5.27 and this figure display the summary statistics of all numerical attributes in the data frame.

The summary statistic includes *count* (number of values that are not empty for every numerical attribute), *mean* (the average values for every numerical attribute), and *std* (standard deviation for every numerical attribute). Next it shows *min* (minimum value for every numerical attribute), *25%* (25% percentile* for every numerical attribute), *50%* (50% percentile* for every numerical attribute), *75%* (75% percentile* for every numerical attribute) and *max* (maximum value for every numerical attribute).

According to the statistic of total parking, there are 12416 data in this attribute. Total parking has a mean value of 57.95 and a standard deviation of 31.92. Then with

CHAPTER 5

a minimum value of 28 and a maximum value of 100. Furthermore, 25% percentile with the value of 28, 50% percentile with the value of 40, 75% percentile with the value of 100.

According to the statistic of weather, there are 12416 data in this attribute. Weather has a mean value of 1.22 and a standard deviation of 0.91. However mean and standard deviation is not important in this case because Weather is a nominal data convert to numeric which represent 0 as cloudy, 1 as rainy and 2 as sunny. Then “Weather” has a minimum value of 0 and a maximum value of 2. Furthermore, 25% percentile with the value of 0, 50% percentile with the value of 2, 75% percentile with the value of 2. As a result, most weather are in the condition of sunny as more than 50% of the data are 2.

According to the statistic of empty, there are 12416 data in this attribute. Empty has a mean value of 30.91 which mean average 30.91 parking is empty in the dataset and a standard deviation of 31.8. Then with a minimum value of 0 and a maximum value of 100. Furthermore, 25% percentile with the value of 4, 50% percentile with the value of 27, 75% percentile with the value of 40.

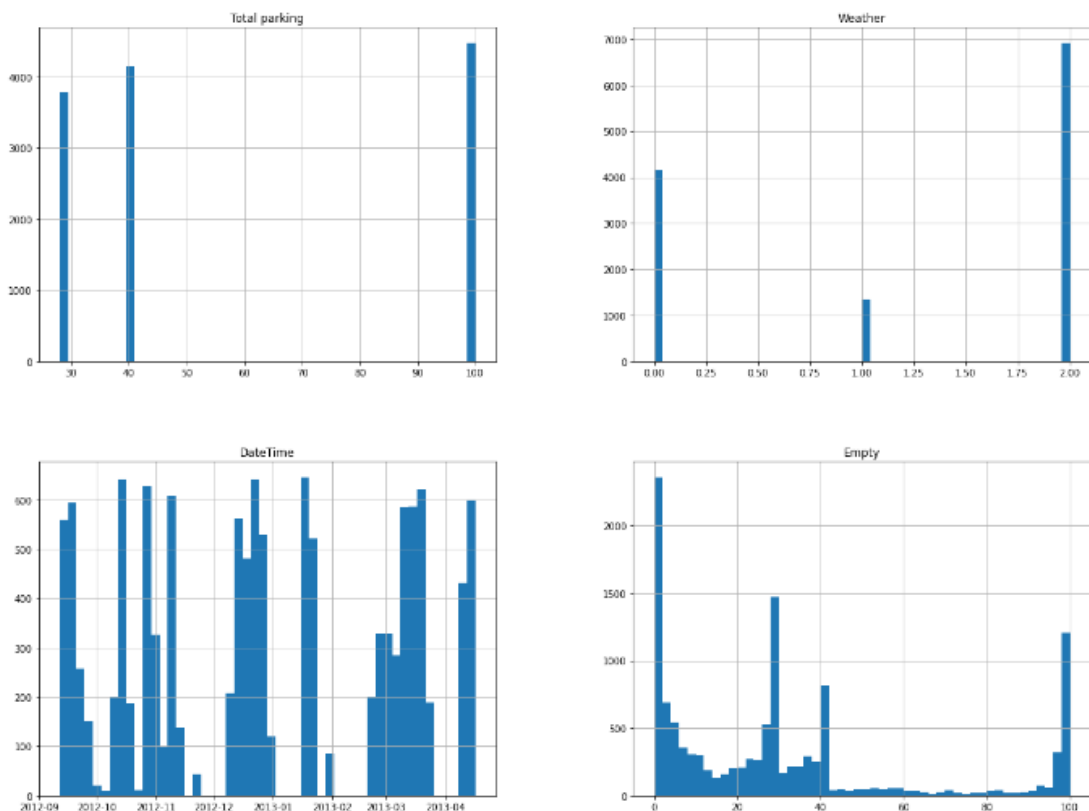


Figure 5.28: Histogram of each numerical attribute in pk data frame

The output for the code (*plots the histogram of each numerical attribute*) in appendix is display in Figure 5.28 and this figure display the histogram for “*Total parking*”, “*Weather*”, “*DateTime*” and “*Empty*”.

The “*Total parking*” histogram shows that there are only three types of parking lots, which include the capacity of 28, 40 and 100.

Then, the “*Weather*” histogram shows that most of the data are 2 (2 is sunny), which the quantity is close to seven thousand, the next highest is 0 (0 is cloudy), which the quantity is around four thousand, and 1 (1 is rainy) happened the less among them which is around one thousand and five hundred.

Furthermore, the “*DateTime*” histogram shows the distribution of the dataset in date, and one bar contains around five days. This “*DateTime*” histogram also reveals that some of the days in the histogram are empty, meaning the data set is not in continuously.

Lastly, there are three peaks in the “*Empty*” histogram. After analysing this histogram, the result is that three parking lot are frequently in zero capacity and a full capacity condition. The total capacity of the first parking lot is 28, the next is 40, and the last is 100.

Out[40]:

	Total parking	Weather	Empty
Total parking	1.000000	0.004159	0.610662
Weather	0.004159	1.000000	-0.057591
Empty	0.610662	-0.057591	1.000000

Figure 5.29: correlation matrix of pk data frame

The output for the code (*display correlation matrix*) is display in Figure 5.29 and this figure shows the correlation matrix of the “*pk*” data frame. According to this figure, empty and total parking strongly correlate with a value of 0.61, a strong correlation relationship.

Next, the relationship between empty and weather has a correlation value of negative 0.057, and this value is not expected as the statement in the literature review. The reason for this might be because of the characteristic of the dataset. This dataset is collected from a university, and the class conducted in the university is not cancel by weather like rainy unless it is extreme weather like a hurricane or blizzard.

The weather condition in this dataset (cloudy, rainy and sunny) do not affect the scheduling of the classes. Hence, no matter in rainy, sunny or cloudy, the class is still being conducted, and the student still needs to attend it. In conclusion, the weather did not affect much on the number of empty parking because this dataset is collected at a university.

Then, the relationship between total parking and weather has a correlation value of 0.004, which is to be expected given that weather conditions are unlikely to significantly impact the overall parking capacity of a lot. In conclusion, the correlations show that weather is less critical in this dataset.

```
Out[18]: 2    6912
         0    4162
         1    1342
         Name: Weather, dtype: int64
```

Figure 5.30: Count unique values of weather

Figure 5.30 shows that the "Weather" column mostly contains the value 2, indicating a sunny day with 6,912 occurrences, which is over half the total dataset of 12,416. The next highest value is 0, representing a cloudy day with only 4,162 occurrences. The least common weather condition in the dataset is 1, representing a rainy day with only 1,324 occurrences.

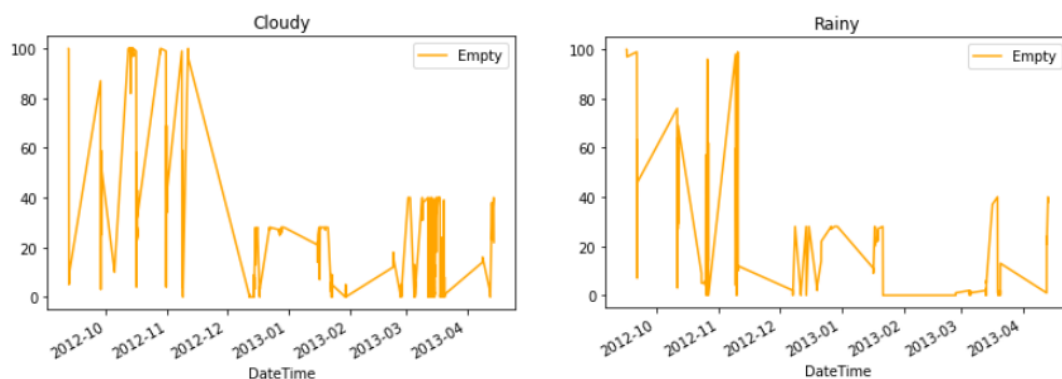


Figure 5.31: Graph plot with Empty and DateTime in Cloudy

Figure 5.32: Graph plot with Empty and DateTime in Rainy

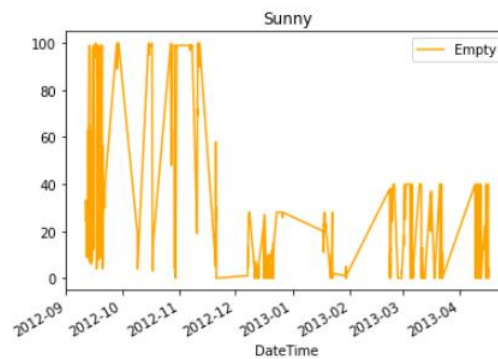


Figure 5.33: Graph plot with Empty and DateTime in Sunny

The code (*plots graph according to weather types*) in appendix plot total three figures (Figure 5.31, Figure 5.32 and Figure 5.33). The figures above show three types of weather (cloudy, rainy and sunny) in the PKLot dataset with “*Empty*” as the y-axis and “*DateTime*” as the x-axis.

In Figure 5.31, there is only 4162 datasets in cloudy, and the slash line is the empty data that are missing in the *DateTime* because the weather other than cloudy is filtered out.

Next, in Figure 5.32, there is only 1342 datasets in rainy, and the slash line is the empty data that are missing in the Date Time because the weather other than rainy is filtered out.

Lastly, in Figure 5.33, there is only 6912 datasets in sunny, and the slash line is the empty data that are missing in the *DateTime* because the weather other than sunny is filtered out.

5.4.3 Train on prediction model

The code (*import the needed libraries*) in the appendix imports all the necessary libraries that will be used to develop and evaluate the ML models. These include the SVR, LR, and decision tree regression libraries, as well as loss functions such as MSE, MAE, and MAPE. Other relevant libraries, such as NumPy, Pandas, and Scikit-learn, and more are also imported as shown in appendix.

First, run the data pre-processing codes in appendix (done in the 4.3.2 Data processing section):

1. Read the CSV file into a data frame
2. Create a new column name “*DateTime*” by combining the “*Date*” and “*Time*”
3. Convert “*DateTime*” to *DateTime* data type
4. Create a new column of the “*DateTime_year*”, “*DateTime_month*”, “*DateTime_week*”, “*DateTime_day*”, “*DateTime_hour*”, “*DateTime_minute*” and “*DateTime_dayofweek*” using the “*DateTime*” column
5. Create a new column name “*Empty*” by minus “*Occupied*” with “*Total parking*”
6. Drop unneeded columns (“*Location*”, “*Date*”, “*Time*”, “*DateTime*” and “*Occupied*”)
7. Convert all numerical columns (“*Total parking*”, “*Empty*”, “*Weather*” and “*DateTime_week*”) to integer data type
8. Select “*Total parking*” with the value of 100 as the training and testing dataset.

At step number four, the timestamp cannot fit the ML model, so a discrete component is applied in *DateTime*.

```
Shape of original dataset, train: (12416, 10)
x: shape= (3131, 11) type= <class 'numpy.ndarray'>
y: shape= (3131,) type= <class 'numpy.ndarray'>
```

Figure 5.34: The shape of the original training set and the processed training set x and y

Second, the code (*split dataset and pre-processing*) in appendix is the last data pre-processing steps before the ML training start.

This start by split the dataset into *X* as input matrix, *y* as output vector, and *X* and *y* into training, validation and testing sets. The data is split into a training set with a size of 70%, validation set with a size of 15% and a testing set with a size of 15%. Next, the weather column is extracted and applied with one-hot encoding instead of the ordinary label. Then, standardize the numeric data is applied due to the data contains varying scales. After merging the numerical and weather datasets into a single *NumPy* array for fitting the ML model, the data type of *y_train*, *y_val* and *y_test* sets should be converted to *NumPy* arrays as well.

Figure 5.34 show the original dataset has 12416 rows of data and 10 columns. After pre-processing this dataset into x_{train} and y_{train} , the x_{train} has 3131 rows of data (only including total parking equal to 100) with 11 columns (drop one feature to create y and apply one-hot encoding to the weather column). For the y_{train} it has 3131 rows of data.

After running the code above the ML training is ready to go. The performance of the SVR algorithm is evaluated and make comparison by applying two other ML algorithms like LR and Decision Tree Regression. The function “*show10results*” in appendix is created to display the prediction value for the 10 random samples selected from the training set.

```

regr = SVR()
regr.fit(X_train_tr, y_train)
y_pred = regr.predict(X_train_tr)
Result for the 10 random samples:
actual =    55 pred =    45
actual =   100 pred =    82
actual =    98 pred =    91
actual =    82 pred =    87
actual =   100 pred =    81
actual =   100 pred =    86
actual =     0 pred =    60
actual =    98 pred =    83
actual =   100 pred =    88
actual =    48 pred =    37
<--- regr train --->
MSE = 657.0135377309122
RMSE= 25.63227531318498
MAE = 19.35284297177313
MAPE= 2.3085e+15

```

Figure 5.35: Performance output of the SVR model

Next, create a regressor object using the SVR library and train this model using the training set. After the model fit the data, the evaluation of the model's performance on the training set is started by predicting the y_{train} set (the empty parking space). The “*show10results*” function is called to display the prediction value for the 10 random samples selected from the training set.

Furthermore, to evaluate how well the created model fits the data, the function code of MSE, RMSE, MAE and MAPE in appendix are used. The performance output is shown in Figure 5.35. MSE result is 657.01. RMSE result is 25.63. MAE result is 19.35. Lastly, MAPE result is 2.3085e+15.


```

k_fold_scores = cross_val_score(regr, X_train_tr, y_train,
                                scoring = 'neg_mean_squared_error', cv=5)
regr_rmse_scores = np.sqrt(-k_fold_scores)
display_scores(regr_rmse_scores)

Scores: [26.03993561 28.00853098 26.77838432 24.95344072 27.24402081]

Mean: 26.60486248941612
Standard deviation: 1.0445338328109461

```

Figure 5.36: Result of 5-fold cross validation with SVR using the training set

The code above shows applying the 5-fold cross-validation score function on the SVR model. The RMSE score is likely be overly optimistic when evaluated directly on the training set. Hence cross-validation is used to obtain a more accurate RMSE value. K-fold cross-validation is performed on the model by calling the *cross_val_score* function from *sklearn*. In this case, a 5-fold cross-validation (*cv=5*) is performed on the SVR model with an output score of negative mean squared error. This output is subsequently converted to RMSE and display the mean of the RMSE, 26.6 (in Figure 5.36). The standard deviation of the RMSE, which is 1.0445 (in Figure 5.36).

```

lin_reg = LinearRegression()
lin_reg.fit(X_train_tr, y_train)
y_pred = lin_reg.predict(X_train_tr)
Result for the 10 random samples:
actual =    100 pred =    94
actual =     10 pred =    47
actual =     37 pred =    76
actual =     55 pred =    47
actual =     99 pred =    40
actual =      7 pred =    60
actual =     97 pred =    82
actual =     16 pred =    48
actual =     82 pred =    67
actual =     57 pred =    46
<--- lin_reg train --->
MSE = 795.0574975369098
RMSE= 28.196763955051825
MAE = 22.464712706643176
MAPE= 1.6063e+15

```

Figure 5.37: Performance output of the LR model

The code above shows training a model with LR. Before building a model using LR, the first step is to import the LR library from *sklearn*.

