

CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURE FOR TROPICAL FRUIT CLASSIFICATION

MUHAMMAD HAZIQ BIN SHAARI

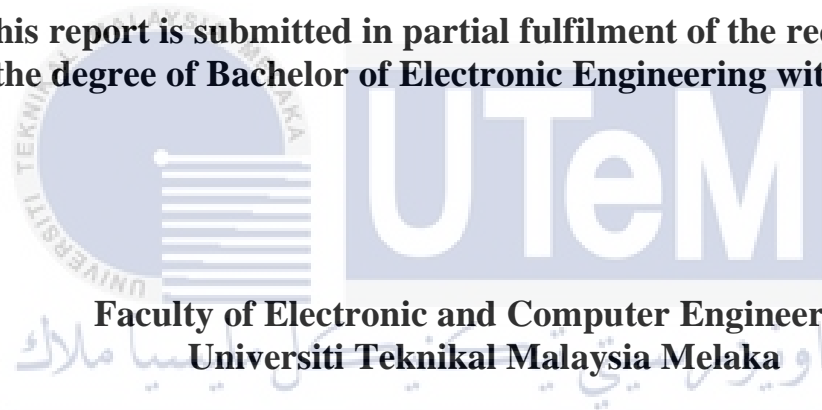


UNIVERSITI TEKNIKAL MALAYSIA MELAKA

CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURE FOR TROPICAL FRUIT CLASSIFICATION

MUHAMMAD HAZIQ BIN SHAARI

**This report is submitted in partial fulfilment of the requirements
for the degree of Bachelor of Electronic Engineering with Honours**



**Faculty of Electronic and Computer Engineering
Universiti Teknikal Malaysia Melaka**

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2022

DECLARATION

I declare that this report entitled “Convolutional Neural Network (CNN) architecture for tropical fruit classification” is the result of my own work except for quotes as cited in the references.



اونيورسيٲى ٲكنيكل ماليسيا ملاك

Signature :

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

Author : Muhammad Haziq bin Shaari

Date : 18 Jun 2022

APPROVAL

I hereby declare that I have read this thesis and in my opinion this thesis is sufficient in terms of scope and quality for the award of Bachelor of Electronic Engineering with Honours.



Signature : 
UNIVERSITI TEKNIKAL MALAYSIA MELAKA

Supervisor Name : Dr. Abd Shukur bin Ja'afar

Date : 18 Jun 2022

DEDICATION

I dedicate this thesis to Allah SWT for giving me everything to complete this project
with ease.



ABSTRACT

Convolutional Neural Network (CNN) is a robust computer vision algorithm from artificial intelligence (AI) that use deep learning networks to analyse visual imagery. The predecessor, machine learning, is also a type of AI but with a lower performance than deep learning. The aim of this project is to design a CNN model architecture layer on par with pre-trained models such as ResNet50, VGG16, MobileNetV2, and DenseNet121 using 15 classes of tropical fruit with total of 15000 images. The model was designed on Google Colab using TensorFlow and Keras library. Next, the network model needs to be trained, test, and optimized by tuning the hyperparameters. Then, all simulation results were compared to the well-known pre-trained models in terms of accuracy and confusion matrix. At the end of this project, the H-CNN able to reach 90.13% accuracy with balance distribution across all of the tropical fruit classification.

ABSTRAK

Rangkaian Saraf Konvolusional (CNN) ialah algoritma penglihatan komputer yang mantap daripada kecerdasan buatan (AI) yang menggunakan rangkaian pembelajaran mendalam untuk menganalisis pengimejan visual. Sebelum ini, pembelajaran mesin, juga merupakan jenis AI tetapi dengan prestasi yang lebih rendah daripada pembelajaran mendalam. Matlamat projek ini adalah untuk mereka bentuk lapisan seni bina model CNN yang setanding dengan model pra-latihan seperti ResNet50, VGG16, MobileNetV2 dan DenseNet121 dengan menggunakan 15 kelas buah tropika dengan berjumlah 15000 imej. Model ini telah direka bentuk pada Google Colab menggunakan Pustaka TensorFlow dan Keras. Seterusnya, model rangkaian perlu dilatih, diuji dan dioptimumkan dengan penalaan hiperparameter. Kemudian, semua hasil simulasi dibandingkan dengan model pra-latihan yang terkenal dari segi ketepatan dan matriks kekeliruan. Pada akhir projek ini, H-CNN mampu mencapai ketepatan 90.13% dengan taburan yang seimbang yang merentasi semua klasifikasi buah tropika.

ACKNOWLEDGEMENTS

Foremost, I would like to thank my family, my parent Siti Kalsom binti Samsuri, siblings, Muhammad Hafiz, Natasha, Muhammad Hazri and Muhammad Hamim Hakim, for always supporting me spiritually throughout my life.

Next, I would like to express my sincere gratitude to my supervisor Dr. Abd Shukur bin Ja'afar for the guidance of my PSM, for his patience, motivation, enthusiasm, and immense knowledge. His encouragement helped me in all the time of research and writing of this thesis. I could not imagine having a better supervisor and mentor for my PSM study.

My sincere thanks also go to my love, Nurul Hidayah binti A.Halim, my friends, Ammar, Afiq, Amirul Hafiz, Khairul Naim, Firzan, Shahmir, Farhan, and Arif for the sleepless nights we were working together before deadlines and for all the fun we have had in the last four years.

Besides my colleagues, I would like to thank the rest of my thesis committee, Professor Madya Dr. Wong Yan Chie,w and Ts. Siti Aisah binti Mat Junos@Yunus, for their encouragement, insightful comments, and hard questions.

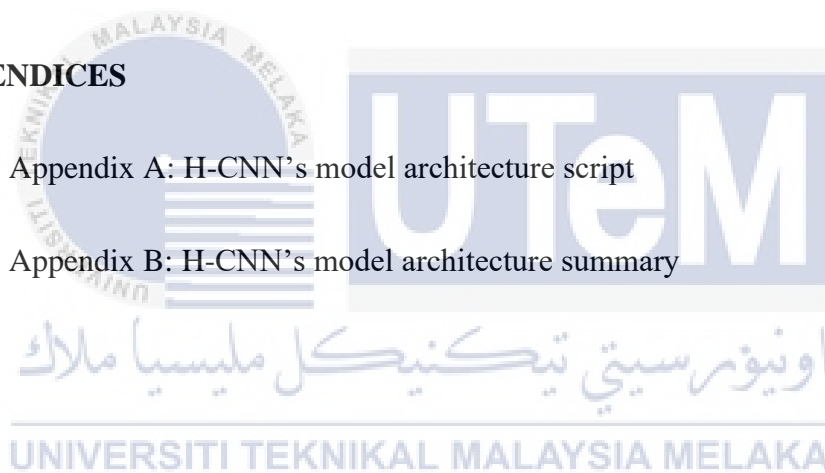
TABLE OF CONTENTS

Declaration	
Approval	
Dedication	
Abstract	i
Abstrak	ii
Acknowledgements	iii
Table of Contents	iv
List of Figures	viii
List of Tables	x
List of Symbols and Abbreviations	xii
List of Appendices	xiii
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement	1
1.2 Objective and Scope	2
1.2.1 Objective	2
1.2.2 Scope of Work	2

1.3	Project Significant	3
CHAPTER 2 LITERATURE SURVEY		4
2.1	Artificial Intelligence (AI)	4
2.1.1	Deep Learning	5
2.2	Convolutional Neural Network (CNN)	5
2.2.1	Input	6
2.2.2	Convolution Layer – The Kernel	7
2.2.3	Pooling Layer	9
2.2.4	Activation Functions (Non-Linearity)	10
2.2.5	Fully Connected Layer	11
2.2.6	Confusion matrix	12
2.2.6.1	Accuracy	12
2.2.6.2	Precision	13
2.2.6.3	Recall	13
2.2.6.4	F1 Score	13
2.2.6.5	Support	14
2.3	Literature Review	14
CHAPTER 3 METHODOLOGY		23
3.1	Design of solution	23
3.1.1	Block Diagram	24

3.1.1.1	Fetch Image from Database	25
3.1.1.2	Image pre-processing	25
3.1.1.3	Design CNN model architecture	25
3.1.1.4	Train the CNN model architecture	25
3.1.1.5	Test and optimise the CNN model architecture	26
3.1.1.6	Compare with transfer learning architectures	26
3.1.1.7	Model inferencing	26
3.1.2	Project flowchart	26
3.1.2.1	Pre-processing	26
3.1.2.2	Design CNN model architecture	27
3.2	Modern tools	29
3.2.1	Software	29
3.2.2	Hardware	30
CHAPTER 4 RESULTS AND DISCUSSION		31
4.1	Chronology of simulation	31
4.1.1	Manually filter the dataset	32
4.1.2	Add new dataset to balance the dataset	40
4.1.3	Tune the hyperparameters	47
4.1.3.1	Number of filters for each convolutional layer	47
4.1.3.2	Number of epochs	48

4.2	Inference of H-CNN model	51
4.2.1	Software	51
4.2.2	Hardware	56
4.3	Comparison with other pre-trained models	59
4.3.1	Confusion matrix	60
4.4	Environment and sustainability	63
	CHAPTER 5 CONCLUSION AND FUTURE WORKS	65
	REFERENCES	67
	APPENDICES	71
5.1	Appendix A: H-CNN's model architecture script	71
5.2	Appendix B: H-CNN's model architecture summary	72



LIST OF FIGURES

Figure 2.1: Classic CNN architecture [8]	6
Figure 2.2: 4x4x3 RGB image [10]	7
Figure 2.3: 5x5 filter (kernel) convolving the input image [8]	8
Figure 2.4: Movement of the kernel [10]	8
Figure 2.5: Pooling layer (2x2) convolving the input [7]	9
Figure 2.6: Graph representation of ReLU [11]	10
Figure 2.7: Architecture of fully connected layers [7]	11
Figure 2.8: Confusion matrix for binary classification [12]	12
Figure 3.1: General process block diagram	24
Figure 3.2: Project flowchart	28
Figure 4.1: Enhancement of classification accuracy process	32
Figure 4.2: Learning curve before and after dataset filtering	39
Figure 4.3: Confusion matrix before adding a new dataset	40
Figure 4.4: Confusion matrix after adding a new dataset	46
Figure 4.5: Learning curves before and after changing number of epochs	48
Figure 4.6: H-CNN initial confusion matrix	50
Figure 4.7: H-CNN final confusion matrix	51
Figure 4.8: Software inferencing code	52

Figure 4.9: Software inferencing test images	52
Figure 4.10: NVIDIA Jetson Nano full setup	57
Figure 4.11: Hardware inferencing block diagram	57
Figure 4.12: H-CNN confusion matrix	61
Figure 4.13: ResNet50 confusion matrix	61
Figure 4.14: VGG16 confusion matrix	62
Figure 4.15: MobileNetV2 confusion matrix	62
Figure 4.16: DenseNet121 confusion matrix	63
Figure 4.17: Sustainable development goals [22]	63
Figure 4.18: 9th goal infographic [24]	64



LIST OF TABLES

Table 2.1: Accuracy comparison on different types of datasets to VGG model [14]	14
Table 2.2: CNN architecture [15]	15
Table 2.3: Effect of data augmentation [15]	16
Table 2.4: Comparison to different model approaches [15]	16
Table 2.5: Deep learning vs Machine learning model [17]	17
Table 2.6: Comparison CNN model and transfer learning [19]	18
Table 2.7: Effect of picture size to CNN classification accuracy [1]	18
Table 2.8: A list of methods to mitigate overfitting [20]	19
Table 2.9: Brief overview of CNN architectures [21]	19
Table 2.10: Summary of previous studies	20
Table 3.1: Software modern tools	29
Table 3.2: Hardware modern tools	30
Table 4.1: Number of images per class before and after manually filter the dataset	33
Table 4.2: Example of unsuitable images dataset for each class	34
Table 4.3: Comparison before and after dataset filtering	39
Table 4.4: Example of added images	41
Table 4.5: After adding a new dataset	46
Table 4.6: Result from changing number of filters	47

Table 4.7: Result from changing number of epochs	49
Table 4.8: Software inferencing output	53
Table 4.9: Hardware inferencing output	58
Table 4.10: Model comparison details	60



LIST OF SYMBOLS AND ABBREVIATIONS

CNN : Convolutional Neural Network

AI : Artificial Intelligence

SVM : Support Vector Machine

ANN : Artificial Neural Network

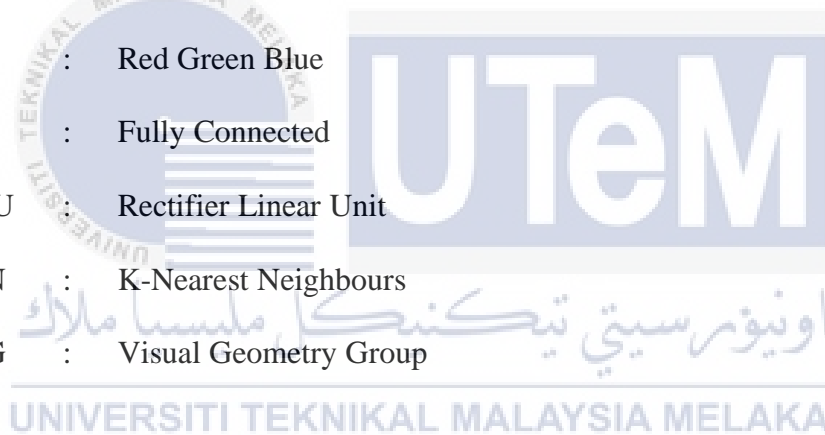
RGB : Red Green Blue

FC : Fully Connected

ReLU : Rectifier Linear Unit

KNN : K-Nearest Neighbours

VGG : Visual Geometry Group



LIST OF APPENDICES

Appendix A: H-CNN's model architecture script.....	71
Appendix B: H-CNN's model architecture summary.....	72



CHAPTER 1

INTRODUCTION



Classification is about categorizing objects into groups. A type of classification is where multiple classes are predicted. In CNN architecture, it comprises convolutional layers, max-pooling layers, and fully connected layers. In the first layer, the input is processed, and an output is produced in the last layer. The classification network selects the category based on which output response has the highest output value.

