



**DESIGN OF VEHICLE DETECTION AND CLASSIFICATION
THROUGH IMAGE PROCESSING TECHNIQUE FOR
SURVEILLANCE SYSTEM**

This report is submitted in accordance with requirement of the University Teknikal Malaysia Melaka (UTeM) for Bachelor Degree of Manufacturing Engineering (Hons.)



PEONG YING CYING

FACULTY OF MANUFACTURING ENGINEERING

2021

DECLARATION

I hereby, declared this report entitled “Design Of Vehicle Detection And Classification Through Image Processing Technique For Surveillance System” is the result of my own research except as cited in references.

Signature

:

Author's Name

: Peong Ying Cying

Date

: 8 February 2021



APPROVAL

This report is submitted to the Faculty of Manufacturing Engineering of Universiti Teknikal Malaysia Melaka as a partial fulfilment of the requirement for Degree of Manufacturing Engineering (Hons). The member of the supervisory committee is as follow:



ABSTRAK

Dengan kemudahan teknologi pada masa kini, sistem pengawasan telah ditingkatkan untuk melakukan pekerjaan mengumpulkan informasi untuk tujuan analisis, mengesan dan mengklasifikasikan objek untuk pengurusan lalu lintas dan bukan hanya untuk tujuan keselamatan. Walau bagaimanapun, kamera litar tertutup konvensional (CCTV) tidak disertakan dengan pemprosesan lebih lanjut pada video yang menyebabkan ketidakmampuan melakukan analisis statistik pada kenderaan. Selanjutnya, adalah tidak produktif untuk mengklasifikasikan kenderaan dengan menggunakan tenaga manusia secara manual. Tujuan projek ini adalah untuk mencipta algoritma untuk pengesanan dan pengelasan kenderaan melalui teknik pemprosesan gambar. Sistem yang dicipta merangkumi sistem perkakasan dan perisian yang berkompromi dengan kamera, komputer riba dan MATLAB. Teknik pemprosesan gambar yang akan digunakan adalah pembelajaran mendalam iaitu rangkaian saraf konvolusional (CNN) yang dibina dengan menggunakan MATLAB. Dua jenis model CNN yang telah dilatih diadopsi dalam projek ini adalah SqueezeNet dan GoogleNet. Jenis klasifikasi kenderaan adalah Sedan, SUV dan MPV. Sistem yang dibangunkan akan disahkan dari segi ketepatan, penarikan dan ketepatan. Hasil dari projek tersebut ialah sistem yang dibangunkan dapat melakukan pengesanan dan klasifikasi kenderaan dengan ketepatan keseluruhan 86.7% untuk SqueezeNet dan 97.5% untuk GoogleNet. Masa pengiraan setiap gambar adalah 0.092s untuk SqueezeNet dan 0.194s untuk GoogleNet.

ABSTRACT

With the ease of technology nowadays, the surveillance system has been upgraded to do such works of gather information for analysis purpose, detect for tracking and classification of object and traffic management despite only for safety purpose. However, the conventional closed-circuit television (CCTV) is not embedded with the further processing on the video were cause to the inability to conduct the vehicle statistics analysis. Furthermore, it is non-productive to classify the vehicle by manually using the manpower. This project aims to develop an algorithm for vehicle detection and classification through the image processing technique. The developed system includes the hardware and software system which compromise of a camera, laptop and MATLAB. The image processing technique that will be used is a deep learning convolutional neural network (CNN) which is constructing by using the MATLAB. Two types of pre-trained CNN models are adopted in this project are the SqueezeNet and GoogleNet. Types of classification of the vehicle are Sedan, SUV and MPV. The developed system will be validated in terms of the accuracy, recall and precision. The result of the project is the developed system can perform the detection and classification of a vehicle with an overall accuracy of 86.7% for SqueezeNet and 97.5% for GoogleNet. The computational time per image is 0.092s for SqueezeNet and 0.194s for GoogleNet.

DEDICATION

Only

My beloved father, Peong Ah Kim

My appreciated mother, Lim Chin Low

My adored brothers, Peong Chee Hao and Peong Wei hao

for giving me moral support, money, cooperation, encouragement and also understandings

Thank You So Much & Love You All Forever

اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

ACKNOWLEDGEMENT

In the name of GOD, the most gracious, the most merciful, with the highest praise to GOD that I manage to complete this final year project successfully without difficulty.

My respected supervisor, Dr. Ruzaidi Bin Zamri for the great mentoring, kind supervision, advice and guidance that was given to me throughout the project. Besides I would like to thank to Engineer Assistant Encik Azwan and Encik Fairuz for lending the equipment in Makmal Simulasi 2 for this project. Furthermore, I would like to give a special thanks to my best friends who gave me much motivation and cooperation mentally in completing this report especially to, Leng Yee and Javin Khong for scientific advice while Hock Seng and Qi Hang for FYP report advice. They had given their critical suggestion and comments throughout my research. Thanks for the great friendship.

Finally, I would like to thank everybody who was important to this FYP report, as well as expressing my apology that I could not mention personally each one of you.

TABLE OF CONTENTS

ABSTRAK	i
ABSTRACT	ii
DEDICATION.....	iii
ACKNOWLEDGEMENT.....	iv
TABLE OF CONTENTS.....	v
LIST OF TABLES.....	x
LIST OF FIGURES.....	xi
LIST OF ABBREVIATIONS	xiv
LIST OF SYMBOLS	xvi
CHAPTER 1.....	1
INTRODUCTION.....	1
1.0 Background of Study.....	1
1.1 Problem Statement.....	2
1.2 Objective.....	3
1.3 Scope.....	3
CHAPTER 2.....	5
LITERATURE REVIEW.....	5
2.0 Literature Review.....	5
2.1 Conventional Digital Image Processing Method.....	5
2.2 Artificial Intelligence, Machine Learning, Deep Learning.....	7
2.2.1 Artificial neural network.....	8

2.2.2 Disadvantage of ANN	9
2.3 Deep Learning - Convolution Neural Network (CNN)	9
2.3.1 History of Convolution Neural Network (CNN)	9
2.3.2 Related CNN implement on vehicle detection	10
2.4 Basic Structure of Convolutional Neural Network (CNN)	10
2.4.1 Convolutional layer	11
2.4.2 Pooling layer	12
2.4.3 Fully connected layer	13
2.4.4 Activation function	13
2.4.5 Softmax layer	14
2.5 Type of Pretrained Network	15
2.5.1 YOLO V3	15
2.5.2 VGG-16	15
2.5.3 AlexNet	16
2.5.4 SqueezeNet	17
2.5.5 GoogleNet	17
2.6 Type of Dataset	18
2.7 Data Pre-processing	21
2.8 Training Method	24
2.8.1 Transfer learning	25
2.8.2 Training from scratch	25
2.8.3 Training Time - GPU	26
2.9 Hyperparameter Optimization	26
2.9.1 Learning rate	27
2.9.2 Epoch, batch and iteration	27
2.9.3 Related setting on hyperparameter in training CNN	28
2.10 Overfitting	31

2.11 Measurement of Performance	32
2.12 Summary	33
CHAPTER 3.....	34
METHODOLOGY	34
3.0 Introduction	34
3.1 Overview Of Methodology	34
3.2 Literature Review	35
3.3 Project Planning	36
3.4 System Development	37
3.4.1 Hardware system	37
3.4.2 Software system	39
3.5 Procedure for System Development	40
3.5.1 Hardware setup	40
3.5.2 Software installation	42
3.6 Programming Development	46
3.6.1 Dataset preparation and pre-processing	48
3.6.2 Create CNN algorithm	52
3.6.3 Experimental setup for training options	55
3.6.4 Output result	56
3.7 Testing	57
3.8 Data Analysis and Validation	58
3.9 Programming Code	59
3.10 Summary	64
CHAPTER 4.....	65
RESULT AND DISCUSSION	65

4.0 Introduction	65
4.1 Result for Validation Dataset	65
4.1.1 Comparison of precision	68
4.1.2 Comparison of recall	69
4.1.3 Comparison of accuracy and training time	71
4.2 Result on Testing Dataset	73
4.2.1 Precision	75
4.2.2 Recall	76
4.3 Comparison and Analysis on Performance Measurement Between Two Datasets	78
4.4 Computational Time Per Image	80
4.5 Summary	82
CHAPTER 5	83
CONCLUSION AND RECOMMENDATION	83
5.0 Conclusion	83
5.1 Financial Implication	84
5.2 Sustainability	85
5.3 Complexity	85
5.4 Long Life Learning (LLL)	86
5.5 Basic Entrepreneurship (BE)	86
5.6 Limitation	87
5.7 Recommendation and Suggestion	88
REFERENCES	90
APPENDIX A	100
APPENDIX B	101
APPENDIX C	104

APPENDIX D..... 111
APPENDIX E..... 119



LIST OF TABLES

3. 1	List of components for hardware with their functions.	38
3. 2	Specification of the hardware component.	38
3. 3	List of Vehicle Model.	49
3. 4	Dataset amount for each dataset.	51
3. 5	Training options.	55
3. 6	Example of confusion matrix – Interpret on Detect No Vehicle class.	58
3. 7	Explanation of programming code for each part.	62
4. 1	Calculation of precision and recall - SqueezeNet.	67
4. 2	Calculation of precision and recall – GoogleNet.	68
4. 3	Tabulated result of accuracy and training time.	72
4. 4	Confusion matrix for testing dataset by SqueezeNet.	73
4. 5	Calculation for precision and recall – SqueezeNet (Testing Dataset).	74
4. 6	Confusion matrix for testing dataset by GoogleNet.	74
4. 7	Calculation of precision and recall – GoogleNet (Testing Dataset).	75
4. 8	Data collection for the time taken per image in SqueezeNet.	80
4. 9	Data collection for time taken per image in GoogleNet.	81
5. 1	Bill of material (BOM).	84

LIST OF FIGURES

1. 1	Types of the vehicle to be classified: Sedan, SUV, MPV.	4
2. 1	Relationship of AI, ML, DL.	8
2. 2	Each neuron from the input is connected to the output.	8
2. 3	Basic structure of CNN.	11
2. 4	Connection of neurons to the reception field and feature map formula.	11
2. 5	Example of max pooling and average pooling.	12
2. 6	ReLU.	14
2. 7	Result of the proposed method.	18
2. 8	Zero padding and scales up.	22
2. 9	Example type of augmentation technique.	22
2. 10	Loss function vs learnable parameter.	24
2. 11	Example of epoch, batch and iteration.	28
2. 12	Hyperparameter of the system network.	29
2. 13	Accuracy over epoch graph.	31
2. 14	Formula of accuracy, recall and precision.	32
3. 1	Process flow chart.	35
3. 2	System development relationship.	37
3. 3	MATLAB software.	40

3. 4	Procedure for setup of camera.	41
3. 5	Procedure to transfer video clips into laptop.	41
3. 6	Create account.	42
3. 7	Download the MATLAB latest version R2020a.	43
3. 8	Choose the Windows for the installer.	43
3. 9	Accept the license agreement.	44
3. 10	Follow the instruction from licensing to confirmation.	44
3. 11	Select all product. Make sure the needed apps and toolbox are downloaded.	45
3. 12	Click install at confirmation.	45
3. 13	Start installing the MATLAB software.	46
3. 14	Flow chart for image processing.	47
3. 15	Top five most sales brand of car in 2019.	48
3. 16	Detect No Vehicle category.	50
3. 17	MPV category.	50
3. 18	SUV category.	50
3. 19	Sedan category.	50
3. 20	Example of testing dataset.	51
3. 21	Fire module.	53
3. 22	Inception module in GoogleNet.	54
3. 23	Plot of confusion matrix from MATLAB. (Jadhav <i>et al.</i> , 2020)	57
3. 24	Formula for accuracy, recall and precision.	58
3. 25	Import training and validation dataset.	60
3. 26	Modify the convolution layer and classification layer.	60
3. 27	Change the hyperparameter.	61

3. 28	Example of training in progress.	61
4. 1	Confusion matrix of validation dataset by SqueezeNet.	66
4. 2	Confusion matrix of validation dataset by GoogleNet.	67
4. 3	Histogram graph for comparison of precision between two CNN models.	68
4. 4	Histogram graph for comparison of recall between two CNN models.	69
4. 5	Example of bad images in the Sedan class.	70
4. 6	Training process – SqueezeNet.	71
4. 7	Training process – GoogleNet.	71
4. 8	Comparison of precision between two CNN model in testing dataset.	75
4. 9	Comparison of recall between two CNN models in testing dataset.	76
4. 10	Example of false predicted images.	77
4. 11	Comparison of average precision between two CNN models and two datasets.	78
4. 12	Comparison of average recall between two CNN models and two datasets.	79
4. 13	Comparison the accuracy for two models on validation dataset and test dataset.	79
4. 14	Comparison the computational time per image for SqueezeNet and GoogleNet.	81

LIST OF ABBREVIATIONS

AI	-	Artificial Intelligence
ANN	-	Artificial Neural Network
APA	-	American Psychological Association
BIT	-	Beijing Institute Of Technology
CCTV	-	Closed-Circuit Television
CLAHE	-	Contrast Limited Adaptive Histogram Equalization
CNN	-	Convolution Neural Network
COCO	-	Common Object In Context
CPU	-	Central Processing Unit
CUDA	-	Computer Unified Device Architecture
3D	-	3 Dimensional
DL	-	Deep Learning
Fc	-	Fully Connected
FN	-	False Negative
FP	-	False Positive
FPS	-	Frame Per Second
GPU	-	Graphics Processing Unit
GTX	-	Giga Texel Shader Extreme
HDL	-	Hardware Description Language
HOG	-	Histogram Of Oriented Gradient
IEEE	-	Institute Of Electrical And Electronics Engineers

IP	-	Internet Protocol
JPG	-	Joint Photographic Group
LSVH	-	Large Scale Variance Highway
ILSVRC	-	ImagNet Large Scale Visual Recognition Challenge
ML	-	Machine Learning
MPV	-	Multi Purpose Vehicle
NAG	-	Nesterov Accelerated Gradient
PNG	-	Portable Networkgraphics
PX	-	Pixel
RAM	-	Random-access Memory
ReLU	-	Rectified Linear Unit
RGB	-	Red, Green, Blue
ROI	-	Region Of Interest
SD	-	Secure Digital
SGD	-	Stochastic Gradient Descent
SIFT	-	Scale-invariant Feature Transform
SMQT	-	Successive Means Of the Quantization Transform
SVM	-	Support Vector Machine
SUV	-	Sport Utility Vehicle
TN	-	True Negative
TP	-	True Positive
VOC	-	Visual Object Classes
YCrCB	-	Luminance, Chroma red, Chroma Blue
YOLO	-	You Only Look Once

LIST OF SYMBOLS

%	-	Percentage
s	-	Seconds



CHAPTER 1

INTRODUCTION

1.0 Background of Study

A surveillance system is a system that consists of a camera connected to a computer device or IP network for observing purpose in that area. Many of the surveillance systems are installed on the traffic light, home, parking lot, highway and corridor. In traditional, it was used for monitoring safety. With the ease of technology nowadays, the surveillance system has been upgraded to do such works of gather information for analysis purpose, detect for tracking and classification of object and traffic management despite only for safety purpose.

Vehicle detection and classification plays an important role in helping the surveillance system for further application of traffic monitoring, traffic management, vehicle tracking, vehicle statistics analysis, parking system monitoring and many more. The vehicle detection means to provide information by localization of the vehicle. The vehicle classification will later separate the recognise vehicle belongs to which category. The conventional method for vehicle detection and classification was done by using the instalment of sensor laid out under the road to collect data and analyse the needed information (Wang *et al.*, 2019). Later with the development of the computer vision technology, where the surveillance system was able to embed with the machine vision. The image processing techniques were then being implemented to detect and classify the vehicle. The image processing technique is to perform some operation on the image as

input and extract certain useful information needed as output from the image. There are two types of image processing which are the analogue image processing like hardcopies for printouts and photographs and digital image processing which manipulate the image by using the computer.

The traditional method for vehicle detection and classification in surveillance system through image processing are such as using the scale-invariant feature transform (SIFT) feature matching and extraction, gaussian mixture model, histogram of oriented gradient (HOG) and support vector machine (SVM). However, with the rapid upgrade of digital technology, the conventional methods are unable to meet the requirement for accurate precision for detection and classification. Therefore, the intelligence system with the deep learning algorithm has been introduced to process vehicle detection and classification. Deep learning is significantly proved from previous research that it shows a robust increase in performance for image recognition and classification field due to the recent advancement in the Graphics Processing Unit (Farak, 2018).

In this study, the Convolution Neural Network (CNN) which is a kind of deep learning from artificial intelligence will be implemented into the image processing for the detection and classification of the vehicle in the surveillance system.

1.1 Problem Statement

Surveillance system places an important role in controlling the management and safety of the traffic. In conjunction with the increase in the population of the human being in a country, a huge volume of vehicles has also been increased on road. In related to the safety management on road, the conventional closed-circuit television (CCTV) is not efficiently to track on the vehicle on the road since it is not embedded with the further processing on the video. So, when there is a crime happen in the traffic area. It is unable to filter, classify and track the vehicle efficiently. Furthermore, the detection and classification of the

vehicle are also very important in the automated car parking system. The current car parking system is not intelligent to guide the consumer to find free space for parking. This lead to problems of time-consuming when the consumer has to traverse through multi-storeyed and multi parking slots to find free space (Bonde *et al.*, 2014). The situation is worsened when consumer keeps on find space but the parking slots already filled. The management in the car parking system is hard to achieve. It is also non-productive to classify the vehicle by manually using the manpower. Thus, by implementing the technology of deep learning in image processing of the video surveillance system will significantly help in improving all the problem statement listed above.

1.2 Objective

The objectives of this study are:

1. To develop an algorithm for vehicle detection and classification through an image processing technique.
2. To validate the developed system for vehicle detection and classification in terms of accuracy.

1.3 Scope

To archive the objective of the study, the scopes are as shown below:

1. The types of classification are Sedan, SUV and MPV.
2. The image processing of the detection and classification of a vehicle is using Convolutional Neural Network algorithm.
3. The software used to conduct the experiment is by using MATLAB.



Figure 1. 1:Types of the vehicle to be classified: Sedan, SUV, MPV.

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

CHAPTER 2

LITERATURE REVIEW

2.0 Literature Review

In this chapter, the topic discussed the information in detail for the traditional method and advanced artificial intelligence method related to the detection and classification of the vehicle from past research. The details are such as the explanation of types, the different of architecture and several parameters needed to affect the accuracy, precision and recall of the result.



2.1 Conventional Digital Image Processing Method

Based on Chen *et al.* (2018), there are two categories of vision-based vehicle detection algorithms for the conventional method. These two categories are motion-based approaches and hand-crafted feature-based approaches.

The motion-based approaches are such as the optical flow, frame subtraction and background subtraction. Optical flow is time-consuming since it is complex in tracking the computed motion vector of each pixel of the image. Frame subtraction detects the moving object by subtracting or calculate the differences of two consecutive frames. Frame subtraction is not suitable to detect motion that is too slow or too fast. Background

subtractions use the technique of modelling the distribution of background and foreground to detect the vehicle. However, background subtraction that based on Derman and Salah, (2018) stated that it is low performance and does not improve vehicle detection since it required more work for the machine.

While the hand-crafted featured based approaches are such as SIFT (Scale-invariant feature transform) and Histogram of Oriented Gradients (HOG). SIFT approach is among the best method in feature detector as it is applicable for multiscale images. Neeru and Kaur (2016) stated that the SIFT algorithm locates the points in an image which are invariant to scale and shift through the representation of orientation invariant feature vector. SIFT is invariance to image rotation, scaling, transition, illumination changes and 3D camera view. HOG define the image as a group of local histograms. HOG is used in computer vision to extract feature from the image for object detection. HOG focus on the structure of the object by calculating the gradient and orientation of the edge of the localized regions of an image (Kaplan and Şaykol, 2018). However, both the two traditional methods have low feature representation.

Huang and Zhang (2017) applied SVM to classify the different sub-categories that describe the attribute of a part of the vehicle model. Support Vector Machine is a supervised learning technique to perform the classification task. It finds the hyperplane that maximizes the margin between two classes to perform the classification task. The algorithm of SVM starts to define the optimal hyperplane through maximum margin. Then, it extends the optimal hyperplane for non-linearly separable problems to minimizes the misclassification. Lastly, it maps data to high dimensional space for the easiness to classify with linear decision surfaces (Bassma *et al.*, 2018).

Research had been done by Kaplan and Şaykol (2018) to compare the effectiveness and accuracy of the conventional method versus the deep learning method. The two-algorithm selected to be compared were The SVM based with Histogram Oriented Gradients (HOG) versus deep learning-based YOLO implementation. The experiment result showed that YOLO algorithm is more accurate than the SVM algorithm with an

accuracy of 81.9% over 57.8%. This means deep learning has higher accuracy over the conventional method.

2.2 Artificial Intelligence, Machine Learning, Deep Learning

Artificial intelligence by mean is the ability of the machine to behave like a human mind in learning and problem-solving. The programmed machine is based on the principle of human intelligence. It can carry task range from simple to more complex and human-like (Frankenfield, 2021). Examples of AI that integrated into our daily lives are such as Siri, Alexa, Google assistant etc. The limitation in human in the acquisition of large data is the time constraints. The process of learning requires knowledge and experience along many years. With the implementation of the AI, the machine can learn the needed knowledge in a significantly shorter amount of time than human (Mintz and Brodie, 2019).

Machine learning is a subset of artificial intelligence. It improves the performance from experience its learn. The working principle for machine learning in image processing there is a need to manually extract the features of the vehicle such as wheels and lights. This combination of attributes then be learned and to predict the image given. The downside of machine learning is the programmer need to know the of the vehicle very well so that the label made is correct. This labelling process of features also very time consuming when there are lots of attributes and image dataset (Bini, 2018).

Deep learning introduced automatic feature extraction to solve for the problem in machine learning algorithms. This means in deep learning, the neural network will decide itself the characteristics of vehicle can be used to represent the class reliably (Gibson, 2017). Convolution neural network is a type of deep learning in image processing. It is modified from the artificial neural network. Figure 2.1 shows the relationship between Artificial Intelligence, Machine Learning and Deep Learning.

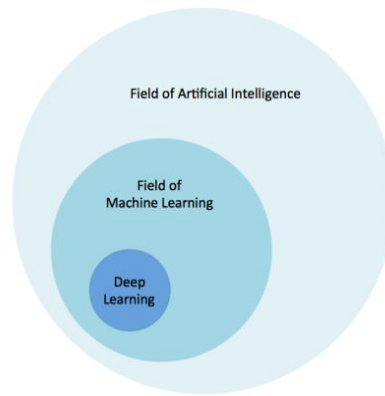


Figure 2. 1: Relationship of AI, ML, DL (Gibson, 2017).

2.2.1 Artificial neural network

Artificial Neural Network is an information processing algorithm inspired by the biological nervous system. It acts like a brain to solve specific problems through a large number of highly interconnected neurons. The neuron is a processing element that has many input and output and each input and overall bias in has a weight. The changes in output depending on the changes of weight and bias (Sladojevic *et al.*, 2016).

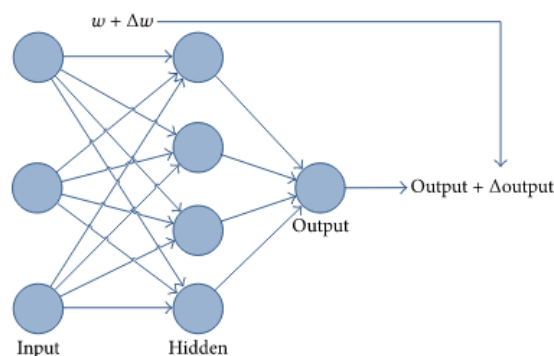


Figure 2. 2: Each neuron from the input is connected to the output (Srdjan Sladojevic *et al.*, 2016).

2.2.2 Disadvantage of ANN

The low performance of ANN is due to the fully connected layer where all the neurons from the input are connected to the hidden layer and then connected to the output. This will increase the weighted connected to be trained. Hence, the risk of overfitting can occur, and it required a large amount of data and computational sources. Moreover, it doesn't have a local correlation between the pixels in the divergent region in an image (Niessner *et al.*, 2017). Thus, an advance development from Artificial Neural Network to Convolutional Neural Network has been introduced to decrease the weight parameters and biases.

2.3 Deep Learning - Convolution Neural Network (CNN)

Convolutional Neural Network is a deep learning algorithm that processes the input image through several layers by given importance weights and biases in various aspects to differentiate one from others. The machine must learn the image features through the process of training to do the task of detection and classification. When compared with the traditional method, CNN can counter the environmental change since it requires no manual feature selection like SVM (Huang and Zhang, 2017). Furthermore, CNN can effectively do the task of detection and classification of the object simultaneously. This means that the CNN system is capable to meet the criteria of fast and accurate in detect and recognize the fast-moving vehicle on the road (Derman and Salah, 2018).

2.3.1 History of Convolution Neural Network (CNN)

Convolutional Neural Network was inspired by the biological theory of virtual Cortex. According to Guo *et al.*, (2017), David Hubel and Torsen Wiesel proposed the visual structure model based on cat visual cortex in 1962 for the first time using the

concept of the receptive field. Hubel and Wiesel implemented the basic visual cell types which were the simple cells and complex cells in creating a cascading model to fulfil pattern recognition tasks. Later in 1980, DR Fukushima proposed the first hierarchical structure Neocognition which consists of two consecutive layers called “S-cells” and “C-cells” for image processing. The Neocognitron model used the terms of “simple-to-complex” by using mathematical operations as a computational model for visual pattern recognition. The first work using the CNN model was introduced by YanLeCun for handwriting recognition by introduced a multi-layer artificial neural network called LeNet-5. However, LeNet-5 has low performance on a large scale- image and video classification. In recent years, many types of CNN architecture have been introduced to improve the performance of CNN on image processing.

2.3.2 Related CNN implement on vehicle detection

Tarmizi and Aziz (2018) in their research use CNN for detecting vehicle for autonomous vehicles under various weather condition. The dataset is getting from MATLAB, KITTI Vision and Intercity Roads where it's then being separate into the training set and testing set with the ratio of 177:118. The CNN network consists of convolution layer, ReLu layer, Max pooling layer, fully connected layer, softmax layer and a classification layer. The result shows that the CNN algorithm can detect the vehicles in bad weather such as poor light condition. The accuracy of the vehicle detection for sunny weather is 97.3%, for night weather is 61.4%, for snowy weathers are 73.4% and 98.7%.

2.4 Basic Structure of Convolutional Neural Network (CNN)

The basic structure of CNN consists of convolutional layers, pooling layers and fully connected layer. Normally, the architecture consists of a repetitive stack of several convolution layers, pooling layers and lastly with one or more fully connected layer. It is called a forward propagation when input data transformed into output through these layers.

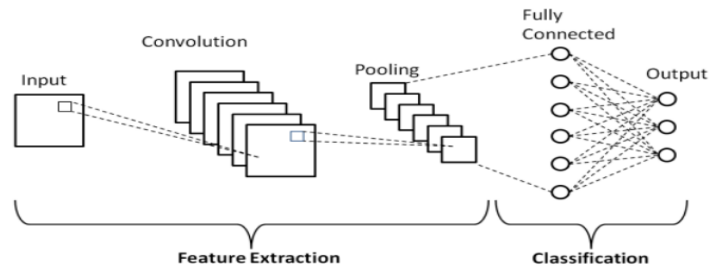
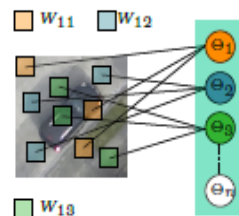


Figure 2. 3: Basic structure of CNN (Phung and Rhee, 2019).

2.4.1 Convolutional layer

Convolution layer is layer consists of a set of learnable filters which is small in size. The filters will move across the whole input image and convolved an output named feature map. The filter is also called kernel or feature detector. The feature map then becomes the input for the next layer. The size of the feature maps is depending on the number of the kernels and number of shifting steps of the filter (Frag, 2018). The figure below shows the connection of neurons to the region called the receptive field. The whole image shares the calculated weight matrices from each neuron. The filter or kernel is the calculated weight matrix and the result is n feature maps. The n is the depth of the following data volume. The formula is as below where the w_{in} is the weight matrix for channel i and neuron n , x_i is the input value, b_n is a bias and the output, o_k is feature map (Niessner *et al.*, 2017).



$$\hat{o}_k = f\left(\sum_i W_{in} * x_i + b_n\right)$$

Figure 2. 4: Connection of neurons to the reception field and feature map formula (Niessner *et al.*, 2017).

Based on the result from Bautista *et al.* (2016), The filter size and number of filter show no significant improves on the accuracy of vehicle detection in the low-resolution input image. It shows that feature information from an image is finite. Adding more detector no provide additional information. Besides, the experiment also shows that more number of stages means deeper the model is will increase the accuracy. This is due to the result in high order features such as corner and edges. However, the accuracy may drop due to too much pooling.

2.4.2 Pooling layer

The next layer is the pooling layer. Pooling layer is used by a given threshold to check whether the feature map has important information or not. There are several types of pooling layer based on research from the previous journal. The types of pooling layer are such as max pool layer, average pooling layer, global average pooling layer and global max pooling layer. Max-Pooling layer is the most famous use of pooling layer. Max - pooling layer means taking the maximum value from each neuron at the previous layer to reduce dimensionality reduction and parameter of an input image.

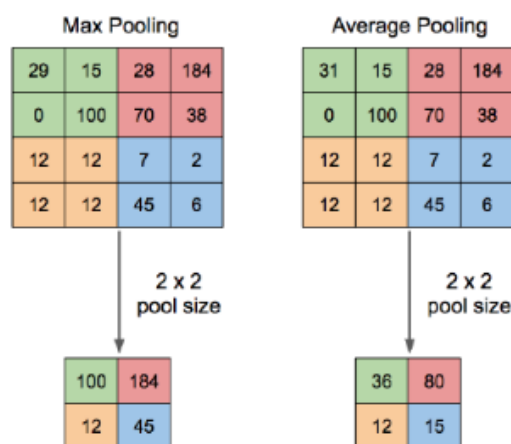


Figure 2. 5: Example of max pooling and average pooling (Yani *et al.*, 2019).

According to Zhang *et al.* (2018), global average pooling makes full use of the features in the intermediate layer so that more important information can be provided to final global features. The research put the global average pooling operation at feature map from third convolutional layer to output an average value. It is equivalent to dimensionality reduction operation. It can increase the efficiency in describing the characteristics of vehicle image by fasten the parameter learning process and improve the recognition accuracy by avoiding the risk of overfitting. Pooling action can lower down computational complexity, accelerate convergence rate and reduce the dimensionality of the feature map. However, based on Solovyev *et al.* (2019), the global max pooling is much simpler as finding the maximum value is easier than compute a mean value from the mathematical point of view. Both efficiencies are approximately the same, the different is just in term of hardware complexity.



2.4.3 Fully connected layer

The fully connected layer is also called a dense layer. It is a layer where each input from feature maps are connected to each output through the learnable weight. When the feature extraction is done from convolution layers, the feature map is downsampled by a pooling layer. They are later mapped to a fully connected layer to output for the classification task. Usually, the number of class to be classified is same as the number of output nodes in a fully connected layer (Patil and Rane, 2021).

2.4.4 Activation function

An activation function is important in the design of the network as it can affect the convergence speed of the network model. Therefore, the selection of the activation function need to be considered appropriately. Based on Zhang *et al.* (2018), the common

type of activation function is Sigmoid, Tanh and ReLU. However, the traditional method Sigmoid and Tanh produce not sparse output and both face the problem of produce gradient disappearance. While ReLU can solve the gradient disappearance problem and train network in a supervised way.

Rectified Linear Unit (ReLU) is the nonlinear activation function which is useful for interactions and non-linearity in the network. Interaction is the output prediction is dependent by another variable while non-linearity means the output is not dependent on the input. ReLU is used to speed up the training and reduce gradient computation to give a better prediction. This is because ReLU only updated the positive portion during the training process. ReLU is represented by function of $F(x) = \max(0, x)$. The layer gives value X when receives positive input and give value 0 when receives negatives input (Tarmizi and Aziz, 2018).

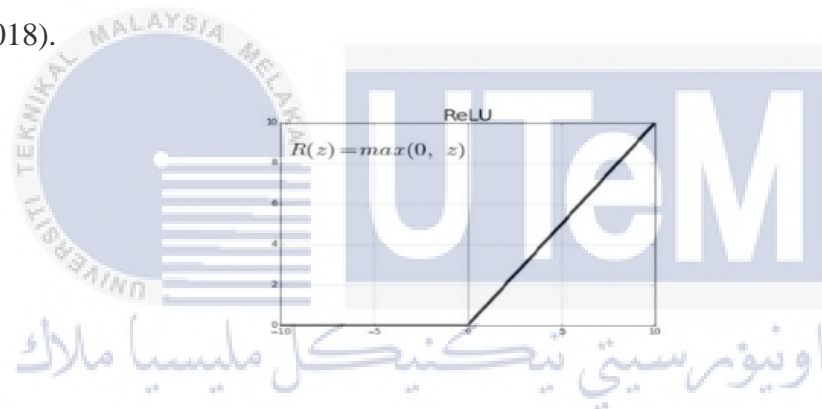


Figure 2. 6 : ReLu (Tarmizi and Aziz, 2018).

2.4.5 Softmax layer

Softmax layer is applied after fully connected layer for multi-class classification. Softmax layer help in differentiating and train the data more easily. Hsu *et al.* (2018) use the softmax layer to differentiate the vehicle object from the background. It is widely used in recognition class such as digit recognition and character recognition and many more in the machine learning system. However, the hardware mapping of Softmax layer involved much exponential and division which may cause the occurrence of overflow thus affect the computation accuracy (Hu *et al.*, 2019). The softmax layer is implemented to determine the

anchors belong to the foreground or background. Later, the classification layer classifies the category of the object (Zhang *et al.*, 2019).