Next, create a regressor object using the LR library and train this model using the training set. After the model fit the data, the evaluation of the model's performance on the training set is started by predicting the *y_train* set (the empty parking space), the

target variable. The “*show10results*” function is called to display the prediction value for the 10 random samples selected from the training set.

Furthermore, to evaluate how well the created model fits the data, the function code of MSE, RMSE, MAE and MAPE in appendix are used. The performance output is shown in Figure 5.37. MSE result is 795.06. RMSE result is 28.20. MAE result is 22.46. Lastly, MAPE result is 1.6063e+15.

```
k_fold_scores = cross_val_score(lin_reg, X_train_tr, y_train,
                                scoring = 'neg_mean_squared_error', cv=5)
lin_reg_rmse_scores = np.sqrt(-k_fold_scores)
display_scores(lin_reg_rmse_scores)
Scores: [27.77571994 28.23343075 28.88632639 27.39118448 29.01746439]

Mean: 28.260825191744097
Standard deviation: 0.6254769634305356
```

Figure 5.38: Result of 5-fold cross validation with LR using the training set

The code above shows applying the 5-fold cross-validation score function on the LR model. The RMSE score is likely be overly optimistic when evaluated directly on the training set. Hence cross-validation is used to obtain a more accurate RMSE value. K-fold cross-validation is performed on the model by calling the *cross_val_score* function from *sklearn*. In this case, a 5-fold cross-validation (cv=5) is performed on the LR model with an output score of negative mean squared error. This output is subsequently converted to RMSE and display the mean of the RMSE, 28.26 (in Figure 5.38). The standard deviation of the RMSE, which is 0.6255 (in Figure 5.38).

```
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(X_train_tr, y_train)
y_pred = tree_reg.predict(X_train_tr)
```

```

Result for the 10 random samples:
actual =      19 pred =      19
actual =      97 pred =      97
actual =      32 pred =      32
actual =      13 pred =      13
actual =      88 pred =      88
actual =      40 pred =      40
actual =      49 pred =      49
actual =      52 pred =      52
actual =      54 pred =      54
actual =     100 pred =     100
<--- tree_reg train --->
MSE = 0.006177345845183597
RMSE= 0.07859609306564543
MAE = 0.00629141255139339
MAPE= 1.8456e-04

```

Figure 5.39: Performance output of the decision tree regression model

The code above shows training a model with Decision Tree Regressor. Before building a model using decision tree regressor, the first step is to import the decision tree regressor library from *sklearn*.

Next, create a regressor object using the Decision Tree Regressor library and train this model using the training set. After the model fit the data, the evaluation of the model's performance on the training set is started by predicting the *y_train* set (the empty parking space), the target variable. The “*show10results*” function is called to display the prediction value for the 10 random samples selected from the training set.

Furthermore, to evaluate how well the created model fits the data, the function code of MSE, RMSE, MAE and MAPE in appendix are used. The performance output is shown in Figure 5.39. MSE result is 0.0062. RMSE result is 0.0786. MAE result is 0.0063. Lastly, MAPE result is 1.8456e-04.

```

k_fold_scores = cross_val_score(forest_reg, X_train_tr, y_train,
                                scoring = 'neg_mean_squared_error', cv=5)
tree_rmse_scores = np.sqrt(-k_fold_scores)
display_scores(tree_rmse_scores)

Scores: [4.69104355 4.82207788 5.17502986 5.34007658 6.17129851]

Mean: 5.239905275893977
Standard deviation: 0.5210691704996271

```

Figure 5.40: Result of 5-fold cross validation with decision tree regressor using the training set

The code above shows applying the 5-fold cross-validation score function on the decision tree regression model. The RMSE score is likely be overly optimistic when

evaluated directly on the training set. Hence cross-validation is used to obtain a more accurate RMSE value. K-fold cross-validation is performed on the model by calling the *cross_val_score* function from *sklearn*. In this case, a 5-fold cross-validation (*cv=5*) is performed on the LR model with an output score of negative mean squared error. This output is subsequently converted to RMSE and display the mean of the RMSE, 5.24 (in Figure 5.40) The standard deviation of the RMSE, which is 0.5211 (in Figure 5.40).

5.4.4 Fine tune the prediction model

There are several parameters in SVR that undergo changes during hyperparameter tuning, including *C*, *gamma*, *degree*, *kernel*, and *epsilon*. The parameter *C* is responsible for controlling the margin of the hyperplane. If the value of *C* is too low, it may result in an underfitting issue, whereas a high value of *C* may lead to an overfitting issue [54].

The next parameter, *gamma*, determines the influence of a single training example on the separation line. A low *gamma* value gives importance to points far from the potential separation line, while a high *gamma* value gives importance to points near the possible line. In this case, only the '*scale*' and '*auto*' options for *gamma* are applied for hyperparameter tuning.

The parameter *degree* defines the degree of the polynomial kernel function used in SVR. In this dataset, it was observed that a degree value of approximately 3 provides optimal results. However, for other types of kernel functions, this parameter will be ignored.

Next, the parameter *kernel* includes four types: *linear*, *poly*, *rbf*, *sigmoid*, and *precomputed*. After manually testing the parameter kernel, the results show that "*poly*" and "*rbf*" have the most optimized results for this dataset.

Lastly, the parameter *epsilon* determines the width of the tube around the hyperplane [55]. Points that fall within this tube are considered accurate predictions and are not penalized by the algorithm. The parameter *epsilon* was manually tested within the range of 0 to 10, and the results indicated that an *epsilon* value of approximately 7 was optimal for this dataset.

The first parameter grid, denoted by “*param_grid_svr2*”, was created after several tests and an understanding of each parameter. The grid consisted of a *C* value range of 5000 to 10000 in increments of 1000, *gamma* values of *scale* and *auto*, *degree* values of 1 and 3, kernel types of *rbf* and *poly*, and epsilon values of 6, 7, and 8, as shown in the appendix. The second parameter grid, denoted by “*param_random_svr*”, consisted of 20 randomly generated *C* values between 10 to 1000, with the same *gamma* and *kernel* values as “*param_grid_svr2*”, *degree* values of 1, 3, and 5, and 10 *epsilon* values randomly generated between 5 to 10.

```

grid_svr2=perform_search_svr('Grid',regr_svr, param_grid_svr2)
print('SVR_Best estimator:', grid_svr2.best_estimator_)
print('SVR_Best hyperparameter settings:', grid_svr2.best_params_)
print('the best model(best score) =', np.sqrt(-grid_svr2.best_score_))
[CV 1/5] END C=10000, degree=3, epsilon=8, gamma=scale, kernel=poly;; score=0.775 total time= 30.7s
[CV 2/5] END C=10000, degree=3, epsilon=8, gamma=scale, kernel=poly;; score=0.806 total time= 30.2s
[CV 3/5] END C=10000, degree=3, epsilon=8, gamma=scale, kernel=poly;; score=0.795 total time= 41.3s
[CV 4/5] END C=10000, degree=3, epsilon=8, gamma=scale, kernel=poly;; score=0.774 total time= 55.0s
[CV 5/5] END C=10000, degree=3, epsilon=8, gamma=scale, kernel=poly;; score=0.765 total time= 35.3s
[CV 1/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=rbf;; score=0.875 total time= 6.9s
[CV 2/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=rbf;; score=0.878 total time= 5.8s
[CV 3/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=rbf;; score=0.877 total time= 6.3s
[CV 4/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=rbf;; score=0.868 total time= 5.4s
[CV 5/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=rbf;; score=0.873 total time= 6.4s
[CV 1/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=poly;; score=0.768 total time= 7.6s
[CV 2/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=poly;; score=0.800 total time= 6.9s
[CV 3/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=poly;; score=0.789 total time= 9.4s
[CV 4/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=poly;; score=0.767 total time= 9.8s
[CV 5/5] END C=10000, degree=3, epsilon=8, gamma=auto, kernel=poly;; score=0.758 total time= 12.8s
done
time took: 6361.84011387825
SVR_Best estimator: SVR(C=10000, degree=1, epsilon=7)
SVR_Best hyperparameter settings: {'C': 10000, 'degree': 1, 'epsilon': 7, 'gamma': 'scale', 'kernel': 'rbf'}
the best model(best score) = nan

```

Figure 5.41: Output after performed grid search on SVR model

Perform the first fine tuning technique, grid search, on the SVR model by using the function code (*returns the model after a grid search or random search has been performed*) in appendix with the input fine tuning technique name “*Grid*” and the parameter grid named “*param_grid_svr2*” as in appendix. This fine tuning took 6361.84 seconds to finish, and the best hyperparameter for this model is shown in Figure 5.41.

```

random_svr2=perform_search_svr('Random',regr_svr, param_random_svr)
print('the best model(best score) =', np.sqrt(-random_svr2.best_score_))
print('SVR_Best estimator:', random_svr2.best_estimator_)
print('SVR_Best hyperparameter settings:', random_svr2.best_params_)

Performing Random Search...done
time took: 215.92864775657654
the best model(best score) = 13.004852539255248
SVR_Best estimator: SVR(C=982.1242917475304, degree=1, epsilon=7.314731798359686)
SVR_Best hyperparameter settings: {'kernel': 'rbf', 'gamma': 'scale', 'epsilon': 7.314731798359686, 'degree': 1, 'C': 982.1242917475304}

```

Figure 5.42: Output after performed random search on SVR model

Perform the second tuning technique, random search, on the SVR model by using the function code (*returns the model after a grid search or random search has been performed*) in appendix with the input fine tuning technique name "Random" and the parameter grid named "param_random_svr" as in appendix. This fine tuning took 215.93 seconds to finish, and the best hyperparameter for this model is shown in Figure 5.42.

```
perform_search_randomPlus=perform_search_randomPlus(param_random_svr)

Performing Random Search Plus...
For 20 times.....

No_0 MSE is 737.8176359485099
No_1 MSE is 179.97479463813605 ----time took: 254.05408906936646
No_2 MSE is 197.22996205339953 ----time took: 121.33799600601196
No_3 MSE is 181.2743880626155 ----time took: 236.09348821640015
No_4 MSE is 193.57042158646922 ----time took: 133.24931740760803
No_5 MSE is 200.04448804652787 ----time took: 146.44625210762024
No_6 MSE is 197.40247580459294 ----time took: 161.07053804397583
No_7 MSE is 198.4210653393532 ----time took: 158.91597485542297
No_8 MSE is 187.61778057420975 ----time took: 202.48800826072693
No_9 MSE is 181.2743880626155 ----time took: 234.0438404083252
No_10 MSE is 193.5513843582489 ----time took: 155.5195972919464
No_11 MSE is 190.52592945583064 ----time took: 209.67320370674133
No_12 MSE is 184.84652860258393 ----time took: 217.6690547466278
No_13 MSE is 193.66467701639138 ----time took: 123.9607150554657
No_14 MSE is 187.61778057420975 ----time took: 175.054851770401
No_15 MSE is 193.57042158646922 ----time took: 158.55393242835999
No_16 MSE is 194.8747960977421 ----time took: 155.07399129867554
No_17 MSE is 197.40247580459294 ----time took: 163.26728200912476
No_18 MSE is 179.97479463813605 ----time took: 247.80920100212097
No_19 MSE is 193.5513843582489 ----time took: 128.7051646709442
No_20 MSE is 193.5513843582489 ----time took: 133.90691900253296

Best parameters--> {'kernel': 'rbf', 'gamma': 'scale', 'epsilon': 5.6508419334444815, 'C': 982.1242917475304}
Best MSE--> 179.97479463813605

done
time took: 3519.8314275741577
```

Figure 5.43: Output after performed random search plus on SVR model

There is no existing library available to perform the random search plus method. Therefore, a self-defined function for Random Search Plus was created based on the study [52]. This function utilizes all the parameters available in "param_random_svr", except for the parameter C , where only two C values are used at a time (*one select ascendingly, one select randomly*). This explains why there are 20 iterations of Random Search being performed, as shown in Figure 5.43. During this hyperparameter tuning, the original SVR model's MSE error is recorded and compared to the MSE result of each Random Search iteration. Finally, the Random Search iteration with the smallest MSE error is outputted as an optimized SVR model.

Perform the third fine tuning technique, random search plus, on the SVR model by using the function code (*returns the model after a random search plus has been performed*) in appendix with the input parameter grid named "*param_random_svr*" as in appendix. This fine tuning took 3519.83 seconds to finish. After it run 20 times random search, the best hyperparameter for this model is shown in Figure 5.43.

```
perform_search_parameterOpt_loop=perform_search_parameterOpt_loop(100, param_random_svr)
No_ 89 MSE is 334.6242787472128 -----time took: 1.5782148838043213
No_ 90 MSE is 265.0104763768305 -----time took: 0.910456657409668
No_ 91 MSE is 198.4210653393532 -----time took: 0.9584286212921143
No_ 92 MSE is 316.81535339515466 -----time took: 1.974910020828247
No_ 93 MSE is 194.31042864760335 -----time took: 1.0988125801086426
No_ 94 MSE is 332.7300533120856 -----time took: 2.1978182792663574
No_ 95 MSE is 198.72411783762226 -----time took: 1.0046985149383545
No_ 96 MSE is 349.01705591398826 -----time took: 1.1780495643615723
No_ 97 MSE is 317.82753309839063 -----time took: 2.7640888690948486
No_ 98 MSE is 197.57864781935038 -----time took: 1.0051312446594238
No_ 99 MSE is 205.40678564570348 -----time took: 0.9577083587646484
No_ 100 MSE is 548.5643380130731 -----time took: 0.4400923252105713

Best parameters--> {'kernel': 'rbf', 'C': 982.1242917475304, 'gamma': 'scale', 'epsilon': 7.440606656840263}
Best MSE--> 179.72393969122018

done
time took: 140.26222252845764
```

Figure 5.44: Output after performed parameter optimization loop on SVR model

Since there was no library available to perform the parameter optimization loop method, a self-defined function for Random Search Plus was created based on the methodology described in [53]. This function tests every combination of hyperparameters by randomly selecting one value for each parameter in "*param_random_svr*" and repeats the testing according to the input number of iterations, as shown in Figure 5.44 with 100 iterations. The optimized SVR model is determined by selecting the model with the smallest MSE error among all models generated in the iterations of the Random Search Plus function.

Perform the last fine tuning technique, parameter optimization loop, on the SVR model by using the function code (*returns the model after a parameter optimization loop has been performed*) in appendix with the input of 100 loops and the parameter grid named "*param_random_svr*" as in appendix. This fine tuning took 140.26 seconds to finish. After 100 loops, the best hyperparameter for this model is shown in Figure 5.44.

5.5 Comment and highlight the feasibility of the proposed method

There is a significant amount of noise present in the images captured in the open parking space. This noise includes parking spaces blocked by trees, oil leaks in the parking lot, ground fissures on the parking lot, and faded parking lines. Furthermore, the lighting conditions in the images are affected by the weather, and at times, the parking lot photo may be overexposed due to sunlight. All of these conditions make it challenging to detect empty and occupied parking spaces accurately. Fortunately, a robust object detection algorithm called YOLOv4 is employed here, which has the capability to handle all of these conditions with minimal time and resource costs. YOLOv4 utilises CNN for object detection, and it has high object detection speed and accuracy, resulting in fewer background faults.