1.1 Problem Statement

First, machine learning has been the way for image classification for quite some time due to its simplicity and fewer computer resources. However, the performance in terms of accuracy compared to deep learning is quite a gap. According to P. Wang, E.

Fan, and P. Wang, [1] the image classification results from the MNIST dataset for the machine learning model, Support Vector Machine (SVM) is 88% while CNN is 98%. Machine learning did not perform too well for a large dataset, unlike CNN. Secondly, the size of the model will be the main factor to avoid using a transfer learning method. A pre-trained model size is much bigger than a designed CNN due to its nature being pre-trained by thousands if not millions of images typically on a large-scale image-classification task. Thus, it has unnecessary weight for a certain application where it would not be fully utilized [2]. This research will dive deeper into how CNN works and what affects its performance. This will give more insights into the structure of the CNN model. To overcome these problems, designing CNN architecture could produce a better AI model with a smaller size.

1.2 Objective and Scope

1.2.1 Objective

The objectives of this project are:

1. To design a Convolutional Neural Network architecture for tropical fruit classification.
2. To optimize the CNN model and compare the result in terms of confusion matrix and accuracy, between pre-trained transfer learning models and designing a model.
3. To inference the designed model on software and hardware.

1.2.2 Scope of Work

To complete this project smoothly, several scopes of work has been determined to ensure the main idea where and how to progress accordingly. Firstly, a dataset for

tropical fruit needs to be collected first. These datasets can be found on Kaggle and any image search engine such as Google Image. Next, the hardest part is to design the CNN model architecture. This process will be done using a TensorFlow and Keras library of Python language since it is the most reliable coding language to develop a CNN. The platform will be on Google Colab due to demand in graphical processing power to train the model effectively. Right after the model is done, it needs to be optimized to get the best result. After that, a transfer learning method will be used to train the model with the same dataset. Then, the comparison of the simulation results will be analyzed. Finally, the designed CNN model will be inferenced on software and hardware.

1.3 Project Significant

This project can be implemented on real life application for more efficient output. Currently in supermarket, the fruits need to be weighted first before the price sticker will be printed. If the worker is not there, then the whole system is jammed. Imagine a self-service automatic process that can eliminate this type of problems. This is just one of the examples to justify how significant this project can be. Basically, any application that need a visual imagery detection might need a CNN technology. Below is some of the famous applications of CNN [3]:

- Facial Recognition
- Analyzing Documents
- Historic and Environmental Collections
- Understanding Climate
- Grey Areas
- Advertising

CHAPTER 2

LITERATURE SURVEY



This chapter will discuss about the CNN theory and previous research that has been done.

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2.1 Artificial Intelligence (AI)

AI has progressed from humble beginnings to a global impact. The definition of AI and what should and should not be included has evolved over time. According to Kaplan and Haenlein [4], AI defined as “a system’s ability to correctly interpret external data, to learn from such data, and to use those learnings to achieve specific goals and tasks through flexible adaptation”. According to European Commission’s 2018 [5] definition of AI, “AI refers to systems that display intelligent behaviors by analyzing their environment and taking action – with some degree of autonomy – to achieve specific goals”.

2.1.1 Deep Learning

Deep learning is a type of AI that mimic the human way of thinking to absorb a certain types of knowledge. To make it simpler, deep learning can be view as a way to automate predictive analytics. While machine learning algorithms are linear, deep learning algorithms are arranged in a hierarchy of increasing complexity and abstraction. According to Y. LeCun, Y. Bengio, and G. Hinton, [6] “Deep learning enables computer models with several processing layers to discover data representations with multiple levels of abstraction. These techniques have significantly advanced the state-of-the-art in speech recognition, visual object recognition, object detection, and numerous other fields, including drug development and genomics. Deep learning identifies intricate structure in big data sets by utilizing the backpropagation algorithm to recommend how a machine's internal parameters used to compute the representation in each layer from the representation in the previous layer should be modified. Deep convolutional neural networks have advanced the processing of pictures, video, speech, and audio, while recurrent neural networks have shed light on sequential data such as text and speech.”

2.2 Convolutional Neural Network (CNN)

CNN is a deep learning architecture, an outstanding type of multiple layer neural network. The architecture of CNN is inspired to mimic a way of human brain. Although it became famous after the shocking achievement of AlexNet in 2012, the history itself originated from 1959. A CNN is a class of artificial neural network (ANN), where it is most applied to analyze visual imagery because of the ability to learn a significant patterns from the pixels of the image and can identify the type of image more intelligently. A deep CNN model has several number of layers where each

layer extracts the significant patterns and feed it into the next layer that can be seen in Figure 2.1. The early layers learn and take out the high-level features, and the deeper layers learns and take out the low-level features [7].

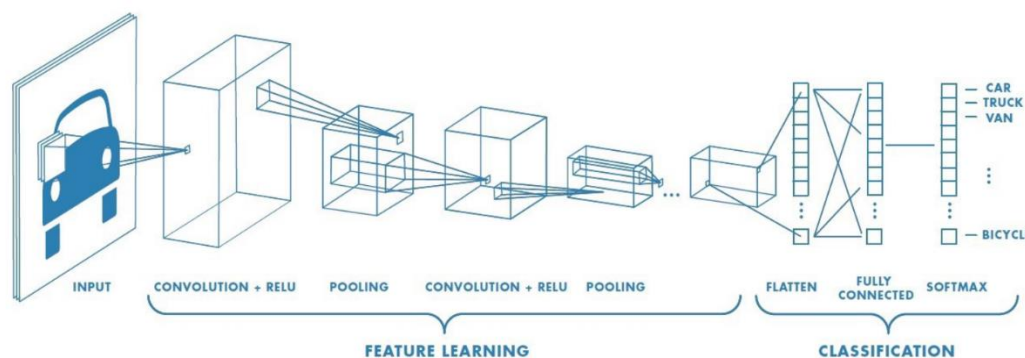


Figure 2.1: Classic CNN architecture [8]

2.2.1 Input

Before anything else, CNN needs an input to go further process. By input image, it means a lot of images (thousands) for it to learn efficiently. The more the input images, the better for the model to learn. Let's take a simple example for a better understanding. In Figure 2.2, a Red Green Blue (RGB) image is the input. It has been separated by its three-color planes, Red, Green, and Blue. There are many other types of color spaces in images such as Grayscale, RGB, HSV, CMYK, etc. Now in this image, the resolution is 4x4, which is really small for a picture. Nowadays, images resolution has been much more complex for example SD (640x480), HD (1280x720), FULL HD (1920x1080), 4K (3840x2160), 8K (7680x4320) and etc [9]. Imagine how computationally intensive things would get for these modern image resolutions. The role of the CNN is to reduce the images into a form which is easier to process, without losing an important feature to get a good prediction [10].

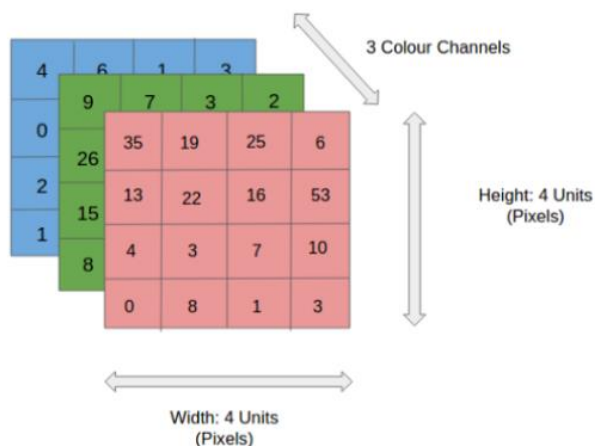


Figure 2.2: 4x4x3 RGB image [10]

2.2.2 Convolution Layer – The Kernel

Convolutional layer is the principal part of any CNN architecture. It consists of a set of convolutional filters (also known as kernels) which convolved the input image (N-dimensional metrics) to generate an output feature map as shown in Figure 2.3. A filter (kernel) can be described as a grid of discrete values or numbers, where each value is known as the weight of this kernel. During the early stage of training process of an CNN model, all the weights of a kernel are assigned with random numbers (best to refer a similar model). Then, with each training epoch, the weights are tuned, it repeats the same process that helps the kernel learning process to extract significant features more deeply [7]. The main objective of the Convolution Operation is to extract the high-level features such as edges, from the input images. It need not to be limited to only one Convolutional Layer, there can be a multiple depends on the model usage. Usually, the first layer is to be capturing a low-level feature such as edges, color, gradient orientation, etc. With added layers, the architectures adapt to the high-level features as well, giving us a network, which has the wholesome understanding of images in the dataset, like how human would [10].

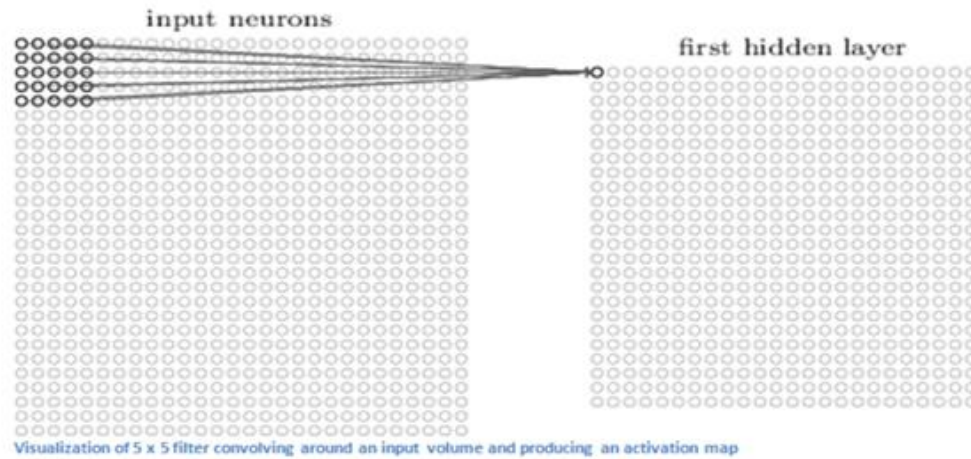


Figure 2.3: 5x5 filter (kernel) convolving the input image [8]

The stride also one of the important keys to design a kernel. Stride can be described as the taken step size along the horizontal or vertical position of 1 to the kernel. Figure 2.4 shows how the kernel move from left to right and then starts again in row below. If the stride length = 1, it is non-strided, because the kernel will go through each pixel of input image [10].

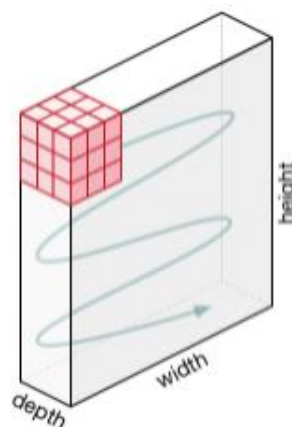


Figure 2.4: Movement of the kernel [10]

2.2.3 Pooling Layer

Pooling layers are used to sub-sample the feature maps generated by the convolution process. For example, it takes bigger feature maps and compresses them to smaller feature maps like in Figure 2.5. When decreasing the feature maps, the most dominating features in each pool step are always preserved. Like the convolution operation, the pooling operation is conducted by defining the pooled region size and the stride of the operation [7]. Through dimensionality reduction, the computer power required to process the data is reduced [10]. Pooling techniques such as max pooling, min pooling, average pooling, gated pooling, tree pooling, and others are used in pooling layers. Max pooling is the most well-known and often utilized pooling technique.