2.5 Type of Pretrained Network

The design of the layer of the network can be different according to the situation of the task. Normally, the more depth the network is, the more accurate the detection and classification. However, when the network is too deep, it will increase the weight to the network which may decrease the performance of the task. Therefore, the consideration of the network depth, number of convolution layer, number of feature maps and size of pooling layer has to be done appropriately to suit to the situation of data (Pelt and Sethian, 2017). Some of the common widely used network been used for the system in related to vehicle detection and classification such as YOLO, AlexNet, VGG, GoogleNet, SqueezeNet and many more.

2.5.1 YOLO V3

Based on Roy and Rahman (2019), YOLO V3 is the fastest model used in object detection algorithm. It consists of several layers of detection combined with 53 layers of neural network for classification into a total of 106 layers deep. It can process 45 images per second with a good processor in a computer. Yolo-V3 finds the probability of an object in each bounding boxes which predict from the reframing of the input image into fixed grids. Hence, Yolo-V3 does not require complex pipelining which may increase the work to the system. Thus, the detection performance can be optimized.

2.5.2 VGG-16

While VGG-16 is another set of deep convolutional neural network which is simple and practicable. VGG-16 network consists of 13 convolution layers, 5 maximum pooling layers, 3 fully connected layers and a softmax layer. The recognition accuracy can be improved by stack the convolution kernel and maximum pooling layer repeatedly (Zhang *et al.*, 2019). The researchers use VGG-16 to recognise the colour of the vehicle and it is shown that the average recognition is 92.68%. It is proofed that by using VGG-16 with some parameter adjusting, the accuracy recognition can be high, and the average processing time of a picture is 0.008ms which is considered faster. The hyperparameter they used are 256 batch size, an initial learning rate of 0.01 with a dropout rate of 0.5 and the maximum number of iterations is 2240000.

2.5.3 AlexNet

AlexNet is created to classify the large ImageNet dataset. AlexNet is consists of 5 convolution layers, 2 normalization layers, 3 pooling layers and 3 fully- connected layer. The normalization layer is used to moderate the neighbour of excited neurons (Molina-Cabello *et al.*, 2018).

Based on Seng *et al.* (2018), VGG16 network shows a great performance for detection and classification of the vehicle with accuracy up to 97.6% in the complex background image. While by using the AlexNet network, the accuracy is about 93%. (Sheng *et al.*, 2018) in their experiment tested the classification of the vehicle into brand categories such as BMW, Ford, Audi, Chevrolet, Mercedes -Benz and Volkswagen. The dataset is augmented with a mirroring technique to increase the training pictures. The dataset then being tested with AlexNet, VGGnet-16, Vggnet-19, GoogleNet, ResNet50 and ResNet101. The result showed that the resnet101 and VGGnet-19 reach the best accuracy among other networks. While AlexNet has the lowest accuracy. ResNet101 and VGG19 have a deeper network and smaller convolution kernels. Therefore, they both are good in feature extraction on images and thus can performances image recognition better.

2.5.4 SqueezeNet

Based on Muhammad *et al.* (2019), the different distance of the camera from an object and the vary in size of the object of a dataset can increase the challenges for object detection algorithm. The experiment compared on two pre-trained neural networks which were the AlexNet and SqueezeNet. The research showed that both accuracies showed similarly but SqueezeNet is more suitable to be implemented in CCTV due to its model size of 3MB compared to 238MB of model size for AlexNet. SqueezeNet also has a lesser rate of false. Lee *et al.* (2019) stated that deep learning model SqueezeNet requires less than 5MB of space thus is good to be implemented for real-time applications. The size of SqueezeNet is 11 times smaller than GoogLeNet and 53 times smaller than AlexNet. The result showed the time of processing was 108.8ms per image for SqueezeNet model. Tsang (2019) stated that smaller model brings advantages of such as required less communication during training, required less bandwidth to export new model from the cloud and the third advantage is more feasible to deploy on hardware with limited memory.

2.5.5 GoogleNet

GoogleNet architecture comes from Inception module which consists of convolutional layers, max pooling layers, concatenating layers, dropout layers, fully connected layer and a softmax layer. GoogleNet using the Inception Module to improve computational efficiency (Seo and Shin, 2019).

Guo *et al.* (2017) in their research used GoogleNet as a base network to classify the type of vehicle in a 720P video dataset. The result showed that the performance by using the GoogleNet network with an input size of 256 can get 94.99%. They then experiment with random resized/burred data to increase performance. The accuracy of input size 256 with burred data for the performance has increased to 98.37%. Figure 2.11 shows the result of the experiment.

Input Image Size	GoogleNet	GoogleNet with Blur
64	9.55%	91.85%
128	66.71%	98.02%
256	94.99%	98.37%

Figure 2. 7: Result of the proposed method (Guo *et al.* 2017).

Seo and Shin (2019) in their research used CNN to categories the electric vehicle types for electric vehicle parking and charging system. They used GoogleNet as the main network and perform the transfer learning technique. They remove the last fully connected layer and pretrained the network with large-scale ImageNet dataset. They then fine-tuned the last fully connected layer with their electric vehicle dataset. They divided the dataset into the training set, validation set and test set with the ratio of 6:3:1. The result shows that the overall accuracy of training is 0.912, validation is 0.776 and test is 0.776. The accuracy improved with the increasing step.

Derman and Salah (2018) in their research used the network model of SVM to compare with CNN, Tiny -YOLO to do the vehicle detection and classification. The Tiny-Yolo network is a Darknet-19 model that consists of 9 convolution layers, 6 max polling layers, one average pooling layer and one softmax layer. Tiny- YOLO is simple and required small GPU resources. They tested with BIT-Vehicle dataset and TPS dataset. The result showed that Tiny -Yolo performed well in BIT dataset and SVM worked well in TPS dataset. The reason was due to the acquired dataset was imbalanced which may be caused to overfitting that affected the result.

2.6 Type of Dataset

Dataset is another important point in training object detection and classification. Learning (2020) Stated that preparing dataset needed a lot of work. The first step is to

decide what task that the model needed to solve. The second is what type of data should be gathered. The dataset must properly represent the task in different situation and condition. If not, the model can wrongly predict the wrong features as important information and learned it. Another intense work for preparing dataset was the labelling of the dataset. It was difficult to automate and needed to be done by a human. In the research showed that jpg and png do not show significant differences in clinical actual performance. It is suggested not to use large PNG images to train model because JPG images are smaller in size thus model generation is faster and reduce the requirement for high storage (Jones *et al.*, 2019).

The dataset will be separated into the training set and testing set. The general ratio for the split of dataset for training and testing are such as 9:1, 8:2,7:3 and 6:4. According to Pajaziti *et al.* (2019), the classifier will be more accurate when the number of the dataset is large. However, the greater number of a dataset means longer training time will be taken. There are two types of images in the dataset which are positive and negative images. Positive images mean the images contain an object to be detected while negative images mean the images do not contain the object to be detected. Espinosa *et al.* (2017) prepared the dataset with another class for the urban environment related object means the other possible object detected other than the vehicle. All the images were collected from different perspectives and angle. There are many large numbers of dataset there are normally used in training vehicle detection and classification. The dataset can be either downloaded from the internet or personal capture the images through video frames.

Based on Roecker *et al.* (2018), BIT- Vehicle dataset consists of 9850 vehicle images which are wide range in changes of scale, surface colour, illumination and position of the vehicle. All images are in high resolution which is 1600 x 1200px and 1920 x 1080px. The vehicle in the images may contain more than one, thus, some data pre-processing action such as cropping has to take before entering the training process.

KITTI is a large vehicle dataset that contains various scales of the vehicle in various scenes. It contains a total of 14999 vehicle images for training and testing. The

dataset divides the sample into three difficulty level which is easy, moderate and hard according to occlusion, size and truncation. The minimum height of 40x for easy level, a minimum height of 25px for moderate and approximately corresponds to object within 47m for hard level (Li, 2017).

PASCAL VOC is a popular dataset that using for training data for detection, classification and segmentation. PASCAL VOC dataset focuses on compact objects like human, animals, vehicle and door object. Another dataset which is COCO created by Microsoft has more classes than PASCAL VOC also widely for general purpose (Pan and Ogai, 2019).

Zhang *et al.* (2017) in the research used VeRi and VehicleID dataset which was getting from real-word surveillance videos. VeRi dataset had 619 vehicles captures by using 20 surveillance cameras in a traffic scene. The vehicle was captured at different point of view, different illuminations and different resolution. While VehicleID contained 20267 vehicles in a total of 221763 images. The images were captured from the front or back.

Some research created their dataset for training and testing of the vehicle on detection and classification. LSVH is one of the examples for own constructed data set in the highway. It consisted of 16 videos with various in time, weather, scenes and resolution (Hu *et al.*, 2019). Based on Huttunen *et al.* (2016), they self-created two types of dataset. One type of dataset captured by one camera where the background was static. The dataset was collected over a long period in a day thus the dataset had significant changes in environmental conditions. Another dataset was created by two cameras in different point thus resulting in different background and different angle and direction of the vehicle. This dataset can be used to evaluate the network to learn the appearance of background or actual vehicle.

Munich Vehicle dataset contains 20 original large-scale aerial images with different types of vehicle such as car, bus, truck, etc. Deng *et al.* (2017) separate the dataset to half

for training and training and undergo data pre-processing technique such as augmentation for increase the data to avoid overfitting.

San *et al.* (2019) disclaimed that separating classes of vehicles into the sedan, SUV, crossover etc require more quality images to extract more features to divide into more detailed manner. Kannojia and Jaiswal (2018) stated that the reduction in the resolution of the dataset had decreased the performance score of CNN in terms of accuracy, precision and F1 score.

2.7 Data Pre-processing

Data pre-processing is an operation where the input images are being enhanced or suppress some unwanted distortion before a move to further process. Han *et al.* (2018) stated that the input data must undergo data reprocessing such as sizing and normalization before entering the training and testing phase. However, CNN demand less pre-processing operation than other neural networks just for eliminating unconcerned differences

Hashemi (2019) stated that in CNN, image resizing will be very important when the input images are not in a fixed size. In research, two types of image scaling method which were scaling up using interpolation and zero paddings had been used to do a comparison. The result showed that zero padding did not affect the accuracy of classification however it can reduce the training time due to the neighbouring zero input units. Thus, the synaptic weights from input units will not need to be updated. Therefore, the corresponding convolutional unit will not be activated in the next layer.

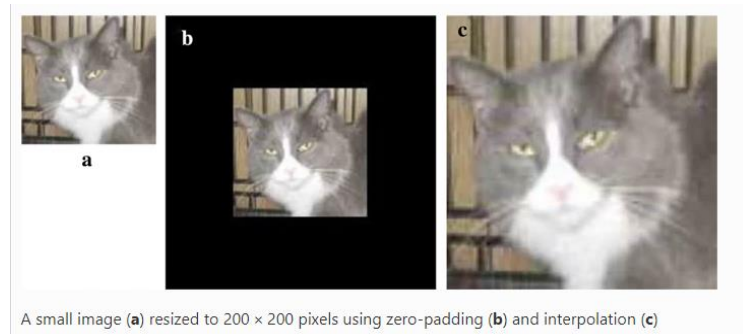


Figure 2. 8: Zero padding and scales up (Hashemi, 2019).

Data augmentation is a necessary step when working with the vehicle classification based on deep learning neural network. It can improve the performances of the automatic learning model. The data augmentation techniques are such as random rotations, flips, cropping, etc (Hicham *et al.*, 2018). Data augmentation is useful when training data size is not large. Kim and Lim (2017) augmented the original image through the aspect ratio keeping without stretching the original images through the rotate and flip technique. The research stated that there was no significant performance difference between stretched images and aspect ratio keeping images.



Figure 2. 9: Example type of augmentation technique (Kim and Lim, 2017).

Other data pre-processing such as convert the RGB colour space image to YCrCb colour space image through the standard conversion formula. YCrBr is consists of three

components which are the Y is computed from nonlinear RGB, Cr is storing a difference between red and luma component, and Cb is storing a difference between blue and luma component. Therefore, YCrBr colour space is luminance independent which makes it more attractive to colour image segmentation and feature extraction and hence give better performance (Awang *et al.*, 2020).

Hicham *et al.* (2018) in their research used deep learning CNN and data augmentation to test the vehicle detection and classification. There were four categories of the vehicle which were school bus, ambulance, police car and Moroccan transport Company which were total of 2400 samples images. The data augmentation technique had been used were such as flip, random rotation and cropping. The CNN architectural consisted of 6 convolution layer, 4 pooling layers and one softmax layer. The experiment showed a good result that the precision of the classification of the vehicle was about 0.88 to 0.90 and the recall was about 0.83.

Xin and Wang (2019) stated that all possible changes in the same image can extend the effective size of training data. This will then change the result of the image training as it helps the network to deal with all problem occur in the real use of classifiers. Chen (2019) in the research used five different image enhancement algorithms which were the contrast limited adaptive histogram equalization (CLAHE), the successive means of the quantization transform (SMQT), the adaptive gamma correction, the wavelet transform, and the Laplace operator to explore the effect of image enhancement algorithms on the performance of CNN models in deep learning and transfer learning. The results showed image-enhancing algorithm may improve the performance for the complete CNN model but not suitable for transfer learning. It can reduce the performance of transfer learning-based pre-trained CNN model. The reason may be due to the pre-trained CNN model was trained on natural images. Therefore the parameter was suitable for extracting features from natural images.

Conclude, data preprocessing such as resizing, data augmentation are necessary steps in CNN while image-enhancing may seem to have a negative impact on transfer learning on CNN.

2.8 Training Method

Training is a process of finding kernels in convolutional layers and weights in the fully connected layer. Two processes will be needed in training a network are forward propagation and backward propagation. Forward propagation uses a loss function to find the model performance under kernels and weights. The loss function is used to measure the compatibility between output predictions of the network. Example of the loss function is cross-entropy. While backpropagation and gradient descent are used to update the loss value of kernels and weight. Gradient descent is an optimization algorithm that keeps updating the parameter of kernel and weight to minimize the loss. The learnable parameter is updated in the negative direction of the gradient with an arbitrary step size determined based on a hyperparameter called the learning rate. The formula of update of the learnable parameter is as $w := w - \alpha * (\partial L / \partial w)$, where w is the learnable parameter, α is learning rate, L is loss function (Patil and Rane, 2021).

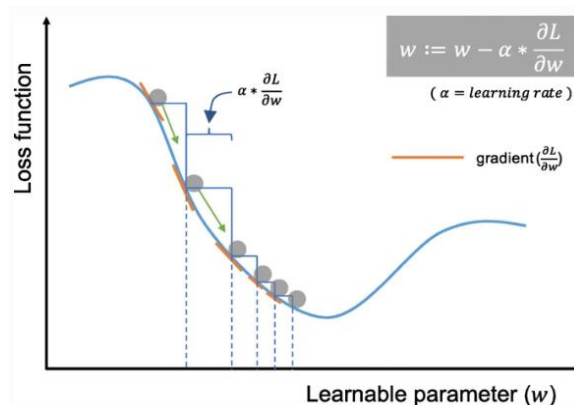


Figure 2. 10: Loss function vs learnable parameter (Patil and Rane, 2021).

2.8.1 Transfer learning

Transfer learning is introduced to optimize the save of time and achieve better performance. It is used to improve the learning skills in the target task by given knowledge from source-task. Transfer learning also means by using a trained model for one task as a starting point to train for the second task. Therefore, the knowledge from the previously learned task can be pass to the next related task (Shu, 2019).

Transfer learning has two methods which are feature extraction and fine-tuning. Common methods in fine-tuning are frozen some layers and fine-tune the remaining layers. Feature extraction is keeping the weights of the pre-trained model intact. Feature extraction trains the new classifier on target dataset by using the embedding it produces. While fine-tuning initial the weights of a pre-trained model to train on target dataset. The choosing of both techniques is based on proximity both source and target dataset and the size of the dataset. The overfitting may occur when training the small dataset in a large number of layers (Boulent *et al.*, 2019).

Zhuo *et al.* (2017) in their experiment used AlexNet and GoogleNet as the based network to test the accuracy of training with or without fine-tune. The dataset comes from VehicleDataset and it was separated into training and test samples. The result showed that with the employed of fine-tuning, the average of 2% higher classification can be obtained if compared to without fine-tuning. GoogleNet network with fine-tuning achieved better accuracy if compared with AlexNet.

2.8.2 Training from scratch

Training from scratch means the network weight is randomly initialized rather than inherited from the previous model. It has a higher risk in overfitting as the input weight is defined from input data. Furthermore, it required a larger training dataset. However, this

approach can handle the more specific task of object detection and classification and thus can improve the performance (Boulent *et al.*, 2019).

2.8.3 Training Time - GPU

Nithiyaraj and Arivazhagan (2020) stated that CPUs is hard to handle for large dataset when training using a deep learning algorithm. There is a need for graphics processors GPUs such as Titan, GeForce and NVIDIA's Tesla to reduce the training time typically faster in 10 to 30 times when compared to CPUs. The models of the GPUs are such as NVIDIA GeForce GTX 1080, NVIDIA Titan Z, and NVIDIA Tesla V100 GPU which are powerful in processing.

Chung and Sohn (2018) stated that the graphical processing unit (GPU) is needed to learn a CNN model. Without GPU, it is unable to compute matrices of weight parameters in parallel. This research used NVIDIA Quadro K5200 as GPU to conduct the experiment. The result showed that the GPU took 42 seconds to run a single epoch. The maximum number of epochs was set at 5000. Therefore, the computing time can be done within two days. The GPU can affect the training time of the network.

2.9 Hyperparameter Optimization

Hyperparameter is a parameter that can be adjusted to has a direct impact on the machine training of CNN. It is usually set before the learning process begins. The examples of the hyperparameter are such as batch size, epoch, learning rate, weight and momentum (Do *et al.*, 2020).

2.9.1 Learning rate

Learning rate hyper-parameter is adjusting the weights of the network to loss gradient. Learning Rate will initially start with a value and the value will decrease as the optimization progresses. The learning rate can be represented as $lr (1 + \gamma k)^{-\alpha}$, where lr is the learning rate, parameter k is the current iteration, γ and α are the parameters to compute learning rate (Zhou *et al.*, 2016).

2.9.2 Epoch, batch and iteration

Based on (Do *et al.*, 2020), the entire large dataset needed to be divided to pass into the neural network. Epoch is the forward pass and backwards pass of all training data through the network. Batch is the size constant number of the dataset in a forward/backward pass which is normally determined by the total size of the dataset, number of weights and memory space of GPU. The batch size can be range from 1 to 128 where the higher the batch size, the required memory space of GPU will be larger. While iterations are the total number of passes. Below shows the example of the epoch, batch size and iteration.

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

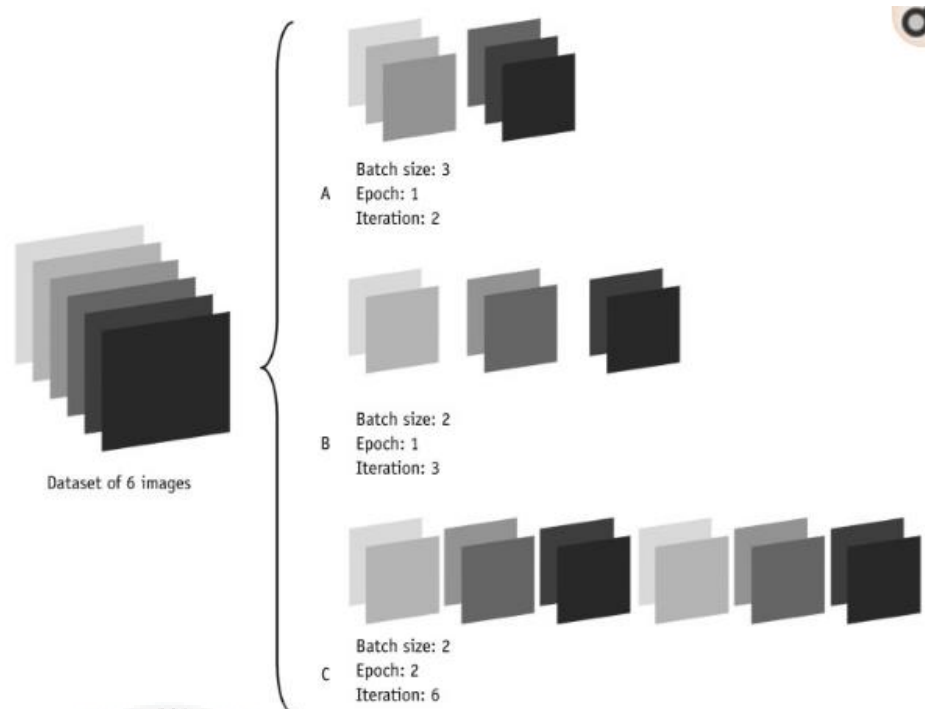


Figure 2. 11: Example of epoch, batch and iteration (Do *et al.*, 2020).

2.9.3 Related setting on hyperparameter in training CNN

Huttunen *et al.* (2016) in their research did the comparison between the CNN network and SVM with SIFT on detecting and classifying the vehicle. The dataset was as explained in subtopic dataset and it was then split into the training set and test set in a ratio of 9:1. The CNN architectural consisted of 2 convolution layer, two fully connected layers and one output layer. Each convolution layer has one max pool layer. Then ReLu and Dropout regularized was added between each layer. The hyperparameter was as the figure below. The learning rate was set to very small. The result showed that CNN is better than SVM with SIFT in term of accuracy with about 97% and littler error in recognition of vehicle. This may due to the high-level feature of CNN.

Hyperparameter	Range	Selected Value
Number of Convolutional Layers	1 – 4	2
Number of Dense Layers	0 – 2	2
Input Image Size	{64, 96, 128, 160}	96
Kernel Size on All Convolutional Layers	{5, 9, 13, 17}	5
Number of Convolutional Maps	{16, 32, 48}	32
Learning rate	$10^{-5} - 10^{-1}$	0.001643

Figure 2. 12: Hyperparameter of the system network (Huttunen *et al.*, 2016).

Guo *et al.*(2017) in their research discussed the learning rate method and different algorithm in solving the optimal parameters during the training. The result showed that the recognition rate of each algorithm improved with the increase of the number of iterations. Multistep is the best among the other algorithm in learning rate set. The multistep adjust the value of step value for basic learning rate while the fixed method keeps the basic learning rate constant and has universal applicability. The experiment also showed that stochastic gradient descent (SGD) with momentum is better than original SGD and NAG. However, it is gradient descends slowly in early-stage and before stable with the increase of the number of iterations.

Roecker *et al.* (2018) in their research proposed a new CNN architecture to classify vehicle into bus, microbus, minivan, SUV, sedan and truck. The CNN architectural consisted of 4 convolution layers, 2 spatial pooling layers, 3 fully-connected layers and one softmax layer. The model is tested on BIT vehicle dataset. The network training was from scratch start with the multinomial logistic regression and follows with the data augmentation and dropout method for regularized to prevent overfitting. The hyperparameter optimization approaches were using the mini-bath gradient descent with Adam Algorithm. The batch size was 128 while the exponential decay rate for beta1 is 0.9 and beta2 is 0.999. The learning rate was 10^{-3} and will stop the training process after 64 epochs. The accuracy of the test was 93.90% and this was due to the data augmentation, depth of network and regularization that prevented the model to overfit which had decreased the error.

He (2018) in the research set the learning rate started at 0.001 for the first 100Kiteration. It will then drop to 50% after every 30K iterations. The total iterations were

200K and the batch size was 16 using SGD. Each convolution will be applied a batch normalization. The result showed that the average precision is 67.25% with a detection rate of 28 frames per second with a moderate GPU which is suitable for real-time implementation.

Rangarajan *et al.* (2018) in their research adopted Alexnet and VGG16 these two pre-trained deep learning algorithms to perform for the classification of tomato crop disease. They fine-tuned the hyperparameters such as weight and bias learning rate and mini-batch size and be analyzed in term of classification accuracy and execution time. The size of dataset was set small size with a total of 373 images and train in 10 epochs. The mini-batch size was set with 2, 12, 22 and 32. The result showed that for VGG16 model, the execution time was increased when the mini-batch size increased. While AlexNet decreased in time for the execution time when mini-batch size increased. The reason was stated to be the time taken for processing in VGG16 was affected by the depth of the model. While for accuracy, increased of mini-batch size decreased the accuracy of VGG16 model whereas for Alexnet did not shows the clear correlation of accuracy. For the fine-tuned in weight and bias for learning rate, the accuracy for VGG16 was decreased with increased weight and bias for learning rate. Therefore, VGG 16 performed better with small mini-batch size and small weight and bias for learning rate. While AlexNet was good in minimum execution time and provided good accuracy that slightly low to VGG16.

From the review of past journals, there is no significant best value of epoch and mini-batch size that can be implemented to achieve high accuracy of the task. This dataset and CNN models may work fine with this hyperparameter value while others may not. The general concept is smaller batch size means more gradient updates per epoch thus longer training time. Most of the setting of value for learning rate and momentum is 0.0001 and 0.09. While foregoing stated that more training epochs will cause CNN tends to memorize the images and loses the generalization of the characteristics learned from the image.

2.10 Overfitting

Overfitting happens when the model train the data too well where it memorizes all the details in data including the noise on training data. This phenomenon has a negative effect on the performance of the model on testing data where the model difficult in coping with information from testing data. The overfitting happens when there is a presence of noise, the limited size of the training set and the complexity of classifiers (Ying, 2019).

Shorten and Khoshgoftaar (2019) stated plotting the training and validation accuracy at each epoch during training can discover the overfitting problem. The plot at the left shows the sign of overfitting when the validation error starts to increase as the training rate continue to decrease. While the graph at right shows the desired relationship between training and testing error where the validation error is continuing to decrease with training error.

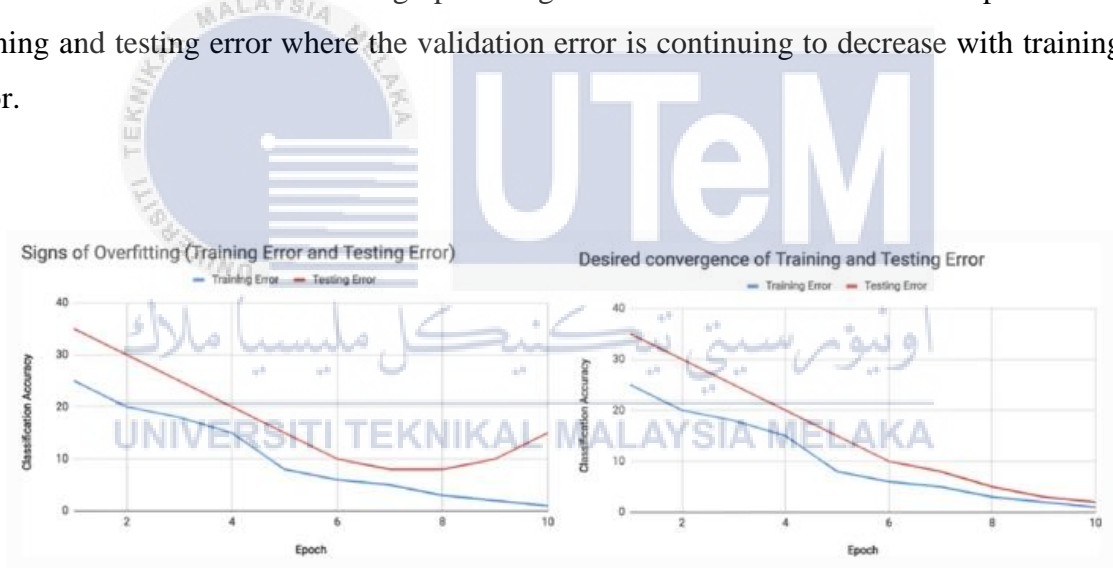


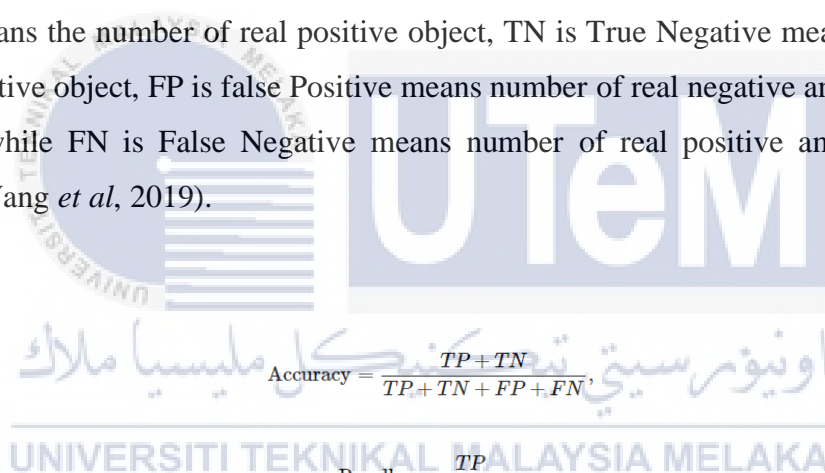
Figure 2. 13: Accuracy over epoch graph (Shorten and Khoshgoftaar, 2019).

Past research showed that overfitting can be reduced through the solutions of reducing the model complexity, early stop before convergence in training, weight decay by constraining parameter using ℓ_2 -regularization. The several regularizations have been introduced such as data augmentation, dropout, dropConnect, stochastic pooling and disturblabel. Data augmentation is randomly generated more data as discussed in subtopic 2.7. While dropout updates only remaining weights in each batch iteration by discarding the hidden neurons. DropConnect updates randomly selected subset weight. Stochastic

pooling changes the deterministic pooling operation and it then chooses one input as pooling result in probability (Xie *et al.*, 2016.).

2.11 Measurement of Performance

The performance of object detection and classification in CNN is defined as accuracy, recall and precision. Accuracy is all the proportion of data is classified correctly. The recall is the proportion of data classified positive among all real positive data. Precision is the percentage of data that are real positive among all classified positive in the CNN network. The formula for accuracy, recall and precision are as below. The TP is True positive means the number of real positive object, TN is True Negative means the number of real negative object, FP is false Positive means number of real negative among predicted positives, while FN is False Negative means number of real positive among predicted negative (Wang *et al.*, 2019).



$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$

$$\text{Recall} = \frac{TP}{TP + FN},$$

$$\text{Precision} = \frac{TP}{TP + FP},$$

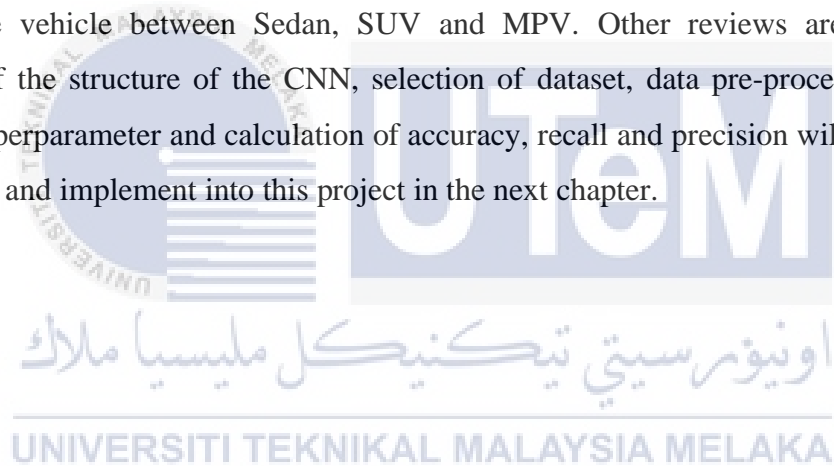
Figure 2. 14: Formula of accuracy, recall and precision (Wang *et al.*, 2019).

The recall and precision both are very important and sometimes a tradeoff between these two has to be made to suit for the situation. For example, the medical field always aims for high recall over high precision. This is because the system needs to ensure that least missed out of patient that got the disease. In this case, the proportion of correctly identified in all disease case is call recall rate. For the situation of high precision over recall is when the system is more important to do the task of for example the proportion of true salmon to all fish which are considered as salmon. For situation precision and recall

are same, The F1 score is calculated to represent the standard for performance measurement (Chen, 2019).

2.12 Summary

As a summary, this chapter shows the study of past research on vehicle detection and classification. The review shows that CNN is proved can do the detection and classification vehicle with robust accuracy if compared to the conventional digital image processing method. Most of the researches are done in detect the vehicle and classified them according to brand, colour, volume and separate between truck, motorcycle taxi and ambulances. For this project, the CNN algorithm will be implemented to detect and classify the vehicle between Sedan, SUV and MPV. Other reviews are such as the construct of the structure of the CNN, selection of dataset, data pre-processing, training method, hyperparameter and calculation of accuracy, recall and precision will be taken into considering and implement into this project in the next chapter.



CHAPTER 3

METHODOLOGY

3.0 Introduction

The methodology section is about to provide an overview of the flow of the project and the process of data collection and method in conducting a specific experiment to achieve the objective of the project. This chapter will focus on the first objective of this project which is to develop an algorithm for vehicle detection and classification through an image processing technique. A process flow chart is created to represent each sequence for a visual guide. Each of the processes for fulfilling the objective number 1 for this project will be discussed further in this chapter.

3.1 Overview Of Methodology

The process flow chart shows the overall processes to be carried out during the whole project in these 15 weeks of a semester. The flow chart is created to graphically represent the sequence of whole processes from beginning to end. It is used to visualize and understand how this project works and what steps to be done before proceeding to the next step to fulfil the objective of this project. Figure 3.1 shows the process flow chart of this project.

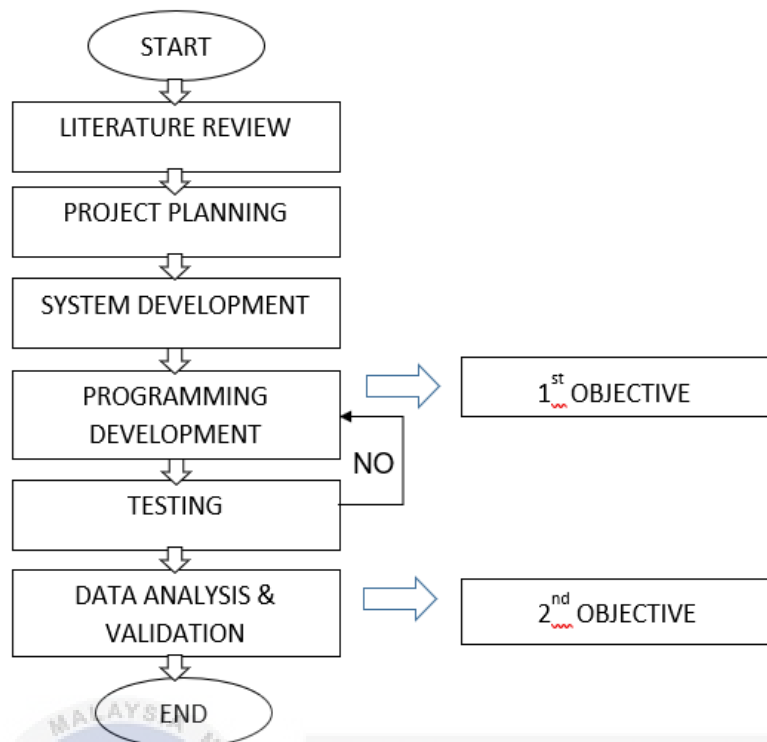


Figure 3. 1: Process flow chart.