Next, SVR is used to develop the parking vacancy prediction model. The expected output of the prediction model is the number of empty parking spaces. This prediction output is created by anticipating the input features collected from the parking lot, which include the total parking space, date and time and the weather type. SVR is well-suited for nonlinear and time series problems, making it a good choice for developing the parking vacancy prediction model.

Additionally, implementing a user-friendly GUI for the parking vacancy prediction model would increase accessibility for users without programming knowledge. This could potentially expand the usage of the model to a wider audience. The PySimpleGUI framework, which is known for its simplicity and cost-efficiency, was chosen for the development of the GUI.

5.6 Concluding Remark

A hybrid prediction model is developed using the YOLOv4 framework and SVR ML algorithm. In other words, this model is formed by a custom YOLOv4 object model and an SVR model. This hybrid prediction model only requires the input dataset in an image format, making the data collection process much easier in real life. The system requirements to develop the model include a laptop as the hardware, software such as Jupyter Notebook in the Anaconda environment, and main software libraries such as

CHAPTER 5

scikit-learn, Darknet YOLOv4, and PySimpleGUI. Four datasets from Brazil, Taiwan, Italy, and China are used in this study.

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSIN

6.1 Model Testing and Performance Metrics (YOLOv4 model)

The performance and work results of the YOLOv4 model

```

_9000: 2.402813, 3.582072 avg loss, 0.000001 rate, 12.967691 seconds, 576000 images, 0.257028 hours left
Saving weights to /mydrive/yolov4/training/yolov4-custom_9000.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_last.weights
Saving weights to /mydrive/yolov4/training/yolov4-custom_final.weights
If you want to train from the beginning, then use flag in the end of training command: -clear

```

Figure 6.1: Training result

- Max batch is 9000
- Total loss is 2.402813
- The average loss is 3.582072
- Currently learning rate on 6000 iterations is 0.000001
- Time spends on process the batch 12.96769 seconds
- Number of images utilised overall during training 576000 images
- Time estimates to complete the max batch or iteration 0.257028 hours

Based on the training results depicted in Figure 6.1, it can be observed that the YOLOv4 model was trained with 9000 iterations or max batch, resulting in a total loss of 2.402813 and an average loss of 3.582072. According to [47], training should continue until the average loss value falls below 0.3, as this parameter is a crucial indicator of the model's performance. A lower average loss value indicates a better-performing model, although it should not fall below 0.05. However, it is noteworthy that the average loss for this custom YOLOv4 model increased from 2.947 at 6000 iterations to 3.582 at 9000 iterations. This suggests that the standard loss value may not be applicable in this case, as the dataset used in this study has different characteristics when compared to the dataset used in this study [47].

Furthermore, at the beginning of training the learning rate is set to 0.001, after training for 6000 iterations it decreases to 0.00001, then at 9000 iterations it decreases to 0.000001. Learning rate decay was used and it helped with both optimization and generalization during training. The batch processing time was 12.96769 seconds, and a

total of 576000 images were used overall during training. Finally, the estimated time remaining to complete the maximum batch or iteration in training is 0.257028 hours.

```

[YOLO]
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.570
avg_outputs = 489910
Allocate additional workspace_size = 134.22 MB
Loading weights from /mydrive/yolov4/training/yolov4-custom_last.weights...
seen 64, trained: 576 K-images (9 Kilo-batches_64)
Done! Loaded 162 layers from weights-file

calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
872
detections_count = 52977, unique_truth_count = 49732
class_id = 0, name = space-empty, ap = 99.98% (TP = 26624, FP = 130)
class_id = 1, name = space-occupied, ap = 99.97% (TP = 23089, FP = 148)

for conf_thresh = 0.25, precision = 0.99, recall = 1.00, F1-score = 1.00
for conf_thresh = 0.25, TP = 49713, FP = 278, FN = 19, average IoU = 94.51 %

IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.999779, or 99.98 %
Total Detection Time: 28 Seconds

```

Figure 6.2: Training result with mAP@IoU=0.5

```

[YOLO]
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.570
avg_outputs = 489910
Allocate additional workspace_size = 134.22 MB
Loading weights from /mydrive/yolov4/training/yolov4-custom_last.weights...
seen 64, trained: 576 K-images (9 Kilo-batches_64)
Done! Loaded 162 layers from weights-file

calculation mAP (mean average precision)...
Detection layer: 139 - type = 28
Detection layer: 150 - type = 28
Detection layer: 161 - type = 28
872
detections_count = 52977, unique_truth_count = 49732
class_id = 0, name = space-empty, ap = 99.97% (TP = 26622, FP = 132)
class_id = 1, name = space-occupied, ap = 99.96% (TP = 23088, FP = 149)

for conf_thresh = 0.25, precision = 0.99, recall = 1.00, F1-score = 1.00
for conf_thresh = 0.25, TP = 49710, FP = 281, FN = 22, average IoU = 94.51 %

IoU threshold = 75 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.75) = 0.999678, or 99.97 %
Total Detection Time: 402 Seconds

```

Figure 6.3: Training result with mAP@IoU=0.75

Precision, recall, and F1 score are presented in Figure 6.2 and Figure 6.3, and these values were computed using the final weights. mAP is then calculated using IoU. IoU measures the overlap between the predicted and ground truth bounding boxes, and its value ranges from 0 to 1. For this model, the average IoU values for 0.5 and 0.75 IoU thresh are both 94.51%. Furthermore, the model achieved a high level of accuracy in detecting the intended targets, as evidenced by the mAP values of 99.98% and 99.97% at 50% and 75%, respectively, in the test dataset.

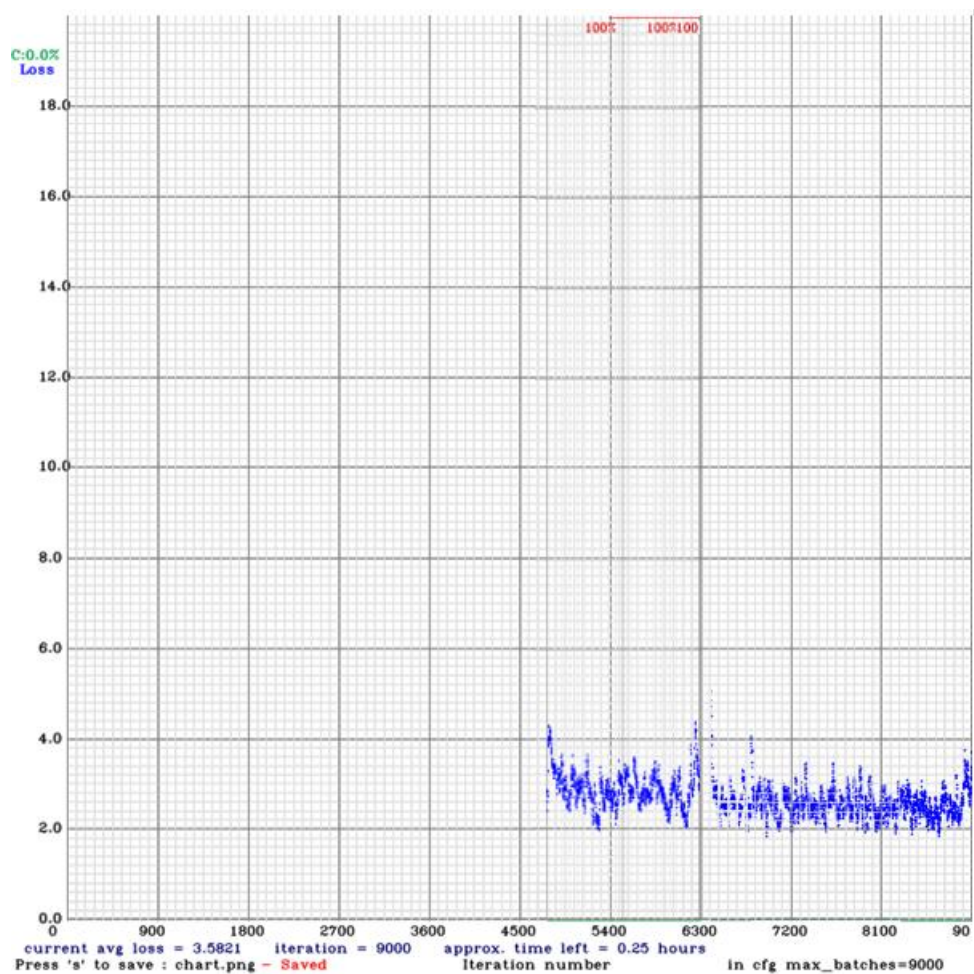


Figure 6.4: Loss and mAP chart

In addition, the loss and mAP chart presented in Figure 6.4 only covers the range from 4800 to 9000 iterations. This is because the last training was continued from 4800 iterations, and several loss and mAP charts from each continuous training were overlapped to show the overall loss during training. Due to usage limits on Google Colab, the training process for the custom model had to be segmented, and the loss and mAP chart before 4800 iterations were not included. Figure 6.4 indicates that the loss starts to increase after 9000 iterations, suggesting that the training should be stopped, or the learning rate should be lowered. Upon completion of the 9000 iterations, a "*bad.list*" file was generated to store any label files that may have issues.

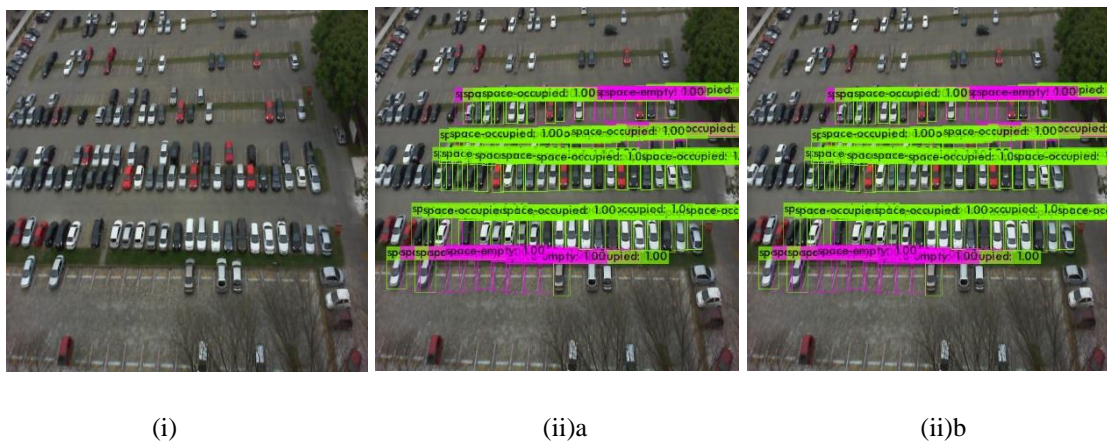


Figure 6.5: PKLot- PUCPR (i) test data, (ii) prediction result on the test data with (a) 0.2, (b) 0.7 thresh value

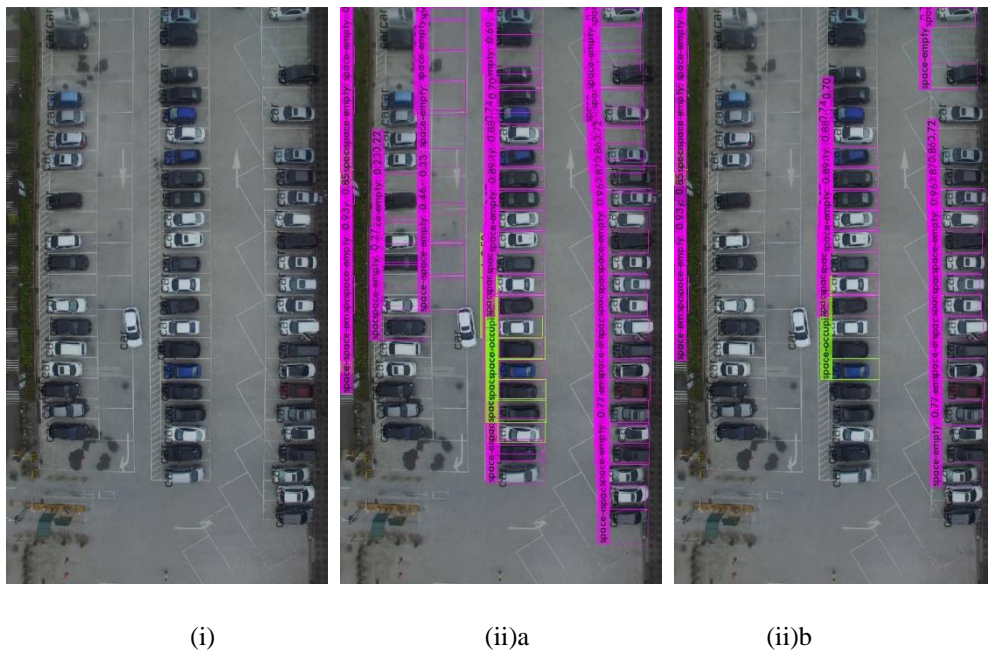


Figure 6.6: CARPK (i) test data, (ii) prediction result on the test data with (a) 0.2, (b) 0.7 thresh value



Figure 6.7: CNRPark+EXT (i) test data, (ii) prediction result on the test data with (a) 0.2, (b) 0.7 thresh value



Figure 6.8: Aerial View of Parking Lot (i) test data, (ii) prediction result on the test data with (a) 0.2, (b) 0.7 thresh value

The trained custom YOLOv4 model was evaluated on four unseen datasets, namely PK Lot, CARPK, CNRPark+EXT, and Aerial View of Parking Lot. The first test was conducted on the PK Lot sample dataset using a thresh value of 0.2 and 0.7. The results are presented in Figure 6.5, which show that the model was able to detect all the parking spaces in the centre area only, regardless of the thresh value used. This issue is due to the limited number of labelled parking spaces in the centre area of the PK Lot dataset. Additionally, the model performed well and achieved a perfect result on this dataset. With the sign of the loss is increasing when the training approach 9000 iteration, it indicating the model has potential for overfitting.

The second test was conducted on the CARPK sample dataset using the same threshold value as before. The results are presented in Figure 6.6. At a threshold value of 0.2, the model performed poorly, wrongly identifying more than half of the occupied parking spaces as vacant and mistaking the pavement markings on the road as empty parking spaces. With a threshold value of 0.7, the model performed slightly better, with every bounding box correctly positioned and unaffected by the pavement markings on the road. However, the model still failed to correctly identify objects, as it identified more than half of the occupied parking spaces as vacant. This can be explained by the fact that this dataset contains incorrect pavement markings, which can mislead the model into mistaking them for parking lines and output a poor result. Moreover, the overfitting issue with this model can explain why it failed to correctly identify objects.

The third test was conducted on the CNRPark+EXT sample dataset using the same threshold value as before. The results are presented in Figure 6.7. However, at a threshold value of 0.2, the model incorrectly identified the car top as an empty parking space. Even with a threshold value of 0.7, the model still could not accurately identify the occupied parking spaces in the sample dataset. These prediction results are poor as the model could not correctly identify the objects and misidentified the car top as an empty parking space. It can be concluded that this dataset is not suitable for the model as the images only show a single straight line under the car, making it difficult for both humans and machines to identify it as a parking space.