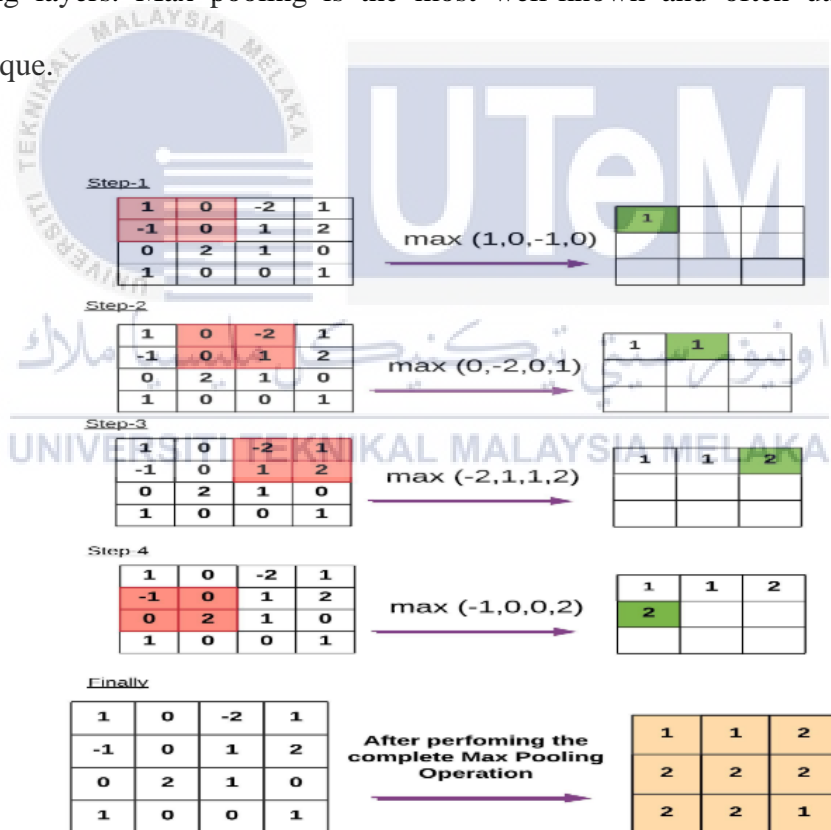


Figure 2.5: Pooling layer (2x2) convolving the input [7]

2.2.4 Activation Functions (Non-Linearity)

In any neural network-based model, the activation function's role is to translate the input to the output, where the input value is derived by computing the weighted sum of neuron input and then adding bias to it (if there is a bias). In other words, the activation function produces the matching output to determine whether a neuron will fire or not for a particular input as depicts in Figure 2.6. In CNN architecture, after each learnable layers (layers with weights, i.e., convolutional and Fully Connected (FC) non-linear activation layers are used. The CNN model can learn more complicated things and manage to translate inputs to outputs non-linearly thanks to the non-linearity behavior of those layers. In this project, Rectifier Linear Unit (ReLU) will be used as activation function in CNN [7].

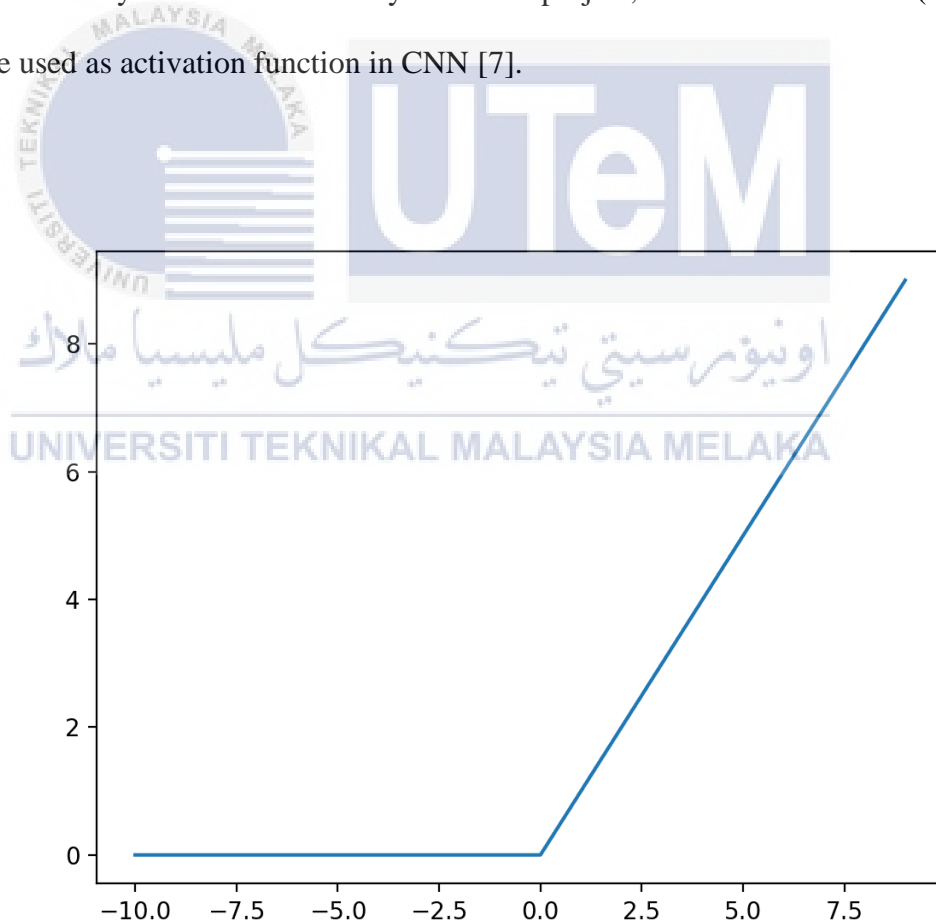


Figure 2.6: Graph representation of ReLU [11]

ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. The main advantage of ReLU is that it requires very minimal computation load compared to others. Below (2.1) is the mathematical equation of ReLU [11].

$$f(x)\text{ReLU} = \max(0, x) \quad (2.1)$$

2.2.5 Fully Connected Layer

The last part of CNN architecture layers is FC. The way this layer works is that it looks at the output of the previous layer (should be the activation of high-level features) and the number of classes N [10]. The FC layers receive input from the last convolutional or pooling layer, which is in the form of a set of metrics (feature maps), flatten them to make a vector, and then feed it into the FC layer to construct the final CNN output as in Figure 2.7. In other words, FC is used as the classifier of the CNN architecture [7].

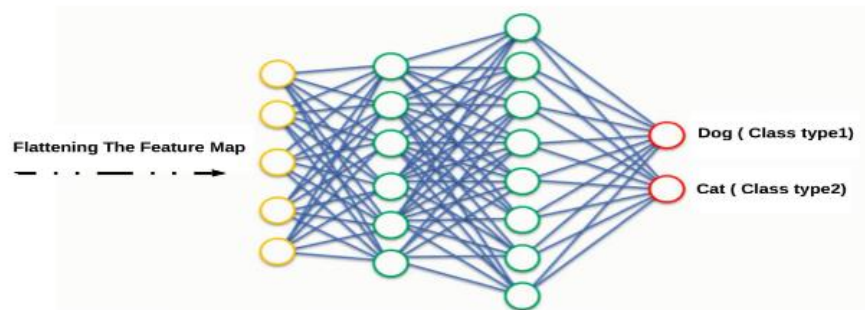


Figure 2.7: Architecture of fully connected layers [7]

2.2.6 Confusion matrix

A confusion matrix is a table that is used to evaluate the accuracy performance of a classification model on a set of test data which the true values are known. Confusion matrix shows where the model is getting confused by plotting a table that reports with each row corresponding to the true class and each column to the predicted class like in Figure 2.8. The confusion matrix simple to understand, but the terminologies used can be confusing. The output “TN” stands for True Negative which shows the number of negative samples classified correctly. Similarly, “TP” stands for True Positive which indicates the number of positive examples classified accurately. The term “FP” shows False Positive value, for example the number of actual negative examples classified as positive, and “FN” means a False Negative value which is the number of actual positive examples classified as negative.

		Predicted	
		Negative	Positive
Actual	Negative	TN	FP
	Positive	FN	TP

Figure 2.8: Confusion matrix for binary classification [12]

2.2.6.1 Accuracy

Accuracy can be calculated from formula (2.2) [12], and it's the most intuitive performance measure. It is simply a ratio of correctly predicted observation to the total observations. One may think that, if accuracy is high, then it is a good model. Accuracy is a great measure but only when the datasets is symmetric where values of false positive and false negatives are almost same. Therefore, other parameters are needed to evaluate the performance of the model [13].

$$Accuracy = \frac{TN + TP}{TN + FP + FN + TP} \quad (2.2)$$

2.2.6.2 Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations [13]. High precision relates to the low false positive rate as shown in equation (2.3)[12].

$$Precision = \frac{TP}{TP + FP} \quad (2.3)$$

2.2.6.3 Recall

Recall is the ratio of correctly predicted positive observations to all observations in actual class – yes [13]. It is calculated from equation (2.4) [12].

$$Recall = \frac{TP}{TP + FN} \quad (2.4)$$

2.2.6.4 F1 Score

F1 Score is the weighted average of precision and recall. Therefore, this score takes both false positives and false negatives into account as in equation (2.5) [12]. Intuitively, it is not easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if the class is uneven distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both precision and recall [13].

$$F1 = 2 * \frac{precision * recall}{precision + recall} \quad (2.5)$$

2.2.6.5 Support

Support is the number of samples of the true response that lie in that class [13].

2.3 Literature Review

Move on to the next discussion, a high accuracy in simulation does not guarantee a same accuracy percentage in real world application. This may happen due to the real situation where the model is being tested for example the lighting, background, etc. According to Josh Janzen's experiment, he stated that "most real-world applications of CNN for image recognition are not going to be that similar to ImageNet base of images" [2]. There is study that related to this, based on G. Zeng [14] did an experiment by using 4 types of dataset to train the model as in Table 2.1. The first one is pure online photos, second one is pure shooting photos, third one is train online photos first then shooting photos, and the last one is shooting first and then online photos. The result shows how this type of dataset may influence the accuracy.

Table 2.1: Accuracy comparison on different types of datasets to VGG model
[14]

VGG model with different input images	Top-1 accuracy (%)
VGG - Online	89.6
VGG - Shooting	94.0
VGG - Online-Shooting	95.6
VGG - Shooting-Online	90.5

Now, the study will focus on more to designing CNN model performance. From previous research, it is known that there are not many studies on this topic. This is due to complication of the process that can consume a lot of time. From Y. Zhang, Z. Dong, and X. Chen et al.[15] studies, they designed a 13-layer deep CNN model

Table 2.3: Effect of data augmentation [15]

Test Set	No. of Image	Data Augmentation F1	No augmentation F1
Clean	1800	94.94%	93.72%
Background	173	89.60%	84.39%
Decay	136	94.12%	86.03%
Unfocused	145	91.03%	86.90%
Occlusion	161	92.55%	86.34%

Table 2.4: Comparison to different model approaches [15]

Approach	Overall Accuracy
PCA+ kSVM	88.20%
PCA + FSCABC	89.11%
WE + BBO	89.47%
FRFE + BPNN	88.99%
FRFE + IHGA	89.59%
13-layer CNN	94.94%

This project only focuses on deep learning because it is more superior than machine learning models. As N. Saranya, K. Srinivasan, S. Kumar et al. [17] did a comparison of fruit classification between machine learning models, K-Nearest Neighbours (KNN) and Support Vector Machine (SVM) versus deep learning models CNN in Table 2.5 where the KNN, SVM and CNN produced 48.63%, 60.65% and 96.49% respectively.

Table 2.5: Deep learning vs Machine learning model [17]

TITLE	DATASET	METHOD	RESULT
	(FRUIT 360)		
Fruit Classification Using Traditional Machine Learning	Size 100x100	KNN	48.63%
and Deep Learning Approach	Red Apple: 492 Banana: 490 Orange: 479 Pomegranate: 246 Hold out (70/30)	SVM 13-layer CNN model	60.65% 96.49% (20 epochs)

Now, in deep learning itself, there are two methods that can be used for training the model as stated previously, by designing CNN model architecture and transfer learning method. On the internet, the CNN recommendation between these two methods will always lean to transfer learning method. Instead of building a CNN model architecture, transfer learning can easily import a pre-built and pre-trained model such as VGG, GoogleNet, ResNet, DenseNet, etc. Simpler to say, transfer learning is like by taking a model trained on a large dataset and transfer its knowledge to a smaller dataset. That means transfer learning requires a smaller amount of dataset [18]. However, the complication of the architecture of transfer learning is very complex. Not only it took a lot of computational power, but it also took a lot of size. According to R. Pathak and H. Makwana [19] studies on designing CNN and transfer learning models shows their 6-layer CNN model as listed in Table 2.6, outperformed the other transfer learning VGG-16, AlexNet, LeNet-5, and VGG-19 with 98.23%, 90.81%, 83.56%, 82.93%, and 76.48% respectively. Bear in mind that designing CNN model indeed can surpass transfer learning, but it is error prone and time consuming.