3.2 Literature Review

The literature review is important as the first step to carry out this project to help in better understanding of the topic. It provides the foundation on the topic from previous study and research to serve as a guideline and reference. The source of journals for creates this literature review normally gets from various of the verified and trustable website such as IEEE, Science Direct and Google Scholar to ensure the source is legit. The journals are downloaded from the internet website and stored in a folder in the laptop with numbering from one onward for easiness of organized. Important data have been extracted from those journals such as the definition, method to conduct the experiment, result and discussion or finding from their study. The data then be analysed and sorted based on the parameter to write in different sub title in the literature review. The main data from the literature review are the dataset for this project need to collect manually through google and video recording, the CNN algorithm need to adopt pre-trained neural networks that are small in size,

parameter and depth for a suit to the small dataset. The data are documented by using Microsoft Words. The detail of the literature review is explained more in chapter 2 of this report. The list of journals is stated at the end of this report in Reference with correct APA format of citation.

3.3 Project Planning

Project planning is a process on managing the project through establish of scope in related to define the objective to attain them. It is the heart of the project life cycle to act as a guideline to organize the process to focus on objectives and scopes so that it will not fall out of track progression.

To attain the objectives, this project is developed by a combination of hardware and software system. This system will be explained in detail in next subtopic regarding the connection and how its interface in between. In related to scopes which listed in chapter 1, the preparation of dataset collection for two types of datasets will be conducted through the collection of images from Google for training and validation dataset. While a camera is used to collect data set for the testing of the algorithm. The testing dataset will be collected by taking a video from the roadside. The images of vehicles are then cut from the frame and cropped out the desired region of interest. Four types of categories are Detect No Vehicle, MPV, SUV and Sedan. Dataset pre-processing such as dataset separation, dataset augmentation and dataset resizing is done before continuing to image processing process. The image processing algorithm will be conducted by using MATLAB software to execute the first objective of the project. Only Convolution Neural Network (CNN) algorithm will be implemented to do the image processing on detection and classification of vehicle. The technique of transfer learning and fine-tune on pre-trained CNN will be adopted. The algorithm will be trained and tested several times until the output result shows the expected percentage of accuracy. An analysis will be done between the result on validation dataset and result on test dataset to execute the second objective.

3.4 System Development

The system for this project is made by the combination of hardware with software to achieve the first objective. Hardware is physical components such as the computer device and camera. While the software system used in this project is MATLAB which mainly use for designing the algorithm for vehicle detection and classification. The relationship between the hardware system and software system is as below:

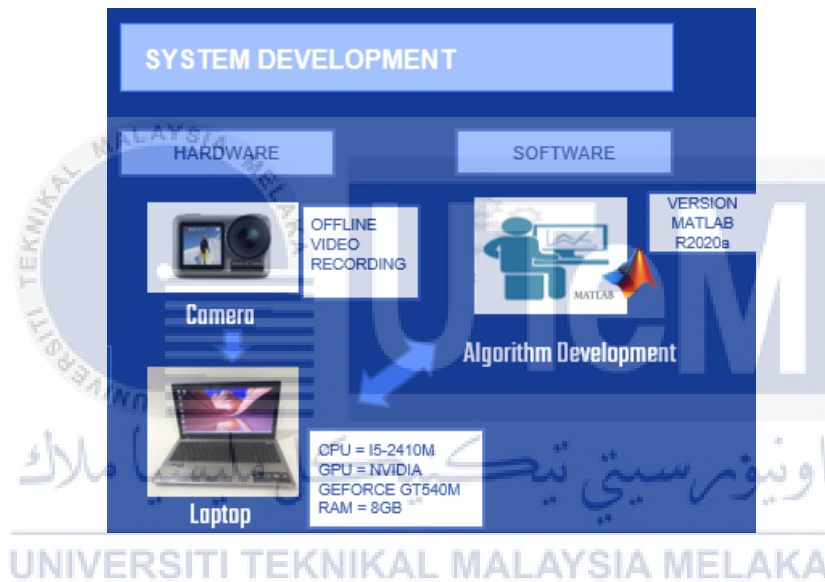


Figure 3. 2: System development relationship.

3.4.1 Hardware system

The hardware of this project is made up of a camera and computer device. These two components perform different tasks. The camera used to capture images for the collection of the test dataset. While the computer device is acting as an interface between the camera and software system to prepare the dataset, develop the programming for image processing and display the output. Table 3.1 shows the hardware components model and Table 3.2 shows its description.

Table 3. 1: List of components for hardware with their functions.



Component	Model	Function
Camera	DJI Osmo Action 	To capture video clips containing vehicles.
Laptop	Asus A53S 	To interface between the camera and software system to prepare the dataset, develop the programming for image processing and display the output.

Table 3. 2: Specification of the hardware component.

DJI Osmo Action	
Sensor	1/2.3 CMOS
Lens	FOV:145° f/2.8
Max Image Size	4000 x 3000 pixels
Storage	MicroSD – 64GB
Video	720P to 4K
Video Format	MOV, MP4
Battery	1300 mAh LiPo
Asus A53S	
Processor	Intel® Core™ i5-2410M
CPU	2.3GHz
GPU	NVIDIA GEFORCE GT 540M CUDA 2GB
RAM	8GB
Storage	299GB
Operating System	Windows 10 Home, 64-bit operating system
LAN Speed	65.0 Mbps

Further research and troubleshoot will be conducted to ensure the hardware are synced to each other smoothly by referring to the website platform of YouTube and MATLAB official website for visual guidance on installation.

3.4.2 Software system

MATLAB is the main software that will be used in this project in designing the algorithm for vehicle detection and classification. MATLAB is developed by MathWorks from 1984. It is a high-performance language for technical computing by using familiar mathematical notation. MATLAB has been utilized to many functions such as math computation, algorithm development, modelling, simulation, prototyping, data analysis, visualization, scientific and engineering graphics, Graphical User Interface building and many more. It is a trustable software that been used by many industries for research development and analysis. It is also be used in university as a standard instructional tool for mathematics, science and engineering base. In MATLAB there is an application-specific solution called toolboxes to solve different problems. The examples of toolboxes are such as signal processing and communication toolbox, image processing and computer vision toolbox, test and measurement toolbox, code generation toolbox, machine learning and deep learning toolbox, control system design and analysis toolbox, etc. These toolboxes help automatically generate a MATLAB program to automate the work once we get the results with a suitable algorithm with our data. The MATLAB algorithms can be converted to other programming languages such as C++, HDL, CUDA, Python to run on an embedded processor. The toolbox involved in this project is machine learning and deep learning toolbox with the Deep Network Designer apps.

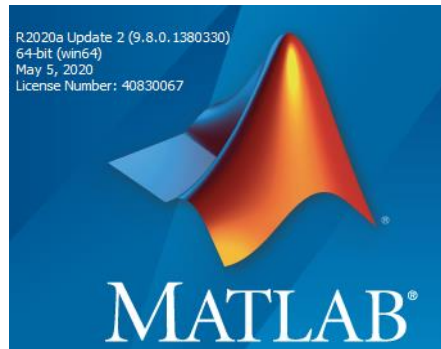


Figure 3. 3: MATLAB software.

3.5 Procedure for System Development

This section will show the interfacing between the hardware and software to sync to each other so that the system can receive the data and perform the image processing task without errors.

3.5.1 Hardware setup

The hardware setup is the step to prepare the hardware to be in a ready state before continuing to the development of programming and algorithm. The hardware including the setup of the camera to take the video clips of vehicles from the roadside and the insert of video clips to the laptop for further process.

The steps of camera setup are as below:



Figure 3. 4: Procedure for setup of camera.

The steps of transfer video clips into the laptop:



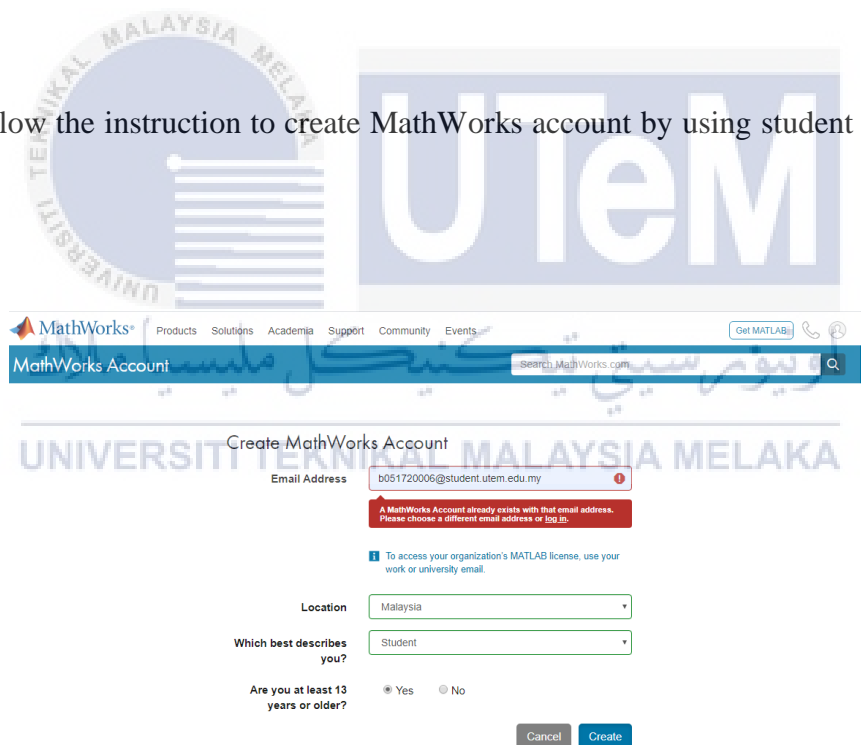
Figure 3. 5: Procedure to transfer video clips into laptop.

The video clips are now saved in the laptop and are ready for the next process of dataset preparation.

3.5.2 Software installation

MATLAB software can be downloaded from its website link at <http://www.mathworks.com>. This software is offered for industry, university and personal use and has to pay for the software. However, since UTeM has the MATLAB Campus-Wide License, the student can access freely by creating an account using student email UTeM.

Step 1: Follow the instruction to create MathWorks account by using student UTeM email address.



The screenshot displays the MathWorks account creation interface. At the top, there is a navigation bar with the MathWorks logo and links for Products, Solutions, Academia, Support, Community, and Events. Below this is a search bar and a 'Get MATLAB' button. The main heading is 'Create MathWorks Account'. The form includes an 'Email Address' field with the value 'b051720006@student.utem.edu.my' and a red error message: 'A MathWorks Account already exists with that email address. Please choose a different email address or EDU ID.' Below the email field is an information icon and a note: 'To access your organization's MATLAB license, use your work or university email.' The 'Location' dropdown menu is set to 'Malaysia'. The 'Which best describes you?' dropdown menu is set to 'Student'. At the bottom, there are radio buttons for 'Are you at least 13 years or older?' with 'Yes' selected. The page also features the MathWorks logo and navigation links like 'Products', 'Solutions', 'Academia', 'Support', 'Community', and 'Events'.

Figure 3. 6: Create account.

Step 2: Sign in to download the MATLAB latest version in GET MATLAB. Version for this project is using R2020a. Clicks installer for Windows.

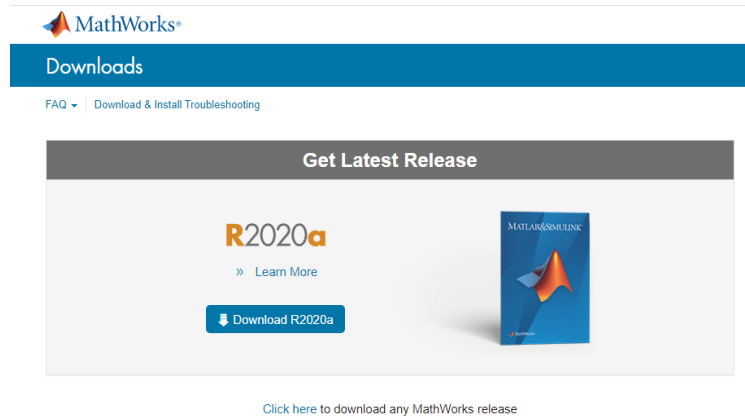


Figure 3. 7: Download the MATLAB latest version R2020a.

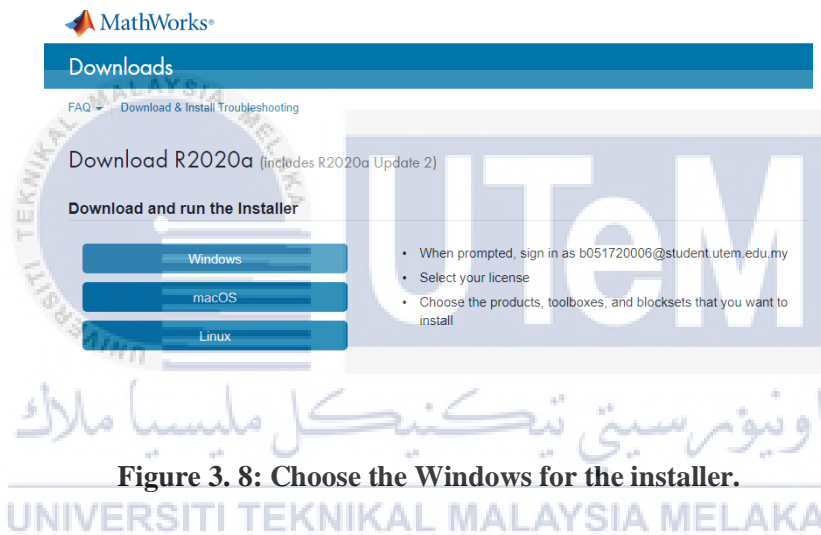


Figure 3. 8: Choose the Windows for the installer.

Step 3: Open the downloaded file and install the MATLAB software in the laptop by following the instruction.

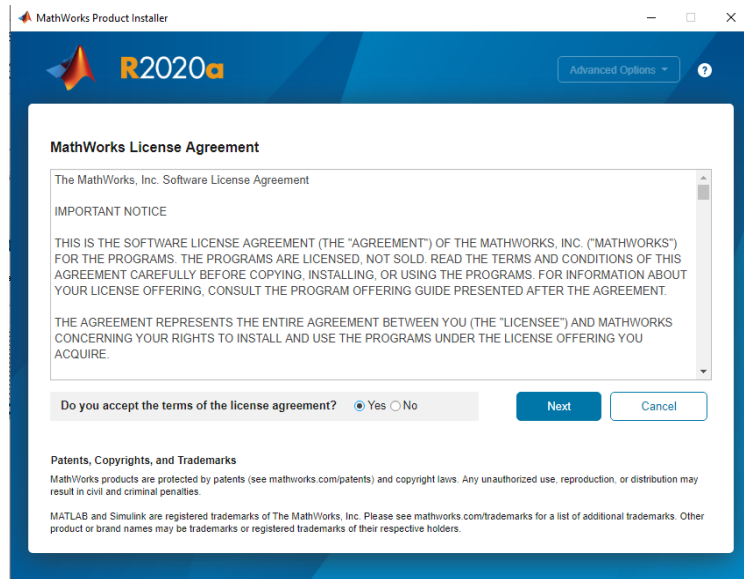


Figure 3. 9: Accept the license agreement.



Figure 3. 10: Follow the instruction from licensing to confirmation.

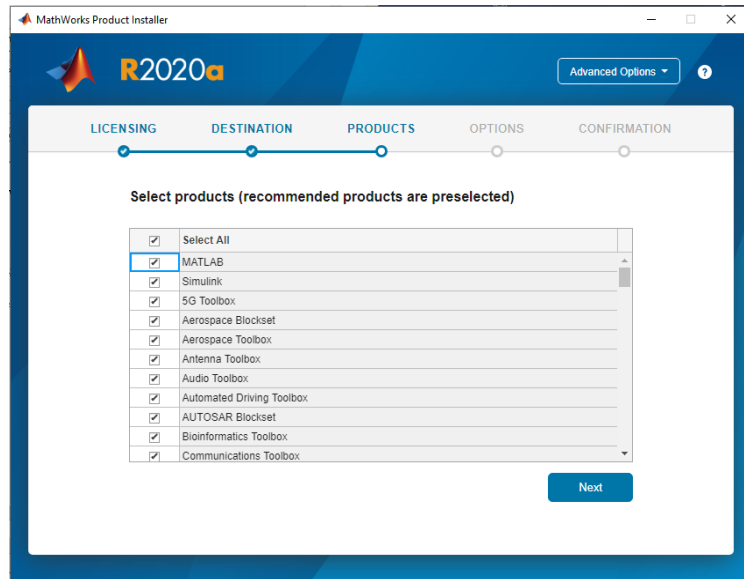


Figure 3. 11: Select all product. Make sure the needed apps and toolbox are downloaded.



Figure 3. 12: Click install at confirmation.

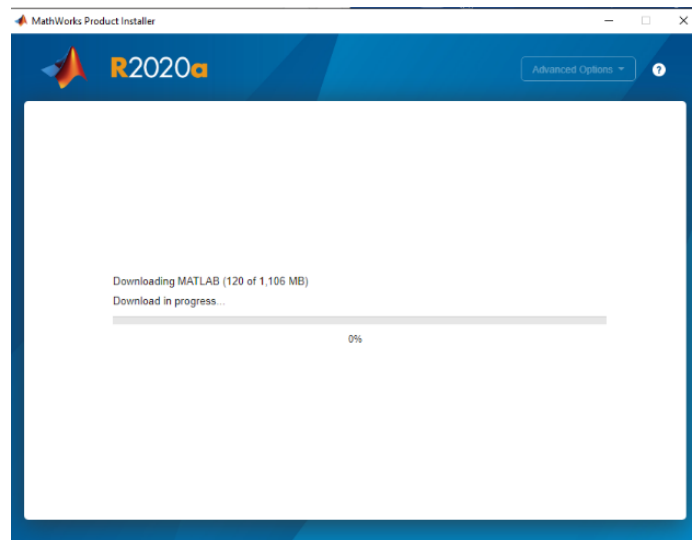


Figure 3. 13: Start installing the MATLAB software.

After done the installation, the MATLAB icon will appear on the desktop.

3.6 Programming Development

Programming development is a process where designing the algorithm for image processing on vehicle detection and classification. This process will be carried out in MATLAB software. A draft of the sequence is first generated and tested to make sure the flow of the sequence is smooth and simplify to achieve the objective of this project. The sequence of the image processing flow is in Figure 3.14. It starts with collect the dataset then followed by image processing using the pre-trained neural network which is done in MATLAB and lastly testing the trained neural network in terms of accuracy for detection and classification types of vehicle.

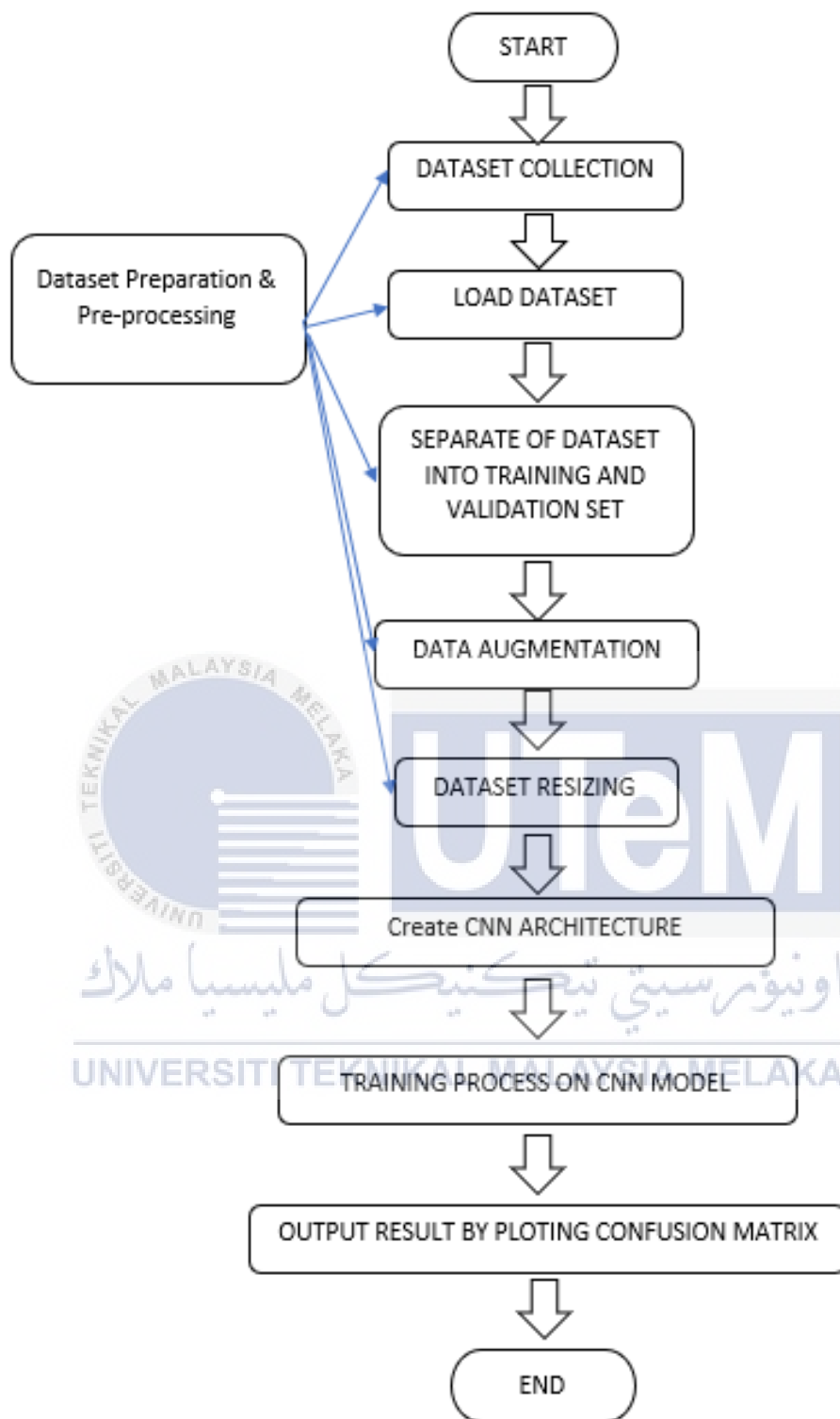


Figure 3. 14: Flow chart for image processing.

3.6.1 Dataset preparation and pre-processing

The input data for developing the image processing of vehicle is called dataset. The dataset must contain the images with non-vehicle to train the model to differentiate between the condition that contains no vehicle or contain the three types of the vehicle. The size of the dataset is set to be in between hundreds. The angle of the vehicle to be trained and classified is limited to three angles which are 45° from the front view of the vehicle, side view of the vehicle and 45° from back view of the vehicle.

There are two datasets for this project. The training and validation dataset for train the CNN model while the testing dataset for test the developed algorithm. To ensure the collection of the dataset for training and validation is correct in term of the categories of the vehicle into MPV, SUV and Sedan, a survey on the model of the vehicle correspond to the MPV, SUV and Sedan has been made. Five top brands of vehicles sales in 2019 which are the Perodua, Proton, Toyota, Honda and Nissan are the most popular car brand that Malaysia people will choose when purchasing a new car.

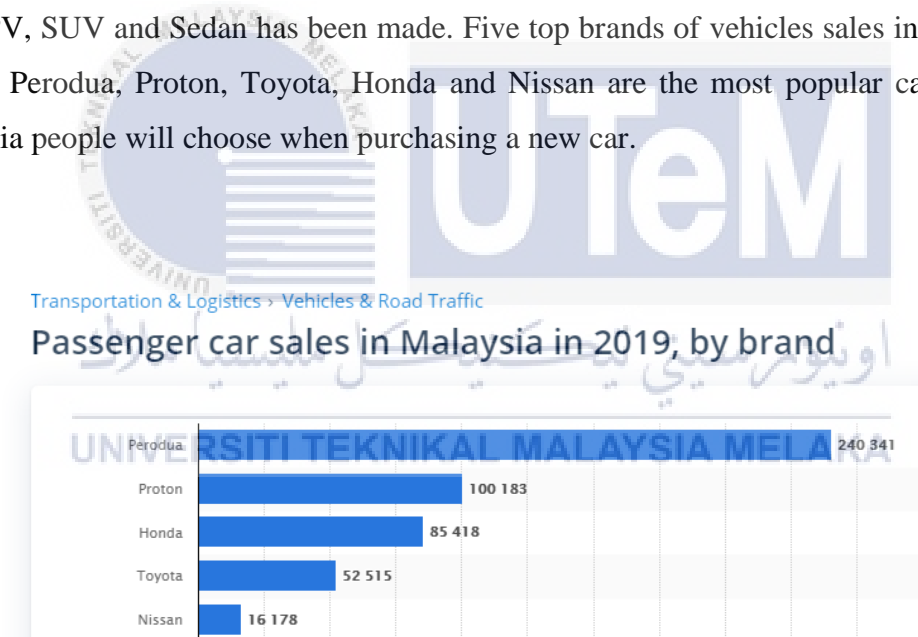


Figure 3. 15: Top five most sales brand of car in 2019.

[\(https://www.statista.com/statistics/516276/passenger-vehicle-sales-in-malaysia-by-brand/\)](https://www.statista.com/statistics/516276/passenger-vehicle-sales-in-malaysia-by-brand/)

From the top five brands of vehicle, the models of MPV, SUV and Sedan have extracted out according to the CarBase.my website. CarBase.my is a website in Malaysia that provides a comprehensive database of the vehicle that currently sale in Malaysia. The database including the prices, specifications, body types, brands and images of vehicles.

The list of the vehicle model for Sedan, SUV and MPV for the top five brands is tabulated in Table 3.3.

Table 3. 3: List of Vehicle Model.

Brand	Sedan	SUV	MPV
Perodua			
	Bezza	Aruz	Alza
Proton			
	Saga Persona Perdana* Inspira* Preve	X50 X70	Ertiga* Exora
Toyota			
	Vios Corolla Altis Camry	Rush C-HR Fortuner RAV4 Harrier	Avanza Innova Vellfire Alphard Previa*
Honda			
	City Civic Accord	BR-V HR-V CR-V	Odyssey Stream*
Nissan			
	Almera Teana Sylphy* Latio sedan*	X-Trail Murano*	Serena S-Hybrid Grand Livina* Elgrand*

The collection of vehicle images is according to the model listed above. The images are either downloaded from CarBase.my or downloaded from google search. The background of images most probably contains a real-life situation. The images for category Detect No Vehicle are all downloaded from google and most probably are images of the roadside. Total of 396 images is collected for 4 classes for training and validation dataset.



Figure 3. 16: Detect No Vehicle category.

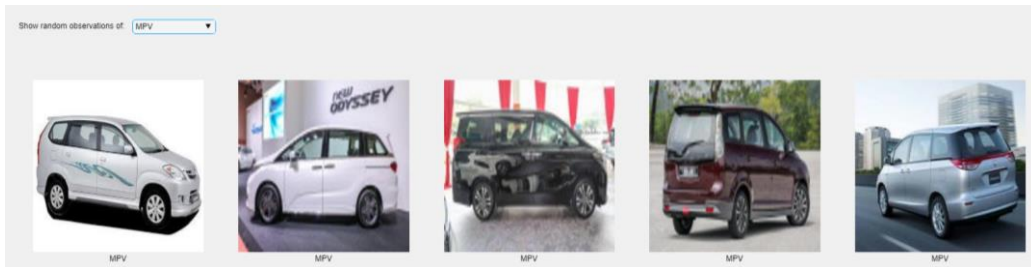


Figure 3. 17: MPV category.



Figure 3. 18: SUV category.

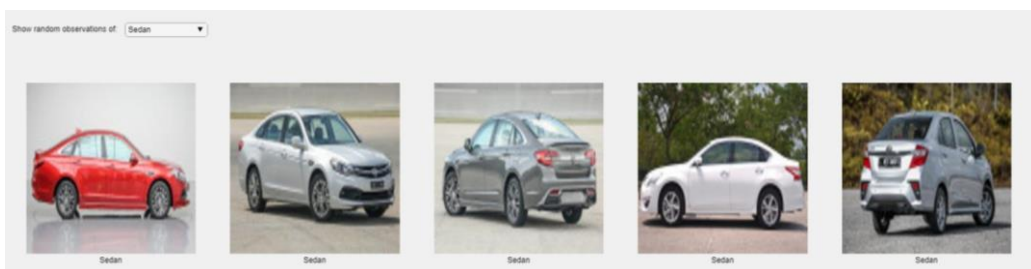


Figure 3. 19: Sedan category.

The testing dataset is collected by using the camera to record video clips at the roadside in the housing area. The dataset collection will be done at the day time to ensure the lighting is sufficient to produce clear and good condition dataset. After that, manually

snapshot of the frame that contains the images for non-vehicle and image that contain the vehicle for MPV, SUV and Sedan. Then the region of interest is cropped out.



Figure 3. 20: Example of testing dataset.

Total of 120 images is collected for testing dataset. The images for three categories of the vehicle are saved in three subfolders in one main folder. The name of subfolder is the four categories of the vehicle which are Detect No Vehicle, MPV, SUV and Sedan. This is same as both training and validation dataset and testing dataset.

This whole training and validation dataset is loaded to MATLAB and be separated into a training dataset and validation dataset with the ratio of 8:2 to train and validate the algorithm. Total of images 317 are used for training and the rest of 79 images are used for validation. The testing dataset is loaded into MATLAB during the testing of the algorithm. Both datasets are balanced in amount for each category.

Table 3. 4: Dataset amount for each dataset.

Categories	From Google		From offline video
	Training Dataset	Validation Dataset	Testing Dataset
Detect No Vehicle	79	20	30
MPV	79	20	30
SUV	79	20	30
Sedan	79	20	30

Data augmentation is a process to improve the network accuracy by adding more variety to training data through techniques of such as flipping, rotation and many more mention in chapter 2. Thus, data augmentation is saving time and can increase accuracy at the same time. The data augmentation is done for training data only. In this project, only the reflection of training data in x-axis which is flipping in left or right is adopted because not all technique will be suitable for a different type of dataset. Some can cause a decrease in the accuracy of the result. Since the collected datasets are all various in size, therefore, both the training dataset and validation dataset are then resized to image size that required by the input layer from CNN architectural.

3.6.2 Create CNN algorithm

Pretrained neural network is selected to be adopted in this project. As mention in chapter 2, there are two types of training which are training from scratch and through transfer learning. Both types of training have an advantage and disadvantage. It depends on the situation. In this project, due to time constraint and hardware limitation, the dataset used is on the range of hundreds. In this case, the transfer learning will well suit if compared to training from scratch since transfer learning required a smaller number of training images while training from scratch required for thousand or million number of training images. Transfer learning is referred to use of the pre-trained network that been trained on a large image dataset and then transfer the learnt rich feature to learn on a new task. The pre-trained neural networks that been chosen for this project are SqueezeNet and GoogleNet. Both pre-trained neural networks are very well known and its functionality can be proved from past research.

3.6.2.1 SqueezeNet

SqueezeNet was proposed by Iandola *et al.* (2017). It is a small CNN architecture that was a total of 18 depth. This CNN model can achieve the same accuracy as AlexNet with 50 x lesser parameter. The size of SqueezeNet is 0.5 MB which is 510x smaller than AlexNet. The input size for SqueezeNet is 227 x 227 x 3. Three strategies had been

introduced by the author to reduce the CNN parameter while maintaining accuracy. The first strategy is to use 1×1 filter for most of the convolutional layers as this filter size has 9X fewer parameter. The second strategy is to decrease the number of input channels to filter size of 3×3 through the Squeeze layer. The third strategy is having large activations maps through the technique of downsampling late such as maintain the stride in the network to increase the classification accuracy. Fire module is a combination of squeeze convolution layer with 1×1 filter that feeds into the expand layer that consists of 1×1 filter size and 3×3 filter size of convolution layer. The fire module introduces the small filter size and limits the number of input channels to filter size of 3×3 . There are 8 fire modules in SqueezeNet that serve as the basic building block for SqueezeNet model. Figure 3.21 shows the fire module.

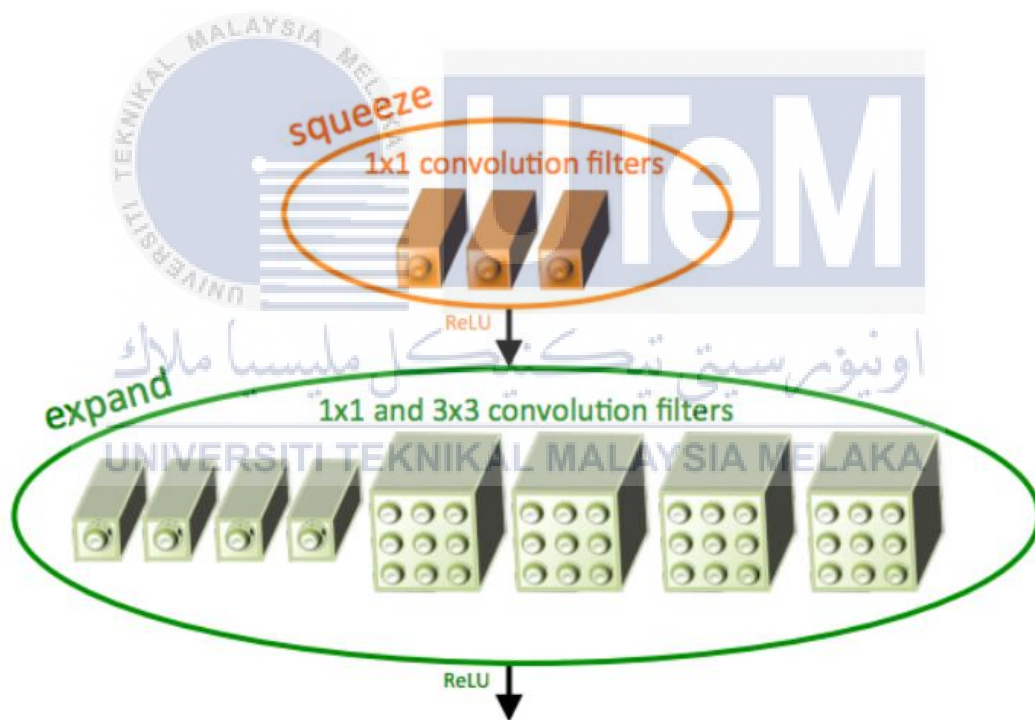


Figure 3. 21: Fire module.