During the last test on the Aerial View of Parking Lot sample dataset using the same threshold value as before, there was an issue with overlapping anticipated and true bounding boxes at a threshold value of 0.2, highlighting the importance of IoU for this testing. The results are presented in Figure 6.8. At a threshold value of 0.7, the overlapping issue did not occur, but the results still showed that the model needs further improvement, as it was unable to identify every object in the image and misidentified a few occupied parking spots as empty, possibly due to the lighting and colour of the car tops being similar to that of the road, especially since it was a cloudy day with limited sunlight.

6.2 Model Testing and Performance Metrics (Prediction model)

The performance and work results of the prediction model

```
Shape of X_val_num: (671, 8)
Shape of X_val_cat: (671,)
Shape of X_val_tr(after
transformed weather set): (671, 11)
```

Figure 6.9: The processed test set x

```
pk_filter_large.Empty.mean()
Out[5]: 57.02749832327297
```

Figure 6.10: Mean of empty features after filter with total parking equal 100

The validation set's weather column is taken out and applied using one-hot encoding rather than the standard label using the code (*pre-processing validation set*)

in appendix. This is due to the data itself has various scales, and the numerical data in the dataset needs to be standardised. Finally, create a single NumPy array by combining the numerical and weather sets to predict using a ML model. In Figure 6.9, shows the shape of validation set after pre-processing. The code (*pre-processing test set*) in appendix is executed afterward, which functions basically as the validation set.

Both the x validation and test sets contain 671 rows of data after pre-processing (only includes total parking equal to 100) and 11 columns (*after applying one-hot encoding to the weather column*). The mean of empty features after filter with total parking equal to 100 is 57.027 and it used to calculate the accuracy of the model using RMSE.

Then it is ready to perform testing on the created model with the validation and test sets. The function code (*shows the evaluation result on train, validation and test sets*) is utilised to print the evaluation result on train, validation and test sets by input the ML model.

```
show_3result(regr,'regr')
<--- regr validate --->
MSE = 737.8176359485099
RMSE= 27.162798750285468
MAE = 21.231218639335907
MAPE= 2.9220e+15

<--- regr test --->
MSE = 735.578557048572
RMSE= 27.121551523623644
MAE = 21.03588438062626
MAPE= 1.5783e+15
```

Figure 6.11: Performance testing on the SVR model

Perform testing using the validation and test sets on the SVR model by using the function code (*shows the evaluation result on train, validation and test sets*) in appendix. The performance outputs for the validation and test sets are shown in Figure 6.11. For validation set, MSE result is 737.82, RMSE result is 27.16, MAE result is 21.23 and MAPE result is 2.9220e+15. For test set, MSE result is 735.58, RMSE result is 27.12, MAE result is 21.04 and MAPE result is 1.5783e+15.

```
show_3result(lin_reg,'lin_reg')
```



```

<--- lin_reg validate --->
MSE = 859.4842363469412
RMSE= 29.31696158108717
MAE = 23.634343515960417
MAPE= 2.2345e+15

<--- lin_reg test --->
MSE = 851.105158453784
RMSE= 29.173706628637095
MAE = 23.32449069540031
MAPE= 1.1776e+15

```

Figure 6.12: Performance testing on the LR model

Perform testing using the validation and test sets on the LR model by using the function code (*shows the evaluation result on train, validation and test sets*) in appendix. The performance outputs for the validation and test sets are shown in Figure 6.12. For validation set, MSE result is 859.48, RMSE result is 29.32, MAE result is 23.63 and MAPE result is 2.2345e+15. For test set, MSE result is 851.11, RMSE result is 29.17, MAE result is 23.32 and MAPE result is 1.1776e+15.

```

show_3result(tree_reg,'tree_reg')
<--- tree_reg validate --->
MSE = 20.34217265589241
RMSE= 4.510229778613548
MAE = 1.8181936460624983
MAPE= 2.0135e+13

<--- tree_reg test --->
MSE = 15.724200317925924
RMSE= 3.965375179970481
MAE = 1.792272845551534
MAPE= 2.2149e+14

```

Figure 6.13: Performance testing on the decision tree regression model

Perform testing using the validation and test sets on the decision tree regression model by using the function code (*shows the evaluation result on train, validation and test sets*) in appendix. The performance outputs for the validation and test sets are shown in Figure 6.13. For validation set, MSE result is 20.34, RMSE result is 4.51, MAE result is 1.81 and MAPE result is 2.0135e+13. For test set, MSE result is 15.72, RMSE result is 3.96, MAE result is 1.79 and MAPE result is 2.2149e+14.

Following the SVR model is evaluated after performing on four different types of hyperparameter tuning techniques. The function code (*shows the evaluation result on train, validation and test sets*) is utilised to print the evaluation result on train,

validation and test sets by inputting each of the SVR model after performing hyperparameter tuning.

```
In [250]: show_3result(grid_svr2, 'grid_svr2')

<--- grid_svr2 train --->
MSE = 99.46794046442575
RMSE= 9.973361542851325
MAE = 7.3116938331946715
MAPE= 4.8520e+14

<--- grid_svr2 validate --->
MSE = 148.1435299802517
RMSE= 12.171422676920383
MAE = 8.784239418084182
MAPE= 8.5308e+14

<--- grid_svr2 test --->
MSE = 141.52863908016408
RMSE= 11.896580982793505
MAE = 8.846254883462878
MAPE= 4.7698e+14
```

Figure 6.14: Performance testing on the fine-tuned SVR model, obtained using grid search

The performance of the fine-tuned SVR model, obtained using grid search, was evaluated on the training, validation, and test sets using the function code (*shows the evaluation result on train, validation and test sets*) provided in the appendix. The resulting performance outputs for the three sets are summarized in Figure 6.14. The MSE for the training set was found to be 99.47, while the MSE for the validation and test sets are 148.14 and 141.53, respectively. Similarly, the RMSE for the training, validation, and test sets were found to be 9.97, 12.17, and 11.90, respectively, while the MAE results were 7.31, 8.78, and 8.85, respectively. Finally, the MAPE for the training set was 4.852E+14, while the MAPE for the validation and test sets are 8.5308E+14 and 4.7698E+14, respectively.

```
In [244]: show_3result(random_svr2, 'random_svr2')

<--- random_svr2 train --->
MSE = 136.31718661617347
RMSE= 11.675495133662361
MAE = 8.66947715383008
MAPE= 4.3559e+14

<--- random_svr2 validate --->
MSE = 179.5027158507757
RMSE= 13.39786236124165
MAE = 10.034482937221954
MAPE= 6.2625e+14

<--- random_svr2 test --->
MSE = 179.81025862416635
RMSE= 13.409334756958167
MAE = 10.086828778885902
MAPE= 4.6148e+14
```

Figure 6.15: Performance testing on the fine-tuned SVR model, obtained using random search

The performance of the fine-tuned SVR model, obtained using random search, was evaluated on the training, validation, and test sets using the function code (*shows the evaluation result on train, validation and test sets*) provided in the appendix. The resulting performance outputs for the three sets are summarized in Figure 6.15. The MSE for the training set was found to be 136.32, while the MSE for the validation and test sets are 179.50 and 179.81, respectively. Similarly, the RMSE for the training, validation, and test sets were found to be 11.68, 13.40, and 13.41, respectively, while the MAE results were 8.67, 10.03, and 10.09, respectively. Finally, the MAPE for the training set was 4.3559E+14, while the MAPE for the validation and test sets are 6.2625E+14 and 4.6148E+14, respectively.

```
In [27]: show_3result(perform_search_randomPlus, 'Random search plus')

<--- Random search plus  train --->
MSE = 134.49213491500564
RMSE= 11.59707441189396
MAE = 8.142730524094102
MAPE= 4.0003e+14

<--- Random search plus  validate --->
MSE = 179.97479463813605
RMSE= 13.415468483736825
MAE = 9.691244448237631
MAPE= 5.7626e+14

<--- Random search plus  test --->
MSE = 175.58577017546216
RMSE= 13.250878090732785
MAE = 9.612327775791663
MAPE= 3.9977e+14
```

Figure 6.16: Performance testing on the fine-tuned SVR model, obtained using random search plus

The performance of the fine-tuned SVR model, obtained using random search plus, was evaluated on the training, validation, and test sets using the function code (*shows the evaluation result on train, validation and test sets*) provided in the appendix. The resulting performance outputs for the three sets are summarized in Figure 6.16. The MSE for the training set was found to be 134.49, while the MSE for the validation and test sets are 179.97 and 175.59, respectively. Similarly, the RMSE for the training, validation, and test sets were found to be 11.60, 13.42, and 13.25, respectively, while the MAE results were 8.14, 9.69, and 9.61, respectively. Finally, the MAPE for the training set was 4.0003E+14, while the MAPE for the validation and test sets are 5.7626E+14 and 3.9977E+14, respectively.

```
In [30]: show_3result(perform_search_parameterOpt_loop,'Parameter optimization loop')

<--- Parameter optimization loop  train --->
MSE = 136.47186105868803
RMSE= 11.682117147961154
MAE = 8.706254741842596
MAPE= 4.4265e+14

<--- Parameter optimization loop  validate --->
MSE = 179.72393969122018
RMSE= 13.406115757042388
MAE = 10.063347266080852
MAPE= 6.3624e+14

<--- Parameter optimization loop  test --->
MSE = 180.00091421498254
RMSE= 13.41644193573626
MAE = 10.111287436569853
MAPE= 4.6793e+14
```

Figure 6.17: Performance testing on the fine-tuned SVR model, obtained using parameter optimization loop

The performance of the fine-tuned SVR model, obtained using parameter optimization loop, was evaluated on the training, validation, and test sets using the function code (*shows the evaluation result on train, validation and test sets*) provided in the appendix. The resulting performance outputs for the three sets are summarized in Figure 6.17. The MSE for the training set was found to be 136.47, while the MSE for the validation and test sets are 179.72 and 180.00, respectively. Similarly, the RMSE for the training, validation, and test sets were found to be 11.68, 13.41, and 13.42, respectively, while the MAE results were 8.71, 10.06, and 10.11, respectively. Finally, the MAPE for the training set was 4.4265E+14, while the MAPE for the validation and test sets are 6.3624E+14 and 4.6793E+14, respectively.

From the result discussion above, the summary table summarizes all the results is made, and this can be seen in following Table 6.1-2. While Table 6.3 show the two of the parameter grid in details.

Table 6.1: Prediction result

Result/Models		Support vector regression (SVR)	Linear regression (LR)	Decision tree regression
Prediction Result- Training set	MSE	657.01	795.06	0.0062
	RMSE	25.63	28.20	0.0786
	MAE	19.35	22.46	0.0063
	MAPE	2.3085E+15	1.6063E+15	1.85E-04
5-fold CV	Mean of CV	26.60	28.26	5.24
	Standard deviation of CV	1.04	0.63	0.52
Validation set	MSE	737.82	859.48	20.3422
	RMSE	27.16	29.32	4.5102
	MAE	21.23	23.63	1.8182
	MAPE	2.922E+15	2.2345E+15	2.0135E+13
Test set	MSE	735.58	851.11	15.7242
	RMSE	27.12	29.17	3.9654
	MAE	21.04	23.32	1.7923
	MAPE	1.5783E+15	1.1776E+15	2.2149E+14

Table 6.2: Prediction result (after fine-tuning)

Result/SVR Model		Grid search	Random search	Random search plus	Parameter optimization loop
Prediction Result- Training set	MSE	99.47	136.32	134.49	136.47
	RMSE	9.97	11.68	11.60	11.68
	MAE	7.31	8.67	8.14	8.71
	MAPE	4.852E+14	4.3559E+14	4.0003E+14	4.4265E+14
Validation set	MSE	148.14	179.50	179.97	179.72
	RMSE	12.17	13.40	13.42	13.41
	MAE	8.78	10.03	9.69	10.06
	MAPE	8.5308E+14	6.2625E+14	5.7626E+14	6.3624E+14
Test set	MSE	141.53	179.81	175.59	180.00
	RMSE	11.90	13.41	13.25	13.42
	MAE	8.85	10.09	9.61	10.11
	MAPE	4.7698E+14	4.6148E+14	3.9977E+14	4.6793E+14
Number of Loops		-	-	20 times Random search	100 times
Time took (minutes):		106.03	3.60	58.66	2.34
Parameter Grid:		param_grid_svr2	param_random_svr	param_random_svr	param_random_svr
Best Parameters:		{'kernel': 'rbf', 'gamma': 'scale', 'epsilon': 7, 'degree': 1, 'C': 10000}	{'kernel': 'rbf', 'gamma': 'scale', 'epsilon': 7.3147, 'degree': 1, 'C': 982.1243}	{'kernel': 'rbf', 'gamma': 'scale', 'epsilon': 5.6508, 'C': 982.1243}	{'kernel': 'rbf', 'gamma': 'scale', 'epsilon': 7.4406, 'C': 982.1243}

Table 6.3: Parameter Grid

param_grid_svr2	param_random_svr
<pre>{'C': [6000,7000,8000,9000,10000], 'gamma': ['scale', 'auto'], 'degree':[1,3], 'kernel': ['rbf','poly'], 'epsilon':[6,7,8]}</pre>	<pre>{'C': uniform(loc=10, scale=990).rvs(20), 'gamma': ['scale', 'auto'], 'degree':[1,3,5], 'kernel': ['rbf','poly'], 'epsilon': uniform(loc=5, scale=5).rvs(10)}</pre>

According to Table 6.1, the decision tree regression models have an overfitting issue because the MSE and RMSE are so low, with values of 0.0062 and 0.0786. It might be because this ML algorithm is too complex for the dataset. On the other hand, the LR model also has a bit underfitting problem as the MSE and RMSE are slightly higher than the SVR model, which is 795.06 and 28.20, respectively. This can be explained by the fact that LR is a much simpler ML algorithm than SVR. The SVR delivers the best result among the three models, but it still seems to have an underfitting issue. An RMSE of SVR, which is 25.63 ($25.63/57.027=44.96\%$), is not very pleasant compared to the average “empty” value of 57.027.

Although the model created with decision tree regression has the most petite MAE and MAPE. Still, it is not the best model because these values are too small, and obviously, there is an overfitting issue. Then, the MAP and MAPE of the SVR model are 19.35 and $2.3085e+15$, respectively. In this case, the MAPE is larger than 1 due to the fact that the y_{true} values are very close to or equal to zero. Therefore, the MAPE is less significant in the model evaluation. The MAP of the SVR model is slightly lower than that of the LR model. Hence, SVR is more considerable than the other two algorithms. The standard value of MAPE should be 25% or lower, and the MAPE of the decision tree regression model achieves this with a value of 0.018%, but the result is too promising and might lead to model overfitting [49].

Table 6.1 shows a more accurate RMSE and standard deviation of RMSE for every model after applying the 5-fold cross-validation score. First, convert all the negative mean square errors that output as the result of a 5-fold cross-validation score to RMSE, then the mean function is applied, and the calculated value (mean of RMSE) is a more accurate RMSE. A more accurate RMSE for SVR, LR, and decision tree regression are 26.6, 28.26, and 5.24, respectively.

After evaluating all the models on unseen data using validation and test sets, the SVR model has a slight overfitting issue (MSE of the validation set is 2.24 lower than the MSE of the test set), but it still performs the best among the three models, as the MSEs are both the smallest, with values of 737.82 and 735.58, respectively. Subsequently, the LR model has a bit of an overfitting issue too, as the MSE on the validation set slightly decreases to 8.38 when compared to the MSE on the test set. Next,

decision tree regression is overfitting, as the MSE increases significantly (from 0.0062 to 20.34) when testing with the validation set, and the MSE of the test set is 4.61 lower than the MSE of the validation set. The changes is significant in decision tree regression when comparing the changes in the result using a ratio.