Table 2.6: Comparison CNN model and transfer learning [19]

COMPARISON OF THE ACCURACIES	
MODEL	ACCURACY
VGG-16	90.81%
VGG-19	76.48%
LeNet-5	82.93%
AlexNet	83.56%
Proposed Model (designed CNN)	98.23%

From Table 2.7, CNN model's accuracy affected by the input size of dataset. Essentially, the bigger picture size will bring a higher classification accuracy. This is simple to understand as a bigger picture size will have more clearer details that can be seen from our eyes. So does CNN, it will be a lot easier to learn from a clearer picture than a pixelated picture.

Table 2.7: Effect of picture size to CNN classification accuracy [1]

Picture size	Number of categories	CNN classification accuracy
64x64	10	0.71
128x128	10	0.74
256x256	10	0.95

Overfitting is a common problem while training a CNN model. This happens due to a lot of reason but mostly from lack of training data and the CNN model is too complex for the problem solving. In Table 2.8, R. Yamashita, M. Nishio, and K. G. D. Richard et al. [20] listed a methods to mitigate overfitting for CNN model. Starting from top to bottom is the priority that need to be taken care of.

Table 2.8: A list of methods to mitigate overfitting [20]

How to mitigate overfitting
More training data
Data augmentation
Regularization (weight decay, dropout)
Batch normalization
Reduce architecture complexity

Author L. Alzubaidi, J. Zhang, and A. Humaidi et al. [21] listed a brief overview some of the CNN architectures in Table 2.9. VGG, ResNet, DenseNet and MobileNet are a well-established pre-trained model. It has a multiple version due to the community helps it to grow day by days. Below is the strength of each model with ResNet seems like a best model here with 3.57 error rate for ImageNet dataset. Although, it might show another result for another type of dataset.

Table 2.9: Brief overview of CNN architectures [21]

Model	Main finding	Depth	Dataset	Error rate	Input size	Year
VGG	Increased depth and small filter size	16,19	ImageNet	7.3	224x224x3	2014
ResNet	Robust against overfitting due to symmetry	152	ImageNet	3.57	224x224x3	2016

	mapping-based					
	skip links					
DenseNet	Block of layers;	201	CIFAR-	3.46,	224x224x3	2017
	layers connected		10,	17.18,		
	to each other		CIFAR-	5.54		
			100,			
			ImageNet			
MobileNet-	Inverted residual	53	ImageNet	-	224x224x3	2018
V2	structure					

The methods that are shown in Table 2.10 are being implemented for this project on account of results from previous studies that shows its effectiveness in increasing the total accuracy performance of the CNN model architecture.

Table 2.10: Summary of previous studies

Method	Differences		
Combine online and shooting photos dataset [14]	Different input images	Accuracy	
	VGG – Online	89.6%	
	VGG – Shooting	94.0%	
	VGG – Online-Shooting	95.6%	
	VGG – Shooting-Online	90.5%	
Data augmentation [15]	Before (accuracy)	After (accuracy)	
	Clean	93.72%	94.94%
	Background	84.39%	89.60%

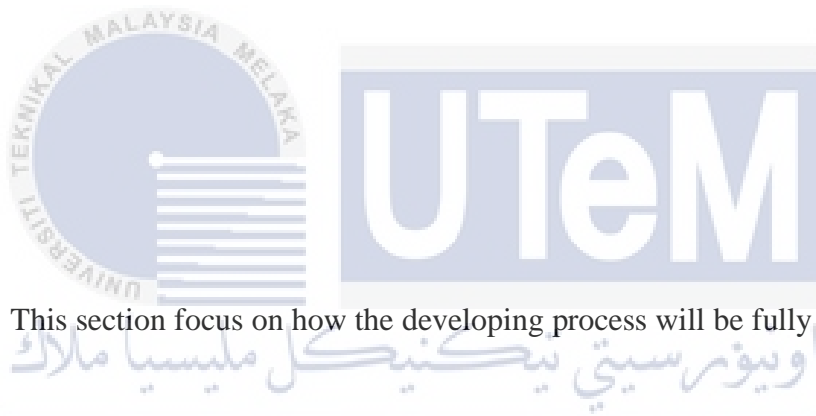
Decay	86.03%	94.12%
Unfocused	86.90%	91.03%
Occlusion	86.34%	92.55%
Deep learning [17]	Method	Accuracy
	KNN	48.64%
	SVM	60.65%
	13-layer CNN model	96.49%
Design CNN model architecture [19]	Model	Accuracy
	VGG-16	90.81%
	VGG-19	76.48%
	LeNet-5	82.93%
	AlexNet	83.56%
	Designed CNN	98.23%
Input size more than 200x200 [1]	Picture size	Accuracy
	64x64	0.71
	128x128	0.74
	256x256	0.95
Pre-trained models[21]	Model	Main finding
	VGG	Increased depth and small filter size
	ResNet	Robust against overfitting due to symmetry mapping-based skip links

DenseNet	Block of layers; layers connected to each other
MobileNet-V2	Inverted residual structure



CHAPTER 3

METHODOLOGY



This section focus on how the developing process will be fully conducted.

3.1 Design of solution

After study the previous research, it is easier to design a roadmap by taking a consideration on what is the most important and least important part to focus on. This may greatly help to reduce time taken to complete the project. As CNN model requires a lot of try and error to achieve a great result. By considering every important component in CNN model will highlight on how the component may help to reduce the training time and increase the accuracy.

3.1.1 Block Diagram

Figure 3.1 represent the important steps taken to complete this project starting with preparing the dataset, design the CNN model, test, and optimize, compare the results, and model inferencing.

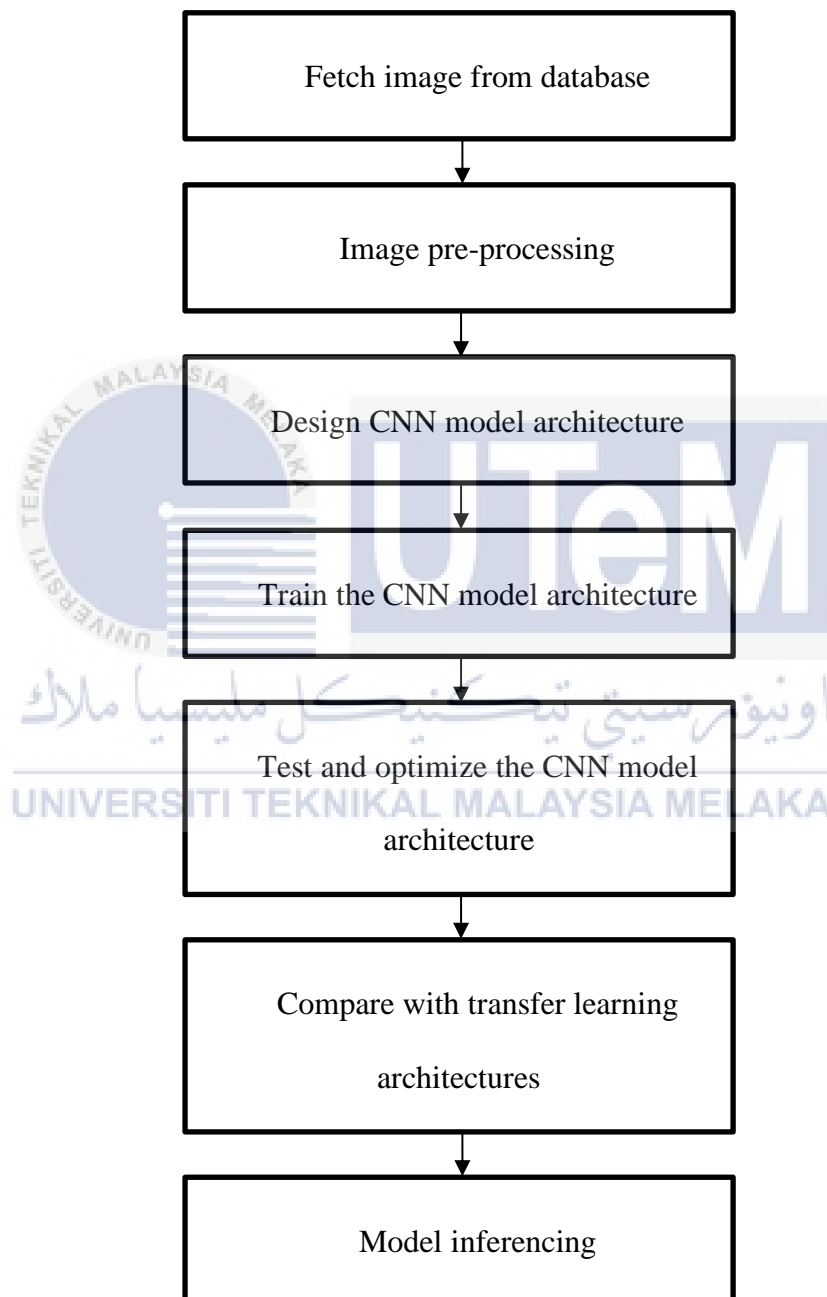


Figure 3.1: General process block diagram

3.1.1.1 Fetch Image from Database

This first block is fetching the dataset images from database, Kaggle. Fruit-262 will be chosen with several images added from self-taken picture, Google Image, and data augmentation. The dataset will be stored in Google Drive where it is a cloud storage that enable to synchronize with Google Colab. The whole dataset will be stored here in one folder, the training and test image will be separated using Python script on Google Colab.

3.1.1.2 Image pre-processing

Image pre-processing prepare the dataset images into a format images before they are used by model training. When the image is pre-processed, it also means that the size of the image is being reduced to the input size.

3.1.1.3 Design CNN model architecture

The third stage is designing CNN model. Here is where the hyperparameters will be decided. The CNN model has several components such as convolution layer, activation function, pooling, fully connected and 'softmax'. The filter size of filter, number of filters, stride, padding weights, bias and activation needs to be assigned for each layer that encounter a convolution process.

3.1.1.4 Train the CNN model architecture

Once the model has been designed, it is ready to train. The training process will be done on Google Colab for faster training time. Google Colab allows user to train their CNN model using a virtual graphic card from Google server. This is helpful for user to use when they face a hardware limitation on their personal computer. The training process undoubtedly uses a lot of computational power.

3.1.1.5 Test and optimize the CNN model architecture

Test process is done right after the training process. Its function is to test the trained model accuracy. The dataset is separated with 80:20 ratio. The 80 is for training and the 20 is for test. The test images can't be the same image as training. The test result of accuracy value is the total accuracy of the model. To get the best result, the CNN model architecture is being optimized by tuning the hyperparameters.

3.1.1.6 Compare with transfer learning architectures

Now, to compare between these models, it is time for transfer learning architectures to be train using a same dataset. This process is a lot easier compares to designed CNN model because it uses a transfer learning method by importing the pre-trained model to train our dataset. No designing layers needed. The training and test process is same just like the previous model. The comparison will be done in terms of accuracy and confusion matrix.

3.1.1.7 Model inferencing

Finally, the designed CNN model architecture inferencing will be done on software and hardware to show its functionality.

3.1.2 Project flowchart

Figure 3.2 depicts the flowchart of the whole project process in details. It consists of two main part, pre-processing, and design CNN model architecture.

3.1.2.1 Pre-processing

In Figure 3.2 left hand side, pre-processing process will be explained in more details. The next is to pre-process the image before being trained. The main objective to do an image processing is to reduce the CNN model's burden during training

process. The explanation behind this is when the image has been pre-processed to a certain criterion such as resolution, centralization of the image, etc., for all images will act as standard specification for the input CNN model. When training, the model is fed only with one standard of image thus reducing the computational power and reducing the time taken to complete the training. In this case, the image will be resized to 208x256.

3.1.2.2 Design CNN model architecture

From Figure 3.2 right hand side, this is where the CNN model architecture is being designed. Not to mention that the early result would not be as expected, this would require a try and error by tuning the hyperparameters. Hyperparameters are the variables which determines the network structure and the variables which determine how the network is trained. Hyperparameters are set before training, every time the hyperparameters are being changed, it needs to retrain again to get the new result. In this project, the hyperparameters that will be focus on is layers, filters, activator, and optimizer.