In this project, the parameters of all layers of SqueezeNet are frozen, and all the parameters of the last convolution layer and the classification layer are removed. Then new convolution layer with the filter size of 1×1 and the number of filters of 4 is replaced. The weight and bias learn rate are both set to 10 times greater so that the new layers learn faster than the transferred layers. The new classification layer is replaced to make sure the output

size equal to the number of class to be classified. The detail of SqueezeNet architecture is attached in Appendix B.

3.6.2.2 GoogleNet

GoogleNet is introduced by Christian Szegedy *et al.* (2015) and had won the ILSVRC at 2014. . GoogleNet has a depth of 22 layers for convolution, size of 27MB and 7M parameters. GoogleNet is faster and less parameter but with better accuracy compared to AlexNet. The main bone in GoogleNet is inception layer. GoogleNet uses the concept of inception layer to cover the bigger area but at the same time maintains fine resolution for small information images. In the inception layer, the previous layer will be convolved in parallel with different sizes of the filter from small to bigger such as 1 x 1, 3 x 3 and 5 x 5. The max pooling layer and 1 x1 convolution layer are added to reduce the dimension of images and thus reduce the parameter. A concatenated layer is added at the end of inception layer to stack the output. There are 9 inception modules in GoogleNet followed with global average pooling at the near end of the network to average the feature map to 1 x 1 from 7 x 7, There is a dropout layer with 0.4 to reduce the overfitting.

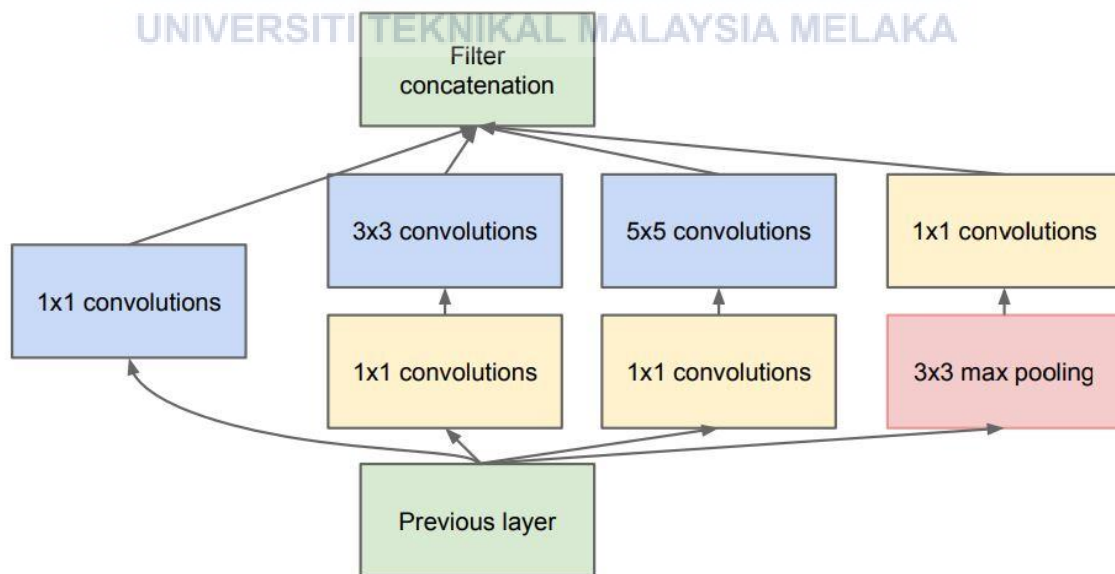


Figure 3. 22: Inception module in GoogleNet.

In this project, the parameters of all layers of GoogleNet are frozen, and all the parameters of the last fully connected layer and the classification layer are removed. Then new fully connected layer is replaced and the output size is set to 4 which are the number of classes to be classified. The weight and bias learn rate are both set to 10 times greater so that the new layers learn faster than in the transferred layers. The new classification layer is replaced to make sure the output size equal to number of class to be classified. The detail of GoogleNet architecture is attached in Appendix C.

3.6.3 Experimental setup for training options

Fine-tune of the pre-trained network in terms of hyperparameter is needed so that the model can well suit to the dataset to obtain the highest accuracy. The hyperparameter such as initial learning rate, max epochs and minibatch size is mainly focused on the tuning. Table 3.5 shows the range for hyperparameter to be tuned. The range is selected from the initial value given from MATLAB and adjusted the value of these three hyperparameter according to the theory from past research. The final chosen hyperparameter value for initial learning rate is 0.0001, the max epoch is 25, and minibatch size is 32. The other training options are tabulated in table 3.5.

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

Table 3. 5: Training options.

Training Options	Value
Solver	sdgm
Execution Environment	CPU
Initial Learn Rate	0.0001
Max Epochs	25
Mini Batch Size	32
Momentum	0.9
L2Regularization	0.0001
Shuffle	Every epoch
Validation Frequency	5
Validation Data	augimdsValidation
Plots	Training Progress

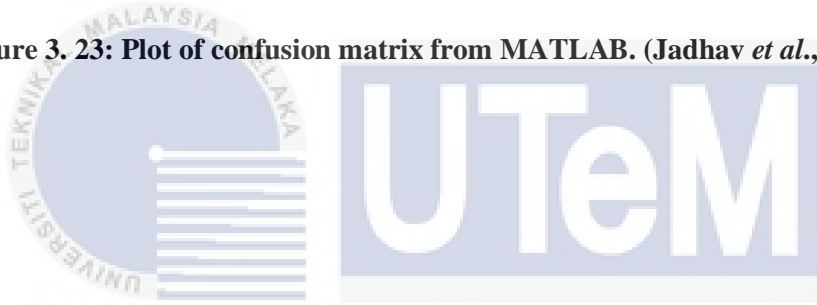
3.6.4 Output result

The output result is displayed through the plot of the confusion matrix. The confusion matrix is a table that displays the performance of the CNN model. It is easy to understand because it makes the direct comparison of true positive, false positive, true negative and false negative to be visualized. It also outputs the precision, recall and overall accuracy in percentage.

In the confusion matrix table, the rows are the output class which is also called predicted class. The columns are the target class which is also called the true class. The diagonal in green colour shows the number of images that been correctly classified in that category. The off-diagonal in pink colour shows the incorrectly classified observation. The column on the far right of the table shows the positive predictive value and false discovery rate in percentages of all the prediction belong to each category is correctly and incorrectly classified. It is also called precision. The row on the bottom of the table shows the true positive rate and false-negative rate in percentages of all the images belong to each category is correctly and incorrectly classified. It is also called the recall. The overall accuracy is showed on the most right below grey colour box. Figure 3.23 shows the example plot of confusion matrix in MATLAB (Jadhav *et al.*, 2020).



Figure 3. 23: Plot of confusion matrix from MATLAB. (Jadhav *et al.*, 2020)



3.7 Testing

After done the training process, the trained CNN is further test on the testing dataset to validate the performance. The performance is measured by using the plot of confusion matrix to show the ability of the trained CNN to find all the relevant vehicle and perform correct classification in real-life static vehicle picture. The testing dataset consists of total 120 images. Each of the images contains one vehicle. Each category consists of 30 images. This test dataset is passed through the trained CNN one time per one image to collect the result and time taken for processing one image. The result is recorded in a table and is tabulated through the plot of a confusion matrix manually. Table 3.6 shows the example of the confusion matrix that interpret on class Detect No Vehicle. The precision, recall and overall accuracy are calculated using the formula stated in Figure 3.24. The TP is True Positive means the number of real positive image of Detect No Vehicle, FP is False Positive means the actual target is false but the output prediction is positive, while FN is False Negative means the actual target is true but the output prediction is false.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$

$$\text{Recall} = \frac{TP}{TP + FN},$$

$$\text{Precision} = \frac{TP}{TP + FP},$$

Figure 3. 24: Formula for accuracy, recall and precision.

Table 3. 6: Example of confusion matrix – Interpret on Detect No Vehicle class.

Example		Target Class				Precision (%)
		Detect No Vehicle	MPV	SUV	Sedan	
Output Class	Detect No Vehicle	TP	FP	FP	FP	Precision Detect No Vehicle
	MPV	FN				Precision MPV
	SUV	FN				Precision SUV
	Sedan	FN				Precision Sedan
Recall (%)		Recall Detect No Vehicle	Recall MPV	Recall SUV	Recall Sedan	Accuracy (%)

3.8 Data Analysis and Validation

The data analysis is done by comparing the result collected from the validation dataset and testing dataset between the two pre-trained CNN models. The comparison and analysis are made in term of performance measurement such as precision, recall and

overall accuracy. The training time for both pre-trained CNN models are also been compared and analysed based on the plot of the graph during training progress. The average computational time per image is calculated to compare between two pre-trained CNN models on how much time has been used to process one image. The comparison and analysis of results are aimed to find out which pre-trained CNN model is performed better and stable in terms of the performance measurement and computational time to judge for the reliability and suitability of the algorithm developed to be implemented for application in a surveillance system.

3.9 Programming Code

Programming is the core of the software developing. The programming code for this project is divided into eight parts. The eight parts are consist of the load initial parameter, import dataset, set training options, create the CNN algorithm, train CNN models, create confusion matrix, load net and test CNN models with the testing dataset. The Deep Network Designer apps in Machine Learning and Deep Learning toolbox is used in this project. The upload of training and validation dataset, the modification of CNN models, the set of hyperparameter and the initial training of models are done in this app. The MATLAB programming code can be auto-generated after the first training in the app. The programming code then saved and ran in the live editor in MATLAB for ease of editing and displaying the result. The complete programming code for this project is attached in the Appendix D and Appendix E

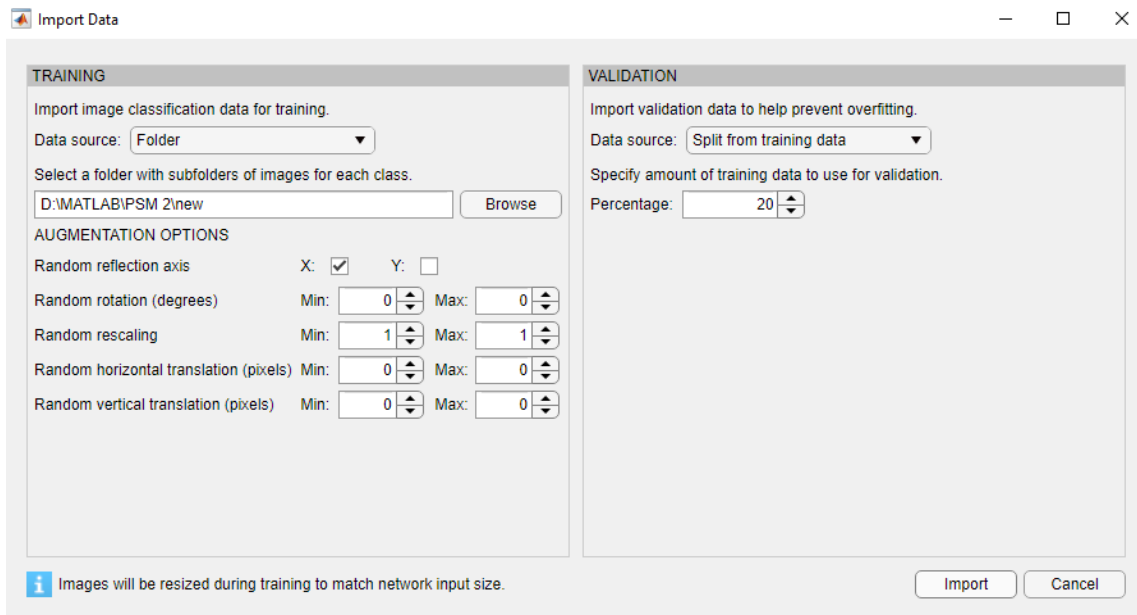


Figure 3. 25: Import training and validation dataset.

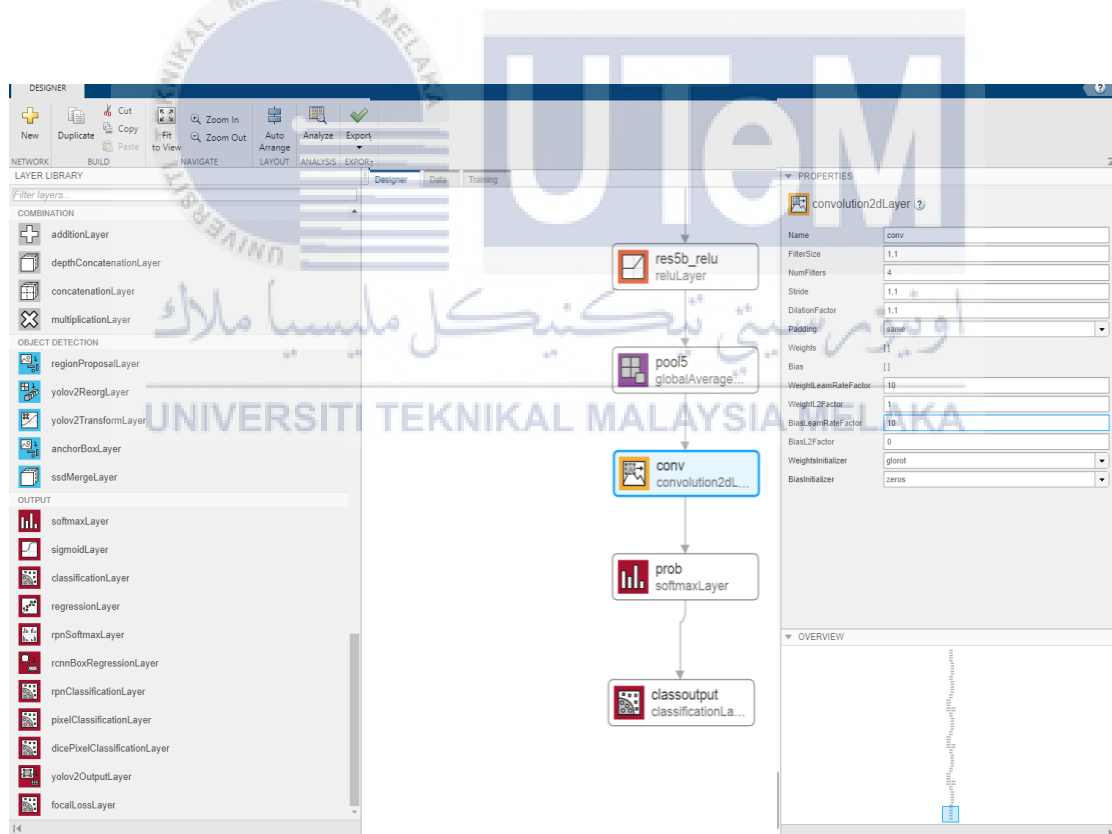


Figure 3. 26: Modify the convolution layer and classification layer.

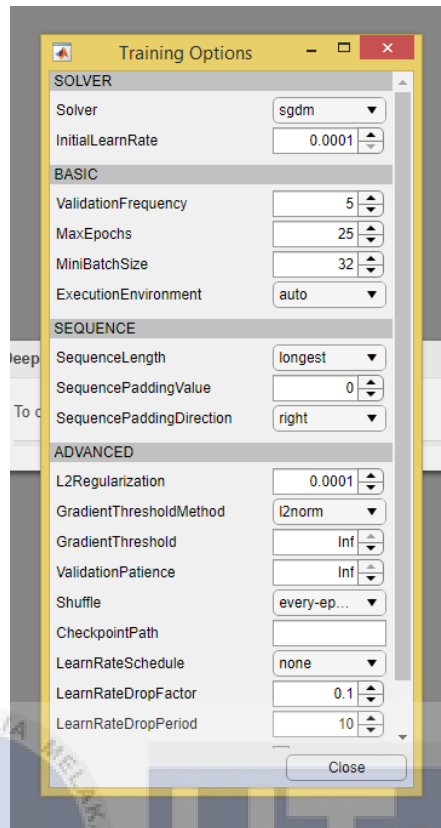


Figure 3. 27: Change the hyperparameter.

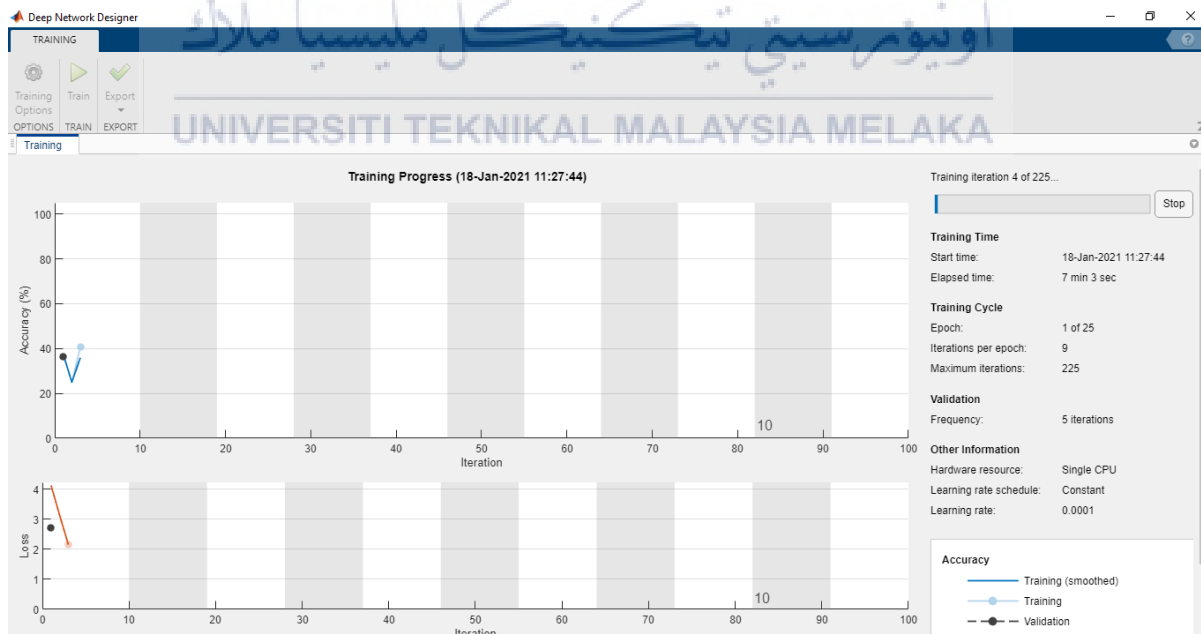


Figure 3. 28: Example of training in progress.

Table 3. 7: Explanation of programming code for each part.

No	Part	Function Code
1	Load Initial Parameter	<pre>trainingSetup = load("C:\Users\Asus\Documents\MATLAB\PSM 2\trainingSetup_2021_01_06_21_22_07.mat");</pre> <p>The trainingSetup is used to load the parameters of the initial pre-trained network for transfer learning. It is downloaded and saved in computer file when downloaded of the pre-trained network.</p>
2	Import Data	<pre>imdsTrain = imageDatastore("D:\MATLAB\PSM 2\new","IncludeSubfolders",true,"LabelSource","foldernames"); [imdsTrain,imdsValidation] = splitEachLabel(imdsTrain,0.8); imageAugmenter = imageDataAugmenter(... 'RandXReflection',true); augimdsTrain = augmentedImageDatastore([227 227 3],imdsTrain,"DataAugmentation",imageAugmenter); augimdsValidation = augmentedImageDatastore([227 227 3],imdsValidation);</pre> <p>The dataset of vehicles that prepared in the folder is imported into by using the imageDatastore function. This function is used to create an object to manage the collection of images' dataset. The splitEachLabel function is used to split the imgaeDatastore into train image datastore and validation image datastore with the ratio of 8:2 randomly. The imageDataAugmenter function is used to augment the training image datastore to increase the images of training images through the reflection of images on X-axis. The augmentedImageDatastore function is used to create an object for storing the resize of training dataset and validation dataset. Both datasets are resized according to the input size of the CNN model. The object augimdsTrain is used for the training dataset and object augimdsValidation is used for validation dataset during the training process.</p>
3	Set Training Options	<pre>opts = trainingOptions("sgdm",... "ExecutionEnvironment","auto",... "InitialLearnRate",0.0001,... "MaxEpochs",25,... "MiniBatchSize",32,... "Momentum", 0.9,... "L2Regularization",0.0001,... "Shuffle","every-epoch",... "ValidationFrequency",5,... "Plots","training-progress",...</pre>

		<pre>"ValidationData",augimdsValidation);</pre> <p>The training options are set by using the trainingOptions function. All values in the hyperparameter are edited inside this code to fine-tune the CNN model.</p>
4	Create CNN Model	<pre>lgraph = layerGraph(); tempLayers = [*linear array of layers]; lgraph = addLayers(lgraph,tempLayers); lgraph = connectLayers(lgraph,"fire2-relu_squeeze1x1","fire2-expand1x1");</pre> <p>layerGraph() is the main function to create the layer graph variable that contains all the network layers. Each branch is consists of a linear array of layers using the function tempLayers[]. Then add the branches of the network to the layer graph using function addLayers. Lastly, connect all the branches of the network to create the network graph using the connectLayers function. The full code is attached in the Appendix.</p>
5	Train Network	<pre>[squeezeenet, traininfo] = trainNetwork(augimdsTrain,lgraph,opts); save squeezeenet</pre> <p>The network is trained by using the trainNetwork function. The input argument for this function is the objects of augimdsTrain, lgraph and opts set from the previous. The training info is saved in name squeezeenet or googlenet based on the CNN model. The net is saved to ensure next time open will not need to rerun the programme.</p>
6	Create Confusion Matrix	<pre>YPred = classify(squeezeenet, augimdsValidation); YTest = imdsValidation.Labels; figure, plotconfusion(YTest, YPred); confMat = confusionmat(YTest, YPred); confMat = bsxfun(@rdivide, confMat, sum(confMat,2)) accuracy = mean(diag(confMat)*100); disp('accuracy') disp(accuracy)</pre> <p>The function classify is used to classify the validation dataset by using the saved net. YTest is the target class. Then the confusion matrix is plotted and displayed in the figure. The accuracy is calculated using the mean of the value in diagonal in confusion matrix and multiply with 100 to get the percentage. Then displayed the accuracy.</p>
7	Load Network	<pre>% load squeezeenet</pre> <p>Load the saved squeezeenet when next time opens the MATLAB. This code is an</p>

		option where it will only be used when needed to load the saved net into the workplace in MATLAB.
8	Test Network	<pre> I = imread('D:\MATLAB\PSM 2\DatasetTest\Sedan\30.png'); tic; I = imresize(I, [227 227], 'nearest'); [label,scores] = classify(squeezenet,I); scores = max(double(scores*100)); time=toc; figure imshow(I) title(join([string(label),",scores,'% '])) </pre> <p>The imread function is used to read one image from the written path in laptop. Change the path to change to the next image for testing. The tic is used to calculate time start at the resizing process. The image is resized to the required input size using the imresize function. The classify function is used to perform the classification task on the resize image by using the net saved. The scores is multiplied with 100 to calculate for the percentage. The toc is used to stop the time calculated from tic. The period between the tic and toc is the processing time for one image. It then outputs the label and scores for the processed image using imshow(I) function. The title is displayed together with the scores.</p>

3.10 Summary

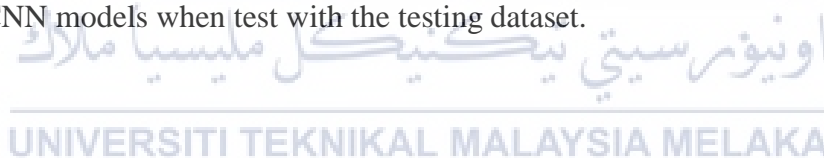
In summarize, this chapter has provided the overall flow of project planning start from the beginning until the end on how to achieve the objectives for this project. Each process has been explained and discussed deeply to has a better understanding of how it related to each other. With the proper plan of methodology, the project start from the literature review, project planning, system development, testing of the system and lastly with data analysis can be carried out accordingly to the schedule. This is important so that the project will not take longer time to finish. Lastly, the methodology is a guideline to set up the experiment to gather data for analysis that will be used in the next chapter.

CHAPTER 4

RESULT AND DISCUSSION

4.0 Introduction

This chapter will discuss the results obtained from the work done according to the methodology in chapter 3. This chapter will be divided into two parts where the first part will discuss the result obtained from the training of both the CNN models which are the SqueezeNet and GoogleNet. While the second part will discuss the validation of result get from both CNN models when test with the testing dataset.



4.1 Result for Validation Dataset

Transfer learning technique has been implemented to train on the selected CNN algorithm to perform the new task on vehicle detection and classification. The two pre-trained CNN algorithms are SqueezeNet and GoogleNet. The training progress is plotted and the result of the training are shown by using a confusion matrix. Many information can be extracted from the confusion matrix. The green colour indicates the true positive, the horizontal orange colour indicates the false positive while vertical orange colour indicates the false negatives. From the true positive, false positive and false negative, precision and recall can be calculated. The vertical light grey indicates the precision while the horizontal light grey colour indicates the recall and lastly the grey colour is the overall accuracy.

Figure 4.1 shows the confusion matrix of validation dataset for SqueezeNet model and Figure 4.2 shows the confusion matrix of validation dataset for GoogleNet.

		Confusion Matrix					
Output Class	Detect No Vehicle	20 25.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%	
	MPV	0 0.0%	19 23.8%	1 1.3%	3 3.8%	82.6% 17.4%	
	SUV	0 0.0%	1 1.3%	19 23.8%	1 1.3%	90.5% 9.5%	
	Sedan	0 0.0%	0 0.0%	0 0.0%	16 20.0%	100% 0.0%	
		100% 0.0%	95.0% 5.0%	95.0% 5.0%	80.0% 20.0%	92.5% 7.5%	
		Target Class					
		Detect No Vehicle	MPV	SUV	Sedan		

Figure 4. 1: Confusion matrix of validation dataset by SqueezeNet.

From Figure 4.1, the confusion matrix shows that 19 out of 20 images for category MPV vehicles are classified correctly. Only one image is classified wrongly to be SUV. For SUV, 19 out of 20 images of SUV vehicles are classified correctly and only one image is classified wrongly as MPV. For Sedan, 16 out of 20 images sedan vehicles are classified correctly. 3 images are classified wrong as MPV and one image is classified wrongly as SUV. The images for Detect No Vehicle category are all classified correctly. Table 4.1 shows the calculation for precision and recall. This data will discuss in next subtopic for precision and recall.

Table 4. 1: Calculation of precision and recall - SqueezeNet.

Class	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision (TP/(TP+FP)) *100 (%)	Recall (TP/(TP+FN)) *100 (%)
Detect No Vehicle	20	0	0	100	100
MPV	19	4	1	82.6	95
SUV	19	2	1	90	95
Sedan	16	0	4	100	80

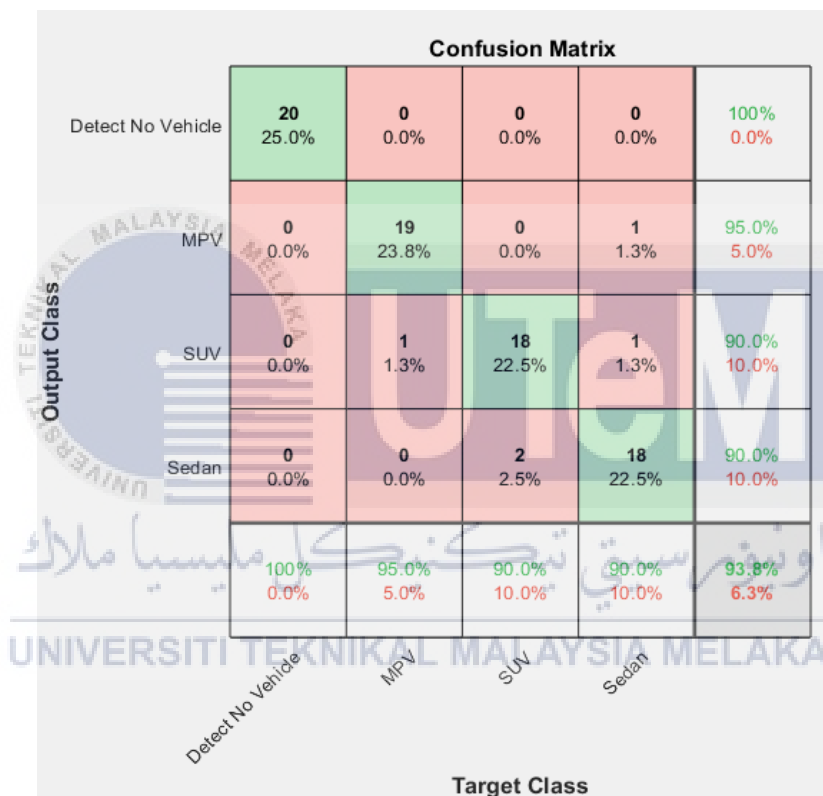


Figure 4. 2: Confusion matrix of validation dataset by GoogleNet.

From Figure 4.2, the confusion matrix shows that 19 out of 20 images for category MPV vehicles are classified correctly. Only one image is classified wrongly to be SUV. For SUV, 18 out of 20 images of SUV vehicles are classified correctly and two images are classified wrongly as Sedan. For Sedan, 18 out of 20 images sedan vehicles are classified correctly. One image is classified wrong as MPV and one image is classified wrongly as SUV. The images for detect no vehicle category are all classified correctly. Table 4.2 shows the calculation for precision and recall. This data will discuss in next subtopic.

Table 4. 2: Calculation of precision and recall – GoogleNet.

Class	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision $(TP/(TP+FP)) * 100$ (%)	Recall $(TP/(TP+FN)) * 100$ (%)
Detect No Vehicle	20	0	0	100	100
MPV	19	1	1	95	95
SUV	18	2	2	90	90
Sedan	18	2	2	90	90

4.1.1 Comparison of precision

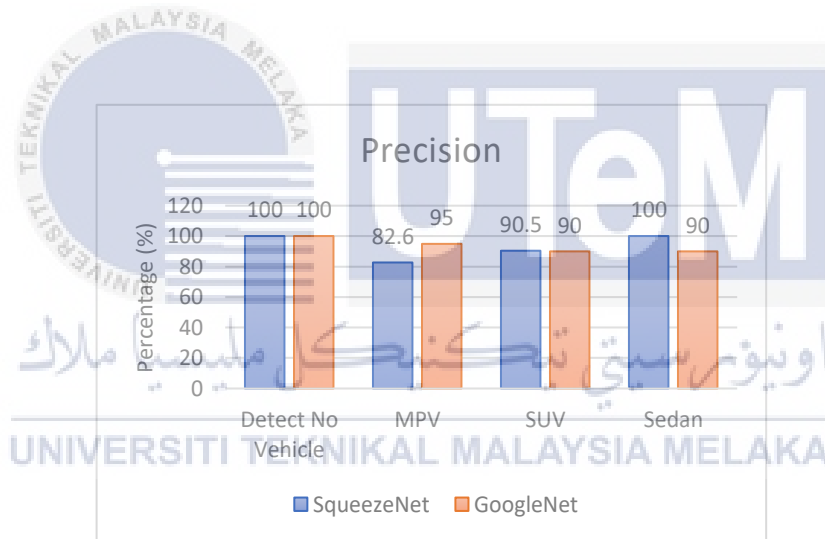


Figure 4. 3: Histogram graph for comparison of precision between two CNN models.

The precision from Table 4.1 and Table 4.2 are being transferred and compared using a histogram. Figure 4.3 shows the histogram for comparison of precision between two CNN model. Based on the histogram, the precision percentage of Detect No Vehicle class is 100% for both SqueezeNet and GoogleNet. The precision percentage of MPV class is 82.6% for SqueezeNet and 95% for GoogleNet. For SUV class, both CNN models scored almost same precision where SqueezeNet with 90.5% which is only 0.5% higher than the precision percentage for GoogleNet. Similarly, for Sedan class, the

SqueezeNet scored better in precision with 100% which is 10% higher than the precision percentage for SqueezeNet.

In precision, the higher the precision percentage, the more accurate the prediction, the lesser the error to classify the images wrongly. The 100% precision fall on Detect No Vehicle class, this is due to the images for this class are having less similarity among the other three class. The images are mostly the road pictures with grass and sky. There is not any vehicle in those images. Therefore, the precision percentage is high due to its easiness to recognize and differentiate between this class with other class that has the vehicle object. While MPV class has the lowest precision in SqueezeNet. This means 19/23 of the images that the predictor classifies as MPV are actually MPV. There are extra 4 images are classified wrongly by the predictor and three images come from Sedan class. This might due to there are few bad images in Sedan class that can make the CNN models misunderstanding on the features of images given.