Since SVR model perform the best among the three ML algorithms, SVR model is being selected to fine-tuning to find the best hyperparameter. Four fine tuning technique is applied. Two different parameter grid is utilised as shown in Table 6.3.

The prediction results of the fine-tuned SVR models are presented in Table 6.2. The SVR model obtained through grid search exhibits an overfitting issue as the validation set MSE is 6.61 lower than the test set MSE, despite having the lowest errors among the four models. On the other hand, the SVR model obtained through random search has a training MSE of 136.32, a validation MSE of 179.50, and a test MSE of 179.81, indicating that it is a good model with no overfitting or underfitting issues. However, the SVR model obtained through random search plus also exhibits overfitting, with a validation MSE decrement of 4.62 when compared to the test MSE. Finally, the SVR model obtained through parameter optimization loop yields similar results to random search, albeit with slightly higher errors on the training, validation, and test sets.

Based on the results, it can be concluded that the fine-tuned SVR model obtained using random search is the most suitable model for this dataset. This model has a good balance between training, validation, and test errors, indicating that it is not overfitting or underfitting. In contrast, the other models either have overfitting or underfitting issues, which may impact their performance on new, unseen data. Therefore, the fine-tuned SVR model obtained using random search is the most appropriate model for this dataset.

```
In [230]: # save the model(random search) to disk
          filename = 'random_svr2.sav'
          joblib.dump(random_svr2, filename)
          print("save successfully...")
          # some time later...

          save successfully...
```

Figure 6.18: Save the fine-tuned SVR model using random search as sav file

Furthermore, the best weight obtained from the fine-tuned SVR model using random search (Figure 6.18) is being utilized for the "Predict Open Space Parking Vacancies" GUI, which is developed using PySimpleGUI.

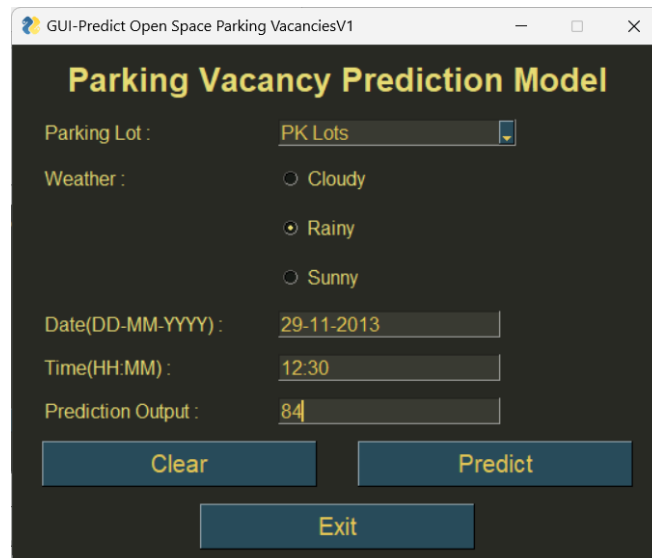


Figure 6.19: Parking Vacancy Prediction Model-GUI

The initial version of the parking vacancy prediction model GUI requires three inputs from the user: weather type, date, and time (Figure 6.19). As the SVR model was developed using only the PK Lots dataset, the current version of the model can only make predictions for PK Lots. Once the user has entered the date and time in the correct format and selected the current weather, they can click the "Predict" button to obtain the parking vacancy prediction based on their input. The "Clear" button can be used to clear all inputs, including weather, date, time, and prediction output. To exit the system, the user can click the "Exit" button.

The model is limited to predicting dates within the monthly range of July to December every year, as it was developed using a dataset collected only from September to November. To address this limitation, the dataset used to build the SVR model should include data from every month of the year. This will ensure that the model can make accurate predictions throughout the year. Based on the current dataset, the model has shown promising results for predicting dates from August until November each year.

6.3 Project Challenges

Testing the YOLOv4 model in this project was a frustrating experience as the software used for performing training and testing on the model was time-consuming. In the current year, Google Colab's usage limit has decreased to 5-7 hours per day, with a cooldown period of 12-16 hours. As a result, the actual time spent on fine-tuning and testing in this research was almost double the expected time.

Performing hyperparameter tuning required significant CPU usage. To accelerate the test speed, a PC in the FYP lab is being utilised. As testing various parameter combinations took considerable time, a laptop was utilized to remotely control the lab PC during working days. However, the lab PC unexpectedly shut down twice, resulting in the loss of project software, test records, and important data. Although some critical tests were saved on the drive, most of the data was lost. Resulting in spending nearly three extra hours to reinstall necessary software and reconfigure the environment.

6.4 Objective Evaluation

The trained custom YOLOv4 model has demonstrated its ability to accurately detect occupied and vacant parking spaces, even in the presence of various types of noise, including obstacles, weather conditions, road conditions, and lighting conditions, as evidenced by its high accuracy on the PK Lots sample dataset. However, when tested on three other datasets, the results were poor, likely due to the pavement markings on the road being too similar to the boundary lines around the parking spaces. A significant point to consider is that these datasets were collected from different countries, with varying parking lot styles and car designs, which may have contributed to the model's decreased accuracy. Furthermore, this study did not consider nearby parking lots as part of the model's training data.

Next, an SVR model is developed to predict current and future empty parking spaces for the parking lot dataset that the model is trained with. This model takes into account the weather conditions of the day, and weather conditions during the desired prediction date will be requested as one of the input features. Additionally, an interactive GUI for the parking prediction model is created with PySimpleGUI, which allows individuals without programming experience to predict the availability of empty parking spaces using the SVR model. However, the current version of the prediction

model is limited to making predictions within the months from July to December, as the dataset used to train the model only covers the period from September 2012 to November 2012 (3 months only).

Overall, the hybrid prediction model is fully developed with the ability to predict empty parking space using an interactive GUI. The model includes a custom YOLOv4 object detection model and an SVR ML algorithm, which together enable accurate detection of occupied and vacant parking spaces. The model also takes into account weather conditions as an input feature for prediction. While the current version of the model is limited to making predictions within the months of July to December, it represents a promising development in the field of parking prediction models.

6.5 Concluding Remark

In this chapter, the performance of the hybrid prediction model, which consists of a custom YOLOv4 object detection model and an SVR model, is evaluated. The testing and evaluation of both the YOLOv4 object detection model and the SVR model are discussed in detail. Additionally, the challenges encountered during the fine-tuning process for both models are documented. Finally, the objective of this study is evaluated based on the performance and improvement of the hybrid prediction model.

CHAPTER 7 CONCLUSION AND RECOMMENDATION

7.1 Conclusion

In conclusion, this project aimed to overcome the issue of vehicle parking by developing a hybrid open parking space prediction model using ML and YOLOv4 object detector. The purpose of the model was to predict available parking space, which would maximize the utilization of parking space and reduce idling, ultimately having a positive impact on the environment. Through the literature review, it was found that SVR and YOLO object detection algorithms built with CNN were the most suitable algorithms for developing the prediction model. SVR is capable of handling nonlinear and time series problems effectively, while YOLO is known for its ability to perform real-time object detection with high accuracy. As a result, SVR was used to develop the parking vacancy prediction model, and YOLO was chosen to develop the object detection model.

During the YOLOv4 training, various environmental factors such as weather, road condition, and obstacles like trees or lamp posts were included in the training dataset. The custom YOLOv4 model was developed and tested on unseen data, and it was able to generate a label text file that was used as the dataset in developing the ML model. Several configurations were tested, and it was found that the first version of the custom YOLOv4 model showed the most promising performance. The training continued until 9000 iterations, where the model showed signs of overfitting. However, when tested on three other datasets, the model did not perform well, likely due to differences in the datasets used to train the model. Additionally, a new type of noise was identified, where pavement markings on the road were mistakenly identified by the model as empty parking spaces. As a result, future improvements for this model could include training with additional datasets specifically for detecting parking spaces with a reasonable noise included and testing on different parameter combinations in the configuration file.

Next, the parking vacancy prediction model was developed using SVR. The dataset used to train the model included weather conditions, but the performance analysis revealed that this feature had little impact on the number of empty parking spaces. This issue was attributed to the dataset characteristics. Additionally, two other

ML algorithms were compared to the SVR model, and the results indicated that the SVR model performed better. The SVR model was then fine-tuned using four hyperparameter tuning techniques, and the model fine-tuned with the random search method produced the most promising results, with the least overfitting issue and a lower error rate than the other three techniques. Ultimately, the SVR model fine-tuned with the random search method was integrated into the parking vacancy prediction GUI model.

However, the dataset used to train the SVR model is limited from September 2012 to November 2012 because only the dataset with large parking spaces is used, thereby the SVR model could not make a prediction for the month of January till June for every year. Next, an idea to include nearby parking conditions as a feature in the SVR model is not feasible for predicting parking vacancies because the prediction model is applied using an ML algorithm.

Finally, an interactive GUI has been developed using PySimpleGUI to enable users, particularly those without programming experience, to test the prediction model and predict the current and future availability of open parking spaces. The GUI has been designed with user-friendliness in mind, allowing non-technical users to easily access and utilize the prediction model.

7.2 Recommendation

It is recommended to increase the dataset used to train the custom YOLOv4 model by including all available datasets in the future. For object detection research, other research datasets typically have similar features and styles, with a single training image having only a few or a dozen objects. In contrast, the dataset used in this project contains a total of 100 small objects for most images. Therefore, it is important to consider the characteristics and features of the dataset used, as it has a significant impact on the model accuracy. Next, it is essential to increase the standards for dataset selection and cleaning processes to ensure that the dataset used is free from errors and contains characteristics suitable for the model. Otherwise, the ML model may have an underfitting issue, leading to lower performance. Using inadequate or unsuitable data can result in lower model performance, making it crucial to consider dataset quality and suitability for the model.

For the custom YOLOv4 model, further testing could be conducted on different parameter combinations in the configuration file. However, given the time investment required, such improvements may not yield significant results. Therefore, it would be more practical to select a sufficient amount of diverse and suitable datasets. Additionally, future work could involve including datasets with different road conditions, such as painted parking spaces, incorrect pavement markings, faded markings, and old markings. Furthermore, developing the object detection model with a higher version of the YOLO framework could achieve a more precise and accurate results.

Lastly, the SVR model can be improved by revising it to include more input features such as the nearby parking conditions and the occurrence of public holidays on the day. The nearby parking condition feature can be implemented by considering driver parking behavior, their priorities or preferred parking spaces in the parking lot using a rating scale. Additionally, a bigger dataset that includes data from each month of the year should be prioritized as the current model is unable to accurately predict parking vacancies in the first half of the year. Given the expected increase in the number of input features, it is recommended that various types of ANN algorithms be tested for developing the prediction model. ANNs are a suitable option when dealing with high complexity, similar to SVR. In particular, ANNs have been found to be effective for handling large numbers of input features, making them a viable alternative to SVR.

REFERENCES

- [1] Q. Fu, X. Yang, and Z. Niu, 2014. Bi-level objective model of optimal parking lot recommendation based on parking guidance signs. *Application Research of Computers*, vol. 31, no. 10, pp. 3017–3019, 2014
- [2] Mambo Malaysia, “Idling wastes fuel and money (how much you could have saved?),” Mambo Malaysia, 12-Aug-2021. [Online]. Available: <https://www.mambomalaysia.com/vehicle-idling-wastes-fuel-money/>. [Accessed: 12-Nov-2021].
- [3] Z. Zhao and Y. Zhang, “A comparative study of parking occupancy prediction methods considering parking type and parking scale,” *Journal of Advanced Transportation*, 14-Feb-2020. [Online]. Available: <https://www.hindawi.com/journals/jat/2020/5624586/>. [Accessed: 17-Nov-2021].
- [4] S. Gautam, “People in Kuala Lumpur waste 25 minutes every day looking for parking,” GMP Blog, 06-Apr-2021. [Online]. Available: <https://blog.getmyparking.com/2020/03/04/people-in-kuala-lumpur-waste-25-minutes-every-day-looking-for-parking/>. [Accessed: 07-Nov-2021].
- [5] J. Fan, Q. Hu, and Z. Tang, “Predicting vacant parking space availability: An SVR method with Fruit Fly Optimisation,” *IET Intelligent Transport Systems*, vol. 12, no. 10, pp. 1414–1420, 2018.
- [6] R. Pupale, “Support vector machines(svm) - an overview,” Medium, 11-Feb-2019. [Online]. Available: <https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989>. [Accessed: 12-Nov-2021].
- [7] A. Yadav, “Support Vector Machines(SVM),” Medium, 22-Oct-2018. [Online]. Available: <https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589>. [Accessed: 12-Nov-2021].
- [8] C. Deng, J. Wu, and X. Shao, “Reliability assessment of machining accuracy on support vector machine,” *Intelligent Robotics and Applications*, pp. 669–678, 2008.
- [9] H. Motulsky and A. Christopoulos, *Fitting models to biological data using linear and nonlinear regression: A practical guide to curve fitting*. Oxford University Press, 2004.

REFERENCES

- [10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-Learn: Machine learning in Python,” *The Journal of Machine Learning Research*, 01-Feb-2011. [Online]. Available: <https://dl.acm.org/doi/10.5555/1953048.2078195>. [Accessed: 13-Nov-2021].
- [11] K. Dhiraj, “Top 4 advantages and disadvantages of support vector machine or SVM,” *Medium*, 26-Dec-2020. [Online]. Available: <https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107>. [Accessed: 12-Nov-2021].
- [12] A. Sethi, “Support vector regression in machine learning,” *Analytics Vidhya*, 01-Apr-2020. [Online]. Available: <https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/>. [Accessed: 13-Nov-2021].
- [13] T. Sharp, “An introduction to support vector regression (SVR),” *Medium*, 06-May-2020. [Online]. Available: <https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2>. [Accessed: 13-Nov-2021].
- [14] Javatpoint, “Linear regression in machine learning” *www.javatpoint.com*, 2021. [Online]. Available: <https://www.javatpoint.com/linear-regression-in-machine-learning>. [Accessed: 13-Nov-2021].
- [15] EliteDataScience, “Modern machine learning algorithms: Strengths and weaknesses,” *EliteDataScience*, 08-Jul-2021. [Online]. Available: <https://elitedatascience.com/machine-learning-algorithms>. [Accessed: 13-Nov-2021].
- [16] Y. Wu, L. Li, and L. Li, “Chi-square test neural network: A new binary classifier based on backpropagation neural network,” *arXiv.org*, 04-Sep-2018. [Online]. Available: <http://arxiv.org/abs/1809.01079>. [Accessed: 16-Nov-2021].
- [17] O. N. AL-Allaf, “Fast backpropagation neural network algorithm for reducing convergence time of BPNN image compression,” *ICIMU 2011 : Proceedings of the 5th international Conference on Information Technology & Multimedia*, 2011.