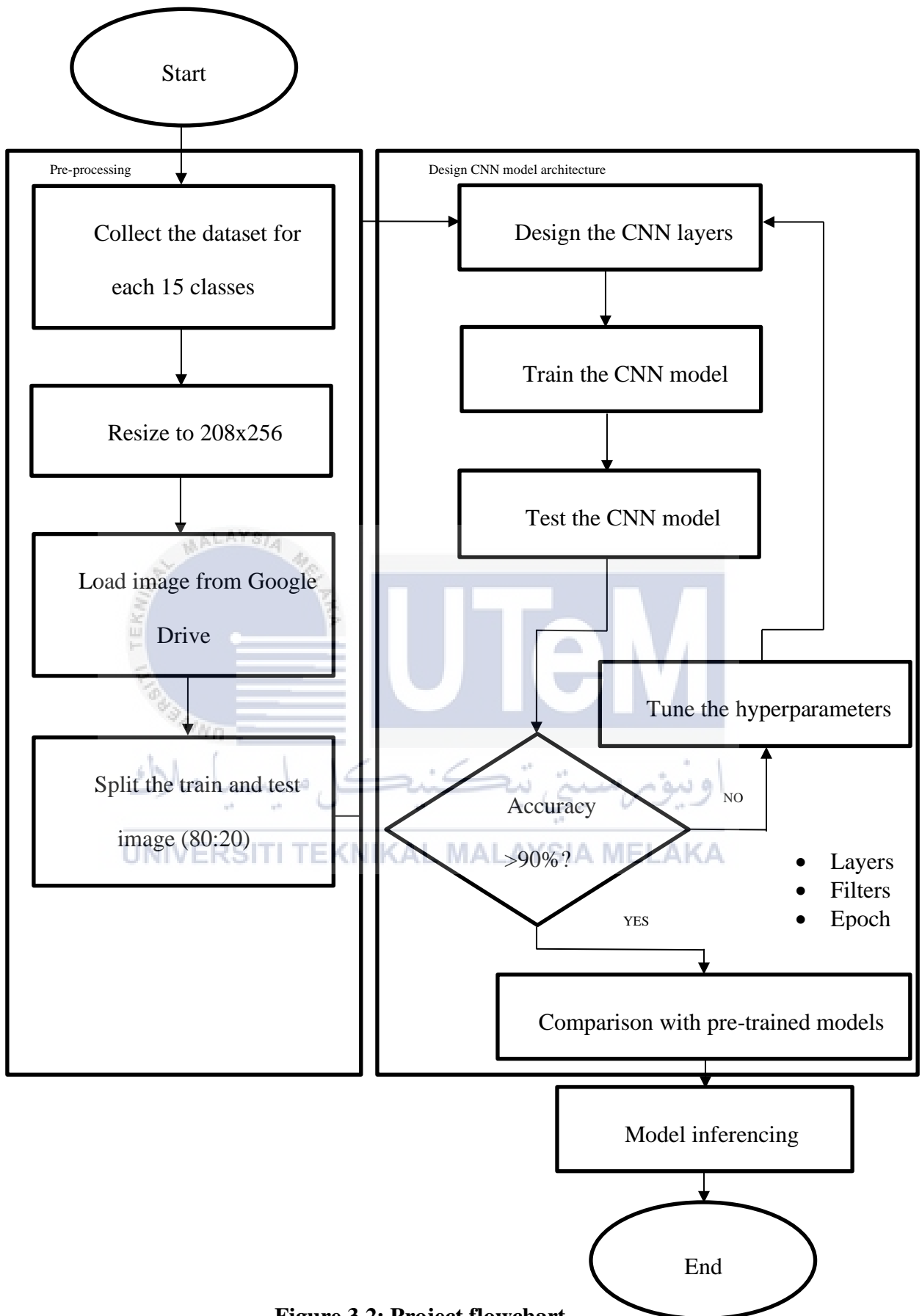


Figure 3.2: Project flowchart

3.2 Modern tools

To perform all the tasks discussed in previous section, it requires a several tools. Considering this project related to AI and deep learning, it demands a high-power computing resources. This project modern tools consists of two main section, software, and hardware.

3.2.1 Software

While completing this project, a laptop running on Windows 11 installed with software in Table 3.1 runs smoothly. On Kaggle, Fruit-262 datasets was downloaded as the main resource for dataset. There are still a lot of datasets available on the website but for this project, only one dataset was being chosen. Once the dataset is ready to be trained, it will be stored on Google Drive as a cloud storage. Next, the most used software was Google Colab Pro. Here is where the design, train, test, optimization, and inferencing of the CNN model architecture was done. The reason for Google Colab Pro was used instead of local hardware is because the graphical processing unit offered is faster. Lastly, TensorFlow and Keras library was used to write the CNN model architecture's script.

Table 3.1: Software modern tools

Platform name	Application
Windows 11	Operating system
Kaggle	Datasets community
Google Drive	Cloud storage for datasets
Google Colab Pro	Design, train, test, optimise, and inference the CNN model architecture
TensorFlow and Keras	Machine learning Python's library

3.2.2 Hardware

All the software mentioned previously ran on the laptop specifications as in Table

3.2. Table 3.2: Hardware modern tools

Table 3.2: Hardware modern tools

Acer Nitro 5 laptop specification	
Processor	Intel ® Core™ i5-8300 CPU @ 2.30 GHz (8 CPUs)
GPU	NVIDIA GeForce GTX 1050 4GB VRAM
RAM	12 GB
Storage	512 GB SSD



CHAPTER 4

RESULTS AND DISCUSSION



This chapter will focus on analysis of the results obtained from simulation and hardware implementation.

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

4.1 Chronology of simulation

A multiple step needs to be done to achieve the best result from the designed CNN model. There are just too many variables in CNN model that can cause a variation of result. Thus, a try and error method were conducted to observe the differentiations of result from each step in Figure 4.1.

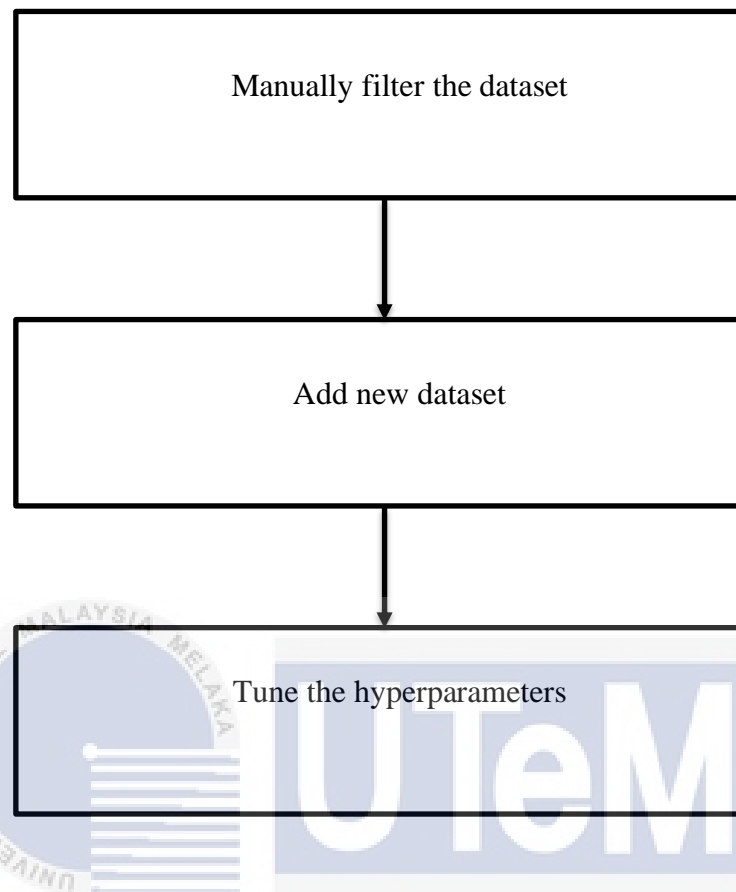


Figure 4.1: Enhancement of classification accuracy process




4.1.1 Manually filter the dataset

In this step, manual selection of image dataset must be done because there are too many unsuitable images inside the Fruit-262 dataset folders that caused the CNN model face a hardship to learn the significant features of each fruit classes. However, this method will cause the number of datasets for each class to be more imbalance. Table 4.1 below shows list number of images for each class before and after this process, and Table 4.2 shows some of the example of unsuitable fruit images.

Table 4.1: Number of images per class before and after manually filter the dataset

Dataset class	Number of images	
	Before	After
Apple	1171	1085
Banana	1162	1103
Cempedak	1051	893
Coconut	1024	487
Dragonfruit	1035	666
Durian	1026	827
Honeydew	1008	911
Mango	1025	904
Mangosteen	1105	1002
Orange	1087	1024
Papaya	1027	651
Pineapple	1037	889
Rambutan	1054	1001
Watermelon	996	916
Wax Apple	1017	701

Table 4.2: Example of unsuitable images dataset for each class

Class of fruit	Example images
Apple	
Banana	
Cempedak	

Coconut



Dragonfruit



اونيورسيتي تيكنيكل مليسيا ملاك

Durian

UNIVERSITI TEKNIKAL MALAYSIA MELAKA



Honeydew



Mango



Mangosteen

UNIVERSITI TEKNIKAL MALAYSIA MELAKA



Orange



Papaya



Pineapple



Rambutan



Watermelon



Wax apple



Following Figure 4.2, both before and after result shows a healthy learning curve. The after result of the trained H-CNN model indeed generates a huge improvement in accuracy from 68.69% to 84.15% refer to Table 4.3. This is due to an uncomplicated images dataset that will allow the H-CNN model to learn more efficiently. The time taken to train the model also slightly decreased due to a lower total amount of dataset.

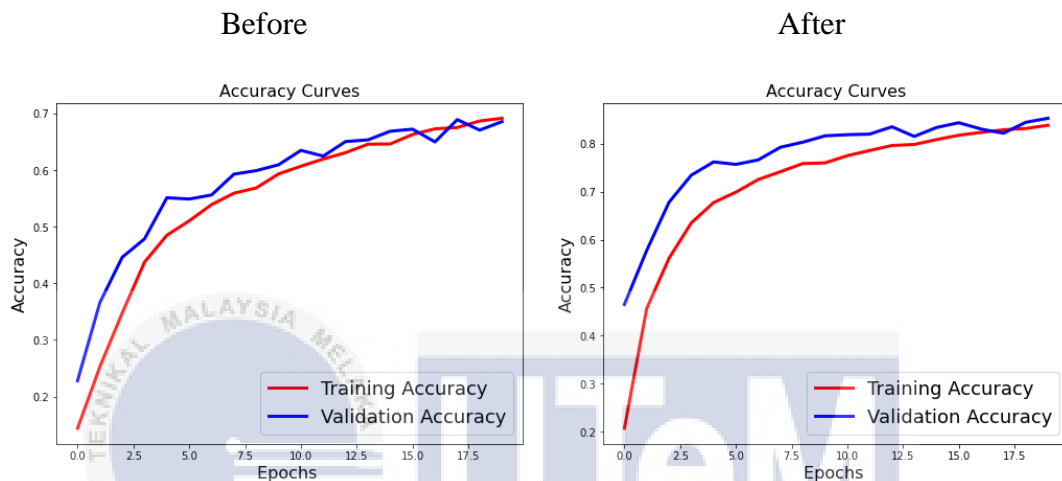


Figure 4.2: Learning curve before and after dataset filtering

Table 4.3: Comparison before and after dataset filtering

H-CNN	Result	
	Before	After
Accuracy	68.69%	84.15%
Loss	0.99069	0.45295
Training time	12.933 minute (776 sec)	11.866 minute (712 sec)
Total parameters	164,815	164,815
Epoch	20	20
Batch size	256	256

4.1.2 Add new dataset to balance the dataset

From the latest progress, after the confusion matrix has been generated as shown in Figure 4.3, the imbalance of the dataset can be noticed clearly. The brightness of the blue box indicates the density of dataset inside each class. An imbalance of dataset could be the result to be biased towards specific class that have the higher amount of dataset. To eliminate this problem, the amount of dataset for each class must be in the same value, in this case the amount is 1,000 set of images to make it fair and square for the H-CNN to learn. Table 4.4, shows the example of the added images for each class.