4.1.2 Comparison of recall

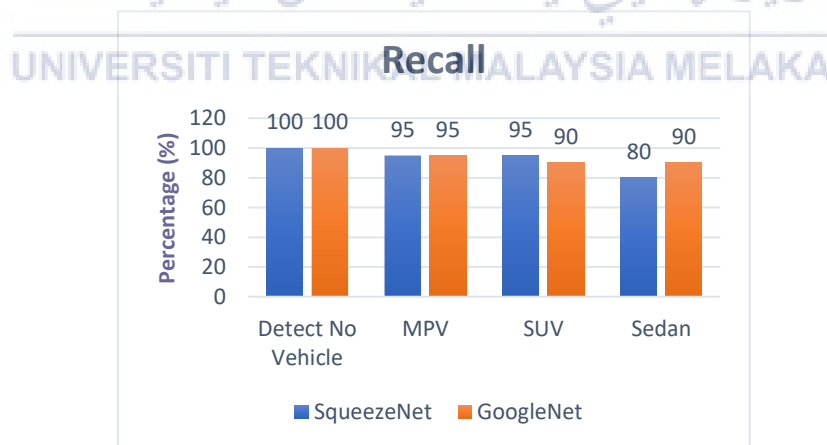


Figure 4. 4: Histogram graph for comparison of recall between two CNN models.

The recall from Table 4.1 and Table 4.2 are being transferred and compared using a histogram. Figure 4.8 shows the histogram for comparison of recall between the two CNN models. Based on the histogram, the recall percentage of Detect No Vehicle

class is 100% for both SqueezeNet and GoogleNet. Similarly, the recall percentage of MPV class is 95% for both SqueezeNet and GoogleNet. For SUV class, the SqueezeNet scored better in recall with 95% which is 5% higher than the recall percentage for GoogleNet. In vice versa, for Sedan class, the GoogleNet scored better in recall with 90% which is 10% higher than the recall percentage for SqueezeNet.

For both CNN models, the Sedan class has the lowest recall over the other three class. An assumption made in this scenario is there are some bad images in Sedan class that made the classifier hard to predict it into the right class. Figure 4.5 below shows some of the bad images. Some reasons can cause the happening of bad images in the dataset. It might be because during the dataset pre-processing where the resize of the images into the required input size of the CNN model. The images (a) may be stretched and caused to the important features in Sedan go different and thus result in false prediction. Some of the images of sedan type vehicle, (b) and (d) are not in full body size. Thus, the features in these images are not completed and might not be able to classify as Sedan class. The human mistake in sorting the front view vehicle in Sedan class (c). Those bad images can cause to the misunderstanding of the CNN model to classify those images as MPV or SUV.

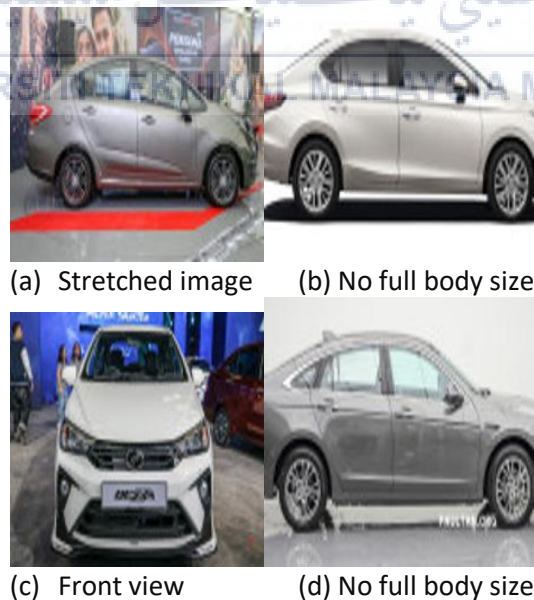


Figure 4. 5: Example of bad images in the Sedan class.

4.1.3 Comparison of accuracy and training time

During the training of CNN models, the accuracy for the training dataset and validation dataset has been plotted out in the training progress graph. The blue colour line is the training accuracy while the black line with dots is the validation accuracy. From the training progress graph, whether the training is overfitting or not can be detected. The validation frequency in plotting the accuracy is per 5 iterations. Normally the accuracy of the model will be increased with iterations. The number of iterations is determined by max epoch and mini-batch size. The smaller the mini-batch size, the greater the iteration. The greater the max epoch, the greater the iterations. Greater iterations mean longer training time. In this project, the mini-batch size, max epoch and hyperparameters are set to same to ease to the comparison. Figure 4.6 and Figure 4.7 shows the training progress for SqueezeNet and GoogleNet.

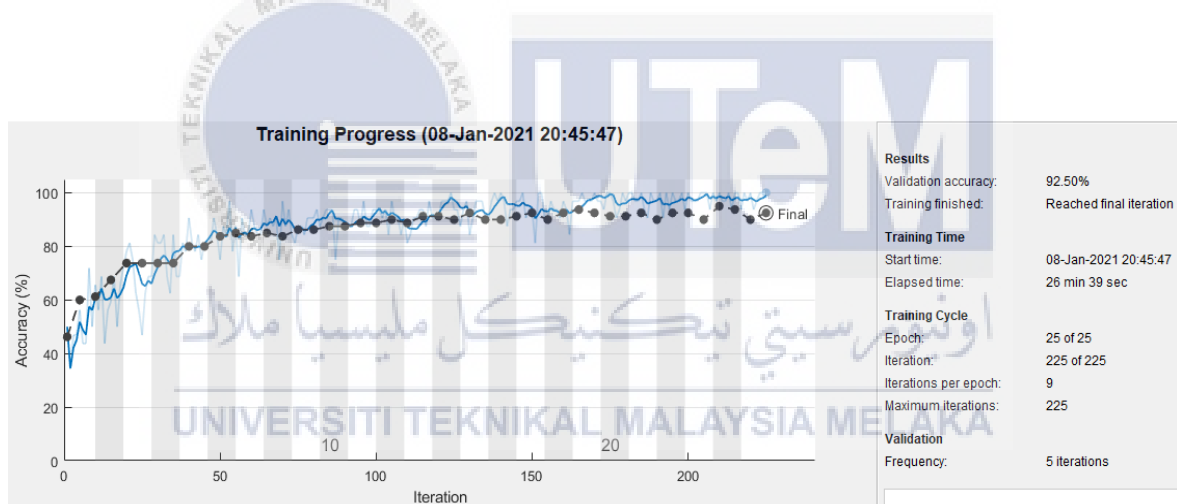


Figure 4. 6: Training process – SqueezeNet.

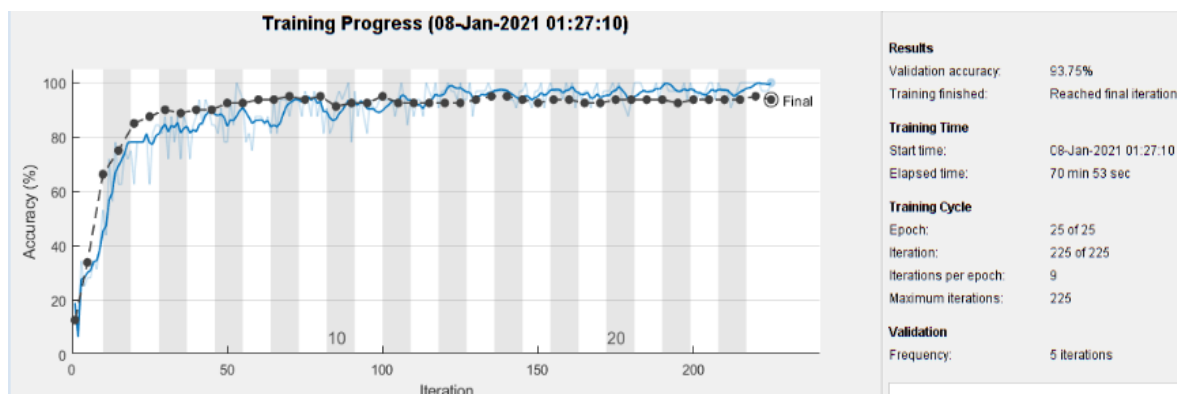


Figure 4. 7: Training process - GoogleNet

Table 4. 3: Tabulated result of accuracy and training time.

Model	Training Accuracy	Validation Accuracy	Training Time
SqueezeNet	100	92.5	26 mins 39 sec
GoogleNet	100	93.8	70 mins 53 sec

Based on the data tabulated in Table 4.3, the validation accuracy for SqueezeNet is 92.5% while for GoogleNet is 93.8%. It shows that GoogleNet has higher validation accuracy when compared to SqueezeNet. However, the difference is not significant where only 1.3% different in accuracy between both pre-trained CNN models. While for training time, with the same hyperparameter setting for the training option, the training time for SqueezeNet was 26mins 39 seconds and for GoogleNet was 70 mins 53seconds. GoogleNet took 44 mins 14 sec longer to complete the training. Due to the ease to compare the SqueezeNet and GoogleNet, the hyperparameter of the max epoch are set to be same therefore the GoogleNet continue to train even the maximum accuracy has been reached.

Based on the theory, the CNN model with deeper architecture results in better accuracy but the poor side is training time will be longer (Lee et al., 2019). The parameters of CNN architecture will also affect on the training time. The larger the parameters, the longer the training time. GoogleNet has larger parameter and greater depth when compared to SqueezeNet. Therefore, GoogleNet used longer time for training and better accuracy of the classifier is achieved (Seo & Shin, 2019).

The difference between the training accuracy and validation accuracy from the plot of training progress can indicate whether the training is overfitting (Shorten & Khoshgoftaar, 2019). For GoogleNet, this model is perfect fitting since the validation accuracy point for every 5 iterations is near to training accuracy. While for SqueezeNet, the gap between the validation accuracy and training accuracy is getting bigger when getting near to the end of the training. This shows that this model is having little overfitting where it fit nicely to training data but slightly poor on validation data.

4.2 Result on Testing Dataset

The trained CNN models are further tested on the testing dataset to validate the performance measurement. The images containing the object are snapshotted from the recorded video and separated into 30 images per categories. The total of images dataset to be 120 images. The 120 images are tested one by one manually using the programming code explained in Table 3.7. The results are recorded and tabulated on the confusion matrix. The precision, recall and accuracy are calculated using the formula stated in figure 3.24 in chapter 3. Table 4.4 shows the confusion matrix for testing dataset by SqueezeNet model and Table 4.6 shows the confusion matrix for the testing dataset by GoogleNet.

Table 4. 4: Confusion matrix for testing dataset by SqueezeNet.

SqueezeNet		Target Class				Precision
		Detect No Vehicle	MPV	SUV	Sedan	
Output Class	Detect No Vehicle	26	0	0	0	100%
	MPV	4	30	12	0	65.2%
	SUV	0	0	18	0	100%
	Sedan	0	0	0	30	100%
Recall		86.7%	100%	60%	100%	86.7%

From Table 4.4, The confusion matrix shows that 26 out of 30 images for category Detect No Vehicle are classified correctly. Four images are classified wrongly to be MPV. For SUV, 18 out of 30 images of SUV vehicles are classified correctly and 12 images are classified wrongly as MPV. For MPV and Sedan, both categories are all classified

correctly. Table 4.6 shows the calculation for precision and recall for SqueezeNet in the testing dataset.

Table 4. 5: Calculation for precision and recall – SqueezeNet (Testing Dataset).

Class	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision $(TP/(TP+FP)) * 100$ (%)	Recall $(TP/(TP+FN)) * 100$ (%)
Detect No Vehicle	26	0	4	100	86.7
MPV	30	16	0	65.2	100
SUV	18	0	12	100	60
Sedan	30	0	0	100	100

Table 4. 6: Confusion matrix for testing dataset by GoogleNet.

GoogleNet		Target Class				Precision
		Detect No Vehicle	MPV	SUV	Sedan	
Output Class	Detect No Vehicle	30	0	0	0	100%
	MPV	0	30	0	3	91%
	SUV	0	0	30	0	100%
	Sedan	0	0	0	27	100%
Recall		100%	100%	100%	90%	97.5%

From Table 4.6, the confusion matrix shows that 27 out of 30 images for category Sedan are classified correctly. Only three images are classified wrongly to be MPV. For categories Detect no Vehicle, MPV and Sedan, both three categories are all classified correctly. Table 4.7 shows the calculation for precision and recall for SqueezeNet in the testing dataset.

Table 4. 7: Calculation of precision and recall – GoogleNet (Testing Dataset).

Class	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision $(TP/(TP+FP)) * 100$ (%)	Recall $(TP/(TP+FN)) * 100$ (%)
Detect No Vehicle	30	0	0	100	100
MPV	39	3	0	91	100
SUV	38	0	0	100	100
Sedan	27	0	3	100	90

The precision and recall from Table 4.5 and Table 4.7 will be discussed in next subtopic for the comparison between SqueezeNet and GoogleNet in the testing dataset.

4.2.1 Precision

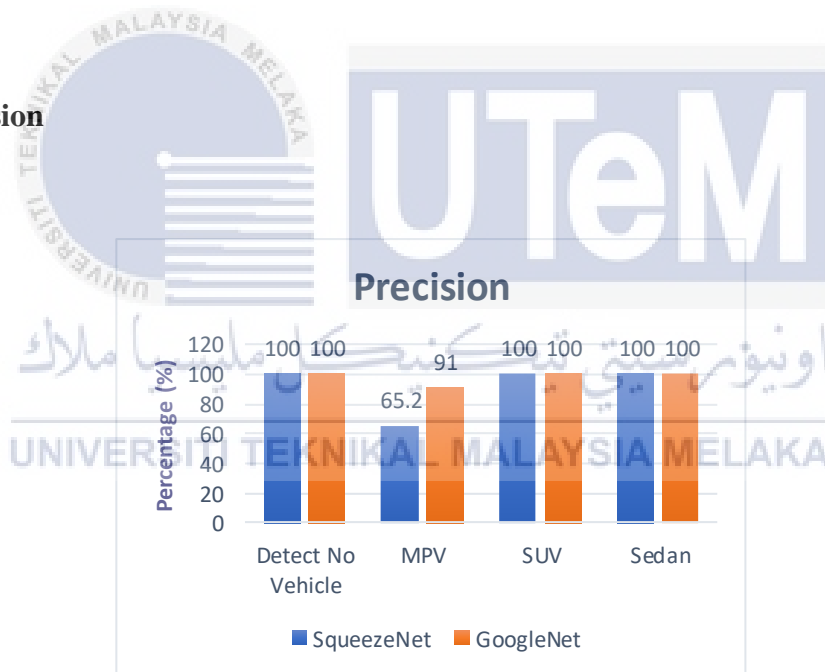


Figure 4. 8: Comparison of precision between two CNN model in testing dataset.

The precision from Table 4.5 and Table 4.7 are being transferred and compared using a histogram. Figure 4.8 shows the histogram for comparison of precision between two CNN model. Based on the histogram, the precision percentage of Detect No Vehicle class, SUV class and Sedan class are 100% for both SqueezeNet and GoogleNet. The precision percentage of MPV class is 65.2% for SqueezeNet and 91% for GoogleNet.

The precision for MPV class is lower than the other three classes. This is because all the misclassified images from other class are predicted as MPV class. This may be due to vehicle type for MPV is purely like rectangular. Since the rectangular shape is the basic shape for most vehicles, therefore, the images that are hard to detect the small features differences of the vehicles might be classified as MPV. The high order features such as edges and corner in Sedan and SUV may be hard to be detected in SqueezeNet model (Bautista et al., 2016).

4.2.2 Recall

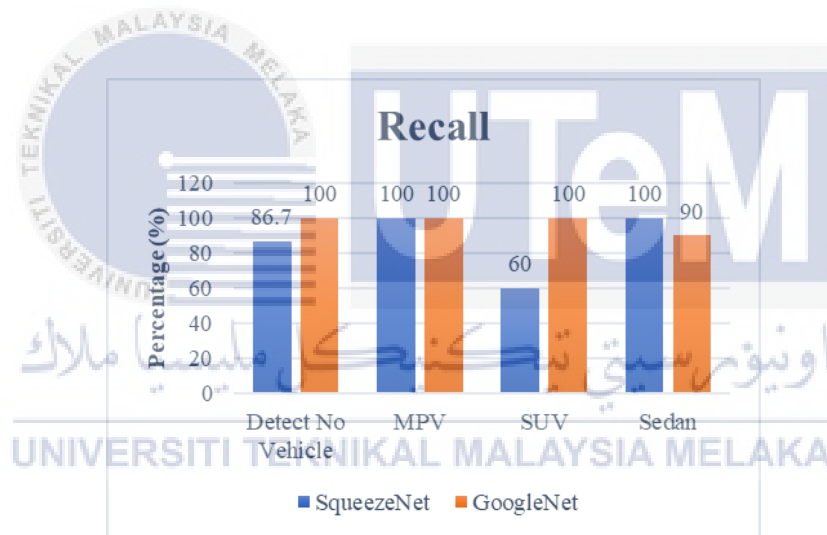


Figure 4. 9: Comparison of recall between two CNN models in testing dataset.

The recalls from Table 4.5 and Table 4.7 are being transferred and compared using a histogram. Figure 4.9 shows the histogram for comparison of recall between the two CNN models. Based on the histogram, the recall percentage of Detect No Vehicle class is 86.7% for SqueezeNet and 100% for GoogleNet. The recall percentage of MPV class is 100% for both SqueezeNet and GoogleNet. For SUV class, the GoogleNet scored better in recall with 100% which is 40% higher than the recall percentage for SqueezeNet. For Sedan class, the SqueezeNet scored better in recall with 100% which is 10% higher than the recall percentage for GoogleNet.

The most significant difference is at class SUV for SqueezeNet. The reason for SUV class has the lower recall in SqueezeNet is due to 18 images are classified correctly, the rest 12 images are classified wrongly as MPV. The reason may be due to SUV and MPV are sharing the same similarity in body type of vehicle. The different in small features may be hard to detect by the model. The resize of images further effect on the shape of the vehicle. While for Detect No Vehicle class, 4 images are classified wrong as MPV in SqueezeNet. Figure 4.10 shows the example of false predicted images for Detect No Vehicle class as MPV class. These images have a rectangular shape of an object such as lorry (a), sideboard (b) and rectangular white fabric (c). This can cause misunderstanding to the CNN models to predict it as MPV. This is because the MPV type of vehicle has the shape almost same to rectangular.

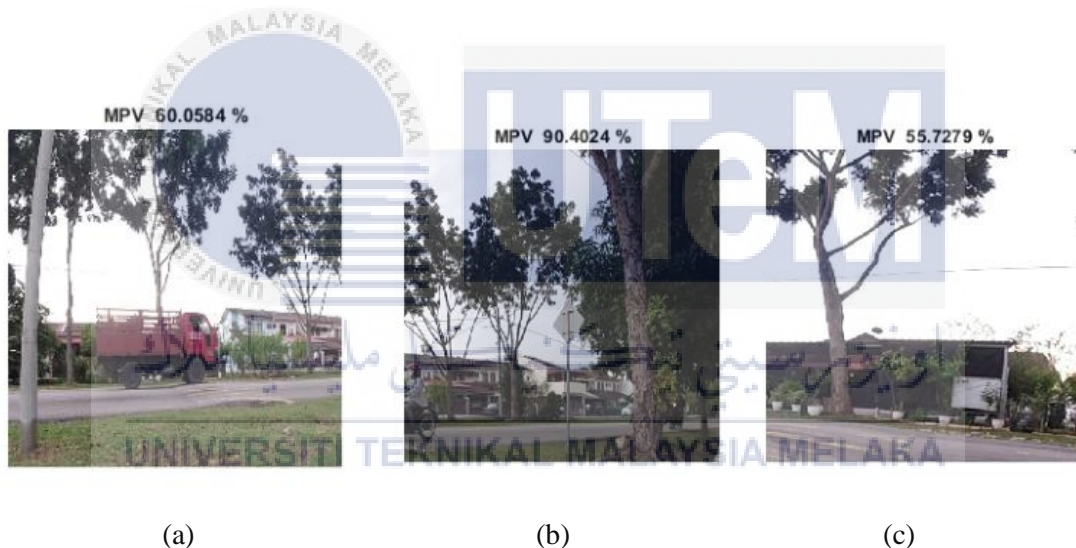


Figure 4. 10: Example of false predicted images.

However, GoogleNet can classify the images for these two classes that mention above all correctly. The is because the GoogleNet architecture is deeper and it is using the inception module as base bone of the architecture. Inception module using the concept of convolving the output from the previous layer in parallel with different sizes of the filter from small to bigger to cover the bigger area and still maintaining fine resolution for small information images (Christian Szegedy, 2015). Therefore, GoogleNet can detect the small features in the images and thus can classify the vehicle correctly.

4.3 Comparison and Analysis on Performance Measurement Between Two Datasets

The result from the two types of datasets are extracted and compared using the histogram. This with the aim to validate the performance measurement by the algorithm build. The performance measurements are average precision, average recall and accuracy for both CNN models. The average of precision and recall are calculated by finding the mean between the 4 classes. Figure 4.11, Figure 4.12 and Figure 4.13 show the histogram for comparison of average precision, average recall and overall accuracy between two CNN models and two datasets.

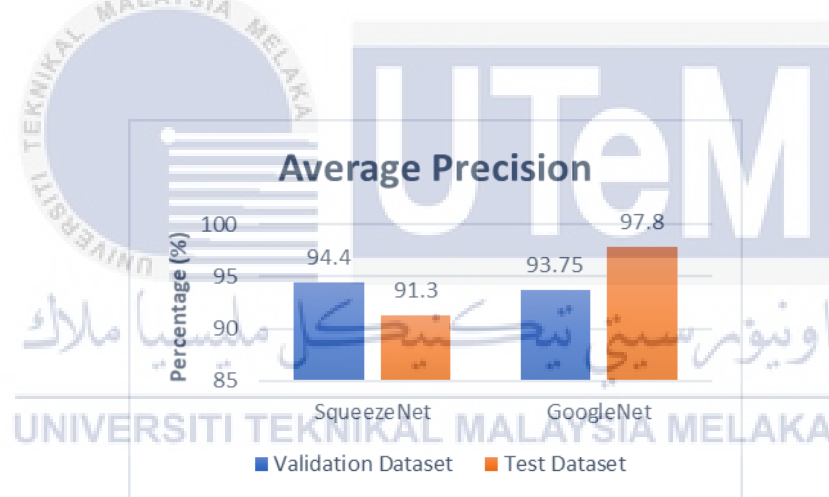


Figure 4. 11: Comparison of average precision between two CNN models and two datasets.

Figure 4.11 shows the average precision for SqueezeNet and GoogleNet on validation dataset and testing dataset. From the histogram, the average precision for SqueezeNet is 94.4% for validation dataset and 91.3% for testing dataset. There is a drop of 3.1% in average precision for SqueezeNet. In contrast, the average precision for GoogleNet is 93.75% for validation dataset and 97.8% for testing dataset. GoogleNet has increased by 4.05% of average precision.

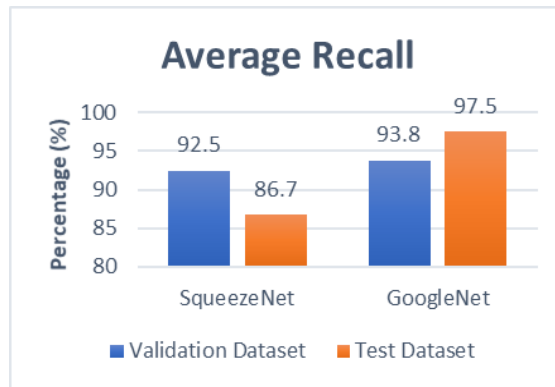


Figure 4. 12: Comparison of average recall between two CNN models and two datasets.

Figure 4.12 shows the average recall for SqueezeNet and GoogleNet on validation dataset and testing dataset. From the histogram, the average recall for SqueezeNet is 92.5% for validation dataset and 86.7% for testing dataset. There is a drop of 5.8% in average recall for SqueezeNet. In contrast, the average recall for GoogleNet is 93.8% for validation dataset and 97.5% for testing dataset. GoogleNet has increased by 3.7% of average recall.

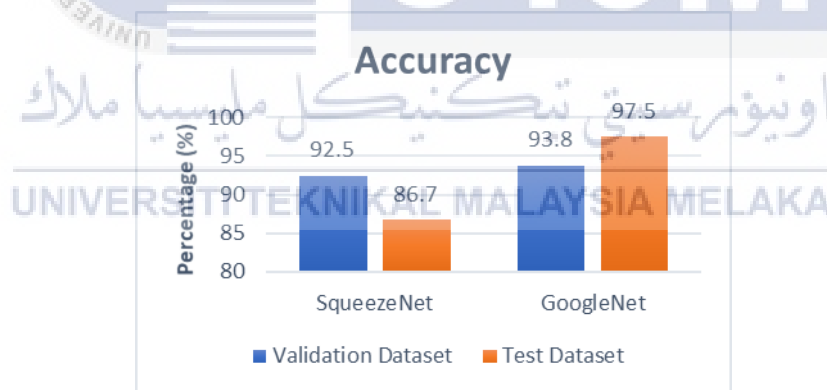


Figure 4. 13: Comparison the accuracy for two models on validation dataset and test dataset.

Figure 4.13 shows the accuracy for SqueezeNet and GoogleNet on validation dataset and testing dataset. From the histogram, the accuracy for SqueezeNet is 92.5% for validation dataset and 86.7% for testing dataset. There is a drop of 5.8% in accuracy for SqueezeNet. In contrast, the accuracy for GoogleNet is 93.8% for validation dataset and 97.5% for testing dataset. GoogleNet has increased by 3.7% of accuracy.

The significant drop in average performance measurement for SqueezeNet can be caused by the overfitting issue that been discussed in sub-topic 4.1.3. The SqueezeNet model undergoing some overfitting where the model memorizes too much detail including the noise on the training dataset. This causes the model unable to generalize and work well on the new images given. The reasons for overfitting might be due to the presence of noise, the limited size of the training set and the complexity of CNN models. While the GoogleNet doesn't have this issue since it is perfectly fit thus the results are close between validation dataset and testing dataset and it can work well for new images given (Ying, 2019).

4.4 Computational Time Per Image

10 images per each class are tested on the two CNN models to further validated on the computation time for processing on an image. The mean of computation time for each class is calculated and overall mean for processing one image is calculated. The data is tabulated in Table 4.8 For SqueezeNet and Table 4.9 For GoogleNet.

Table 4. 8: Data collection for the time taken per image in SqueezeNet.

No	No Vehicle Detected (s)	MPV (s)	SUV (s)	Sedan (s)
1	0.079	0.287	0.061	0.062
2	0.086	0.088	0.078	0.070
3	0.117	0.174	0.093	0.062
4	0.107	0.140	0.076	0.065
5	0.121	0.074	0.098	0.083
6	0.117	0.076	0.069	0.100
7	0.095	0.064	0.065	0.089
8	0.063	0.073	0.103	0.071
9	0.126	0.072	0.129	0.094
10	0.045	0.094	0.067	0.058
Mean	0.095	0.114	0.084	0.075
Overall Mean	0.092			

Table 4. 9: Data collection for time taken per image in GoogleNet.

No	No Vehicle Detected(s)	MPV(s)	SUV (s)	Sedan (s)
1	0.116	0.129	0.120	0.140
2	0.231	0.228	0.274	0.120
3	0.184	0.182	0.173	0.062
4	0.182	0.140	0.172	0.166
5	0.220	0.165	0.168	0.204
6	0.301	0.166	0.172	0.166
7	0.131	0.189	0.395	0.186
8	0.212	0.161	0.278	0.267
9	0.179	0.264	0.364	0.170
10	0.178	0.224	0.209	0.161
Mean	0.193	0.185	0.232	0.164
Overall Mean	0.194			

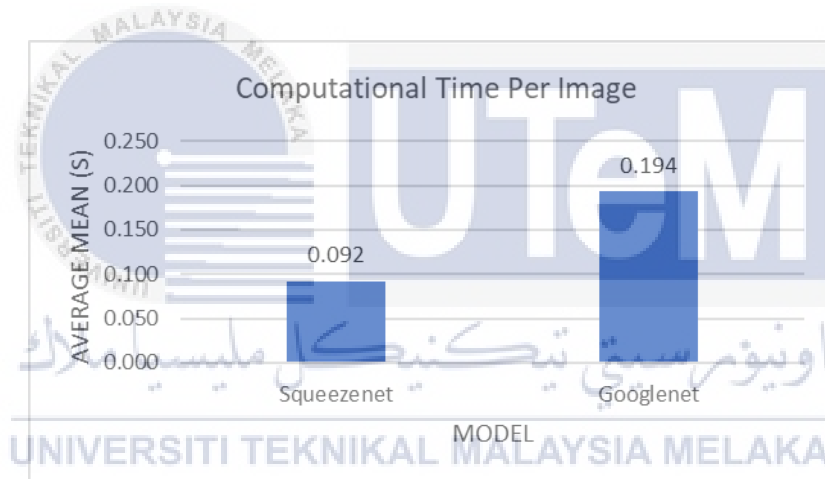


Figure 4. 14: Comparison the computational time per image for SqueezeNet and GoogleNet.

From both Table 4.8 and Table 4.9, there is no significant difference in the computational time for the four classes. The object in images has no significant effect on the computational time. From the overall mean, Figure 4.14 shows the computational time per image for SqueezeNet is 0.092s and GoogleNet is 0.194s. The difference is very obvious where GoogleNet needs an average of longer time to complete one processing on an image if compare to SqueezeNet. The reason is that SqueezeNet has less depth, less parameter and less size when compared to GoogleNet. SqueezeNet is built with aims to reduce the parameter and size of model architecture to make this model faster and can fit into hardware with limited memory. Wang *et al.* (2019) stated that actual traffic flow for

passage time per vehicle is 0.89s. Therefore, it is believed if there is a GPU embedded for the processing of the image, the computational time of the two models will be faster and thus can be coped with real-time vehicle detection and classification in the surveillance system in either traffic flow or automatic car parking system (Nithiyaraj and Arivazhagan, 2020).

4.5 Summary

The programming code can be developed and the first objective can be achieved where the algorithm of vehicle detection and classification can be built with the use of the pre-trained neural network. In terms of result, both developed algorithms can generate high accuracy for both training dataset and validation dataset. The result is further tested on new image dataset. It is proved again that the developed code can detect and classify the vehicle into Sedan, SUV and MPV. It can also differentiate between the situation with no vehicle and has the vehicle. GoogleNet model is done better than SqueezeNet in terms of performances measurement in this project. GoogleNet can achieve high recall and high precision and high accuracy when testing with a new dataset. The computational time per image is 0.194 with no GPU. This means with better GPU it is applicable to be implemented in the real-time surveillance system for managing of vehicle application. Thus, the second objective of this project has been achieved.

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.0 Conclusion

Vehicle detection and classification plays an important role in a part of the surveillance system. It can be further implemented in the automatic parking lot system for ease of counting, analysis and management on the vehicles. It is essential in providing the benefits of reducing the disadvantages of manpower which had been outlined in chapter 2. The detection and classification of vehicles can be done continuously without the need for rest as compared to manpower. So the performance of the system will be constant where the manpower might be reduced with the increase of time. The good accuracy that been achieved from this project has indicated that the machine vision system can replace human vision in term of a long duration of the working hour.

The first objective is to develop an algorithm for vehicle detection and classification through an image processing technique. MATLAB is used to develop the script and run the programming code for detection and classification of vehicle. The image processing technique used is to focus on CNN deep learning. The pre-trained neural network SqueezeNet and GoogleNet are modified and used to train on the new task for this project by using self-collected image dataset. This objective is successfully achieved.

The second objective is to validate the developed system for vehicle detection and classification in terms of accuracy. Both pre-trained CNN shows the result of high

accuracy for the training dataset and validation dataset. During the testing phase, GoogleNet can detect and classify the vehicle with an overall accuracy of 97.5% which is higher than SqueezeNet which is 86.7%. The developed system can detect and classify the vehicle into Sedan, SUV and MPV with good accuracy for GoogleNet. Thus, the second objective is achieved.

In conclusion for this project, the image processing technique of CNN can be implemented to machine vision system to replace human vision to perform the task of vehicle detection and classification. The objectives and scopes are fully accomplished. The following part in this chapter will briefly outline the financial implication, sustainable design and development, complexity, long-life learning (LLL) and basic entrepreneurship (BE) as well as limitations for this vision system project and recommendations and suggestions to minimize the limitation to continually enhance this system.



5.1 Financial Implication

A summary of the bill of material on Table 5.1 shows the overall price to carry out this research. This bill of material is only for reference because the price of the part is varied depends on the method of buying, such as online purchasing or on shop buying. All of the device and material are self-owned.

Table 5. 1: Bill of material (BOM).

Number	Part	Quantity	Price (RM)
1	Laptop	1	2500.00
2	Camera (DJI Osmo Action)	1	1400.00
3	Tripod	1	10.00
4	Micro SD card (64 gb)	1	35.00
5	Universal memory card reader	1	16.00
Total			3691

5.2 Sustainability

The sustainability index for this project is all the material and device used to conduct this whole experiment and study are using the already self-own product. There is no extra money that been used to do this project. Furthermore, the materials and devices used for this project can be used for other tasks in image processing. With proper setup and connection, it can be implemented in the real-time surveillance system. Apart from that, since this project is programming based, therefore there is no need for the cost for the manufacturing of a product. The scopes and methodology for this project are designed to be used fewer components to achieve for the two main objectives of this project. The component such as raspberry pi, camera module, extra keyboard and extra USB cable can be eliminated. The energy consumption mainly used to complete this project is electricity. The electricity is mainly generated by using the natural resource of water and solar. Moreover, if this developed system is deployed for the application, it will not generate the harmful toxic substances that can cause to air pollution, water pollution or other types of negative impact on the environment. Thus, this project is economic friendly achieving sustainable index without comprising the objectives.

5.3 Complexity

The complexity of this project can be categories into three parts. The first part is the complexity of preparation of the dataset. The nature of the dataset will eventually affect the training result of CNN. The size of the images, angle of the vehicles, the ratio of the vehicle to the image size, the resolution of the images, the volume of the dataset and the lighting on the images, the resizing technique are needed to be considered appropriate to output the desired accuracy. The previous research shows no significant discussion on how the dataset impact during the application especially for the ratio of the vehicle to the image size and the resizing technique. Since the training of deep learning, CNN largely depends on the dataset. The second part is the construction of the programming for this project required technical knowledge of programming. Although MATLAB has the machine learning and deep learning toolbox to construct the CNN training and can generate the

code automatically after done the training on Deep designer Network apps. There is still a need to understand the generated MATLAB script to do the troubleshooting and adding the script for testing phrase. The third part is troubleshooting the problem occur on how to improve the accuracy of the algorithm. There are so many possibilities for the cause of low accuracies such as dataset, CNN layer parameter and hyperparameter. It requires critical thinking, analysis, research and patient to justify the problem and create a solution to solve for the problem. This consumes a lot of time as a lot of testing need to be done. These significant findings for the result encompass and accomplish the complex problem solving, complex engineering problems and knowledge profile on the background of techniques used.