REFERENCES

- [18] A. Al-Masri, “How does back-propagation in Artificial Neural Networks Work?,” *Medium*, 29-Jan-2019. [Online]. Available: <https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7>. [Accessed: 13-Nov-2021].
- [19] A. Aslanargun, M. Mammadov, B. Yazici, and S. Yolacan, “Comparison of Arima, neural networks and hybrid models in time series: Tourist arrival forecasting,” *Journal of Statistical Computation and Simulation*, vol. 77, no. 1, pp. 29–53, 2007.
- [20] N. Bora, “Understanding Arima models for machine learning,” Capital One, 09-Nov-2021. [Online]. Available: <https://www.capitalone.com/tech/machine-learning/understanding-arima-models/>. [Accessed: 28-Sep-2022].
- [21] J. Nyambal and R. Klein, “Automated parking space detection using Convolutional Neural Networks,” *arXiv.org*, 14-Jun-2021. [Online]. Available: <http://arxiv.org/abs/2106.07228>. [Accessed: 15-Nov-2021].
- [22] J. M. Ealn Davan, T. W. Koh, D. L. Tong, and K. L. Tseu, “Anticipation of parking vacancy during peak/non-peak hours using convolutional neural network – yolov3 in university campus,” *2021 International Conference on Green Energy, Computing and Sustainable Technology (GECOST)*, 2021.
- [23] S. Saha, “A comprehensive guide to Convolutional Neural Networks-the eli5 way,” *Medium*, 17-Dec-2018. [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. [Accessed: 15-Nov-2021].
- [24] K. Maladkar, “Overview of convolutional neural network in image classification,” *Analytics India Magazine*, 21-Nov-2020. [Online]. Available: <https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/>. [Accessed: 15-Nov-2021].
- [25] S. Bhuiya, “Disadvantages of CNN Models,” *OpenGenus IQ: Computing Expertise & Legacy*, 19-Jun-2020. [Online]. Available: <https://iq.opengenus.org/disadvantages-of-cnn/>. [Accessed: 16-Nov-2021].
- [26] “Colaboratory,” *Google colab*. [Online]. Available: <https://research.google.com/colaboratory/faq.html>. [Accessed: 06-Aug-2022].

REFERENCES

- [27] M. Driscoll, “Jupyter Notebook: An introduction,” *Real Python*, 28-Jul-2022. [Online]. Available: <https://realpython.com/jupyter-notebook-introduction/>. [Accessed: 07-Aug-2022].
- [28] M. Cordeiro, “Why data scientists should use Jupyter notebooks with moderation ?,” *Medium*, 23-Nov-2021. [Online]. Available: <https://towardsdatascience.com/why-data-scientists-should-use-jupyter-notebooks-with-moderation-808900a69eff>. [Accessed: 07-Aug-2022].
- [29] “The world's most popular data science platform,” Anaconda. [Online]. Available: <https://www.anaconda.com/>. [Accessed: 08-Aug-2022].
- [30] “Anaconda and Jupyter Notebook Setup,” *Machine Learning for iOS Developers*, pp. 287–296, 2020.
- [31] S. Ray, “SVM: Support Vector Machine Algorithm in machine learning,” *Analytics Vidhya*, 26-Aug-2021. [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>. [Accessed: 27-Nov-2021].
- [32] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal Speed and accuracy of object detection,” *arXiv.org*, 23-Apr-2020. [Online]. Available: <https://arxiv.org/abs/2004.10934>. [Accessed: 07-Aug-2022].
- [33] “Yolov4 darknet object detection model,” *Roboflow*. [Online]. Available: <https://models.roboflow.com/object-detection/yolov4>. [Accessed: 07-Aug-2022].
- [34] P. R. L. de Almeida, L. S. Oliveira, A. S. Britto, E. J. Silva, and A. L. Koerich, “PKLot – a robust dataset for parking lot classification,” *Expert Systems with Applications*, vol. 42, no. 11, pp. 4937–4949, 2015.
- [35] M.-R. Hsieh, Y.-L. Lin, and W. H. Hsu, “Drone-based object counting by Spatially Regularized Regional Proposal Network,” *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [36] G. Amato, C. Vairo, C. Gennaro, F. Falchi, and F. Carrara, “CNR PARK+EXTA dataset for visual occupancy detection of parking lots,” *CNR Parking Dataset - Dataset for visual occupancy detection of parking lots*. [Online]. Available: <http://cnrpark.it/>. [Accessed: 07-Aug-2022].

REFERENCES

- [37] BraunGe, “Aerial view car detection for Yolov5,” *Kaggle*, 02-May-2022. [Online]. Available: <https://www.kaggle.com/datasets/braunge/aerial-view-car-detection-for-yolov5>. [Accessed: 07-Aug-2022].
- [38] V. Praharsa, “Yolov4 model architecture,” *OpenGenus IQ: Computing Expertise & Legacy*, 11-Jan-2022. [Online]. Available: <https://iq.opengenus.org/yolov4-model-architecture/>. [Accessed: 26-Aug-2022].
- [39] M. J. Douglass, “Book review: Hands-on machine learning with scikit-learn, Keras, and tensorflow, 2nd edition by Aurélien Géron,” *Physical and Engineering Sciences in Medicine*, vol. 43, no. 3, pp. 1135–1136, 2020.
- [40] J. Hui, “Real-time object detection with Yolo, yolov2 and now yolov3,” *Medium*, 27-Aug-2019. [Online]. Available: <https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088>. [Accessed: 08-Aug-2022].
- [41] J. Hui, “Map (mean average precision) for object detection,” *Medium*, 03-Apr-2019. [Online]. Available: <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>. [Accessed: 08-Aug-2022].
- [42] Techzizou, “Train a custom Yolov4 object detector (using Google Colab),” *Medium*, 12-Mar-2022. [Online]. Available: <https://medium.com/analytics-vidhya/train-a-custom-yolov4-object-detector-using-google-colab-61a659d4868>. [Accessed: 08-Aug-2022].
- [43] AlexeyAB, “Alexeyab/Darknet: Yolov4 / scaled-yolov4 / yolo - neural networks for object detection (windows and linux version of darknet),” *GitHub*. [Online]. Available: <https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects>. [Accessed: 08-Aug-2022].
- [44] “OpenCV library, author at opencv,” *OpenCV*. [Online]. Available: <https://opencv.org/author/opencv/>. [Accessed: 08-Aug-2022].
- [45] “Abstract,” *NVIDIA Documentation Center*. [Online]. Available: <https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html#:~:text=NVIDIA%20CUDA%20AE%20Deep,Matrix%20multiplication>. [Accessed: 08-Aug-2022].

REFERENCES

- [46] J. Nelson and J. Solawetz, "Responding to the controversy about yolov5," Roboflow Blog, 04-Mar-2021. [Online]. Available: <https://blog.roboflow.com/yolov4-versus-yolov5/>. [Accessed: 19-Aug-2022].
- [47] Techzizou, "Train a custom yolov4 object detector on Windows," *Medium*, 05-Oct-2021. [Online]. Available: <https://medium.com/geekculture/train-a-custom-yolov4-object-detector-on-windows-fe5332b0ca95>. [Accessed: 20-Aug-2022].
- [48] S. Allwright, "RMSE vs MAPE, which is the best regression metric?," *Stephen Allwright*, 16-Aug-2022. [Online]. Available: <https://stephenallwright.com/rmse-vs-mape/>. [Accessed: 23-Aug-2022].
- [49] D. A. Swanson, "On the relationship among values of the same summary measure of error," *Review of Economics & Finance*, 01-Jan-1970. [Online]. Available: <https://ideas.repec.org/a/bap/journal/150301.html>. [Accessed: 23-Aug-2022].
- [50] V. Paidi, J. Håkansson, H. Fleyeh, and R. G. Nyberg, "Directory of open access journals," *Sustainability*, 01-Mar-2022. [Online]. Available: <https://doaj.org/article/d57a4c9ecc4947739c082ebdbe32a0bb>. [Accessed: 14-Feb-2023].
- [51] "Python guis for humans," PySimpleGUI. [Online]. Available: <https://www.pysimplegui.org/en/latest/#install>. [Accessed: 03-Apr-2023].
- [52] B. Li, "Random search plus: A more effective random search for machine learning hyperparameters optimization," *TRACE*, Oct-2020. [Online]. Available: https://trace.tennessee.edu/utk_gradthes/5849/. [Accessed: 13-Apr-2023].
- [53] "Parameter Optimization Loop Start," KNIME Community Hub. [Online]. Available: <https://hub.knime.com/knime/extensions/org.knime.features.optimization/latest/org.knime.optimization.internal.node.parameter.loopstart.LoopStartParOptNodeFactory>. [Accessed: 13-Apr-2023].
- [54] S. Patel, "Chapter 2 : SVM (Support Vector Machine) - theory," *Medium*, 04-May-2017. [Online]. Available: <https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72>. [Accessed: 18-Apr-2023].

REFERENCES

- [55] S. Dobilas, "Support vector regression (SVR)-one of the most flexible yet robust prediction algorithms," Medium, 12-Feb-2022. [Online]. Available: <https://towardsdatascience.com/support-vector-regression-svr-one-of-the-most-flexible-yet-robust-prediction-algorithms-4d25fbdaca60#:~:text=SVR%20has%20an%20additional%20tunable,not%20penalized%20by%20the%20algorithm.> [Accessed: 18-Apr-2023].
- [56] AlexeyAB, "CFG parameters in the different layers," GitHub, 09-Jul-2020. [Online]. Available: [https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-different-layers.](https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-different-layers) [Accessed: 19-Apr-2023].
- [57] AlexeyAB, "Cudnn error while training -map · issue #7153 · Alexeyab/Darknet," GitHub. [Online]. Available: [https://github.com/AlexeyAB/darknet/issues/7153.](https://github.com/AlexeyAB/darknet/issues/7153) [Accessed: 19-Apr-2023].

APPENDIX A

ISMSI 2023 Acceptance Notification of Full Paper

2023 7th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence (ISMSI 2023)

ISMSI 2023 Acceptance Notification of Full Paper
April 23-24, 2023 | Kuala Lumpur, Malaysia
<http://www.ismsi.org>

Organized by



Technically Sponsored by



Dear WEI JUN, W.J., LEE, MOHAMMAD DALVI, M.D., ESFAHANI, KWAN LEE, K.L., TSEU⁺,

Paper ID: MS029

Title of Full Paper: Predicting Open Parking Space using Deep Learning and Support Vector Regression

Congratulations! We're pleased to inform you that your full paper mentioned above has passed the blind review of the conference technical committees and has been accepted for both publication and oral presentation at the conference 2023 7th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence (ISMSI 2023) will be held in Kuala Lumpur, Malaysia during April 23-24, 2023.

Your paper will be published in **ISMSI 2023 Conference Proceedings** by ACM (ISBN: 978-1-4503-9992-0), which will be indexed by **Ei Compendex, Scopus**, and submitted to be reviewed by Thomson Reuters (ISI Web of Science).

Registration Instructions

Please follow the six steps below to guarantee your registration will be completed on time.

1. Revise your paper according to the review comments in the attachment carefully.
2. Prepare your final revised paper by following the template.
http://www.ismsi.org/acm_template.docx
3. Download and complete the Registration Form.
<http://www.ismsi.org/reg.docx>
4. Finish the payment of registration fee (Payment information can be found in Registration form)
5. Send your **Final Revised Paper, Filled Registration Form (Both .doc and .pdf format), Scanned Payment Proof** to us at sub@ismsi.org by Registration Deadline (Before **April 10, 2023**)

Note:

2023 7th International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence (ISMSI 2023)

- If you pay by Credit Card through the online payment system, please fill your confirmation number in the registration form after your make the payment.
- If you pay by bank transfer, please scan the payment slip as the payment proof for checking.
- **Copyright and final paper needs to be uploaded to press system by authors after conference. Please wait for announcement.**

If you have any problem, please feel free to contact us via sub@ismsi.org for assistance. For the most updated information about the conference, please check the conference website at <http://www.ismsi.org>. The conference schedule will be available on the conference website in early April, 2023.

Yours sincerely,
ISMSI 2023 Organizing Committee
December 30, 2023



APPENDIX

YOLOv4 configuration files

Content for the data file named obj (obj.data):

```
classes = 2
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = /mydrive/yolov4/training
```

Content for the names file named obj (obj.names):

```
space-empty
space-occupied
```

Content for the names file named process (process.py):

```
import glob, os
# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))
current_dir = 'data/obj'

# Percentage of images to be used for the test set
percentage_test = 10;

# Create and/or truncate train.txt and test.txt
file_train = open('data/train.txt', 'w')
file_test = open('data/test.txt', 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))

    if counter == index_test:
        counter = 1
        file_test.write("data/obj" + "/" + title + '.jpg' + "\n")
    else:
        file_train.write("data/obj" + "/" + title + '.jpg' + "\n")
        counter = counter + 1
```

Code used to connect to Google Drive:

```
#mount drive
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')

# Creating a symbolic link allows accessing /content/gdrive/My\ Drive/ through the shorter
path /mydrive.
```

APPENDIX

```
!ln -s /content/gdrive/My Drive/ /mydrive  
  
#Navigate to /mydrive/yolov4  
%cd /mydrive/yolov4
```

Code used to download the weights file for the YOLOv4 pre-trained model:

```
!wget  
https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.  
conv.137
```

YOLOv4 test with "ignore_thresh" value of 0.9 and "iou_normalizer" value of 0.5:

```
159 conv 1024 3 x 3/ 1 15 x 15 x 512 -> 15 x 15 x1024 2.123 BF  
160 conv 21 1 x 1/ 1 15 x 15 x1024 -> 15 x 15 x 21 0.010 BF  
161 yolo  
[yolo] params: iou loss: giou (1), iou_norm: 0.50, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05  
nms_kind: greedynms (1), beta = 0.600000  
Total BFLOPS 79.310  
avg_outputs = 652247  
Allocate additional workspace_size = 134.22 MB  
Loading weights from /mydrive/yolov4/training/yolov4-custom_final.weights...  
seen 64, trained: 384 K-images (6 Kilo-batches_64)  
Done! Loaded 162 layers from weights-file  
  
calculation mAP (mean average precision)...  
Detection layer: 139 - type = 28  
Detection layer: 150 - type = 28  
Detection layer: 161 - type = 28  
1216  
detections_count = 332350, unique_truth_count = 71156  
class_id = 0, name = space-empty, ap = 23.40% (TP = 26480, FP = 85558)  
class_id = 1, name = space-occupied, ap = 29.85% (TP = 25686, FP = 77781)  
  
for conf_thresh = 0.25, precision = 0.24, recall = 0.73, F1-score = 0.36  
for conf_thresh = 0.25, TP = 52166, FP = 163339, FN = 18990, average IoU = 22.44 %  
  
IoU threshold = 50 %, used Area-Under-Curve for each unique Recall  
mean average precision (mAP@0.50) = 0.266239, or 26.62 %  
Total Detection Time: 56 Seconds  
  
Set -points flag:  
`-points 101` for MS COCO  
`-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)  
`-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

Prediction model preprocessing (text label file to CSV file)

Code that read the text files:

```
#store parking datetime in list according to weather status  
cloudy=[]  
rainy=[]  
sunny=[]  
  
with open('C:\\Users\\weiju\\A parking data\\sample\\cloudy.txt') as f:  
    Lines = f.readlines()  
    for line in Lines:  
        cloudy.append(line.strip())  
f.close()  
with open('C:\\Users\\weiju\\A parking data\\sample\\rainy.txt') as f:  
    Lines = f.readlines()  
    for line in Lines:  
        rainy.append(line.strip())
```

APPENDIX

```
f.close()
with open('C:\\Users\\weiju\\A parking data\\sample\\sunny.txt') as f:
    Lines = f.readlines()
    for line in Lines:
        sunny.append(line.strip())
f.close()
```

Code that creating a new data frame to store the data(parking features) extracted from all the files:

```
#create dataframe to store parking features
parkingDataset = ({
    'Location':[],
    'Occupied' :[],
    'Total parking':[],
    'Date':[],
    'Time':[],
    'Weather':[]
})
df = pd.DataFrame(parkingDataset)
print(df)
```

Code that prints out dataframe:

```
print(df)
```

Code that reading label XML file:

```
#store labelfile(xml) that are null in the dataset(original)
loss=[]
with open('C:\\Users\\weiju\\A parking data\\sample\\output.txt') as f:
    Lines = f.readlines()
    for line in Lines:
        loss.append(line.strip())
f.close()

#save label file(xml) information (datetime, occupancy and empty parking) to
dictionary(my_dict)
my_dict={ }
for filename in loss:
    tempFor1=[] #occupy
    temp=[] #empty
    file = minidom.parse(os.path.join("C:\\Users\\weiju\\A parking
data\\missing_xml\\",filename))
    models = file.getElementsByTagName('space')
    for elem in models:
        word=elem.attributes['occupied'].value
        if word=='1':
            tempFor1.append(word)
        else:
            temp.append(word)
```

APPENDIX

```
my_dict[os.path.basename(filename).split('.')[0]] =  
[len(tempFor1),len(temp)+len(tempFor1)]  
print(len(my_dict))
```

Code that appends data into data frame:

```
#retrive data from every labelled file and store it according to pre-create dataframe  
path = "C:/Users/weiju/A parking data/PKLot/PKLot_dataset/"  
for filename in glob.glob(os.path.join(path, '*.txt')):  
    with open(os.path.join(os.getcwd(), filename), 'r') as f: # open in readonly mode  
        fileName=os.path.basename(filename).split('.')[0]  
        location='PKlots'  
        date= datetime.strptime(fileName[:10],"%Y-%m-%d").strftime('%d/%m/%Y')  
        time= datetime.strptime(fileName[11:], '%H_%M_%S').time()  
        temp=fileName.strip()  
  
        if fileName in cloudy:  
            weather=0  
        elif fileName in rainy:  
            weather=1  
        elif fileName in sunny:  
            weather=2  
        else :  
            weather=3  
  
        temp=[]#for empty parking  
        tempFor1=[]#for occupied parking  
        for line in f:  
            words = line.split()  
            if words[0]=='1':  
                tempFor1.append(words[0])  
            else:  
                temp.append(words[0])  
        empty=len(temp)  
        occupied=len(tempFor1)  
        totalParking=len(tempFor1)+len(temp)  
  
        if fileName in my_dict.keys():  
            value=my_dict[fileName]  
            occupied=value[0]  
            totalParking=value[1]  
        new_row = {'Location':location, 'Occupied':occupied, 'Total parking':totalParking,  
        'Date':date,'Time':time,'Weather':weather}  
        df = df.append(new_row, ignore_index=True)
```

Code that saves data frame as csv:

```
#save dataframe as csv  
path= 'C:\\Users\\weiju\\A parking data\\'  
df.to_csv(path+'pklot_weatherNcomplete.csv', index=False)
```

APPENDIX

Prediction model preprocessing

Code used to import the needed libraries:

```
# import python libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import joblib
import random
from datetime import datetime
from time import time
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_absolute_percentage_error
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV,
StratifiedKFold, cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from scipy.stats import uniform, randint
```

Code that used to modify and create features for dataset:

```
pk["DateTime"] = pk["Date"] + ' ' + pk["Time"]
#create new columns to store empty space
pk['Empty'] = (pk['Total parking'] - pk['Occupied'])
#drop unneeded attribute- Occupied, location, duplicate data and time
pk.drop('Location', inplace=True, axis=1)
pk.drop('Occupied', inplace=True, axis=1)
pk.drop('Date', inplace=True, axis=1)
pk.drop('Time', inplace=True, axis=1)
#convert object to DateTime
pk['DateTime'] = pd.to_datetime(pk['DateTime'], format = '%d/%m/%Y %H:%M:%S',
errors = 'coerce')
#convert float to integer
pk['Empty'] = pk['Empty'].astype(np.int64)
pk['Total parking'] = pk['Total parking'].astype(np.int64)
pk['Weather'] = pk['Weather'].astype(np.int64)
```

Code that read the dataset into a data frame:

```
#read dataset (parking lot dataset with null fixed and weather included)
pk = pd.read_csv("pklot_weatherNcomplete.csv")
pk
```

Code that displays description of all attributes:

```
# Quick description all attributes in pk after process
pk.info()
```

Code that checks DateTime redundancy:

APPENDIX

```
#check redundancy DateTime
pk['DateTime']. duplicated().value_counts()
```

Code that check statistics of all numerical attribute:

```
# The statistics of all numerical attributes in pk
pk.describe()
```

Code that plots the histogram of each numerical attribute:

```
# Plot the histogram of each numerical attribute in pk(occupied, total parking, weather and
DateTime)
pk.hist(bins=50, figsize=(20,15))
plt.show()
```

Code that display correlation matrix:

```
# The correlation matrix of pk
# no value with nearly to 1
corr_matrix=pk.corr()
corr_matrix
```

Code that counts number for types of weather:

```
pk['Weather'].value_counts()
```

Code that plots graph according to weather types:

```
#visualization
pk_filter_large = pk[(pk['Weather']==0)]
pk_filter_large.plot(x='DateTime',y='Empty', color='orange')
plt.title('Cloudy')
plt.show()
pk_filter_large = pk[(pk['Weather']==1)]
pk_filter_large.plot(x='DateTime',y='Empty', color='orange')
plt.title('Rainy')
plt.show()
pk_filter_large = pk[(pk['Weather']==2)]
pk_filter_large.plot(x='DateTime',y='Empty', color='orange')
plt.title('Sunny')
plt.show()
```

Prediction model training

Code that perform data pre-processing:

```
#load dataset (parking lot dataset with null fixed and weather included)
pk = pd.read_csv("pklot_weatherNcomplete.csv")
#combine date and time in single column
pk["DateTime"] = pk["Date"] + ' ' + pk["Time"]
#convert object to DateTime
pk['DateTime'] = pd.to_datetime(pk['DateTime'], format = '%d/%m/%Y %H:%M:%S',
errors = 'coerce')
```

APPENDIX

```
pk['DateTime_year'] = pk['DateTime'].dt.year
pk['DateTime_month'] = pk['DateTime'].dt.month
pk['DateTime_week'] = pk['DateTime'].dt.isocalendar().week
pk['DateTime_day'] = pk['DateTime'].dt.day
pk['DateTime_hour'] = pk['DateTime'].dt.hour
pk['DateTime_minute'] = pk['DateTime'].dt.minute
pk['DateTime_dayofweek'] = pk['DateTime'].dt.dayofweek
#create new columns to store empty space
pk['Empty'] = (pk['Total parking'] - pk['Occupied'])
#drop unneeded attribute- Occupied, location, duplicate data and time
pk.drop('Location', inplace=True, axis=1)
pk.drop('Date', inplace=True, axis=1)
pk.drop('Time', inplace=True, axis=1)
pk.drop('DateTime', inplace=True, axis=1)
pk.drop('Occupied', inplace=True, axis=1)
#convert float to integer
pk['Empty'] = pk['Empty'].astype(np.int64)
pk['Total parking'] = pk['Total parking'].astype(np.int64)
pk['Weather'] = pk['Weather'].astype(np.int64)
pk['DateTime_week'] = pk['DateTime_week'].astype(np.int64)
#filter out parking lot type with the parking space size
pk_filter_large = pk[(pk['Total parking']!=100)]
pk_filter_large.info()
```

Code that split dataset and pre-processing:

```
# Split the data set into the input matrix and output vector
X = pk_filter_large.drop('Empty', axis = 1)
y = pk_filter_large['Empty']

## Split the dataset into training and testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=3)
X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,test_size=0.15/0.85,random_s
tate=2)

# split the weather columns to be apply one-hot encoding later
X_train_num = X_train.drop('Weather', axis = 1)
X_train_weather = X_train['Weather']

# # Standardize numeric data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(copy=False)
scaler.fit(X_train_num)
X_train_num_tr = scaler.transform(X_train_num)

# One-hot encode weather column
from sklearn.preprocessing import LabelBinarizer
lb_encoder = LabelBinarizer()
lb_encoder.fit(X_train_weather)
X_train_weather_tr = lb_encoder.transform(X_train_weather)

# combine the transformed numerical and weather set ('Cloudy','Rainy','Sunny')
X_train_tr = np.hstack([X_train_num_tr, X_train_weather_tr])
y_train = y_train.values ; #convert to numpy array
```

APPENDIX

```
y_val = y_val.values  
y_test = y_test.values
```

Code that shows shape of original dataset (train):

```
print('Shape of original dataset, train:', pk.shape)  
print('x: shape=', X_train_tr.shape, 'type=', type(X_train_tr))  
print('y: shape=', y_train.shape, 'type=', type(y_train))
```

Code for “show10results” function:

```
#show10results function  
def show10results(y_train, y_pred):  
    print('Result for the 10 random samples:')  
    selected = np.random.randint(0, len(y_train), 10)  
    for i in selected:  
        print('actual = {:7.0f} pred ={:7.0f}'.format(y_train[i], y_pred[i]))
```

Function code of MSE, RMSE, MAE and MAPE:

```
# Show result of 10 random samples  
show10results(y_train, y_pred)  
# Show the MSE RMSE MAE and MAPE  
regr_mse = mean_squared_error(y_train , y_pred)  
regr_rmse = np.sqrt(regr_mse)  
final_mae = mean_absolute_error(y_train , y_pred)  
final_mape = mean_absolute_percentage_error(y_train, y_pred)  
print('MSE =', regr_mse)  
print('RMSE =', regr_rmse)  
print('MAE=', final_mae)  
print('MAPE=', final_mape)
```

Fine tune the prediction model

Parameter grid named “param_grid_svr2”:

```
param_grid_svr2 = {'C': [6000,7000,8000,9000,10000],  
                  'degree':[1,3],  
                  'gamma': ['scale', 'auto'],  
                  'epsilon':[6,7,8],  
                  'kernel': ['rbf','poly']}
```

Parameter grid named “param_random_svr”:

```
param_random_svr = {'C': uniform(loc=10, scale=990).rvs(20),  
                   'gamma': ['scale', 'auto'],  
                   'degree':[1,3,5],  
                   'kernel': ['rbf','poly'],  
                   'epsilon': uniform(loc=5, scale=5).rvs(10)}
```


APPENDIX

Function code that returns the model after a grid search or random search has been performed:

```
def perform_search_svr(grid_random, algo, parameter):
    k = StratifiedKFold(n_splits=5)
    #identify grid or random search
    if grid_random=='Grid':
        search_obj=GridSearchCV(algo, parameter, refit = True, verbose = 3, cv=k)
    elif grid_random=="Random":
        search_obj=RandomizedSearchCV(estimator=algo,
                                      param_distributions=parameter,
                                      n_iter=100,
                                      scoring='neg_mean_squared_error',
                                      cv=k,
                                      n_jobs=-1
                                      )
    print('Performing ', grid_random, ' Search...', end = ")
    t0 = time()
    # perform grid search
    search_obj.fit(X_train_tr, y_train)
    print('done')
    print('time took:', time() - t0)
    return search_obj
```

Function code that returns the model after a random search plus has been performed:

```
def perform_search_randomPlus(parameter):
    # the number of iterations for random search plus
    num_iterations = len(parameter['C'])
    print('Performing Random Search Plus...\nFor',num_iterations, end = ' times.....\n')
    t0 = time()
    model = SVR()
    model.fit(X_train_tr, y_train)

    # Evaluate the model's performance on a validation set
    y_pred = model.predict(X_val_tr)
    mse = mean_squared_error(y_val, y_pred)
    best_mse = mse
    print('\nNo_0 MSE is ',mse)

    for i in range(num_iterations):
        new_para={
            'kernel': parameter['kernel'],
            'C': (parameter['C'][i], random.choice(parameter['C'])),
            'gamma': parameter['gamma'],
            'epsilon': parameter['epsilon']
        }

    t1 = time()
    k = StratifiedKFold(n_splits=5)
    search_obj=RandomizedSearchCV(estimator=model,
                                  param_distributions=new_para,
                                  n_iter=100,
                                  scoring='neg_mean_squared_error',
```

APPENDIX

```
        cv=k,
        n_jobs=-1)

search_obj.fit(X_train_tr, y_train)
y_pred = search_obj.predict(X_val_tr)
mse = mean_squared_error(y_val, y_pred)
print('No_',i+1,'MSE is ',mse, ' -----time took:', time() - t1)

if mse < best_mse:
    best_params = search_obj.best_params_
    best_mse = mse

# Train a final model with the best hyperparameters on the entire training set
final_model = SVR(kernel=best_params['kernel'], C=best_params['C'],
gamma=best_params['gamma'], epsilon=best_params['epsilon'])
print('\nBest parameters-->',best_params)
print('\nBest MSE-->',best_mse)
final_model.fit(X_train_tr, y_train)
print('\ndone')
print('time took:', time() - t0)
return final_model
```

Function code that returns the model after a parameter optimization loop has been performed:

```
def perform_search_parameterOpt_loop(iteration, parameter):
    print('Performing Random Search Plus...\nFor',iteration, end = ' times.....\n')
    t0 = time()
    model = SVR()
    model.fit(X_train_tr, y_train)

    # Evaluate the model's performance on a validation set
    y_pred = model.predict(X_val_tr)
    mse = mean_squared_error(y_val, y_pred)
    best_mse = mse
    print('\nNo_0 MSE is ',mse)

    # Set the number of iterations for random search plus
    num_iterations = iteration

    for i in range(num_iterations):
        t1 = time()
        params = {
            'kernel': random.choice(parameter['kernel']),
            'C': random.choice(parameter['C']),
            'gamma': random.choice(parameter['gamma']),
            'epsilon': random.choice(parameter['epsilon'])
        }

        model = SVR(kernel=params['kernel'], C=params['C'], gamma=params['gamma'],
epsilon=params['epsilon'])
        model.fit(X_train_tr, y_train)
```

APPENDIX

```
y_pred = model.predict(X_val_tr)
mse = mean_squared_error(y_val, y_pred)
print('No_',i+1,'MSE is ',mse,' -----time took:', time() - t1)

if mse < best_mse:
    best_params = params
    best_mse = mse

# Train a final model with the best hyperparameters on the entire training set
final_model = SVR(kernel=best_params['kernel'], C=best_params['C'],
gamma=best_params['gamma'], epsilon=best_params['epsilon'])
print('\nBest parameters-->',best_params)
print('\nBest MSE-->',best_mse)
final_model.fit(X_train_tr, y_train)
print('\ndone')
print('time took:', time() - t0)
return final_model
```

Predicting model performance result

Code that converts YOLOv4. weight file to PyTorch .pth file:

```
from tool import darknet2pytorch
import torch

# load weights from darknet format
model = darknet2pytorch.Darknet('cfg/yolov4-custom.cfg', inference=True)
model.load_weights('yolov4-custom_best.weights')

# save weights to pytorch format
torch.save(model.state_dict(), 'save/yolov4-pytorch.pth')

# reload weights from pytorch format
model_pt = darknet2pytorch.Darknet('cfg/yolov4-custom.cfg', inference=True)
model_pt.load_state_dict(torch.load('save/yolov4-pytorch.pth'))
```

Code that pre-processing validation set:

```
# split the weather columns to be apply one-hot encoding later
X_val_num = X_val.drop('Weather', axis = 1)
X_val_weather = X_val['Weather']
## Standardize numeric data
X_val_num_tr = scaler.transform(X_val_num)
# One-hot encode weather column
X_val_weather_tr = lb_encoder.transform(X_val_weather)
# combine the transformed numerical and weather set ('Cloudy','Rainy','Sunny')
X_val_tr = np.hstack([X_val_num_tr, X_val_weather_tr])
X_val_tr.shape
```

Code that pre-processing test set:

```
# split the weather columns to be apply one-hot encoding later
```

APPENDIX

```
X_test_num = X_test.drop('Weather', axis = 1)
X_test_weather = X_test['Weather']
## Standardize numeric data
X_test_num_tr = scaler.transform(X_test_num)
# One-hot encode weather column
X_test_weather_tr = lb_encoder.transform(X_test_weather)
# combine the transformed numerical and weather set ('Cloudy','Rainy','Sunny')
X_test_tr = np.hstack([X_test_num_tr, X_test_weather_tr])
X_test_tr.shape
```

Function code that shows the evaluation result on train, validation and test sets:

```
def show_3result(model,modelName):
    model_tr_prd=model.predict(X_train_tr)
    final_mse=mean_squared_error(y_train, model_tr_prd)
    final_rmse = np.sqrt(final_mse)
    final_mae = mean_absolute_error(y_train,model_tr_prd)
    final_mape = mean_absolute_percentage_error(y_train,model_tr_prd)
    print('<--- ',modelName,' train --->')
    print('MSE =',final_mse)
    print('RMSE=', final_rmse)
    print('MAE =', final_mae)
    print('MAPE=', "{:.4e}".format(final_mape))

    model_val_prd=model.predict(X_val_tr)
    final_mse=mean_squared_error(y_val, model_val_prd)
    final_rmse = np.sqrt(final_mse)
    final_mae = mean_absolute_error(y_val,model_val_prd)
    final_mape = mean_absolute_percentage_error(y_val,model_val_prd)
    print("\n<--- ',modelName,' validate --->')
    print('MSE =',final_mse)
    print('RMSE=', final_rmse)
    print('MAE =', final_mae)
    print('MAPE=', "{:.4e}".format(final_mape))

    model_test_prd=model.predict(X_test_tr)
    final_mse=mean_squared_error(y_test, model_test_prd)
    final_rmse = np.sqrt(final_mse)
    final_mae = mean_absolute_error(y_test,model_test_prd)
    final_mape = mean_absolute_percentage_error(y_test,model_test_prd)
    print("\n<--- ',modelName,' test --->')
    print('MSE =',final_mse)
    print('RMSE=', final_rmse)
    print('MAE =', final_mae)
    print('MAPE=', "{:.4e}".format(final_mape))
```

FINAL YEAR PROJECT WEEKLY REPORT

FINAL YEAR PROJECT WEEKLY REPORT

(Project 1)

Trimester, Year: T3,Y3	Study week no.: 2
Student Name & ID: Lee Wei Jun 19ACB03389	
Supervisor: Tseu Kwan Lee	
Project Title: Predicting Open Space Parking Vacancies using Machine Learning	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

- Review on FYP1 writing report
- Reduced plagiarism percentage
- Finalise figures and equations

2. WORK TO BE DONE

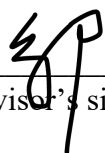
- Study parameter for YOLOv4 configuration file and SVR algorithm
- Update command code for YOLOv4
- Read related article and add to the literature review

3. PROBLEMS ENCOUNTERED


- Previous YOLOv4 command code shown error after run

4. SELF EVALUATION OF THE PROGRESS

A bit late on the work, need to catch up.



Supervisor's signature



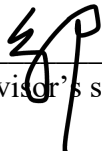
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT


(Project I)

Trimester, Year: T3, Y3	Study week no.: 4
Student Name & ID: Lee Wei Jun 19ACB03389	
Supervisor: Tseu Kwan Lee	
Project Title: Predicting Open Space Parking Vacancies using Machine Learning	

<p>1. WORK DONE [Please write the details of the work done in the last fortnight.]</p> <ul style="list-style-type: none"> ● Finished reading related article and has added in the literature review ● Code fixed ● Studied parameter for YOLOv4 configuration file and SVR algorithm
<p>2. WORK TO BE DONE</p> <ul style="list-style-type: none"> ● Update report format to FYP2 ● Clean dataset for YOLOv4 ● Start to train YOLOv4 with different parameters ● Perform fine tuning on SVR model ● Try and test on new fine tuning ● Rent PC in FYP lab to perform machine learning training
<p>3. PROBLEMS ENCOUNTERED</p> <ul style="list-style-type: none"> ● Still a bit confuse about the parameters for YOLOv4 and SVR algorithm
<p>4. SELF EVALUATION OF THE PROGRESS</p> <p>Should start performing fine tuning test on different combination of parameter, in order to have a deeper understanding about the parameters</p>



 Supervisor's signature




 Student's signature

FINAL YEAR PROJECT WEEKLY REPORT


(Project I)

Trimester, Year: T3,Y3	Study week no.: 6
Student Name & ID: Lee Wei Jun 19ACB03389	
Supervisor: Tseu Kwan Lee	
Project Title: Predicting Open Space Parking Vacancies using Machine Learning	

<p>1. WORK DONE [Please write the details of the work done in the last fortnight.]</p> <ul style="list-style-type: none"> ● Update report format to FYP2 ● Clean dataset for YOLOv4 ● Trained YOLOv4 on different parameters ● Perform several fine tuning test on SVR model ● Rented a PC in FYP lab
<p>2. WORK TO BE DONE</p> <ul style="list-style-type: none"> ● Test more combination of parameters for YOLOv4 ● Test more combination of parameters for SVR model ● Finalise the new fine tuning technique
<p>3. PROBLEMS ENCOUNTERED</p> <ul style="list-style-type: none"> ● PC in FYP lab shut down for no reasons, need to setup the environment again ● Training custom YOLOv4 using Google Colab is time consuming because it has a usage limit with every 5-8 hours of usage, it will follow with 12-16 hours cool down time
<p>4. SELF EVALUATION OF THE PROGRESS</p> <p>Keep up</p>



 Supervisor's signature



 Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project I)

Trimester, Year: T3,Y3	Study week no.: 8
Student Name & ID: Lee Wei Jun 19ACB03389	
Supervisor: Tseu Kwan Lee	
Project Title: Predicting Open Space Parking Vacancies using Machine Learning	

<p>1. WORK DONE [Please write the details of the work done in the last fortnight.]</p> <ul style="list-style-type: none"> ● Finalise the new fine tuning technique
<p>2. WORK TO BE DONE</p> <ul style="list-style-type: none"> ● Test more combination of parameters for YOLOv4 ● Test more combination of parameters for SVR model ● Develop a GUI for parking vacancy prediction
<p>3. PROBLEMS ENCOUNTERED</p> <ul style="list-style-type: none"> ● Overfitting issue for SVR model
<p>4. SELF EVALUATION OF THE PROGRESS</p> <p>Should be continue on the report.</p>

Supervisor's signature

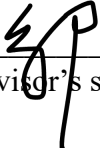
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT


(Project I)

Trimester, Year: T3,Y3	Study week no.: 10
Student Name & ID: Lee Wei Jun 19ACB03389	
Supervisor: Tseu Kwan Lee	
Project Title: Predicting Open Space Parking Vacancies using Machine Learning	

<p>1. WORK DONE [Please write the details of the work done in the last fortnight.]</p> <ul style="list-style-type: none"> ● Developed a GUI for parking vacancy prediction ● Finalize the coding for the machine learning
<p>2. WORK TO BE DONE</p> <ul style="list-style-type: none"> ● Test more combination of parameters for YOLOv4 ● Test more combination of parameters for SVR model ● Continue on report writing
<p>3. PROBLEMS ENCOUNTERED</p> <ul style="list-style-type: none"> ● Mistake found on the coding for the machine learning
<p>4. SELF EVALUATION OF THE PROGRESS</p> <p>Should put more effort on report writing as the deadline is coming soon</p>



 Supervisor's signature



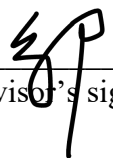
 Student's signature

FINAL YEAR PROJECT WEEKLY REPORT


(Project I)

Trimester, Year: T3,Y3	Study week no.: 11
Student Name & ID: Lee Wei Jun 19ACB03389	
Supervisor: Tseu Kwan Lee	
Project Title: Predicting Open Space Parking Vacancies using Machine Learning	

<p>1. WORK DONE [Please write the details of the work done in the last fortnight.]</p> <ul style="list-style-type: none"> ● testing more combination of parameters for YOLOv4 ● testing more combination of parameters for SVR model
<p>1. WORK TO BE DONE</p> <ul style="list-style-type: none"> ● Evaluate the performance of YOLOv4 and SVR model ● Continue on report writing
<p>3. PROBLEMS ENCOUNTERED</p> <ul style="list-style-type: none"> ● Cannot perform remote control for PC in FYP lab for a day because of electricity problem in university (lightning strike cause short circuit)
<p>4. SELF EVALUATION OF THE PROGRESS</p> <p>Should stop on fine tuning the YOLOv4 and SVR model. Start evaluating the test result.</p>



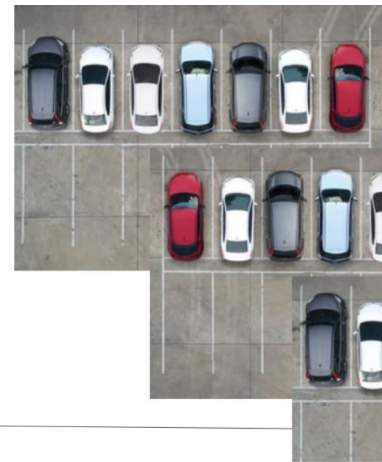
 Supervisor's signature



 Student's signature

POSTER

PREDICTING OPEN SPACE PARKING VACANCIES USING MACHINE LEARNING



INTRODUCTION

This project aims to propose a parking prediction model to overcome the vehicle parking issue. Predictable parking results in lesser idling and thereby reduces the vehicle's gaseous emissions.

OBJECTIVE

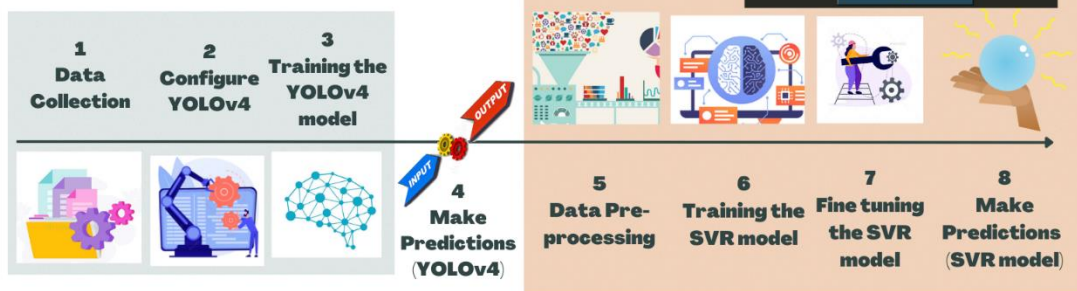
1. To investigate machine learning algorithms and techniques used in the existing studies related to parking prediction.
2. To propose machine learning algorithms concerned with more noises to predict the parking space, like weather and nearby parking lot conditions.
3. To build the model using the proposed support vector regression (SVR) algorithm to predict the available parking space.
4. To test the proposed model's performance by using MSE, RMSE, MAE, and MAPE performance metrics
5. To optimize the performance of an SVR model by employed multiple hyperparameter tuning techniques

PROPOSED METHOD

1. Develop Custom YOLOv4 to make object detection on empty and occupied parking spaces.
2. Develop parking vacancies prediction model using Support Vector Regression (SVR)

WHY PROPOSED MODEL IN THIS PROJECT IS WORABLE?

Previous studies did not concern the weather to predict the empty parking space and there are no existing studies found to develop the prediction model using the dataset generated by the object detection model from the images.



CONCLUSION

An open space parking vacancies prediction model allows people to predict the number of parking vacancies. After making a prediction, searching and waiting for an available parking space can be avoided. As a result, there will be less time wasted and less environmental contamination.

-Currently with an interactive GUI for end user.

Project Developer: Lee Wei Jun

Project Supervisor: Ms Tseu Kwan Lee



PLAGIARISM CHECK RESULT

Turnitin Originality Report

FYP2_report_v1_WeiJun by weijun lee

From Report (FYP2)

Processed on 21-Apr-2023 19:47 +08
ID: 2071219998
Word Count: 19919

Similarity Index		Similarity by Source	
10%		Internet Sources:	6%
		Publications:	7%
		Student Papers:	N/A

sources:

- 1% match (Internet from 14-Dec-2021)
<https://techzizou.com/train-a-custom-yolov4-detector-using-google-colab-tutorial-for-beginners/>
- 1% match (Junkai Fan, Qian Hu, Zhenzhou Tang. "Predicting vacant parking space availability: an SVR method with fruit fly optimisation", IET Intelligent Transport Systems, 2018)
[Junkai Fan, Qian Hu, Zhenzhou Tang. "Predicting vacant parking space availability: an SVR method with fruit fly optimisation", IET Intelligent Transport Systems, 2018](#)
- 1% match (Internet from 07-Oct-2022)
<https://www.hindawi.com/journals/jat/2020/5624586/>
- < 1% match (Junkai Fan, Qian Hu, Zhenzhou Tang. "Predicting vacant parking space availability: an SVR method with fruit fly optimisation", IET Intelligent Transport Systems, 2018)
[Junkai Fan, Qian Hu, Zhenzhou Tang. "Predicting vacant parking space availability: an SVR method with fruit fly optimisation", IET Intelligent Transport Systems, 2018](#)
- < 1% match (Jasvina Mae AP Ealn Davan, Tieng Wei Koh, Dong Ling Tong, Kwan Lee Tseu. "Anticipation of Parking Vacancy During Peak/Non-peak Hours using Convolutional Neural Network – YOLOv3 in University Campus", 2021 International Conference on Green Energy, Computing and Sustainable Technology (GECOST), 2021)
[Jasvina Mae AP Ealn Davan, Tieng Wei Koh, Dong Ling Tong, Kwan Lee Tseu. "Anticipation of Parking Vacancy During Peak/Non-peak Hours using Convolutional Neural Network – YOLOv3 in University Campus", 2021 International Conference on Green Energy, Computing and Sustainable Technology \(GECOST\), 2021](#)
- < 1% match (Internet from 13-Aug-2022)
https://rivan-aditya.github.io/MuBlog/data%20viz/ml/2020/10/23/melb_regression.html

PLAGIARISM CHECK RESULT

Form Title: Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1




FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Lee Wei Jun
ID Number(s)	19ACB03389
Programme / Course	IA
Title of Final Year Project	Predicting Open Space Parking Vacancies using Machine Learning

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)
Overall similarity index: <u>10</u> % Similarity by source Internet Sources: <u>6</u> % Publications: <u>7</u> % Student Papers: <u>0</u> %	ok
Number of individual sources listed of more than 3% similarity: <u>0</u>	
Parameters of originality required, and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.



 Signature of Supervisor
 Name: Tseu Kwan Lee
 Date: 2023 April 23

 Signature of Co-Supervisor
 Name: _____
 Date: _____

FYP 2 CHECKLIST**UNIVERSITI TUNKU ABDUL RAHMAN****FACULTY OF INFORMATION & COMMUNICATION
TECHNOLOGY (KAMPAR CAMPUS)****CHECKLIST FOR FYP2 THESIS SUBMISSION**

Student ID	19ACB03389
Student Name	Lee Wei Jun
Supervisor Name	Ms Tseu Kwan Lee

TICK (√)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

FYP 2 CHECKLIST

I, the author, have checked and confirmed all the items listed in the table are included in my report.



(Signature of Student)

Date: 26-04-2023