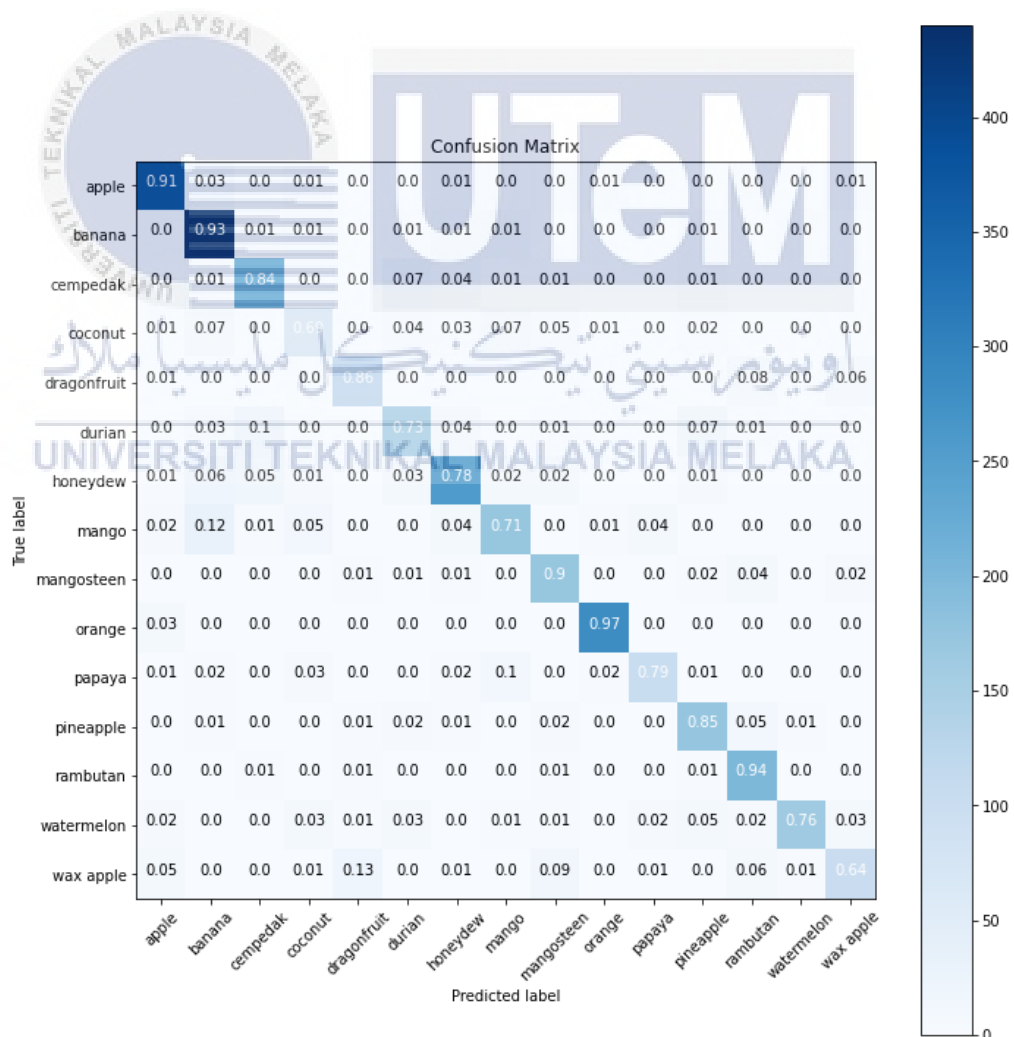






Figure 4.3: Confusion matrix before adding a new dataset

Table 4.4: Example of added images

Class of fruit	Example of image	
	Self-taken image	
Cempedak		
Coconut		

Mango



Papaya



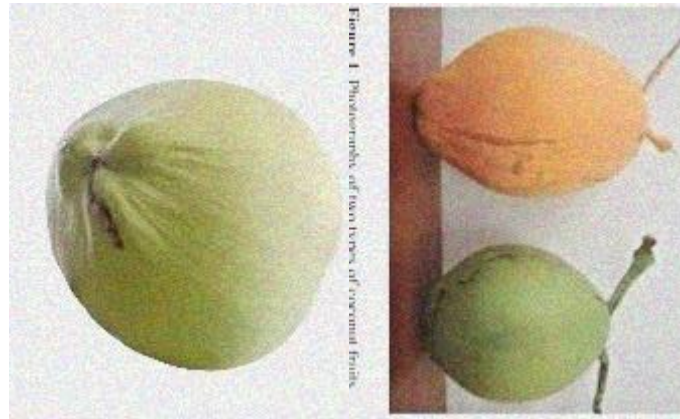
Data augmentation

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

Cempedak



Coconut



Dragonfruit



Mango



Mangosteen



Papaya



Wax Apple



Google Image

Cempedak



Mango



Papaya



 UNIVERSITI TEKNIKAL MALAYSIA MELAKA

The result from this step can be seen clearly from the confusion matrix below in Figure 4.4 that in fact, now the H-CNN has an unbiased dataset which produced a more balanced accuracy. Despite that, the accuracy decreased to 81.63% from 84.15% gained from previous step refers to Table 4.5. This indicates that the difference of 2.52% of accuracy was from a biased class of dataset. The next step will focus on improving the accuracy without changing any dataset anymore.

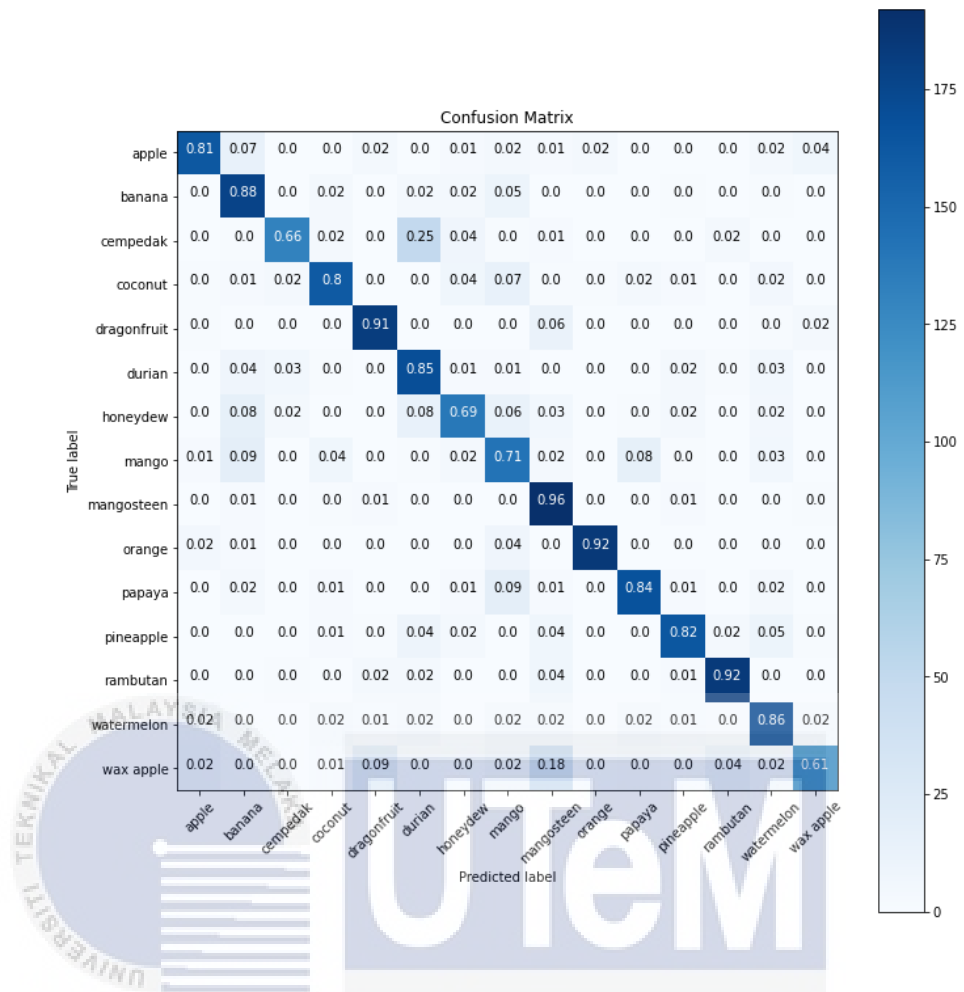


Figure 4.4: Confusion matrix after adding a new dataset

Table 4.5: After adding a new dataset

H-CNN	Result	
	Before	After
Accuracy	84.15%	81.63%
Loss	0.45295	0.54491
Training time	11.866 minute (712 sec)	15.3 minute (918 sec)
Total parameters	164,815	164,815
Epoch	20	20
Batch size	256	256

4.1.3 Tune the hyperparameters

After made some modification to the datasets, the next step is to tune the hyperparameters of the H-CNN model by adjusting the number of filters for each convolutional layer and number of epochs.

4.1.3.1 Number of filters for each convolutional layer

The initial value of hyperparameters can be taken from reputable sources as initial value. In this case, the initial number of filters was taken from Yu-Dong Zhang custom CNN model [15]. However, it is not guaranteed that the value is the best for the H-CNN model due to a different complexity and application for each model. As seen in Table 4.6, by changing the number of filters to 64, 128, 256 and 516 for convolutional layers generates a huge improvement despite using a similar dataset and number of epochs. As expected, the accuracy did rise from 81.63% to 87.83%, but what makes it interesting is despite the total number of parameters massively increase up to 1,153,071 from just 164,815, the training time after tuning the hyperparameters slightly lower than before. This means that the H-CNN model able to learn more efficiently with the new number of filters.

Table 4.6: Result from changing number of filters

H-CNN	Result	
	Before	After
No. of filters for each convolutional layer	L1: 40 L2: 80 L3: 120 L4: 80	L1: 64 L2: 128 L3: 256 L4: 512
Accuracy	81.63%	87.83%

Loss	0.54491	0.37966
Training time	15.3 minute (918 sec)	15.08 minute (905 sec)
Total parameters	164,815	1,153,071
Epoch	20	20
Batch size	256	256

4.1.3.2 Number of epochs

In this step, the main objective is to squeeze the H-CNN model to learn as much as it can by increasing the number of epochs to find the stationary point where it indicates that the H-CNN model had learn enough. Thus, the number of epochs is being set to 30. As expected, there is still a room for an improvement as seen in Table 4.7, the accuracy climbed up to 90.13% with 2.3% margin from before. The training time also rise by 449 seconds which is 7.48 in minute, nonetheless it is not a significant drawback as the main objective is to increase the accuracy as much as it can. Comparing before and after in Figure 4.5, the accuracy curve in after is convincing as the validation accuracy reached the stationary phase.

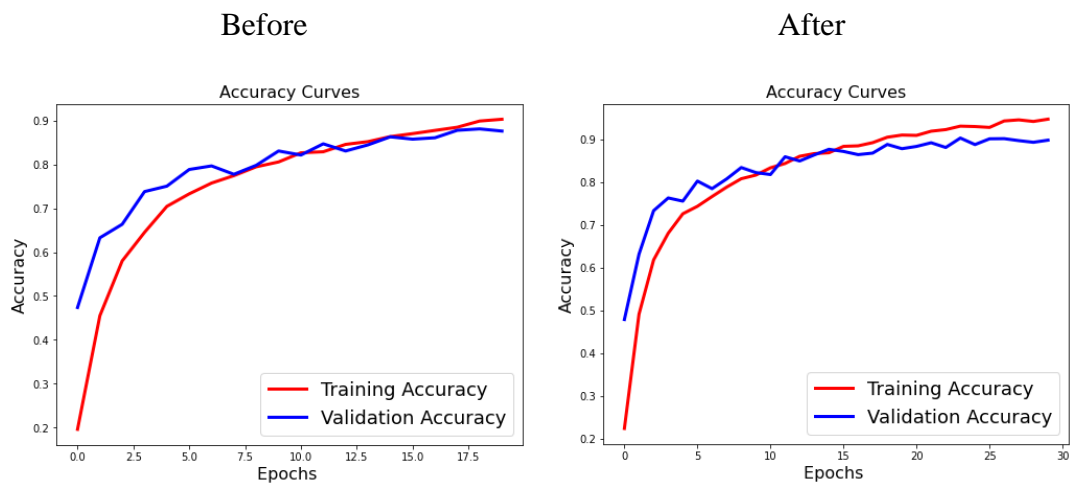
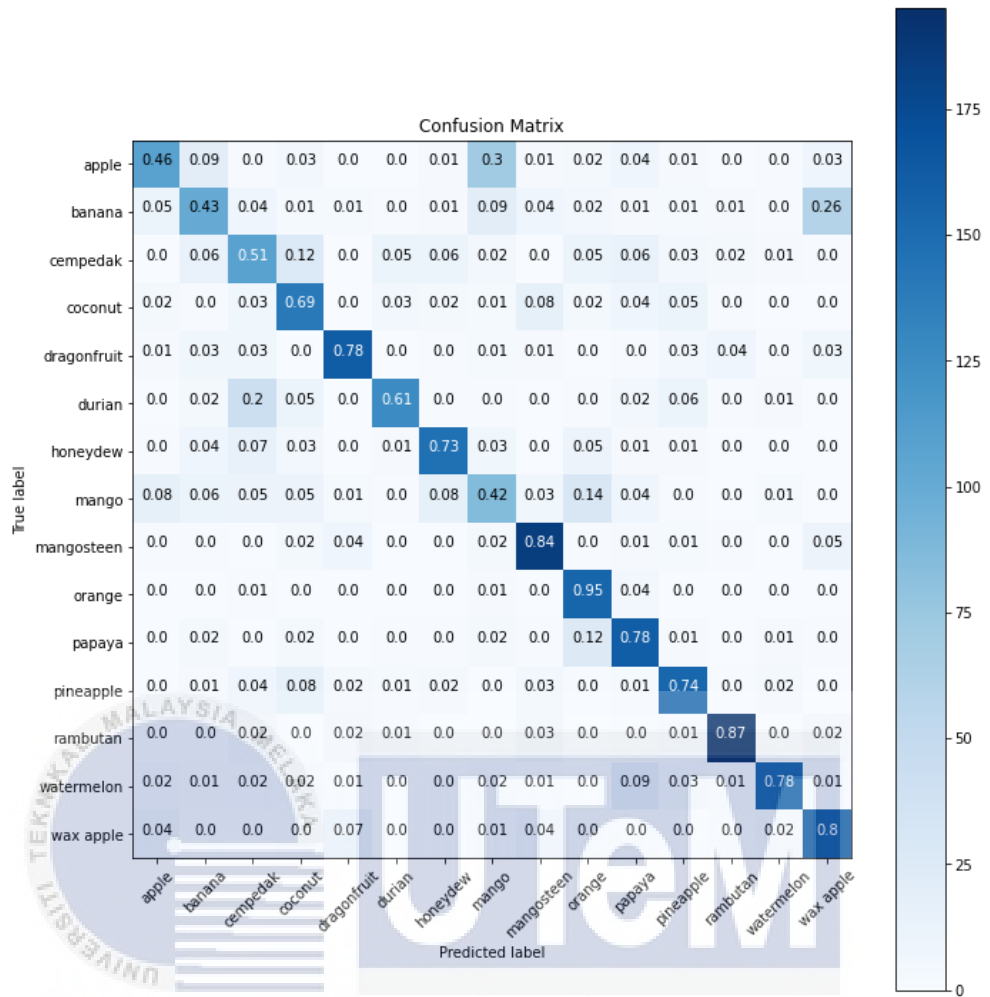