5.4 Long Life Learning (LLL)

This project promotes long life learning (LLL) because it requires self-study to develop the complete algorithm. Long-life learning takes time for personal development because MATLAB programming knowledge is very wide without a border and it can be very complex as well as competitiveness. The application of deep learning in artificial intelligence for image processing is very huge. The image processing technique can be integrated into our daily life in many aspects. It provides long-life learning for the individual to learn deeper into the field. The created algorithm can be deployed to become a stand-alone device and used to perform other tasks. This will need further learning for the application. To enhance the system, the idea of improving the programming should also be done to reduce the error and make the system more stable.

5.5 Basic Entrepreneurship (BE)

The application of this developed algorithm of image processing for vehicle detection and classification is very huge. It can be deployed to other devices to become a

stand-alone device. For example, it can help develop the automatic parking lot system where it can embed in the surveillance system in the camera to detect car and classify the car into different categories and guide the consumer to desire car parking lot. The consumer can directly drive to the parking lot to park their car with the direction given by the system. The management will also be easy where the counting of the vehicle can be done so it will help on managing the parking lot where the consumer can know that is the parking slot is filled or still got a slot. It creates a big potential to be commercialized and bring the technology more into industry 4.0. A further study is needed for its application.

5.6 Limitation

In this project, although all the objectives are fully achieved, some limitations make the project become difficult and impact on the result. The limitations are as follows:

1. Dataset Preparation

Due to the time constraint in prepared the dataset, the dataset prepared is small in volume with total 396 of images. According to research from the past journals, the larger the image dataset, the higher the accuracy of the trained CNN. Although to minimize the problem of the small dataset, transfer learning by using a pre-trained neural network have been selected as the main CNN architectural to perform the training. The result of accuracy still being affected somehow. Another problem associated with dataset preparation is the ratio of the vehicle to the image size and the resize of the dataset can affect the result. Since the region of interest is cropped manually therefore, the size of the vehicle images will be different. Due to the resized of image dataset to be fit into the constant input layer size, the resized image is stretched and the shape of the vehicle has been changed slightly, this may affect in the result different during the real-time application. The angle of placement for the camera to the angle of the vehicle to be detected and classify will need to be considered critically.

2. Hardware Limitation

The specification of the laptop device is low and unable to support for the image processing by using the CNN technique that usually required for high CPU, high GPU and large memory space to make the training time shorter and faster. Otherwise, the laptop device is unable to support and will crack during the training of the algorithm. Therefore, the dataset is set to hundreds and SqueezeNet and GoogleNet are selected to perform the training of CNN. The testing on images is used to replace the real-time testing due to this issue.

5.7 Recommendation and Suggestion

In conjunction to the limitation as mention on above, there are many recommendations and suggestions for upgrading this system to become more stable and high-level by minimizing or eliminating the limitations to eventually provide end-line service to the users.

1. Investigate the effect of the dataset on the result.

There is a need to investigate the ratio of the object to an image in affecting the result. This is because when the object is from the camera, the object will look small in the image. This may result in the system unable to detect and classify the object accurately. Furthermore, the resizing technique for the dataset with different image size then stretched the image making the object different from the real features. This may affect the result too as the system tends to remember the features that been stretched. Therefore, when given the unstretched image foe testing, the system may be unable to classified correctly. This problem is found out during the testing phase of this project. Due to time constraint, only the hypothesis can be roughly pointed out for future improvement. Further experiment will need to carried out to justify on this problem.

2. Upgrade the specification of hardware.

The CPU, GPU and SSD can be upgraded to higher spec to support for smoother and lower the time for image processing process. The image dataset can increase in its volume to cover more brand of vehicle and angle of the vehicle as more dataset higher the accuracy. This would further improve the accuracy of the system and make the vehicle detection and classification can be done at high speed for application on-road or traffic.

3. User-friendly graphical-user-interface (GUI)

The whole system was written in the MATLAB language and run using MATLAB software to output the result of vehicle detection and classification. It is not very practical because the public who cannot read computer language cannot use it. Hence, it is highly recommended to build out a user-friendly screen which provides a graphical user interface where the language used is human-understandable-common language.

4. Open-source software

Since the MATLAB software will need for a license to excess, therefore it will be costly. This has limited the range of uses to excess in MATLAB software. It is also recommended using open-source software such as Python to perform the same task. This is because open-source software Python is free of charge and is also support for the image processing using deep learning.

REFERENCES

- Awang, S., Azmi, N. M. A. N., & Rahman, M. A. (2020). Vehicle Type Classification Using an Enhanced Sparse-Filtered Convolutional Neural Network With Layer-Skipping Strategy. *IEEE Access*, 8, 14265–14277. <https://doi.org/10.1109/access.2019.2963486>
- Bassma, G., Hassan, E. G., & Tayeb, S. (2018). Support vector machines for improving vehicle localization in urban canyons. *MATEC Web of Conferences*, 200, 4–11. <https://doi.org/10.1051/matecconf/201820000004>
- Bautista, C. M., Dy, C. A., Mañalac, M. I., Orbe, R. A., & Cordel, M. (2016). Convolutional neural network for vehicle detection in low resolution traffic videos. *Proceedings - 2016 IEEE Region 10 Symposium, TENSYP 2016*, 277–281. <https://doi.org/10.1109/TENCONSpring.2016.7519418>
- Bini, S. A. (2018). Artificial Intelligence, Machine Learning, Deep Learning, and Cognitive Computing: What Do These Terms Mean and How Will They Impact Health Care? *Journal of Arthroplasty*, 33(8), 2358–2361. <https://doi.org/10.1016/j.arth.2018.02.067>
- Bonde, D. J., Shende, R. S., Kedari, A. S., Gaikwad, K. S., & Bhokre, A. U. (2014). Automated car parking system commanded by Android application. *2014 International Conference on Computer Communication and Informatics: Ushering in Technologies of Tomorrow, Today, ICCCI 2014*, 5(3), 3001–3004. <https://doi.org/10.1109/ICCCI.2014.6921729>
- Boulent, J., Foucher, S., Théau, J., & St-Charles, P. L. (2019). Convolutional Neural Networks for the Automatic Identification of Plant Diseases. *Frontiers in Plant Science*, 10(July). <https://doi.org/10.3389/fpls.2019.00941>
- Chen, L., Ye, F., Ruan, Y., Fan, H., & Chen, Q. (2018). An algorithm for highway vehicle detection based on convolutional neural network. *Eurasip Journal on Image and Video Processing*, 2018(1), 1–7. <https://doi.org/10.1186/s13640-018-0350-2>

- Chen, X. (2019). *Image enhancement effect on the performance of convolutional neural networks*. June, 1–40. www.bth.se
- Chung, J., & Sohn, K. (2018). Image-Based Learning to Measure Traffic Density Using a Deep Convolutional Neural Network. *IEEE Transactions on Intelligent Transportation Systems*, 19(5), 1670–1675. <https://doi.org/10.1109/TITS.2017.2732029>
- Deng, Z., Sun, H., Zhou, S., Zhao, J., & Zou, H. (2017). Toward Fast and Accurate Vehicle Detection in Aerial Images Using Coupled Region-Based Convolutional Neural Networks. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(8), 3652–3664. <https://doi.org/10.1109/JSTARS.2017.2694890>
- Derman, E., & Salah, A. A. (2018). Continuous real-time vehicle driver authentication using convolutional neural network based face recognition. *Proceedings - 13th IEEE International Conference on Automatic Face and Gesture Recognition, FG 2018*, 577–584. <https://doi.org/10.1109/FG.2018.00092>
- Do, S., Song, K. D., & Chung, J. W. (2020). Basics of deep learning: A radiologist's guide to understanding published radiology articles on deep learning. *Korean Journal of Radiology*, 21(1), 33–41. <https://doi.org/10.3348/kjr.2019.0312>
- Espinosa, J. E., Velastin, S. A., & Branch, J. W. (2017). Vehicle detection using alex net and faster R-CNN deep learning models: A comparative study. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10645 LNCS, 3–15. https://doi.org/10.1007/978-3-319-70010-6_1
- Farag, W. (2018). Recognition of traffic signs by convolutional neural nets for self-driving vehicles. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 22(3), 205–214. <https://doi.org/10.3233/KES-180385>
- Guo, Q., Liang, Z., & Hu, J. (2017). Vehicle Classification with Convolutional Neural

- Network on Motion Blurred Images. *DEStech Transactions on Computer Science and Engineering, aiea*, 40–45. <https://doi.org/10.12783/dtcse/aiea2017/14912>
- Guo, T., Dong, J., Li, H., & Gao, Y. (2017). Simple convolutional neural network on image classification. *2017 IEEE 2nd International Conference on Big Data Analysis, ICBDA 2017*, 721–724. <https://doi.org/10.1109/ICBDA.2017.8078730>
- Han, D., Liu, Q., & Fan, W. (2018). A new image classification method using CNN transfer learning and web data augmentation. *Expert Systems with Applications*, 95, 43–56. <https://doi.org/10.1016/j.eswa.2017.11.028>
- Hashemi, M. (2019). Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. *Journal of Big Data*, 6(1). <https://doi.org/10.1186/s40537-019-0263-7>
- Hicham, B., Ahmed, A., & Mohammed, M. (2018). Vehicle Type Classification Using Convolutional Neural Network. *Colloquium in Information Science and Technology, CIST, 2018-October*, 313–316. <https://doi.org/10.1109/CIST.2018.8596500>
- Hsu, S. C., Huang, C. L., & Chuang, C. H. (2018). Vehicle detection using simplified fast R-CNN. *2018 International Workshop on Advanced Image Technology, IWAIT 2018*, 1–3. <https://doi.org/10.1109/IWAIT.2018.8369767>
- Hu, R., Tian, B., Yin, S., & Wei, S. (2019). Efficient Hardware Architecture of Softmax Layer in Deep Neural Network. *International Conference on Digital Signal Processing, DSP, 2018-Novem*, 1–5. <https://doi.org/10.1109/ICDSP.2018.8631588>
- Hu, X., Xu, X., Xiao, Y., Chen, H., He, S., Qin, J., & Heng, P. A. (2019). SINet: A Scale-Insensitive Convolutional Neural Network for Fast Vehicle Detection. *IEEE Transactions on Intelligent Transportation Systems*, 20(3), 1010–1019. <https://doi.org/10.1109/TITS.2018.2838132>
- Huang, K., & Zhang, B. (2017). Fine-grained vehicle recognition by deep Convolutional Neural Network. *Proceedings - 2016 9th International Congress on Image and Signal*

Processing, BioMedical Engineering and Informatics, CISP-BMEI 2016, 465–470.
<https://doi.org/10.1109/CISP-BMEI.2016.7852756>

Huttunen, H., Yancheshmeh, F. S., & Ke, C. (2016). Car type recognition with Deep Neural Networks. *IEEE Intelligent Vehicles Symposium, Proceedings, 2016-Augus*, 1115–1120. <https://doi.org/10.1109/IVS.2016.7535529>

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2017). 50 X Fewer Parameters and < 0.5 Mb Model Size. *Iclr, April 2016*, 1–13.

Jadhav, S. B., Udupi, V. R., & Patil, S. B. (2020). Identification of plant diseases using convolutional neural networks. *International Journal of Information Technology (Singapore)*. <https://doi.org/10.1007/s41870-020-00437-5>

Jones, A. D., Graff, J. P., Darrow, M., Borowsky, A., Olson, K. A., Gandour-Edwards, R., Datta Mitra, A., Wei, D., Gao, G., Durbin-Johnson, B., & Rashidi, H. H. (2019). Impact of pre-analytical variables on deep learning accuracy in histopathology. *Histopathology*, 75(1), 39–53. <https://doi.org/10.1111/his.13844>

Kannoja, S. P., & Jaiswal, G. (2018). Effects of Varying Resolution on Performance of CNN based Image Classification An Experimental Study. *International Journal of Computer Sciences and Engineering*, 6(9), 451–456.
<https://doi.org/10.26438/ijcse/v6i9.451456>

Kaplan, Ö., & Şaykol, E. (2018). Comparison of support vector machines and deep learning for vehicle detection. *CEUR Workshop Proceedings*, 2280(December), 64–69.

Kim, P. K., & Lim, K. T. (2017). Vehicle Type Classification Using Bagging and Convolutional Neural Network on Multi View Surveillance Image. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2017-July*, 914–919. <https://doi.org/10.1109/CVPRW.2017.126>

Learning, T. (2020). *ANI2892*. 1–12.

- Lee, H. J., Ullah, I., Wan, W., Gao, Y., & Fang, Z. (2019). Real-Time vehicle make and model recognition with the residual squeezeNet architecture. *Sensors (Switzerland)*, 19(5). <https://doi.org/10.3390/s19050982>
- Li, B. (2017). 3D fully convolutional network for vehicle detection in point cloud. *IEEE International Conference on Intelligent Robots and Systems, 2017-Septe*, 1513–1518. <https://doi.org/10.1109/IROS.2017.8205955>
- Mintz, Y., & Brodie, R. (2019). Introduction to artificial intelligence in medicine. *Minimally Invasive Therapy and Allied Technologies*, 28(2), 73–81. <https://doi.org/10.1080/13645706.2019.1575882>
- Molina-Cabello, M. A., Luque-Baena, R. M., López-Rubio, E., & Thurnhofer-Hemsi, K. (2018). Vehicle type detection by ensembles of convolutional neural networks operating on super resolved images. *Integrated Computer-Aided Engineering*, 25(4), 321–333. <https://doi.org/10.3233/ICA-180577>
- Muhammad, K., Ahmad, J., Lv, Z., Bellavista, P., Yang, P., & Baik, S. W. (2019). Efficient Deep CNN-Based Fire Detection and Localization in Video Surveillance Applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(7), 1419–1434. <https://doi.org/10.1109/TSMC.2018.2830099>
- Neeru, N., & Kaur, L. (2016). Modified SIFT descriptors for face recognition under different emotions. *Journal of Engineering (United Kingdom)*, 2016. <https://doi.org/10.1155/2016/9387545>
- Niessner, R., Schilling, H., & Jutzi, B. (2017). Investigations On The Potential Of Convolutional Neural Networks For Vehicle Classification Based On Rgb And Lidar Data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4(1W1), 115–123. <https://doi.org/10.5194/isprs-annals-IV-1-W1-115-2017>
- Nithiyaraj, E. E., & Arivazhagan, S. (2020). Survey on Recent Works in Computed Tomography based Computer - Aided Diagnosis of Liver using Deep Learning

Techniques. *International Journal of Innovative Science and Research Technology*, 5(7), 173–181. <https://doi.org/10.38124/ijisrt20jul058>

Pajaziti, A., Bajrami, X., Beqa, F., & Gashi, B. (2019). Development of a vehicle for driving with Convolutional Neural Network. *International Journal of Advanced Computer Science and Applications*, 10(9), 413–420. <https://doi.org/10.14569/ijacsa.2019.0100954>

Pan, X., & Ogai, H. (2019). Fast lane detection based on deep convolutional neural network and automatic training data labeling. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E102A(3), 566–575. <https://doi.org/10.1587/transfun.E102.A.566>

Patil, A., & Rane, M. (2021). Convolutional Neural Networks: An Overview and Its Applications in Pattern Recognition. *Smart Innovation, Systems and Technologies*, 195, 21–30. https://doi.org/10.1007/978-981-15-7078-0_3

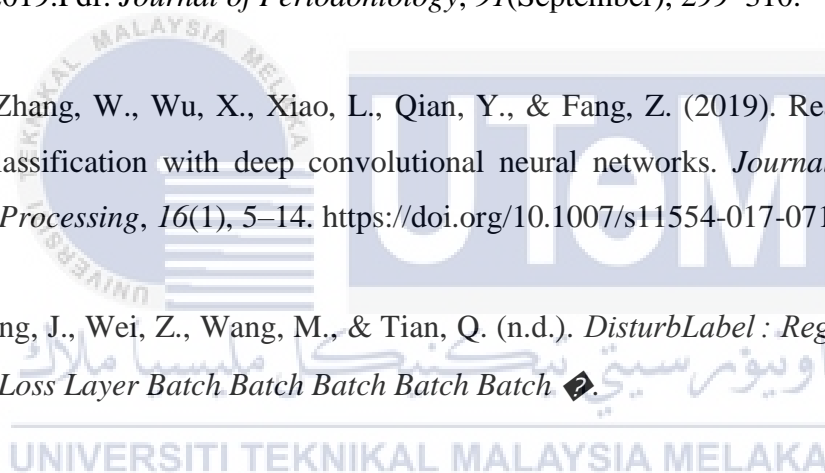
Pelt, D. M., & Sethian, J. A. (2017). A mixed-scale dense convolutional neural network for image analysis. *Proceedings of the National Academy of Sciences of the United States of America*, 115(2), 254–259. <https://doi.org/10.1073/pnas.1715832114>

Phung, V. H., & Rhee, E. J. (2019). A High-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences (Switzerland)*, 9(21). <https://doi.org/10.3390/app9214500>

Rangarajan, A. K., Purushothaman, R., & Ramesh, A. (2018). Tomato crop disease classification using pre-trained deep learning algorithm. *Procedia Computer Science*, 133, 1040–1047. <https://doi.org/10.1016/j.procs.2018.07.070>

Roecker, M. N., Costa, Y. M. G., Almeida, J. L. R., & Matsushita, G. H. G. (2018). Automatic Vehicle type Classification with Convolutional Neural Networks. *International Conference on Systems, Signals, and Image Processing, 2018-June*(June 2018). <https://doi.org/10.1109/IWSSIP.2018.8439406>

- Roy, S., & Rahman, M. S. (2019). Emergency Vehicle Detection on Heavy Traffic Road from CCTV Footage Using Deep Convolutional Neural Network. *2nd International Conference on Electrical, Computer and Communication Engineering, ECCE 2019, February 2019*. <https://doi.org/10.1109/ECACE.2019.8679295>
- San, W. J., Lim, M. G., & Chuah, J. H. (2019). Efficient vehicle recognition and classification using convolutional neural network. *Proceedings - 2018 IEEE International Conference on Automatic Control and Intelligent Systems, I2CACIS 2018, 1*, 117–122. <https://doi.org/10.1109/I2CACIS.2018.8603700>
- Seng, D., Lin, B., & Chen, J. (2018). Convolutional neural network and the recognition of vehicle types. *NeuroQuantology*, *16*(6), 720–727. <https://doi.org/10.14704/nq.2018.16.6.1641>
- Seo, Y., & Shin, K. shik. (2019). Real-time electric vehicle classification for electric charging and parking system using pre-trained convolutional neural network. *ACM International Conference Proceeding Series, Part F1483*, 8–11. <https://doi.org/10.1145/3322645.3322650>
- Sheng, M., Liu, C., Zhang, Q., Lou, L., & Zheng, Y. (2018). Vehicle detection and classification using convolutional neural networks. *Proceedings of 2018 IEEE 7th Data Driven Control and Learning Systems Conference, DDCLS 2018*, 581–587. <https://doi.org/10.1109/DDCLS.2018.8516099>
- Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, *6*(1). <https://doi.org/10.1186/s40537-019-0197-0>
- Shu, M. (2019). Deep learning for image classification on very small datasets using transfer learning. *Creative Components*, 14–21.
- Sladojevic, S., Arsenovic, M., Anderla, A., Culibrk, D., & Stefanovic, D. (2016). Deep Neural Networks Based Recognition of Plant Diseases by Leaf Image Classification. *Computational Intelligence and Neuroscience*, 2016. <https://doi.org/10.1155/2016/3289801>

- Solovyev, R., Kustov, A., Telpukhov, D., Rukhlov, V., & Kalinin, A. (2019). Fixed-point convolutional neural network for real-time video processing in FPGA. *Proceedings of the 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, ElConRus 2019*, 1605–1611. <https://doi.org/10.1109/EIConRus.2019.8656778>
- Tarmizi, I. A., & Aziz, A. A. (2018). Vehicle Detection Using Convolutional Neural Network for Autonomous Vehicles. *International Conference on Intelligent and Advanced System, ICIAS 2018*, 1–5. <https://doi.org/10.1109/ICIAS.2018.8540563>
- Wang, Chin-Wei Yu, Shan-Huey Mandelaris, George A. Wang, H.-L. (2019). Wang2019.Pdf. *Journal of Periodontology*, 91(September), 299–310.
- Wang, X., Zhang, W., Wu, X., Xiao, L., Qian, Y., & Fang, Z. (2019). Real-time vehicle type classification with deep convolutional neural networks. *Journal of Real-Time Image Processing*, 16(1), 5–14. <https://doi.org/10.1007/s11554-017-0712-5>
- Xie, L., Wang, J., Wei, Z., Wang, M., & Tian, Q. (n.d.). *DisturbLabel: Regularizing CNN on the Loss Layer Batch Batch Batch Batch Batch* 
- Xin, M., & Wang, Y. (2019). Research on image classification model based on deep convolution neural network. *Eurasip Journal on Image and Video Processing*, 2019(1). <https://doi.org/10.1186/s13640-019-0417-8>
- Yani, M., Irawan, B., & Setiningsih, C. (2019). Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. *Journal of Physics: Conference Series*, 1201(1). <https://doi.org/10.1088/1742-6596/1201/1/012052>
- Ying, X. (2019). An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, 1168(2). <https://doi.org/10.1088/1742-6596/1168/2/022022>
- Zhang, M., Wang, P., & Zhang, X. (2019). Vehicle color recognition using deep

convolutional neural networks. *ACM International Conference Proceeding Series*, 236–238. <https://doi.org/10.1145/3349341.3349408>

Zhang, Qiang, Zhuo, L., Li, J., Zhang, J., Zhang, H., & Li, X. (2018). Vehicle color recognition using Multiple-Layer Feature Representations of lightweight convolutional neural network. *Signal Processing*, 147, 146–153. <https://doi.org/10.1016/j.sigpro.2018.01.021>

Zhang, Qinghui, Wan, C., & Han, W. (2019). A modified faster region-based convolutional neural network approach for improved vehicle detection performance. *Multimedia Tools and Applications*, 78(20), 29431–29446. <https://doi.org/10.1007/s11042-018-6769-8>

Zhang, Y., Liu, D., & Zha, Z. J. (2017). Improving triplet-wise training of convolutional neural network for vehicle re-identification. *Proceedings - IEEE International Conference on Multimedia and Expo*, July, 1386–1391. <https://doi.org/10.1109/ICME.2017.8019491>

Zhou, Y., Wang, H., Xu, F., & Jin, Y. Q. (2016). Polarimetric SAR Image Classification Using Deep Convolutional Neural Networks. *IEEE Geoscience and Remote Sensing Letters*, 13(12), 1935–1939. <https://doi.org/10.1109/LGRS.2016.2618840>

Zhuo, L., Jiang, L., Zhu, Z., Li, J., Zhang, J., & Long, H. (2017). Vehicle classification for large-scale traffic surveillance videos using Convolutional Neural Networks. *Machine Vision and Applications*, 28(7), 793–802. <https://doi.org/10.1007/s00138-017-0846-2>

Chen-Hang HE, K.-M. L. (2018). Fast Vehicle Detection With Lateral Convolutional Neural Network. *IEEE*, 2341-2345.

Christian Szegedy, W. L. (2015). Going Deeper with Convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 9). Boston, MA, USA: IEEE.

Frankenfield, J. (6 January, 2021). Artificial Intelligence (AI).

Gibson, J. P. (2017). *Deep Learning*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: y O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Tsang, S.-H. (2019). Review: SqueezeNet (Image Classification).



APPENDIX A

JOURNAL REVIEW TABLE

No	Main Topic	Journals
1	Conventional image processing method	6
2	Artificial intelligence, machine learning	6
3	Deep learning CNN	17
4	Pretrained network	12
5	Dataset	13
6	Dataset Preprocessing	7
7	Training	6
8	Hyperparameter optimisation	7
9	Performance measurement	6

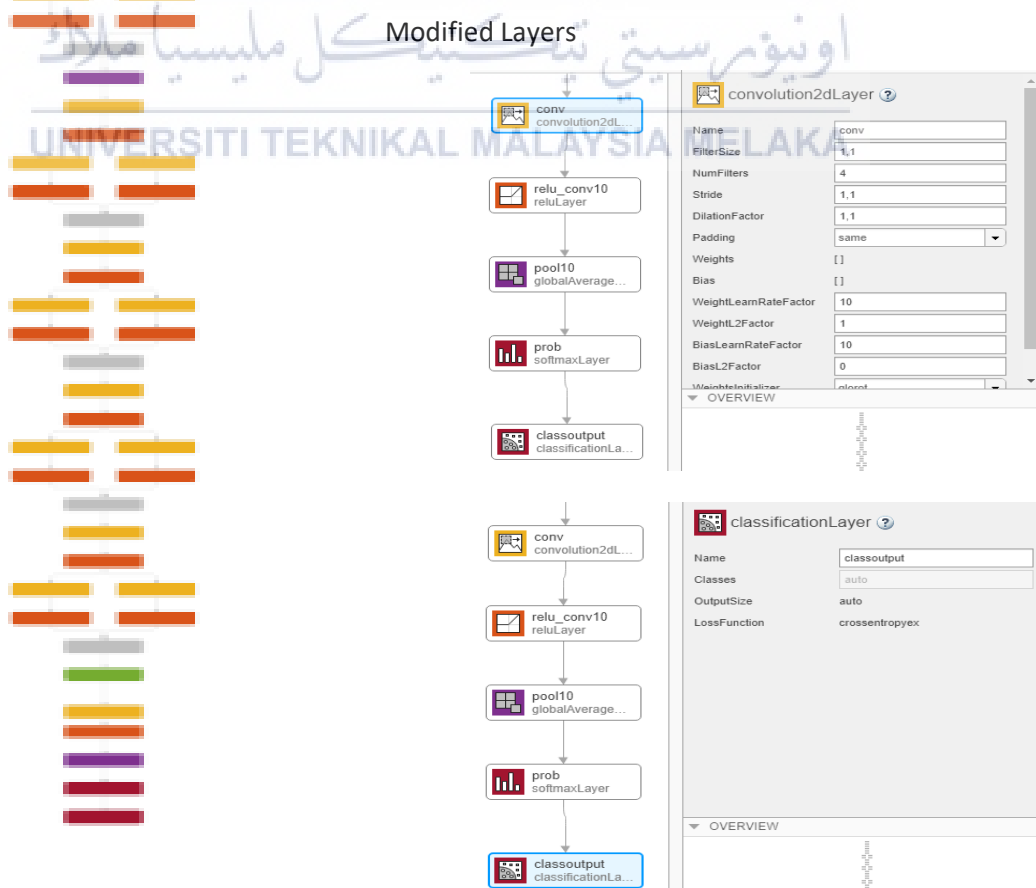
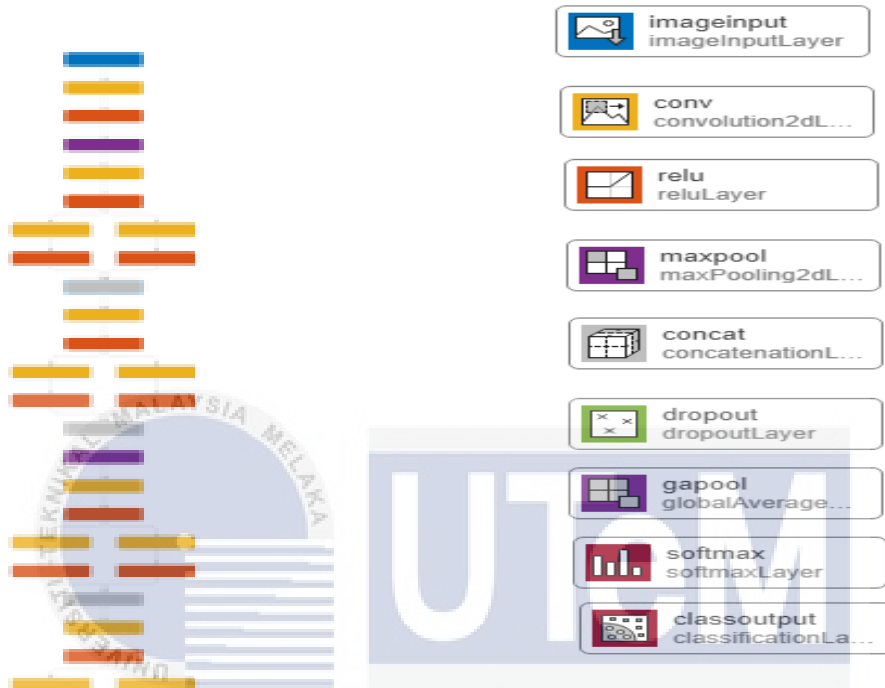
*Review of journals may be repeated for different topics.



APPENDIX B

SQUEEZENET ARCHITECTURE

Different colour indicates different layers. Total layers 68.



PARAMETERS OF EACH LAYER - SQUEEZENET

	Name	Type	Activations	Learnables	Total Learnab...
1	data 227x227x3 images with 'zerocenter' normalization	Image Input	227x227x3	-	0
2	conv1 64 3x3x3 convolutions with stride [2 2] and padding [0 0 0 0]	Convolution	113x113x64	Weights 3x3x3x64 Bias 1x1x64	1792
3	relu_conv1 ReLU	ReLU	113x113x64	-	0
4	pool1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	56x56x64	-	0
5	fire2-squeeze1x1 16 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	56x56x16	Weights 1x1x64x16 Bias 1x1x16	1040
6	fire2-relu_squeeze1x1 ReLU	ReLU	56x56x16	-	0
7	fire2-expand3x3 64 3x3x16 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	56x56x64	Weights 3x3x16x64 Bias 1x1x64	9280
8	fire2-expand1x1 64 1x1x16 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	56x56x64	Weights 1x1x16x64 Bias 1x1x64	1088
9	fire2-relu_expand1x1 ReLU	ReLU	56x56x64	-	0
10	fire2-relu_expand3x3 ReLU	ReLU	56x56x64	-	0
11	fire2-concat Depth concatenation of 2 inputs	Depth concatenation	56x56x128	-	0
12	fire3-squeeze1x1 16 1x1x128 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	56x56x16	Weights 1x1x128x16 Bias 1x1x16	2064
13	fire3-relu_squeeze1x1 ReLU	ReLU	56x56x16	-	0
14	fire3-expand1x1 64 1x1x16 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	56x56x64	Weights 1x1x16x64 Bias 1x1x64	1088
15	fire3-relu_expand1x1 ReLU	ReLU	56x56x64	-	0
16	fire3-expand3x3 64 3x3x16 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	56x56x64	Weights 3x3x16x64 Bias 1x1x64	9280
17	fire3-relu_expand3x3 ReLU	ReLU	56x56x64	-	0
18	fire3-concat Depth concatenation of 2 inputs	Depth concatenation	56x56x128	-	0
19	pool3 3x3 max pooling with stride [2 2] and padding [0 1 0 1]	Max Pooling	28x28x128	-	0
20	fire4-squeeze1x1 32 1x1x128 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28x28x32	Weights 1x1x128x32 Bias 1x1x32	4128
21	fire4-relu_squeeze1x1 ReLU	ReLU	28x28x32	-	0
22	fire4-expand1x1 128 1x1x32 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28x28x128	Weights 1x1x32x128 Bias 1x1x128	4224
23	fire4-relu_expand1x1 ReLU	ReLU	28x28x128	-	0
24	fire4-expand3x3 128 3x3x32 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	28x28x128	Weights 3x3x32x128 Bias 1x1x128	36992
25	fire4-relu_expand3x3 ReLU	ReLU	28x28x128	-	0
26	fire4-concat Depth concatenation of 2 inputs	Depth concatenation	28x28x256	-	0
27	fire5-squeeze1x1 32 1x1x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28x28x32	Weights 1x1x256x32 Bias 1x1x32	8224
28	fire5-relu_squeeze1x1 ReLU	ReLU	28x28x32	-	0
29	fire5-expand3x3 128 3x3x32 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	28x28x128	Weights 3x3x32x128 Bias 1x1x128	36992
30	fire5-expand1x1 128 1x1x32 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28x28x128	Weights 1x1x32x128 Bias 1x1x128	4224
31	fire5-relu_expand3x3 ReLU	ReLU	28x28x128	-	0
32	fire5-relu_expand1x1 ReLU	ReLU	28x28x128	-	0
33	fire5-concat Depth concatenation of 2 inputs	Depth concatenation	28x28x256	-	0
34	pool5	Max Pooling	14x14x256	-	0

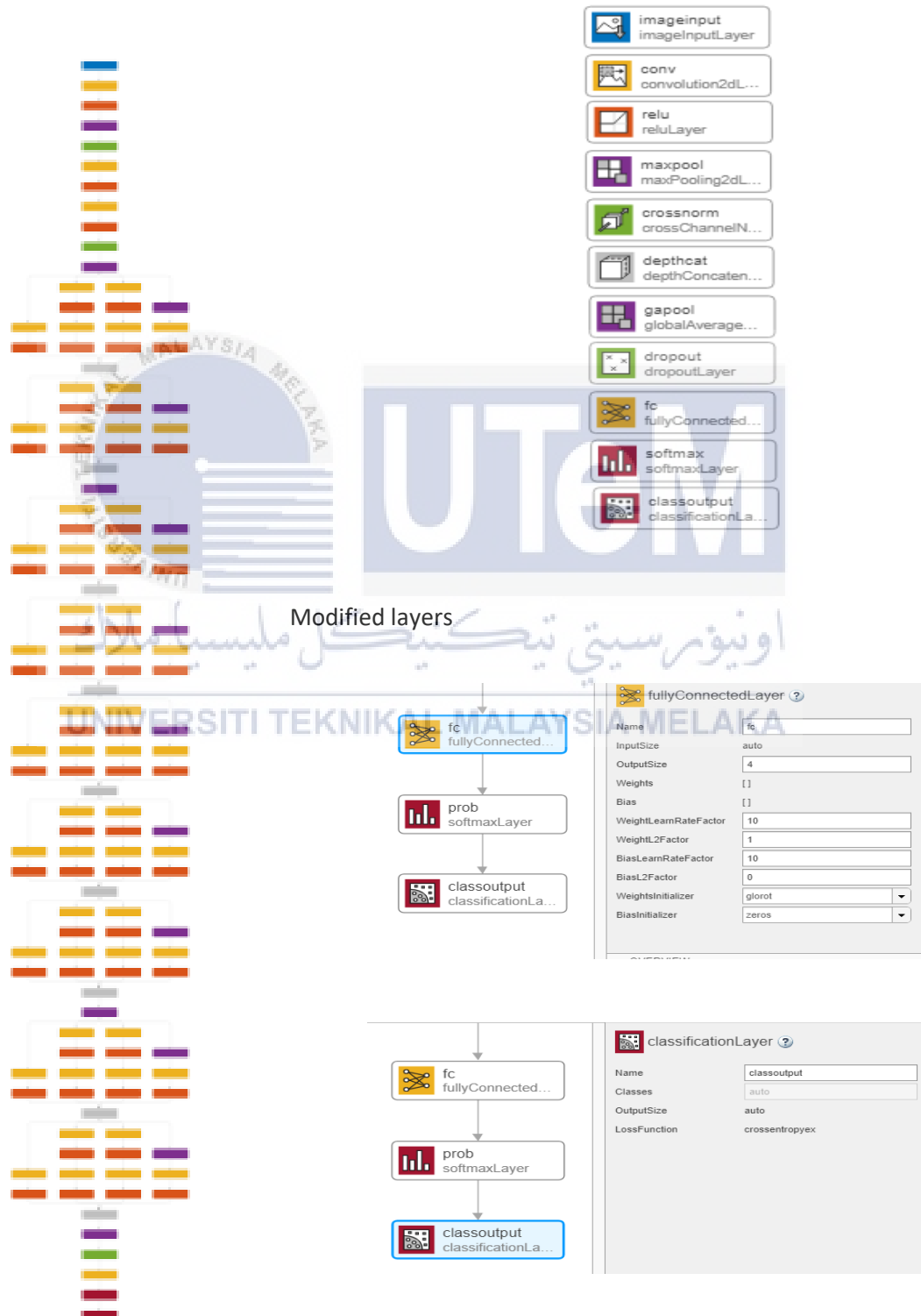
PARAMETERS OF EACH LAYER - SQUEEZENET

34	pool5 3x3 max pooling with stride [2 2] and padding [0 1 0 1]	Max Pooling	14×14×256	-	0
35	fire6-squeeze1x1 48 1x1x256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×48	Weights 1×1×256×48 Bias 1×1×48	12336
36	fire6-relu_squeeze1x1 ReLU	ReLU	14×14×48	-	0
37	fire6-expand1x1 192 1x1x48 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×192	Weights 1×1×48×192 Bias 1×1×192	9408
38	fire6-relu_expand1x1 ReLU	ReLU	14×14×192	-	0
39	fire6-expand3x3 192 3x3x48 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14×14×192	Weights 3×3×48×192 Bias 1×1×192	83136
40	fire6-relu_expand3x3 ReLU	ReLU	14×14×192	-	0
41	fire6-concat Depth concatenation of 2 inputs	Depth concatenation	14×14×384	-	0
42	fire7-squeeze1x1 48 1x1x384 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×48	Weights 1×1×384×48 Bias 1×1×48	18480
43	fire7-relu_squeeze1x1 ReLU	ReLU	14×14×48	-	0
44	fire7-expand3x3 192 3x3x48 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14×14×192	Weights 3×3×48×192 Bias 1×1×192	83136
45	fire7-relu_expand3x3 ReLU	ReLU	14×14×192	-	0
46	fire7-expand1x1 192 1x1x48 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×192	Weights 1×1×48×192 Bias 1×1×192	9408
47	fire7-relu_expand1x1 ReLU	ReLU	14×14×192	-	0
48	fire7-concat Depth concatenation of 2 inputs	Depth concatenation	14×14×384	-	0
49	fire8-squeeze1x1 64 1x1x384 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×64	Weights 1×1×384×64 Bias 1×1×64	24640
50	fire8-relu_squeeze1x1 ReLU	ReLU	14×14×64	-	0
51	fire8-expand1x1 256 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×256	Weights 1×1×64×256 Bias 1×1×256	16640
52	fire8-relu_expand1x1 ReLU	ReLU	14×14×256	-	0
53	fire8-expand3x3 256 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14×14×256	Weights 3×3×64×256 Bias 1×1×256	147712
54	fire8-relu_expand3x3 ReLU	ReLU	14×14×256	-	0
55	fire8-concat Depth concatenation of 2 inputs	Depth concatenation	14×14×512	-	0
56	fire9-squeeze1x1 64 1x1x512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×64	Weights 1×1×512×64 Bias 1×1×64	32832
57	fire9-relu_squeeze1x1 ReLU	ReLU	14×14×64	-	0
58	fire9-expand1x1 256 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×256	Weights 1×1×64×256 Bias 1×1×256	16640
59	fire9-expand3x3 256 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14×14×256	Weights 3×3×64×256 Bias 1×1×256	147712
60	fire9-relu_expand3x3 ReLU	ReLU	14×14×256	-	0
61	fire9-relu_expand1x1 ReLU	ReLU	14×14×256	-	0
62	fire9-concat Depth concatenation of 2 inputs	Depth concatenation	14×14×512	-	0
63	drop9 50% dropout	Dropout	14×14×512	-	0
64	conv 4 1x1x512 convolutions with stride [1 1] and padding 'same'	Convolution	14×14×4	Weights 1×1×512×4 Bias 1×1×4	2052
65	relu_conv10 ReLU	ReLU	14×14×4	-	0
66	pool10 Global average pooling	Global Average Po...	1×1×4	-	0
67	prob softmax	Softmax	1×1×4	-	0
68	classoutput crossentropyex	Classification Output	-	-	0

APPENDIX C

GOOGLNET ARCHITECTURE

Different colour indicates different layers. Total of 144 layers.



PARAMETERS OF EACH LAYER – GOOGLNET

ANALYSIS RESULT				
	Name	Type	Activations	Learnables
1	data 224×224×3 images with 'zerocenter' normalization	Image Input	224×224×3	-
2	conv1-7x7_s2 64 7×7×3 convolutions with stride [2 2] and padding [3 3 3 3]	Convolution	112×112×64	Weights 7×7×3×64 Bias 1×1×64
3	conv1-relu_7x7 ReLU	ReLU	112×112×64	-
4	pool1-3x3_s2 3×3 max pooling with stride [2 2] and padding [0 1 0 1]	Max Pooling	56×56×64	-
5	pool1-norm1 cross channel normalization with 5 channels per element	Cross Channel Nor...	56×56×64	-
6	conv2-3x3_reduce 64 1×1×64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	56×56×64	Weights 1×1×64×64 Bias 1×1×64
7	conv2-relu_3x3_reduce ReLU	ReLU	56×56×64	-
8	conv2-3x3 192 3×3×64 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	56×56×192	Weights 3×3×64×192 Bias 1×1×192
9	conv2-relu_3x3 ReLU	ReLU	56×56×192	-
10	conv2-norm2 cross channel normalization with 5 channels per element	Cross Channel Nor...	56×56×192	-
11	pool2-3x3_s2 3×3 max pooling with stride [2 2] and padding [0 1 0 1]	Max Pooling	28×28×192	-
12	inception_3a-pool 3×3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	28×28×192	-
13	inception_3a-1x1 64 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×64	Weights 1×1×192×64 Bias 1×1×64
14	inception_3a-pool_proj 32 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×32	Weights 1×1×192×32 Bias 1×1×32
15	inception_3a-5x5_reduce 16 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×16	Weights 1×1×192×16 Bias 1×1×16
16	inception_3a-relu_5x5_reduce ReLU	ReLU	28×28×16	-
17	inception_3a-3x3_reduce 96 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×96	Weights 1×1×192×96 Bias 1×1×96
18	inception_3a-relu_3x3_reduce ReLU	ReLU	28×28×96	-
19	inception_3a-3x3 128 3×3×96 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	28×28×128	Weights 3×3×96×128 Bias 1×1×128
20	inception_3a-relu_pool_proj ReLU	ReLU	28×28×32	-
21	inception_3a-5x5 32 5×5×16 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	28×28×32	Weights 5×5×16×32 Bias 1×1×32
22	inception_3a-relu_5x5 ReLU	ReLU	28×28×32	-
23	inception_3a-relu_3x3 ReLU	ReLU	28×28×128	-
24	inception_3a-relu_1x1 ReLU	ReLU	28×28×64	-
25	inception_3a-output Depth concatenation of 4 inputs	Depth concatenation	28×28×256	-
26	inception_3b-1x1 128 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×128	Weights 1×1×256×128 Bias 1×1×128

PARAMETERS OF EACH LAYER – GOOGLNET

	Name	Type	Activations	Learnables
16	inception_3a-relu_5x5_reduce ReLU	ReLU	28×28×16	-
17	inception_3a-3x3_reduce 96 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×96	Weights 1×1×192×96 Bias 1×1×96
18	inception_3a-relu_3x3_reduce ReLU	ReLU	28×28×96	-
19	inception_3a-3x3 128 3×3×96 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	28×28×128	Weights 3×3×96×128 Bias 1×1×128
20	inception_3a-relu_pool_proj ReLU	ReLU	28×28×32	-
21	inception_3a-5x5 32 5×5×16 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	28×28×32	Weights 5×5×16×32 Bias 1×1×32
22	inception_3a-relu_5x5 ReLU	ReLU	28×28×32	-
23	inception_3a-relu_3x3 ReLU	ReLU	28×28×128	-
24	inception_3a-relu_1x1 ReLU	ReLU	28×28×64	-
25	inception_3a-output Depth concatenation of 4 inputs	Depth concatenation	28×28×256	-
26	inception_3b-1x1 128 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×128	Weights 1×1×256×128 Bias 1×1×128
27	inception_3b-3x3_reduce 128 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×128	Weights 1×1×256×128 Bias 1×1×128
28	inception_3b-relu_3x3_reduce ReLU	ReLU	28×28×128	-
29	inception_3b-relu_1x1 ReLU	ReLU	28×28×128	-
30	inception_3b-3x3 192 3×3×128 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	28×28×192	Weights 3×3×128×192 Bias 1×1×192
31	inception_3b-relu_3x3 ReLU	ReLU	28×28×192	-
32	inception_3b-pool 3×3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	28×28×256	-
33	inception_3b-pool_proj 64 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×64	Weights 1×1×256×64 Bias 1×1×64
34	inception_3b-relu_pool_proj ReLU	ReLU	28×28×64	-
35	inception_3b-5x5_reduce 32 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	28×28×32	Weights 1×1×256×32 Bias 1×1×32
36	inception_3b-relu_5x5_reduce ReLU	ReLU	28×28×32	-
37	inception_3b-5x5 96 5×5×32 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	28×28×96	Weights 5×5×32×96 Bias 1×1×96
38	inception_3b-relu_5x5 ReLU	ReLU	28×28×96	-
39	inception_3b-output Depth concatenation of 4 inputs	Depth concatenation	28×28×480	-
40	pool3-3x3_s2 3×3 max pooling with stride [2 2] and padding [0 1 0 1]	Max Pooling	14×14×480	-
41	inception_4a-3x3_reduce 96 1×1×480 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×96	Weights 1×1×480×96 Bias 1×1×96

PARAMETERS OF EACH LAYER – GOOGLNET

ANALYSIS RESULT				
	Name	Type	Activations	Learnables
42	inception_4a-5x5_reduce 16 1x1x480 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x16	Weights 1x1x480x16 Bias 1x1x16
43	inception_4a-relu_5x5_reduce ReLU	ReLU	14x14x16	-
44	inception_4a-1x1 192 1x1x480 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x192	Weights 1x1x480x192 Bias 1x1x192
45	inception_4a-relu_3x3_reduce ReLU	ReLU	14x14x96	-
46	inception_4a-3x3 208 3x3x96 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14x14x208	Weights 3x3x96x208 Bias 1x1x208
47	inception_4a-5x5 48 5x5x16 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	14x14x48	Weights 5x5x16x48 Bias 1x1x48
48	inception_4a-relu_1x1 ReLU	ReLU	14x14x192	-
49	inception_4a-pool 3x3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	14x14x480	-
50	inception_4a-pool_proj 64 1x1x480 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x64	Weights 1x1x480x64 Bias 1x1x64
51	inception_4a-relu_pool_proj ReLU	ReLU	14x14x64	-
52	inception_4a-relu_5x5 ReLU	ReLU	14x14x48	-
53	inception_4a-relu_3x3 ReLU	ReLU	14x14x208	-
54	inception_4a-output Depth concatenation of 4 inputs	Depth concatenation	14x14x512	-
55	inception_4b-5x5_reduce 24 1x1x512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x24	Weights 1x1x512x24 Bias 1x1x24
56	inception_4b-relu_5x5_reduce ReLU	ReLU	14x14x24	-
57	inception_4b-pool 3x3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	14x14x512	-
58	inception_4b-5x5 64 5x5x24 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	14x14x64	Weights 5x5x24x64 Bias 1x1x64
59	inception_4b-relu_5x5 ReLU	ReLU	14x14x64	-
60	inception_4b-3x3_reduce 112 1x1x512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x112	Weights 1x1x512x112 Bias 1x1x112
61	inception_4b-relu_3x3_reduce ReLU	ReLU	14x14x112	-
62	inception_4b-pool_proj 64 1x1x512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x64	Weights 1x1x512x64 Bias 1x1x64
63	inception_4b-relu_pool_proj ReLU	ReLU	14x14x64	-
64	inception_4b-1x1 160 1x1x512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x160	Weights 1x1x512x160 Bias 1x1x160
65	inception_4b-relu_1x1 ReLU	ReLU	14x14x160	-
66	inception_4b-3x3 224 3x3x112 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14x14x224	Weights 3x3x112x224 Bias 1x1x224
67	inception_4b-relu_3x3 ReLU	ReLU	14x14x224	-

PARAMETERS OF EACH LAYER – GOOGLNET

ANALYSIS RESULT				
	Name	Type	Activations	Learnables
	ReLU			
68	inception_4b-output Depth concatenation of 4 inputs	Depth concatenation	14×14×512	-
69	inception_4c-pool 3×3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	14×14×512	-
70	inception_4c-5x5_reduce 24 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×24	Weights 1×1×512×24 Bias 1×1×24
71	inception_4c-pool_proj 64 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×64	Weights 1×1×512×64 Bias 1×1×64
72	inception_4c-1x1 128 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×128	Weights 1×1×512×128 Bias 1×1×128
73	inception_4c-relu_1x1 ReLU	ReLU	14×14×128	-
74	inception_4c-relu_pool_proj ReLU	ReLU	14×14×64	-
75	inception_4c-3x3_reduce 128 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×128	Weights 1×1×512×128 Bias 1×1×128
76	inception_4c-relu_3x3_reduce ReLU	ReLU	14×14×128	-
77	inception_4c-3x3 256 3×3×128 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14×14×256	Weights 3×3×128×256 Bias 1×1×256
78	inception_4c-relu_3x3 ReLU	ReLU	14×14×256	-
79	inception_4c-relu_5x5_reduce ReLU	ReLU	14×14×24	-
80	inception_4c-5x5 64 5×5×24 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	14×14×64	Weights 5×5×24×64 Bias 1×1×64
81	inception_4c-relu_5x5 ReLU	ReLU	14×14×64	-
82	inception_4c-output Depth concatenation of 4 inputs	Depth concatenation	14×14×512	-
83	inception_4d-5x5_reduce 32 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×32	Weights 1×1×512×32 Bias 1×1×32
84	inception_4d-relu_5x5_reduce ReLU	ReLU	14×14×32	-
85	inception_4d-5x5 64 5×5×32 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	14×14×64	Weights 5×5×32×64 Bias 1×1×64
86	inception_4d-3x3_reduce 144 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×144	Weights 1×1×512×144 Bias 1×1×144
87	inception_4d-relu_3x3_reduce ReLU	ReLU	14×14×144	-
88	inception_4d-3x3 288 3×3×144 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14×14×288	Weights 3×3×144×288 Bias 1×1×288
89	inception_4d-relu_3x3 ReLU	ReLU	14×14×288	-
90	inception_4d-relu_5x5 ReLU	ReLU	14×14×64	-
91	inception_4d-1x1 112 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14×14×112	Weights 1×1×512×112 Bias 1×1×112
92	inception_4d-relu_1x1 ReLU	ReLU	14×14×112	-

PARAMETERS OF EACH LAYER – GOOGLNET

	Name	Type	Activations	Learnables
93	inception_4d-pool 3x3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	14x14x512	-
94	inception_4d-pool_proj 64 1x1x512 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x64	Weights 1x1x512x64 Bias 1x1x64
95	inception_4d-relu_pool_proj ReLU	ReLU	14x14x64	-
96	inception_4d-output Depth concatenation of 4 inputs	Depth concatenation	14x14x528	-
97	inception_4e-5x5_reduce 32 1x1x528 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x32	Weights 1x1x528x32 Bias 1x1x32
98	inception_4e-relu_5x5_reduce ReLU	ReLU	14x14x32	-
99	inception_4e-1x1 256 1x1x528 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x256	Weights 1x1x528x256 Bias 1x1x256
100	inception_4e-3x3_reduce 160 1x1x528 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x160	Weights 1x1x528x160 Bias 1x1x160
101	inception_4e-relu_3x3_reduce ReLU	ReLU	14x14x160	-
102	inception_4e-3x3 320 3x3x160 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	14x14x320	Weights 3x3x160x320 Bias 1x1x320
103	inception_4e-5x5 128 5x5x32 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	14x14x128	Weights 5x5x32x128 Bias 1x1x128
104	inception_4e-relu_5x5 ReLU	ReLU	14x14x128	-
105	inception_4e-pool 3x3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	14x14x528	-
106	inception_4e-pool_proj 128 1x1x528 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	14x14x128	Weights 1x1x528x128 Bias 1x1x128
107	inception_4e-relu_pool_proj ReLU	ReLU	14x14x128	-
108	inception_4e-relu_1x1 ReLU	ReLU	14x14x256	-
109	inception_4e-relu_3x3 ReLU	ReLU	14x14x320	-
110	inception_4e-output Depth concatenation of 4 inputs	Depth concatenation	14x14x832	-
111	pool4-3x3_s2 3x3 max pooling with stride [2 2] and padding [0 1 0 1]	Max Pooling	7x7x832	-
112	inception_5a-3x3_reduce 160 1x1x832 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7x7x160	Weights 1x1x832x160 Bias 1x1x160
113	inception_5a-1x1 256 1x1x832 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7x7x256	Weights 1x1x832x256 Bias 1x1x256
114	inception_5a-relu_1x1 ReLU	ReLU	7x7x256	-
115	inception_5a-5x5_reduce 32 1x1x832 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7x7x32	Weights 1x1x832x32 Bias 1x1x32
116	inception_5a-relu_5x5_reduce ReLU	ReLU	7x7x32	-
117	inception_5a-pool 3x3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	7x7x832	-
118	inception_5a-5x5 128 5x5x32 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	7x7x128	Weights 5x5x32x128 Bias 1x1x128

PARAMETERS OF EACH LAYER - GOOGLNET

ANALYSIS RESULT				
	Name	Type	Activations	Learnables
119	inception_5a-relu_5x5 ReLU	ReLU	7×7×128	-
120	inception_5a-relu_3x3_reduce ReLU	ReLU	7×7×160	-
121	inception_5a-3x3 320 3×3×160 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	7×7×320	Weights 3×3×160×320 Bias 1×1×320
122	inception_5a-relu_3x3 ReLU	ReLU	7×7×320	-
123	inception_5a-pool_proj 128 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×7×128	Weights 1×1×832×128 Bias 1×1×128
124	inception_5a-relu_pool_proj ReLU	ReLU	7×7×128	-
125	inception_5a-output Depth concatenation of 4 inputs	Depth concatenation	7×7×832	-
126	inception_5b-1x1 384 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×7×384	Weights 1×1×832×384 Bias 1×1×384
127	inception_5b-relu_1x1 ReLU	ReLU	7×7×384	-
128	inception_5b-pool 3×3 max pooling with stride [1 1] and padding [1 1 1 1]	Max Pooling	7×7×832	-
129	inception_5b-pool_proj 128 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×7×128	Weights 1×1×832×128 Bias 1×1×128
130	inception_5b-relu_pool_proj ReLU	ReLU	7×7×128	-
131	inception_5b-3x3_reduce 192 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×7×192	Weights 1×1×832×192 Bias 1×1×192
132	inception_5b-relu_3x3_reduce ReLU	ReLU	7×7×192	-
133	inception_5b-3x3 384 3×3×192 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	7×7×384	Weights 3×3×192×384 Bias 1×1×384
134	inception_5b-relu_3x3 ReLU	ReLU	7×7×384	-
135	inception_5b-5x5_reduce 48 1×1×832 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×7×48	Weights 1×1×832×48 Bias 1×1×48
136	inception_5b-relu_5x5_reduce ReLU	ReLU	7×7×48	-
137	inception_5b-5x5 128 5×5×48 convolutions with stride [1 1] and padding [2 2 2 2]	Convolution	7×7×128	Weights 5×5×48×128 Bias 1×1×128
138	inception_5b-relu_5x5 ReLU	ReLU	7×7×128	-
139	inception_5b-output Depth concatenation of 4 inputs	Depth concatenation	7×7×1024	-
140	pool5-7x7_s1 Global average pooling	Global Average Po...	1×1×1024	-
141	pool5-drop_7x7_s1 40% dropout	Dropout	1×1×1024	-
142	fc 4 fully connected layer	Fully Connected	1×1×4	Weights 4×1024 Bias 4×1
143	prob softmax	Softmax	1×1×4	-
144	classoutput crossentropyx	Classification Output	-	-

APPENDIX D

SOURCE CODE SQUEEZENET

SQUEEZENET

Load Initial Parameters

Load parameters for network initialization. For transfer learning, the network initialization parameters are the parameters of the initial pretrained network.

```
trainingSetup = load("C:\Users\Asus\Documents\MATLAB\PSM  
2\trainingSetup_2021_01_06_21_22_07.mat");
```

Import Data

Import training and validation data.

```
%Import dataset from folder.  
imdsTrain = imageDatastore("D:\MATLAB\PSM  
2\new", "IncludeSubfolders", true, "LabelSource", "foldernames");  
  
%Split dataset with ratio 8:2.  
[imdsTrain, imdsValidation] = splitEachLabel(imdsTrain, 0.8);  
  
%Image Augmentation by reflection on X axis (flip to left/right).  
imageAugmenter = imageDataAugmenter( ...  
    'RandXReflection', true);  
  
% Resize the images to match the network input layer.  
augimdsTrain = augmentedImageDatastore([227 227  
3], imdsTrain, "DataAugmentation", imageAugmenter);  
augimdsValidation = augmentedImageDatastore([227 227 3], imdsValidation);
```

Set Training Options

Specify hyperparameter to use when training.

```
opts = trainingOptions("sgdm", ...  
    "ExecutionEnvironment", "auto", ...  
    "InitialLearnRate", 0.0001, ...  
    "MaxEpochs", 25, ...  
    "MiniBatchSize", 32, ...  
    "Momentum", 0.9, ...  
    "L2Regularization", 0.0001, ...
```

```

"Shuffle", "every-epoch", ...
"ValidationFrequency", 5, ...
"Plots", "training-progress", ...
"ValidationData", augimdsValidation);

```

Create Layer Graph

Create the layer graph variable to contain the network layers.

```
lgraph = layerGraph();
```

Add Layer Branches

Add the branches of the network to the layer graph. Each branch is a linear array of layers.

```

tempLayers = [
    imageInputLayer([227 227
3], "Name", "data", "Mean", trainingSetup.data.Mean)
    convolution2dLayer([3 3], 64, "Name", "conv1", "Stride", [2
2], "Bias", trainingSetup.conv1.Bias, "Weights", trainingSetup.conv1.Weights)
    reluLayer("Name", "relu_conv1")
    maxPooling2dLayer([3 3], "Name", "pool1", "Stride", [2 2])
    convolution2dLayer([1 1], 16, "Name", "fire2-
squeeze1x1", "Bias", trainingSetup.fire2_squeeze1x1.Bias, "Weights", trainingSe
tup.fire2_squeeze1x1.Weights)
    reluLayer("Name", "fire2-relu_squeeze1x1")];
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
convolution2dLayer([1 1], 64, "Name", "fire2-
expand1x1", "Bias", trainingSetup.fire2_expand1x1.Bias, "Weights", trainingSetu
p.fire2_expand1x1.Weights)
    reluLayer("Name", "fire2-relu_expand1x1")];
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
convolution2dLayer([3 3], 64, "Name", "fire2-expand3x3", "Padding", [1 1 1
1], "Bias", trainingSetup.fire2_expand3x3.Bias, "Weights", trainingSetup.fire2_
expand3x3.Weights)
    reluLayer("Name", "fire2-relu_expand3x3")];
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
depthConcatenationLayer(2, "Name", "fire2-concat")
convolution2dLayer([1 1], 16, "Name", "fire3-
squeeze1x1", "Bias", trainingSetup.fire3_squeeze1x1.Bias, "Weights", trainingSe
tup.fire3_squeeze1x1.Weights)
    reluLayer("Name", "fire3-relu_squeeze1x1")];
lgraph = addLayers(lgraph, tempLayers);

```

```

tempLayers = [
    convolution2dLayer([3 3],64,"Name","fire3-expand3x3","Padding",[1 1 1
1],"Bias",trainingSetup.fire3_expand3x3.Bias,"Weights",trainingSetup.fire3_
expand3x3.Weights)
    reluLayer("Name","fire3-relu_expand3x3");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    convolution2dLayer([1 1],64,"Name","fire3-
expand1x1","Bias",trainingSetup.fire3_expand1x1.Bias,"Weights",trainingSetu
p.fire3_expand1x1.Weights)
    reluLayer("Name","fire3-relu_expand1x1");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    depthConcatenationLayer(2,"Name","fire3-concat")
    maxPooling2dLayer([3 3],"Name","pool3","Padding",[0 1 0 1],"Stride",[2
2])
    convolution2dLayer([1 1],32,"Name","fire4-
squeeze1x1","Bias",trainingSetup.fire4_squeeze1x1.Bias,"Weights",trainingSe
tup.fire4_squeeze1x1.Weights)
    reluLayer("Name","fire4-relu_squeeze1x1");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    convolution2dLayer([1 1],128,"Name","fire4-
expand1x1","Bias",trainingSetup.fire4_expand1x1.Bias,"Weights",trainingSetu
p.fire4_expand1x1.Weights)
    reluLayer("Name","fire4-relu_expand1x1");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    convolution2dLayer([3 3],128,"Name","fire4-expand3x3","Padding",[1 1 1
1],"Bias",trainingSetup.fire4_expand3x3.Bias,"Weights",trainingSetup.fire4_
expand3x3.Weights)
    reluLayer("Name","fire4-relu_expand3x3");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    depthConcatenationLayer(2,"Name","fire4-concat")
    convolution2dLayer([1 1],32,"Name","fire5-
squeeze1x1","Bias",trainingSetup.fire5_squeeze1x1.Bias,"Weights",trainingSe
tup.fire5_squeeze1x1.Weights)
    reluLayer("Name","fire5-relu_squeeze1x1");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
    convolution2dLayer([3 3],128,"Name","fire5-expand3x3","Padding",[1 1 1
1],"Bias",trainingSetup.fire5_expand3x3.Bias,"Weights",trainingSetup.fire5_
expand3x3.Weights)

```

```

    reluLayer("Name", "fire5-relu_expand3x3");
    lgraph = addLayers(lgraph, tempLayers);

    tempLayers = [
        convolution2dLayer([1 1], 128, "Name", "fire5-
expand1x1", "Bias", trainingSetup.fire5_expand1x1.Bias, "Weights", trainingSetu
p.fire5_expand1x1.Weights)
        reluLayer("Name", "fire5-relu_expand1x1");
    ];
    lgraph = addLayers(lgraph, tempLayers);

    tempLayers = [
        depthConcatenationLayer(2, "Name", "fire5-concat")
        maxPooling2dLayer([3 3], "Name", "pool5", "Padding", [0 1 0 1], "Stride", [2
2])
        convolution2dLayer([1 1], 48, "Name", "fire6-
squeeze1x1", "Bias", trainingSetup.fire6_squeeze1x1.Bias, "Weights", trainingSe
tup.fire6_squeeze1x1.Weights)
        reluLayer("Name", "fire6-relu_squeeze1x1");
    ];
    lgraph = addLayers(lgraph, tempLayers);

    tempLayers = [
        convolution2dLayer([1 1], 192, "Name", "fire6-
expand1x1", "Bias", trainingSetup.fire6_expand1x1.Bias, "Weights", trainingSetu
p.fire6_expand1x1.Weights)
        reluLayer("Name", "fire6-relu_expand1x1");
    ];
    lgraph = addLayers(lgraph, tempLayers);

    tempLayers = [
        convolution2dLayer([3 3], 192, "Name", "fire6-expand3x3", "Padding", [1 1 1
1], "Bias", trainingSetup.fire6_expand3x3.Bias, "Weights", trainingSetup.fire6_
expand3x3.Weights)
        reluLayer("Name", "fire6-relu_expand3x3");
    ];
    lgraph = addLayers(lgraph, tempLayers);

    tempLayers = [
        depthConcatenationLayer(2, "Name", "fire6-concat")
        convolution2dLayer([1 1], 48, "Name", "fire7-
squeeze1x1", "Bias", trainingSetup.fire7_squeeze1x1.Bias, "Weights", trainingSe
tup.fire7_squeeze1x1.Weights)
        reluLayer("Name", "fire7-relu_squeeze1x1");
    ];
    lgraph = addLayers(lgraph, tempLayers);

    tempLayers = [
        convolution2dLayer([1 1], 192, "Name", "fire7-
expand1x1", "Bias", trainingSetup.fire7_expand1x1.Bias, "Weights", trainingSetu
p.fire7_expand1x1.Weights)
        reluLayer("Name", "fire7-relu_expand1x1");
    ];
    lgraph = addLayers(lgraph, tempLayers);

    tempLayers = [

```

```

convolution2dLayer([3 3],192,"Name","fire7-expand3x3","Padding",[1 1 1
1],"Bias",trainingSetup.fire7_expand3x3.Bias,"Weights",trainingSetup.fire7_
expand3x3.Weights)
reluLayer("Name","fire7-relu_expand3x3");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
depthConcatenationLayer(2,"Name","fire7-concat")
convolution2dLayer([1 1],64,"Name","fire8-
squeeze1x1","Bias",trainingSetup.fire8_squeeze1x1.Bias,"Weights",trainingSe
tup.fire8_squeeze1x1.Weights)
reluLayer("Name","fire8-relu_squeeze1x1");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
convolution2dLayer([3 3],256,"Name","fire8-expand3x3","Padding",[1 1 1
1],"Bias",trainingSetup.fire8_expand3x3.Bias,"Weights",trainingSetup.fire8_
expand3x3.Weights)
reluLayer("Name","fire8-relu_expand3x3");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
convolution2dLayer([1 1],256,"Name","fire8-
expand1x1","Bias",trainingSetup.fire8_expand1x1.Bias,"Weights",trainingSetu
p.fire8_expand1x1.Weights)
reluLayer("Name","fire8-relu_expand1x1");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
depthConcatenationLayer(2,"Name","fire8-concat")
convolution2dLayer([1 1],64,"Name","fire9-
squeeze1x1","Bias",trainingSetup.fire9_squeeze1x1.Bias,"Weights",trainingSe
tup.fire9_squeeze1x1.Weights)
reluLayer("Name","fire9-relu_squeeze1x1");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
convolution2dLayer([3 3],256,"Name","fire9-expand3x3","Padding",[1 1 1
1],"Bias",trainingSetup.fire9_expand3x3.Bias,"Weights",trainingSetup.fire9_
expand3x3.Weights)
reluLayer("Name","fire9-relu_expand3x3");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
convolution2dLayer([1 1],256,"Name","fire9-
expand1x1","Bias",trainingSetup.fire9_expand1x1.Bias,"Weights",trainingSetu
p.fire9_expand1x1.Weights)
reluLayer("Name","fire9-relu_expand1x1");
lgraph = addLayers(lgraph,tempLayers);

```

```

templayers = [
    depthConcatenationLayer(2,"Name","fire9-concat")
    dropoutLayer(0.5,"Name","drop9")

    %Modified the last convolutional layer.
    convolution2dLayer([1
1],4,"Name","conv","BiasLearnRateFactor",10,"Padding","same","WeightLearnRateFactor",10)
    reluLayer("Name","relu_conv10")
    globalAveragePooling2dLayer("Name","pool10")
    softmaxLayer("Name","softmax")

    %Modified the last classification layer.
    classificationLayer("Name","classoutput")];
lgraph = addLayers(lgraph,templayers);

% clean up helper variable
clear templayers;