Figure 4.5: Learning curves before and after changing number of epochs

Table 4.7: Result from changing number of epochs

H-CNN	Result	
	Before	After
Accuracy	87.83%	90.13%
Loss	0.37966	0.34639
Training time	15.08 minute (905 sec)	22.56 minute (1354 sec)
Total parameters	1,153,071	1,153,071
Epoch	20	30
Batch size	256	256

A confusion matrix is a table that is used to visualize the performance of a classification model on a set of test data for which the true values are known. The y-axis is where the classifier determines the true label and x-axis is what the classifier predicted it to. A huge difference can be seen between initial confusion matrix from Figure 4.6 and the final confusion matrix from Figure 4.7. The final confusion matrix shows a solid thick blue boxes on their own class proving that the H-CNN has improved a lot with methods above. While the initial confusion matrix displays the poor performance before undergoing any optimization process.



UNIVERSITI TEKNIKAL MALAYSIA MELAKA

Figure 4.6: H-CNN initial confusion matrix

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

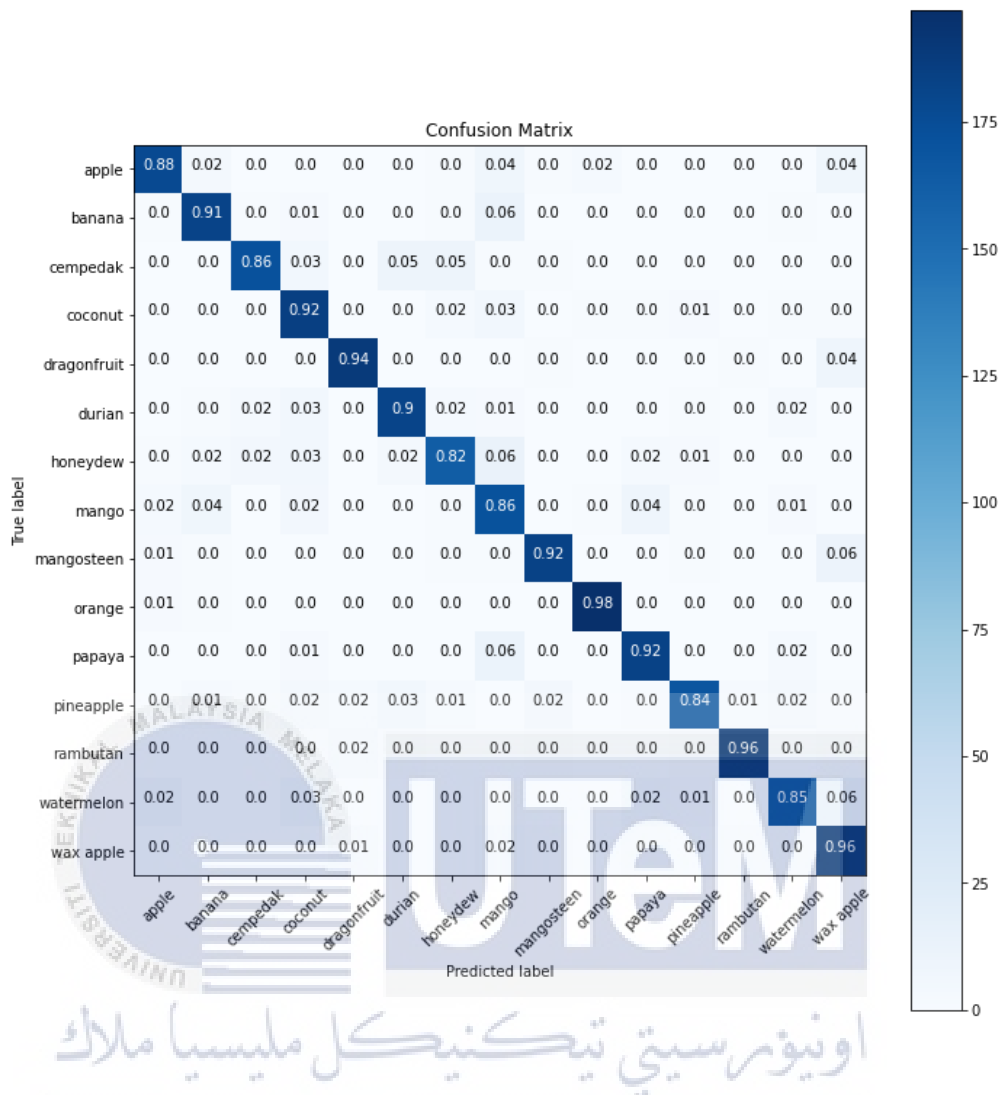


Figure 4.7: H-CNN final confusion matrix

4.2 Inference of H-CNN model

Now, it is time to inference H-CNN to demonstrate the application of the model.

There are 2 ways to accomplish this by via software and hardware.

4.2.1 Software

To run via software, the simplest way is by using the same platform as the model was developed, Google Colab. With a set of script from Figure 4.8, it can mimic a real-world application by predicting a set of random images as in Figure 4.9. The script

performs by taking a random image from 'Model_Test' folder, resize it into model input size 208x256 and predict the image belong to which classes. To demonstrate that the simulation was done fairly, a random set of images was gathered inside a single folder named 'Model_Test'. Note that the images are totally new to the H-CNN model. 75 images were collected from Bing Image Search with 5 images for each class.

```

from keras.preprocessing import image
import numpy as np
import os
import random

sdir=r'drive/My Drive/HAZIQ/INFERENCE/Model_Test/'
flist=os.listdir(sdir)
test_img=random.choice(flist)
test_img=os.path.join(sdir, test_img)
img = image.load_img(test_img, target_size = (208,256))
img = image.img_to_array(img, dtype=np.uint8)
img = np.array(img)/255.0
prediction = model.predict(img[np.newaxis, ...])

print("Probability: ",np.max(prediction[0], axis=-1))
predicted_class = class_names[np.argmax(prediction[0], axis=-1)]
print("Classified: ",predicted_class,'\n')

plt.axis('off')
plt.imshow(img.squeeze())
plt.title("Loaded Image")

```

Figure 4.8: Software inferencing code



Figure 4.9: Software inferencing test images

Table 4.8 shows some of the output from the software inferencing process.

Table 4.8: Software inferencing output

Probability: 0.6968567 Classified: cempedak Text(0.5, 1.0, 'Loaded Image')	Probability: 0.9972408 Classified: dragonfruit Text(0.5, 1.0, 'Loaded Image')
Loaded Image 	Loaded Image 
Probability: 0.5426686 Classified: watermelon Text(0.5, 1.0, 'Loaded Image')	Probability: 0.9812971 Classified: mango Text(0.5, 1.0, 'Loaded Image')
Loaded Image 	Loaded Image 

Probability: 0.9998908
Classified: rambutan

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.5098759
Classified: coconut

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.79612726
Classified: durian

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.99602675
Classified: pineapple

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.9999975
Classified: papaya

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.9529044
Classified: wax apple

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.999998
Classified: orange

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.99999535
Classified: mangosteen

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.9995222
Classified: honeydew

Text(0.5, 1.0, 'Loaded Image')

Loaded Image



Probability: 0.9569744
Classified: banana

Text(0.5, 1.0, 'Loaded Image')

Loaded Image





4.2.2 Hardware

To implement the H-CNN model into a hardware, NVIDIA Jetson Nano was connected with CSI Raspberry Pi Camera V2 as shown in Figure 4.10. From Figure 4.11, firstly save the H-CNN model generated from Google Colab as .hdf5 file. Next, convert the .hdf5 file in ONNX format. Lastly, convert .ONNX file into TensorRT format. Now the model can be inference into NVIDIA Jetson Nano 'imagenet' function. Table 4.9 shows some of the hardware inferencing output.