```

Connect Layer Branches

Connect all the branches of the network to create the network graph.

```

lgraph = connectLayers(lgraph,"fire2-relu_squeeze1x1","fire2-expand1x1");
lgraph = connectLayers(lgraph,"fire2-relu_squeeze1x1","fire2-expand3x3");
lgraph = connectLayers(lgraph,"fire2-relu_expand3x3","fire2-concat/in2");
lgraph = connectLayers(lgraph,"fire2-relu_expand1x1","fire2-concat/in1");
lgraph = connectLayers(lgraph,"fire3-relu_squeeze1x1","fire3-expand3x3");
lgraph = connectLayers(lgraph,"fire3-relu_squeeze1x1","fire3-expand1x1");
lgraph = connectLayers(lgraph,"fire3-relu_expand1x1","fire3-concat/in1");
lgraph = connectLayers(lgraph,"fire3-relu_expand3x3","fire3-concat/in2");
lgraph = connectLayers(lgraph,"fire4-relu_squeeze1x1","fire4-expand1x1");
lgraph = connectLayers(lgraph,"fire4-relu_squeeze1x1","fire4-expand3x3");
lgraph = connectLayers(lgraph,"fire4-relu_expand1x1","fire4-concat/in1");
lgraph = connectLayers(lgraph,"fire4-relu_expand3x3","fire4-concat/in2");
lgraph = connectLayers(lgraph,"fire5-relu_squeeze1x1","fire5-expand3x3");
lgraph = connectLayers(lgraph,"fire5-relu_squeeze1x1","fire5-expand1x1");
lgraph = connectLayers(lgraph,"fire5-relu_expand3x3","fire5-concat/in2");
lgraph = connectLayers(lgraph,"fire5-relu_expand1x1","fire5-concat/in1");
lgraph = connectLayers(lgraph,"fire6-relu_squeeze1x1","fire6-expand1x1");
lgraph = connectLayers(lgraph,"fire6-relu_squeeze1x1","fire6-expand3x3");
lgraph = connectLayers(lgraph,"fire6-relu_expand3x3","fire6-concat/in2");
lgraph = connectLayers(lgraph,"fire6-relu_expand1x1","fire6-concat/in1");
lgraph = connectLayers(lgraph,"fire7-relu_squeeze1x1","fire7-expand1x1");
lgraph = connectLayers(lgraph,"fire7-relu_squeeze1x1","fire7-expand3x3");
lgraph = connectLayers(lgraph,"fire7-relu_expand1x1","fire7-concat/in1");
lgraph = connectLayers(lgraph,"fire7-relu_expand3x3","fire7-concat/in2");
lgraph = connectLayers(lgraph,"fire8-relu_squeeze1x1","fire8-expand3x3");
lgraph = connectLayers(lgraph,"fire8-relu_squeeze1x1","fire8-expand1x1");
lgraph = connectLayers(lgraph,"fire8-relu_expand3x3","fire8-concat/in2");

```

```

lgraph = connectLayers(lgraph, "fire8-relu_expand1x1", "fire8-concat/in1");
lgraph = connectLayers(lgraph, "fire9-relu_squeeze1x1", "fire9-expand3x3");
lgraph = connectLayers(lgraph, "fire9-relu_squeeze1x1", "fire9-expand1x1");
lgraph = connectLayers(lgraph, "fire9-relu_expand3x3", "fire9-concat/in2");
lgraph = connectLayers(lgraph, "fire9-relu_expand1x1", "fire9-concat/in1");

```

Train Network

Train the network using the specified options and training data.

```

[squeezenet, traininfo] = trainNetwork(augimdsTrain, lgraph, opts);
save squeezenet

```

Create Confusion Matrix

Create the confusion matrix for validation dataset.

```

%Classify the validation dataset using the trained squeezenet.
YPred = classify(squeezenet, augimdsValidation);

%Target class.
YTest = imdsValidation.Labels;

%Plot confusion matrix.
figure, plotconfusion(YTest, YPred);
confMat = confusionmat(YTest, YPred);
confMat = bsxfun(@rdivide, confMat, sum(confMat,2))

%Calculate accuracy-mean of all diagonal box confusion matrix x 100%.
accuracy = mean(diag(confMat)*100);
disp('accuracy')
disp(accuracy)

```

Load Network

Use when need to load the trained squeezenet into workspace.

```
load squeezenet
```

Testing Network

Test the network using the testing dataset.

```

%Read one image from folder.
I = imread('D:\MATLAB\PSM 2\DatasetTest\Sedan\30.png');

%Initiate the timer.
tic;

```



```

%Resize the image.
I = imresize(I, [227 227], 'nearest');

%Processing the image.
[label,scores] = classify(squeezenet,I);

%Calculate the score.
scores = max(double(scores*100));

%Close timer.
time=toc;

%Display the result with title of label and score.
figure
imshow(I)
title(join([string(label), ', ', scores, '%']))

```



APPENDIX E

SOURCE CODE GOOGLNET

GOOGLNET

Load Initial Parameters

Load parameters for network initialization. For transfer learning, the network initialization parameters are the parameters of the initial pretrained network.

```
trainingSetup = load("D:\MATLAB\PSM  
2\trainingSetup_2021_01_07__16_18_56.mat");
```

Import Data

Import training and validation data.

```
%Import dataset from folder.  
imdsTrain = imageDatastore("D:\MATLAB\PSM  
2\new", "IncludeSubfolders", true, "LabelSource", "foldernames");  
  
%Split dataset with ratio 8:2.  
[imdsTrain, imdsValidation] = splitEachLabel(imdsTrain, 0.8);  
  
%Image Augmentation by reflection on X axis (flip to left/right).  
imageAugmenter = imageDataAugmenter( ...  
    'RandXReflection', true);  
  
% Resize the images to match the network input layer.  
augimdsTrain = augmentedImageDatastore([224 224  
3], imdsTrain, "DataAugmentation", imageAugmenter);  
augimdsValidation = augmentedImageDatastore([224 224 3], imdsValidation);
```

Set Training Options

Specify hyperparameter to use when training.

```
opts = trainingOptions("sgdm", ...  
    "ExecutionEnvironment", "auto", ...  
    "InitialLearnRate", 0.0001, ...  
    "MaxEpochs", 25, ...  
    "MiniBatchSize", 32, ...  
    "Momentum", 0.9, ...  
    "L2Regularization", 0.0001, ...  
    "Shuffle", "every-epoch", ...  
    "ValidationFrequency", 5, ...
```

```
"Plots", "training-progress", ...
"ValidationData", augimdsValidation);
```

Create Layer Graph

Create the layer graph variable to contain the network layers.

```
lgraph = layerGraph();
```

Add Layer Branches

Add the branches of the network to the layer graph. Each branch is a linear array of layers.

```
tempLayers = [
    imageInputLayer([224 224
3], "Name", "data", "Mean", trainingSetup.data.Mean)
    convolution2dLayer([7 7], 64, "Name", "conv1-
7x7_s2", "BiasLearnRateFactor", 2, "Padding", [3 3 3 3], "Stride", [2
2], "Bias", trainingSetup.conv1_7x7_s2.Bias, "Weights", trainingSetup.conv1_7x7
_s2.Weights)
    reluLayer("Name", "conv1-relu_7x7")
    maxPooling2dLayer([3 3], "Name", "pool1-3x3_s2", "Padding", [0 1 0
1], "Stride", [2 2])
    crossChannelNormalizationLayer(5, "Name", "pool1-norm1", "K", 1)
    convolution2dLayer([1 1], 64, "Name", "conv2-
3x3_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.conv2_3x3_reduce.B
ias, "Weights", trainingSetup.conv2_3x3_reduce.Weights)
    reluLayer("Name", "conv2-relu_3x3_reduce")
    convolution2dLayer([3 3], 192, "Name", "conv2-
3x3", "BiasLearnRateFactor", 2, "Padding", [1 1 1
1], "Bias", trainingSetup.conv2_3x3.Bias, "Weights", trainingSetup.conv2_3x3.We
ights)
    reluLayer("Name", "conv2-relu_3x3")
    crossChannelNormalizationLayer(5, "Name", "conv2-norm2", "K", 1)
    maxPooling2dLayer([3 3], "Name", "pool2-3x3_s2", "Padding", [0 1 0
1], "Stride", [2 2]);
    lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    convolution2dLayer([1 1], 64, "Name", "inception_3a-
1x1", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_3a_1x1.Bias, "We
ights", trainingSetup.inception_3a_1x1.Weights)
    reluLayer("Name", "inception_3a-relu_1x1")];
    lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    convolution2dLayer([1 1], 16, "Name", "inception_3a-
5x5_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_3a_5x5_r
educe.Bias, "Weights", trainingSetup.inception_3a_5x5_reduce.Weights)
    reluLayer("Name", "inception_3a-relu_5x5_reduce")
```

```

convolution2dLayer([5 5],32,"Name","inception_3a-
5x5","BiasLearnRateFactor",2,"Padding",[2 2 2
2],"Bias",trainingSetup.inception_3a_5x5.Bias,"Weights",trainingSetup.incep
tion_3a_5x5.Weights)
reluLayer("Name","inception_3a-relu_5x5");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
maxPooling2dLayer([3 3],"Name","inception_3a-pool","Padding",[1 1 1
1])
convolution2dLayer([1 1],32,"Name","inception_3a-
pool_proj","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_3a_pool_p
roj.Bias,"Weights",trainingSetup.inception_3a_pool_proj.Weights)
reluLayer("Name","inception_3a-relu_pool_proj)];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
convolution2dLayer([1 1],96,"Name","inception_3a-
3x3_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_3a_3x3_r
educate.Bias,"Weights",trainingSetup.inception_3a_3x3_reduce.Weights)
reluLayer("Name","inception_3a-relu_3x3_reduce")
convolution2dLayer([3 3],128,"Name","inception_3a-
3x3","BiasLearnRateFactor",2,"Padding",[1 1 1
1],"Bias",trainingSetup.inception_3a_3x3.Bias,"Weights",trainingSetup.incep
tion_3a_3x3.Weights)
reluLayer("Name","inception_3a-relu_3x3)];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = depthConcatenationLayer(4,"Name","inception_3a-output");
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
convolution2dLayer([1 1],128,"Name","inception_3b-
1x1","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_3b_1x1.Bias,"We
ights",trainingSetup.inception_3b_1x1.Weights)
reluLayer("Name","inception_3b-relu_1x1)];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [
maxPooling2dLayer([3 3],"Name","inception_3b-pool","Padding",[1 1 1
1])
convolution2dLayer([1 1],64,"Name","inception_3b-
pool_proj","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_3b_pool_p
roj.Bias,"Weights",trainingSetup.inception_3b_pool_proj.Weights)
reluLayer("Name","inception_3b-relu_pool_proj)];
lgraph = addLayers(lgraph,tempLayers);

tempLayers = [

```

```

        convolution2dLayer([1 1],32,"Name","inception_3b-
5x5_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_3b_5x5_r
educer.Bias,"Weights",trainingSetup.inception_3b_5x5_reduce.Weights)
        reluLayer("Name","inception_3b-relu_5x5_reduce")
        convolution2dLayer([5 5],96,"Name","inception_3b-
5x5","BiasLearnRateFactor",2,"Padding",[2 2 2
2],"Bias",trainingSetup.inception_3b_5x5.Bias,"Weights",trainingSetup.incep
tion_3b_5x5.Weights)
        reluLayer("Name","inception_3b-relu_5x5");
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],128,"Name","inception_3b-
3x3_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_3b_3x3_r
educer.Bias,"Weights",trainingSetup.inception_3b_3x3_reduce.Weights)
        reluLayer("Name","inception_3b-relu_3x3_reduce")
        convolution2dLayer([3 3],192,"Name","inception_3b-
3x3","BiasLearnRateFactor",2,"Padding",[1 1 1
1],"Bias",trainingSetup.inception_3b_3x3.Bias,"Weights",trainingSetup.incep
tion_3b_3x3.Weights)
        reluLayer("Name","inception_3b-relu_3x3");
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        depthConcatenationLayer(4,"Name","inception_3b-output")
        maxPooling2dLayer([3 3],"Name","pool3-3x3_s2","Padding",[0 1 0
1],"Stride",[2 2]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],16,"Name","inception_4a-
5x5_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4a_5x5_r
educer.Bias,"Weights",trainingSetup.inception_4a_5x5_reduce.Weights)
        reluLayer("Name","inception_4a-relu_5x5_reduce")
        convolution2dLayer([5 5],48,"Name","inception_4a-
5x5","BiasLearnRateFactor",2,"Padding",[2 2 2
2],"Bias",trainingSetup.inception_4a_5x5.Bias,"Weights",trainingSetup.incep
tion_4a_5x5.Weights)
        reluLayer("Name","inception_4a-relu_5x5");
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],192,"Name","inception_4a-
1x1","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4a_1x1.Bias,"We
ights",trainingSetup.inception_4a_1x1.Weights)
        reluLayer("Name","inception_4a-relu_1x1");
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [

```

```

    maxPooling2dLayer([3 3], "Name", "inception_4a-pool", "Padding", [1 1 1
1])
    convolution2dLayer([1 1], 64, "Name", "inception_4a-
pool_proj", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_4a_pool_p
roj.Bias, "Weights", trainingSetup.inception_4a_pool_proj.Weights)
    reluLayer("Name", "inception_4a-relu_pool_proj"]);
    lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    convolution2dLayer([1 1], 96, "Name", "inception_4a-
3x3_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_4a_3x3_r
educate.Bias, "Weights", trainingSetup.inception_4a_3x3_reduce.Weights)
    reluLayer("Name", "inception_4a-relu_3x3_reduce")
    convolution2dLayer([3 3], 208, "Name", "inception_4a-
3x3", "BiasLearnRateFactor", 2, "Padding", [1 1 1
1], "Bias", trainingSetup.inception_4a_3x3.Bias, "Weights", trainingSetup.incep
tion_4a_3x3.Weights)
    reluLayer("Name", "inception_4a-relu_3x3");
    lgraph = addLayers(lgraph, tempLayers);

tempLayers = depthConcatenationLayer(4, "Name", "inception_4a-output");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    convolution2dLayer([1 1], 112, "Name", "inception_4b-
3x3_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_4b_3x3_r
educate.Bias, "Weights", trainingSetup.inception_4b_3x3_reduce.Weights)
    reluLayer("Name", "inception_4b-relu_3x3_reduce")
    convolution2dLayer([3 3], 224, "Name", "inception_4b-
3x3", "BiasLearnRateFactor", 2, "Padding", [1 1 1
1], "Bias", trainingSetup.inception_4b_3x3.Bias, "Weights", trainingSetup.incep
tion_4b_3x3.Weights)
    reluLayer("Name", "inception_4b-relu_3x3");
    lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    maxPooling2dLayer([3 3], "Name", "inception_4b-pool", "Padding", [1 1 1
1])
    convolution2dLayer([1 1], 64, "Name", "inception_4b-
pool_proj", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_4b_pool_p
roj.Bias, "Weights", trainingSetup.inception_4b_pool_proj.Weights)
    reluLayer("Name", "inception_4b-relu_pool_proj"]);
    lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    convolution2dLayer([1 1], 24, "Name", "inception_4b-
5x5_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_4b_5x5_r
educate.Bias, "Weights", trainingSetup.inception_4b_5x5_reduce.Weights)
    reluLayer("Name", "inception_4b-relu_5x5_reduce")

```

```

        convolution2dLayer([5 5],64,"Name","inception_4b-
5x5","BiasLearnRateFactor",2,"Padding",[2 2 2
2],"Bias",trainingSetup.inception_4b_5x5.Bias,"Weights",trainingSetup.incep
tion_4b_5x5.Weights)
        reluLayer("Name","inception_4b-relu_5x5"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],160,"Name","inception_4b-
1x1","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4b_1x1.Bias,"We
ights",trainingSetup.inception_4b_1x1.Weights)
        reluLayer("Name","inception_4b-relu_1x1"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = depthConcatenationLayer(4,"Name","inception_4b-output");
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        maxPooling2dLayer([3 3],"Name","inception_4c-pool","Padding",[1 1 1
1])
        convolution2dLayer([1 1],64,"Name","inception_4c-
pool_proj","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4c_pool_p
roj.Bias,"Weights",trainingSetup.inception_4c_pool_proj.Weights)
        reluLayer("Name","inception_4c-relu_pool_proj"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],24,"Name","inception_4c-
5x5_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4c_5x5_r
educe.Bias,"Weights",trainingSetup.inception_4c_5x5_reduce.Weights)
        reluLayer("Name","inception_4c-relu_5x5_reduce")
        convolution2dLayer([5 5],64,"Name","inception_4c-
5x5","BiasLearnRateFactor",2,"Padding",[2 2 2
2],"Bias",trainingSetup.inception_4c_5x5.Bias,"Weights",trainingSetup.incep
tion_4c_5x5.Weights)
        reluLayer("Name","inception_4c-relu_5x5"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],128,"Name","inception_4c-
1x1","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4c_1x1.Bias,"We
ights",trainingSetup.inception_4c_1x1.Weights)
        reluLayer("Name","inception_4c-relu_1x1"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],128,"Name","inception_4c-
3x3_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4c_3x3_r
educe.Bias,"Weights",trainingSetup.inception_4c_3x3_reduce.Weights)
        reluLayer("Name","inception_4c-relu_3x3_reduce")
    ]

```

```

        convolution2dLayer([3 3],256,"Name","inception_4c-
3x3","BiasLearnRateFactor",2,"Padding",[1 1 1
1],"Bias",trainingSetup.inception_4c_3x3.Bias,"Weights",trainingSetup.incep
tion_4c_3x3.Weights)
        reluLayer("Name","inception_4c-relu_3x3"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = depthConcatenationLayer(4,"Name","inception_4c-output");
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],144,"Name","inception_4d-
3x3_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4d_3x3_r
educate.Bias,"Weights",trainingSetup.inception_4d_3x3_reduce.Weights)
        reluLayer("Name","inception_4d-relu_3x3_reduce")
        convolution2dLayer([3 3],288,"Name","inception_4d-
3x3","BiasLearnRateFactor",2,"Padding",[1 1 1
1],"Bias",trainingSetup.inception_4d_3x3.Bias,"Weights",trainingSetup.incep
tion_4d_3x3.Weights)
        reluLayer("Name","inception_4d-relu_3x3"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],32,"Name","inception_4d-
5x5_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4d_5x5_r
educate.Bias,"Weights",trainingSetup.inception_4d_5x5_reduce.Weights)
        reluLayer("Name","inception_4d-relu_5x5_reduce")
        convolution2dLayer([5 5],64,"Name","inception_4d-
5x5","BiasLearnRateFactor",2,"Padding",[2 2 2
2],"Bias",trainingSetup.inception_4d_5x5.Bias,"Weights",trainingSetup.incep
tion_4d_5x5.Weights)
        reluLayer("Name","inception_4d-relu_5x5"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        maxPooling2dLayer([3 3],"Name","inception_4d-pool","Padding",[1 1 1
1])
        convolution2dLayer([1 1],64,"Name","inception_4d-
pool_proj","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4d_pool_p
roj.Bias,"Weights",trainingSetup.inception_4d_pool_proj.Weights)
        reluLayer("Name","inception_4d-relu_pool_proj"]);
    lgraph = addLayers(lgraph,tempLayers);

    tempLayers = [
        convolution2dLayer([1 1],112,"Name","inception_4d-
1x1","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4d_1x1.Bias,"We
ights",trainingSetup.inception_4d_1x1.Weights)
        reluLayer("Name","inception_4d-relu_1x1"]);
    lgraph = addLayers(lgraph,tempLayers);

```



```

templayers = depthConcatenationLayer(4,"Name","inception_4d-output");
lgraph = addLayers(lgraph,templayers);

templayers = [
    convolution2dLayer([1 1],256,"Name","inception_4e-
1x1","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4e_1x1.Bias,"We
ights",trainingSetup.inception_4e_1x1.Weights)
    reluLayer("Name","inception_4e-relu_1x1");
lgraph = addLayers(lgraph,templayers);

templayers = [
    convolution2dLayer([1 1],160,"Name","inception_4e-
3x3_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4e_3x3_r
educate.Bias,"Weights",trainingSetup.inception_4e_3x3_reduce.Weights)
    reluLayer("Name","inception_4e-relu_3x3_reduce")
    convolution2dLayer([3 3],320,"Name","inception_4e-
3x3","BiasLearnRateFactor",2,"Padding",[1 1 1
1],"Bias",trainingSetup.inception_4e_3x3.Bias,"Weights",trainingSetup.incep
tion_4e_3x3.Weights)
    reluLayer("Name","inception_4e-relu_3x3");
lgraph = addLayers(lgraph,templayers);

templayers = [
    maxPooling2dLayer([3 3],"Name","inception_4e-pool","Padding",[1 1 1
1])
    convolution2dLayer([1 1],128,"Name","inception_4e-
pool_proj","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4e_pool_p
roj.Bias,"Weights",trainingSetup.inception_4e_pool_proj.Weights)
    reluLayer("Name","inception_4e-relu_pool_proj");
lgraph = addLayers(lgraph,templayers);

templayers = [
    convolution2dLayer([1 1],32,"Name","inception_4e-
5x5_reduce","BiasLearnRateFactor",2,"Bias",trainingSetup.inception_4e_5x5_r
educate.Bias,"Weights",trainingSetup.inception_4e_5x5_reduce.Weights)
    reluLayer("Name","inception_4e-relu_5x5_reduce")
    convolution2dLayer([5 5],128,"Name","inception_4e-
5x5","BiasLearnRateFactor",2,"Padding",[2 2 2
2],"Bias",trainingSetup.inception_4e_5x5.Bias,"Weights",trainingSetup.incep
tion_4e_5x5.Weights)
    reluLayer("Name","inception_4e-relu_5x5");
lgraph = addLayers(lgraph,templayers);

templayers = [
    depthConcatenationLayer(4,"Name","inception_4e-output")
    maxPooling2dLayer([3 3],"Name","pool4-3x3_s2","Padding",[0 1 0
1],"Stride",[2 2]);
lgraph = addLayers(lgraph,templayers);

templayers = [

```

```

maxPooling2dLayer([3 3], "Name", "inception_5a-pool", "Padding", [1 1 1
1])
convolution2dLayer([1 1], 128, "Name", "inception_5a-
pool_proj", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_5a_pool_p
roj.Bias, "Weights", trainingSetup.inception_5a_pool_proj.Weights)
reluLayer("Name", "inception_5a-relu_pool_proj");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
convolution2dLayer([1 1], 160, "Name", "inception_5a-
3x3_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_5a_3x3_r
educate.Bias, "Weights", trainingSetup.inception_5a_3x3_reduce.Weights)
reluLayer("Name", "inception_5a-relu_3x3_reduce")
convolution2dLayer([3 3], 320, "Name", "inception_5a-
3x3", "BiasLearnRateFactor", 2, "Padding", [1 1 1
1], "Bias", trainingSetup.inception_5a_3x3.Bias, "Weights", trainingSetup.incep
tion_5a_3x3.Weights)
reluLayer("Name", "inception_5a-relu_3x3");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
convolution2dLayer([1 1], 256, "Name", "inception_5a-
1x1", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_5a_1x1.Bias, "We
ights", trainingSetup.inception_5a_1x1.Weights)
reluLayer("Name", "inception_5a-relu_1x1");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
convolution2dLayer([1 1], 32, "Name", "inception_5a-
5x5_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_5a_5x5_r
educate.Bias, "Weights", trainingSetup.inception_5a_5x5_reduce.Weights)
reluLayer("Name", "inception_5a-relu_5x5_reduce")
convolution2dLayer([5 5], 128, "Name", "inception_5a-
5x5", "BiasLearnRateFactor", 2, "Padding", [2 2 2
2], "Bias", trainingSetup.inception_5a_5x5.Bias, "Weights", trainingSetup.incep
tion_5a_5x5.Weights)
reluLayer("Name", "inception_5a-relu_5x5");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = depthConcatenationLayer(4, "Name", "inception_5a-output");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
convolution2dLayer([1 1], 192, "Name", "inception_5b-
3x3_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_5b_3x3_r
educate.Bias, "Weights", trainingSetup.inception_5b_3x3_reduce.Weights)
reluLayer("Name", "inception_5b-relu_3x3_reduce")
convolution2dLayer([3 3], 384, "Name", "inception_5b-
3x3", "BiasLearnRateFactor", 2, "Padding", [1 1 1

```

```

1], "Bias", trainingSetup.inception_5b_3x3.Bias, "Weights", trainingSetup.incep
tion_5b_3x3.Weights)
    reluLayer("Name", "inception_5b-relu_3x3");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    maxPooling2dLayer([3 3], "Name", "inception_5b-pool", "Padding", [1 1 1
1])
    convolution2dLayer([1 1], 128, "Name", "inception_5b-
pool_proj", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_5b_pool_p
roj.Bias, "Weights", trainingSetup.inception_5b_pool_proj.Weights)
    reluLayer("Name", "inception_5b-relu_pool_proj");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    convolution2dLayer([1 1], 48, "Name", "inception_5b-
5x5_reduce", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_5b_5x5_r
educate.Bias, "Weights", trainingSetup.inception_5b_5x5_reduce.Weights)
    reluLayer("Name", "inception_5b-relu_5x5_reduce")
    convolution2dLayer([5 5], 128, "Name", "inception_5b-
5x5", "BiasLearnRateFactor", 2, "Padding", [2 2 2
2], "Bias", trainingSetup.inception_5b_5x5.Bias, "Weights", trainingSetup.incep
tion_5b_5x5.Weights)
    reluLayer("Name", "inception_5b-relu_5x5");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    convolution2dLayer([1 1], 384, "Name", "inception_5b-
1x1", "BiasLearnRateFactor", 2, "Bias", trainingSetup.inception_5b_1x1.Bias, "We
ights", trainingSetup.inception_5b_1x1.Weights)
    reluLayer("Name", "inception_5b-relu_1x1");
lgraph = addLayers(lgraph, tempLayers);

tempLayers = [
    depthConcatenationLayer(4, "Name", "inception_5b-output")
    globalAveragePooling2dLayer("Name", "pool5-7x7_s1")
    dropoutLayer(0.4, "Name", "pool5-drop_7x7_s1")

    %Modified the last fully connected layer.

fullyConnectedLayer(4, "Name", "fc", "BiasLearnRateFactor", 10, "WeightLearnRate
Factor", 10)

    softmaxLayer("Name", "prob")

    %Modified the last classification layer.
    classificationLayer("Name", "classoutput");
lgraph = addLayers(lgraph, tempLayers);

% clean up helper variable

```

```
clear tempLayers;
```

Connect Layer Branches

Connect all the branches of the network to create the network graph.

```
lgraph = connectLayers(lgraph, "pool2-3x3_s2", "inception_3a-1x1");
lgraph = connectLayers(lgraph, "pool2-3x3_s2", "inception_3a-5x5_reduce");
lgraph = connectLayers(lgraph, "pool2-3x3_s2", "inception_3a-pool");
lgraph = connectLayers(lgraph, "pool2-3x3_s2", "inception_3a-3x3_reduce");
lgraph = connectLayers(lgraph, "inception_3a-relu_pool_proj", "inception_3a-
output/in4");
lgraph = connectLayers(lgraph, "inception_3a-relu_5x5", "inception_3a-
output/in3");
lgraph = connectLayers(lgraph, "inception_3a-relu_1x1", "inception_3a-
output/in1");
lgraph = connectLayers(lgraph, "inception_3a-relu_3x3", "inception_3a-
output/in2");
lgraph = connectLayers(lgraph, "inception_3a-output", "inception_3b-1x1");
lgraph = connectLayers(lgraph, "inception_3a-output", "inception_3b-pool");
lgraph = connectLayers(lgraph, "inception_3a-output", "inception_3b-
5x5_reduce");
lgraph = connectLayers(lgraph, "inception_3a-output", "inception_3b-
3x3_reduce");
lgraph = connectLayers(lgraph, "inception_3b-relu_1x1", "inception_3b-
output/in1");
lgraph = connectLayers(lgraph, "inception_3b-relu_pool_proj", "inception_3b-
output/in4");
lgraph = connectLayers(lgraph, "inception_3b-relu_3x3", "inception_3b-
output/in2");
lgraph = connectLayers(lgraph, "inception_3b-relu_5x5", "inception_3b-
output/in3");
lgraph = connectLayers(lgraph, "pool3-3x3_s2", "inception_4a-5x5_reduce");
lgraph = connectLayers(lgraph, "pool3-3x3_s2", "inception_4a-1x1");
lgraph = connectLayers(lgraph, "pool3-3x3_s2", "inception_4a-pool");
lgraph = connectLayers(lgraph, "pool3-3x3_s2", "inception_4a-3x3_reduce");
lgraph = connectLayers(lgraph, "inception_4a-relu_1x1", "inception_4a-
output/in1");
lgraph = connectLayers(lgraph, "inception_4a-relu_5x5", "inception_4a-
output/in3");
lgraph = connectLayers(lgraph, "inception_4a-relu_pool_proj", "inception_4a-
output/in4");
lgraph = connectLayers(lgraph, "inception_4a-relu_3x3", "inception_4a-
output/in2");
lgraph = connectLayers(lgraph, "inception_4a-output", "inception_4b-
3x3_reduce");
lgraph = connectLayers(lgraph, "inception_4a-output", "inception_4b-pool");
lgraph = connectLayers(lgraph, "inception_4a-output", "inception_4b-
5x5_reduce");
lgraph = connectLayers(lgraph, "inception_4a-output", "inception_4b-1x1");
```

```

lgraph = connectLayers(lgraph, "inception_4b-relu_pool_proj", "inception_4b-
output/in4");
lgraph = connectLayers(lgraph, "inception_4b-relu_1x1", "inception_4b-
output/in1");
lgraph = connectLayers(lgraph, "inception_4b-relu_3x3", "inception_4b-
output/in2");
lgraph = connectLayers(lgraph, "inception_4b-relu_5x5", "inception_4b-
output/in3");
lgraph = connectLayers(lgraph, "inception_4b-output", "inception_4c-pool");
lgraph = connectLayers(lgraph, "inception_4b-output", "inception_4c-
5x5_reduce");
lgraph = connectLayers(lgraph, "inception_4b-output", "inception_4c-1x1");
lgraph = connectLayers(lgraph, "inception_4b-output", "inception_4c-
3x3_reduce");
lgraph = connectLayers(lgraph, "inception_4c-relu_pool_proj", "inception_4c-
output/in4");
lgraph = connectLayers(lgraph, "inception_4c-relu_5x5", "inception_4c-
output/in3");
lgraph = connectLayers(lgraph, "inception_4c-relu_1x1", "inception_4c-
output/in1");
lgraph = connectLayers(lgraph, "inception_4c-relu_3x3", "inception_4c-
output/in2");
lgraph = connectLayers(lgraph, "inception_4c-output", "inception_4d-
3x3_reduce");
lgraph = connectLayers(lgraph, "inception_4c-output", "inception_4d-
5x5_reduce");
lgraph = connectLayers(lgraph, "inception_4c-output", "inception_4d-pool");
lgraph = connectLayers(lgraph, "inception_4c-output", "inception_4d-1x1");
lgraph = connectLayers(lgraph, "inception_4d-relu_3x3", "inception_4d-
output/in2");
lgraph = connectLayers(lgraph, "inception_4d-relu_pool_proj", "inception_4d-
output/in4");
lgraph = connectLayers(lgraph, "inception_4d-relu_5x5", "inception_4d-
output/in3");
lgraph = connectLayers(lgraph, "inception_4d-relu_1x1", "inception_4d-
output/in1");
lgraph = connectLayers(lgraph, "inception_4d-output", "inception_4e-1x1");
lgraph = connectLayers(lgraph, "inception_4d-output", "inception_4e-
3x3_reduce");
lgraph = connectLayers(lgraph, "inception_4d-output", "inception_4e-pool");
lgraph = connectLayers(lgraph, "inception_4d-output", "inception_4e-
5x5_reduce");
lgraph = connectLayers(lgraph, "inception_4e-relu_pool_proj", "inception_4e-
output/in4");
lgraph = connectLayers(lgraph, "inception_4e-relu_3x3", "inception_4e-
output/in2");
lgraph = connectLayers(lgraph, "inception_4e-relu_1x1", "inception_4e-
output/in1");
lgraph = connectLayers(lgraph, "inception_4e-relu_5x5", "inception_4e-
output/in3");

```

```

lgraph = connectLayers(lgraph, "pool4-3x3_s2", "inception_5a-pool");
lgraph = connectLayers(lgraph, "pool4-3x3_s2", "inception_5a-3x3_reduce");
lgraph = connectLayers(lgraph, "pool4-3x3_s2", "inception_5a-1x1");
lgraph = connectLayers(lgraph, "pool4-3x3_s2", "inception_5a-5x5_reduce");
lgraph = connectLayers(lgraph, "inception_5a-relu_1x1", "inception_5a-
output/in1");
lgraph = connectLayers(lgraph, "inception_5a-relu_pool_proj", "inception_5a-
output/in4");
lgraph = connectLayers(lgraph, "inception_5a-relu_5x5", "inception_5a-
output/in3");
lgraph = connectLayers(lgraph, "inception_5a-relu_3x3", "inception_5a-
output/in2");
lgraph = connectLayers(lgraph, "inception_5a-output", "inception_5b-
3x3_reduce");
lgraph = connectLayers(lgraph, "inception_5a-output", "inception_5b-pool");
lgraph = connectLayers(lgraph, "inception_5a-output", "inception_5b-
5x5_reduce");
lgraph = connectLayers(lgraph, "inception_5a-output", "inception_5b-1x1");
lgraph = connectLayers(lgraph, "inception_5b-relu_1x1", "inception_5b-
output/in1");
lgraph = connectLayers(lgraph, "inception_5b-relu_pool_proj", "inception_5b-
output/in4");
lgraph = connectLayers(lgraph, "inception_5b-relu_5x5", "inception_5b-
output/in3");
lgraph = connectLayers(lgraph, "inception_5b-relu_3x3", "inception_5b-
output/in2");

```

Train Network

Train the network using the specified options and training data.

```

[googlenet, traininfo] = trainNetwork(augimdsTrain, lgraph, opts);
save googlenet

```

Create Confusion Matrix

Create the confusion matrix for validation dataset.

```

%Classify the validation dataset using the trained googlenet.
YPred = classify(googlenet, augimdsValidation);

%Target class.
YTest = imdsValidation.Labels;

%Plot confusion matrix.
figure, plotconfusion(YTest, YPred);
confMat = confusionmat(YTest, YPred);
confMat = bsxfun(@rdivide, confMat, sum(confMat,2))

%Calculate accuracy- mean of all diagonal box confusion matrix x 100%.

```

```
accuracy = mean(diag(confMat)*100);  
disp('accuracy')  
disp(accuracy)
```

Load Network

Use when need to load the trained googlenet into workspace.

```
load googlenet
```

Testing Network

Test the network using the testing dataset.

```
%Read one image from folder.  
I = imread('D:\MATLAB\PSM 2\DatasetTest\Detect No Vehicle\31.png');  
  
%Initiate the timer.  
tic;  
  
%Resize the image.  
I = imresize(I, [224 224], 'nearest');  
  
%Processing the image.  
[label,scores] = classify(googlenet,I);  
  
%Calculate the score.  
scores = max(double(scores*100));  
  
%Close timer.  
time=toc;  
  
%Display the result with title of label and score.  
figure  
imshow(I)  
title(join([string(label), ', ', scores, '%']))
```