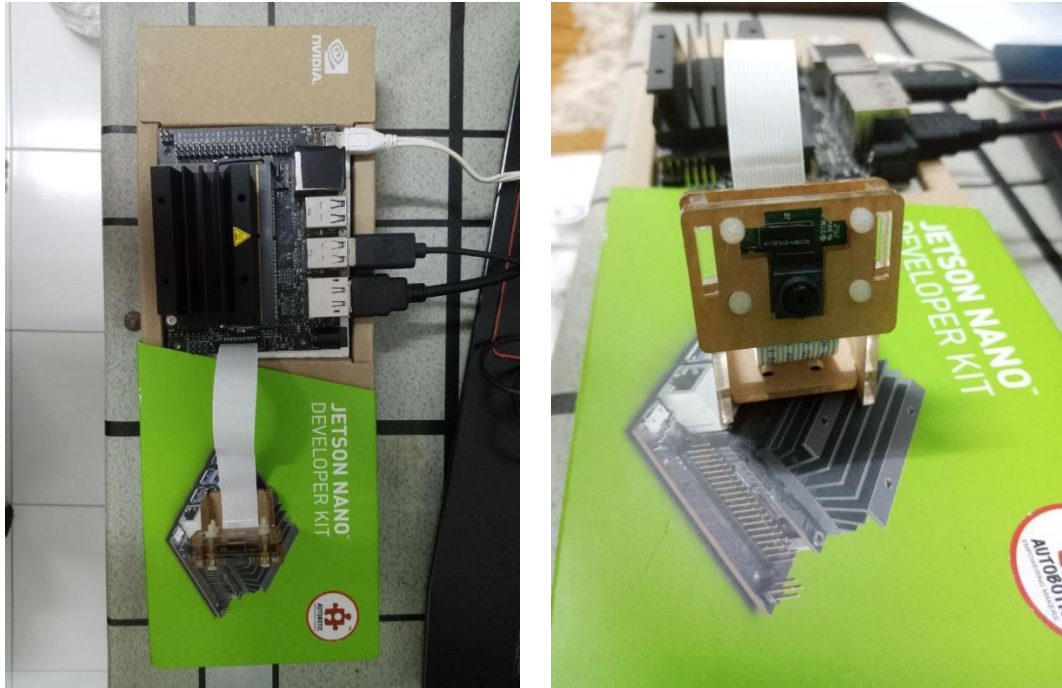


Figure 4.10: NVIDIA Jetson Nano full setup

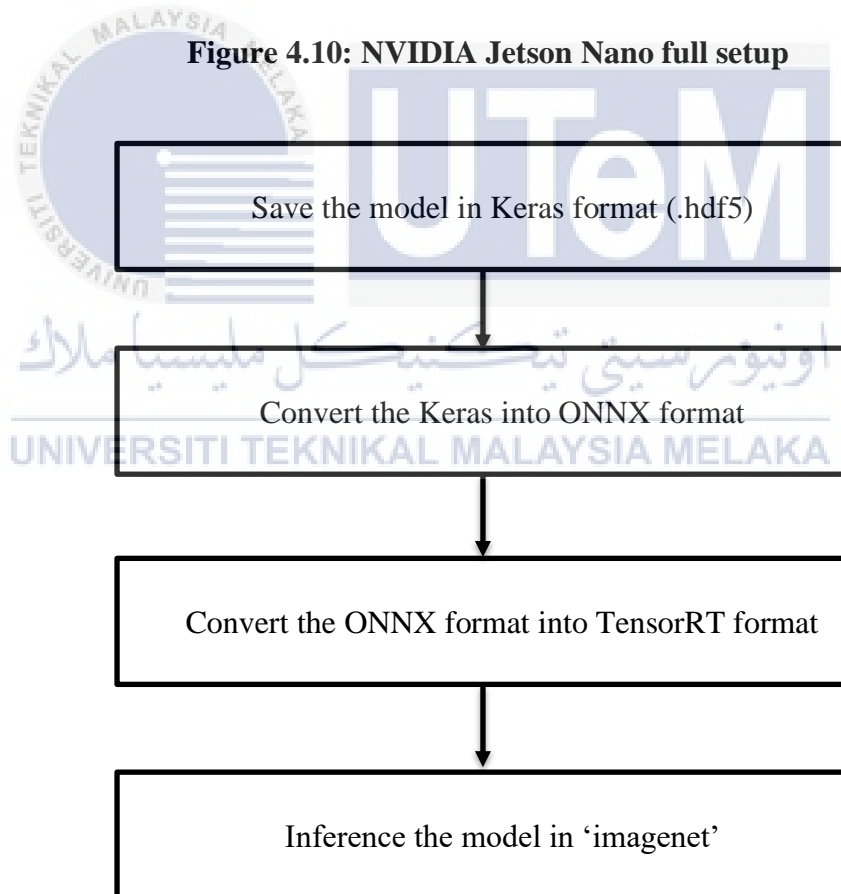
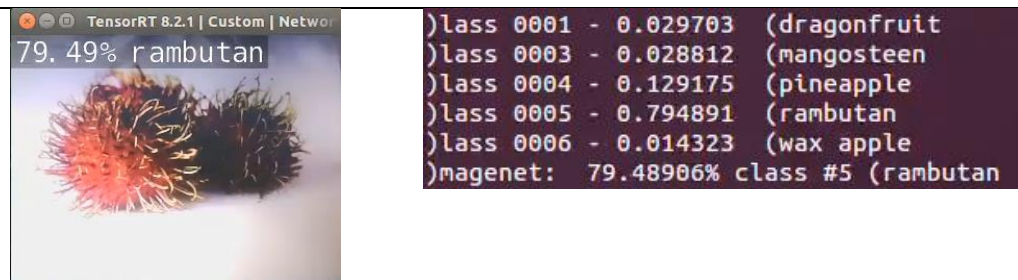


Figure 4.11: Hardware inferencing block diagram

Table 4.9: Hardware inferencing output

 <p>58.45% dragonfruit</p>	<pre>)lass 0001 - 0.584492 (dragonfruit)lass 0003 - 0.134845 (mangosteen)lass 0005 - 0.044370 (rambutan)lass 0006 - 0.233615 (wax apple)magenet: 58.44925% class #1 (dragonfruit</pre>
 <p>88.22% pineapple</p>	<pre>)lass 0002 - 0.059618 (durian)lass 0003 - 0.020476 (mangosteen)lass 0004 - 0.885193 (pineapple)lass 0005 - 0.028455 (rambutan)magenet: 88.51929% class #4 (pineapple</pre>
 <p>54.50% banana</p>	<pre>)lass 0000 - 0.544964 (banana)lass 0002 - 0.157955 (durian)lass 0003 - 0.103759 (mangosteen)lass 0004 - 0.113662 (pineapple)lass 0005 - 0.041035 (rambutan)lass 0006 - 0.036113 (wax apple)magenet: 54.49641% class #0 (banana</pre>
 <p>75.36% durian</p>	<pre>)lass 0002 - 0.753581 (durian)lass 0003 - 0.028086 (mangosteen)lass 0004 - 0.158830 (pineapple)lass 0005 - 0.043900 (rambutan)magenet: 75.35811% class #2 (durian</pre>
 <p>85.78% wax apple</p>	<pre>)lass 0001 - 0.053341 (dragonfruit)lass 0003 - 0.064661 (mangosteen)lass 0005 - 0.015193 (rambutan)lass 0006 - 0.856628 (wax apple)magenet: 85.66275% class #6 (wax apple</pre>



4.3 Comparison with other pre-trained models

Finally, the main objective of this project is to compare designed CNN model architecture with other well established pre-trained models developed by multiple big names such as Oxford University, Google, Microsoft Research etc. In Table 4.10, the comparison shown in terms of accuracy, model size, training time and model complexity. For accuracy, H-CNN comes 3rd out of 4 due to MobileNetV2 suffer an overfitting problem due to the architecture is too complex for dataset used. Nevertheless, the model size is the biggest advantage for H-CNN where it got a massive margin compared to others, but keep in mind that H-CNN was only trained to know 15 types of fruit. The training time does not make a huge difference as the same dataset was being used. As for complexity, H-CNN only use 6-layers of tune able layers compared to the nearest model, VGG16 which use 16-layers and the rest model really uses a lot of layers. Lastly, the total parameters are closely related to the model complexity, whereas the more layers in the architecture (depends on what layer), the higher the total parameters will be.

Table 4.10: Model comparison details

Model	H-CNN	ResNet50	VGG16	MobileNetV	DenseNet12
Name				2	1
Accuracy	90.13%	94.77%	93.93%	77.17%	84.13%
Model size	13 MB	110 MB	60.4 MB	21.2 MB	36 MB
Training time (m)	22	18	41	23	18
Complexity	6-layers	50-layers	16-layers	53-layers	121-layers
Total parameters	1,153,071	25,308,047	15,083,343	3,333,199	7,774,799

4.3.1 Confusion matrix

According to Figure 4.12, 4.13, 4.14, 4.15, and 4.16, all model's confusion matrix and classification report is being shown side by side. Observe that the thick blue box reflects the total accuracy. The blue thickness level represents recall value from classification report. The blue will get thicker with a higher recall value. Recall is the ratio of correctly predicted positive observations to all observations in actual class – yes. Essentially, the higher recall means the better for the confusion matrix.

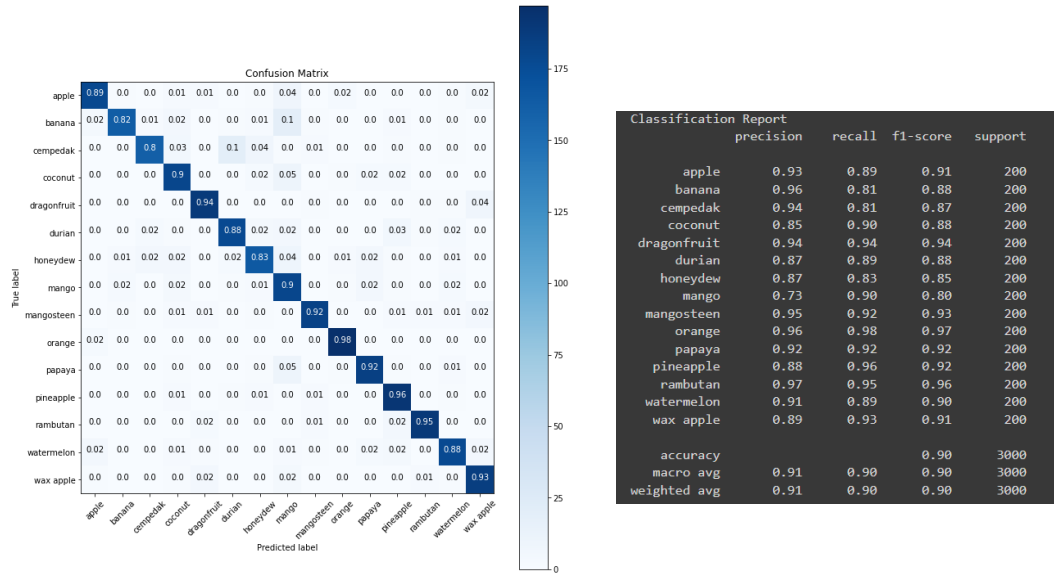


Figure 4.12: H-CNN confusion matrix

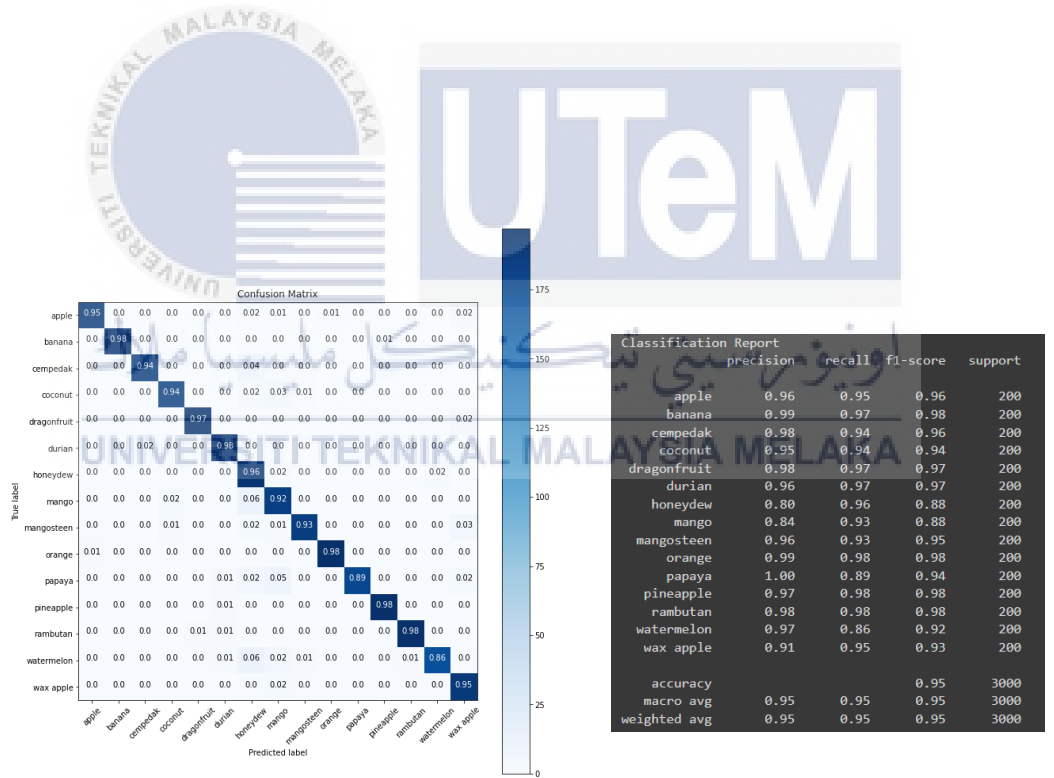


Figure 4.13: ResNet50 confusion matrix

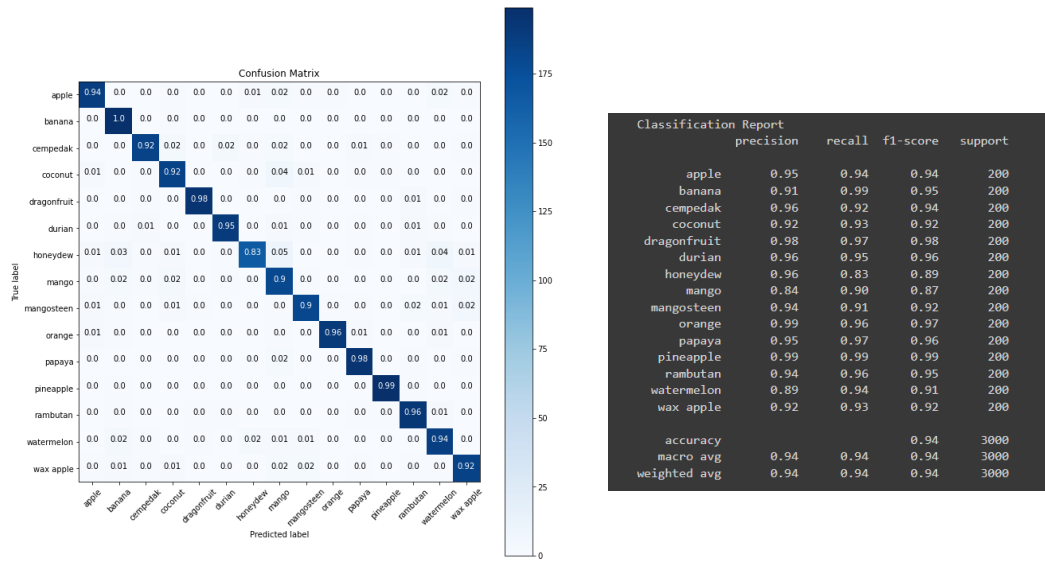


Figure 4.14: VGG16 confusion matrix

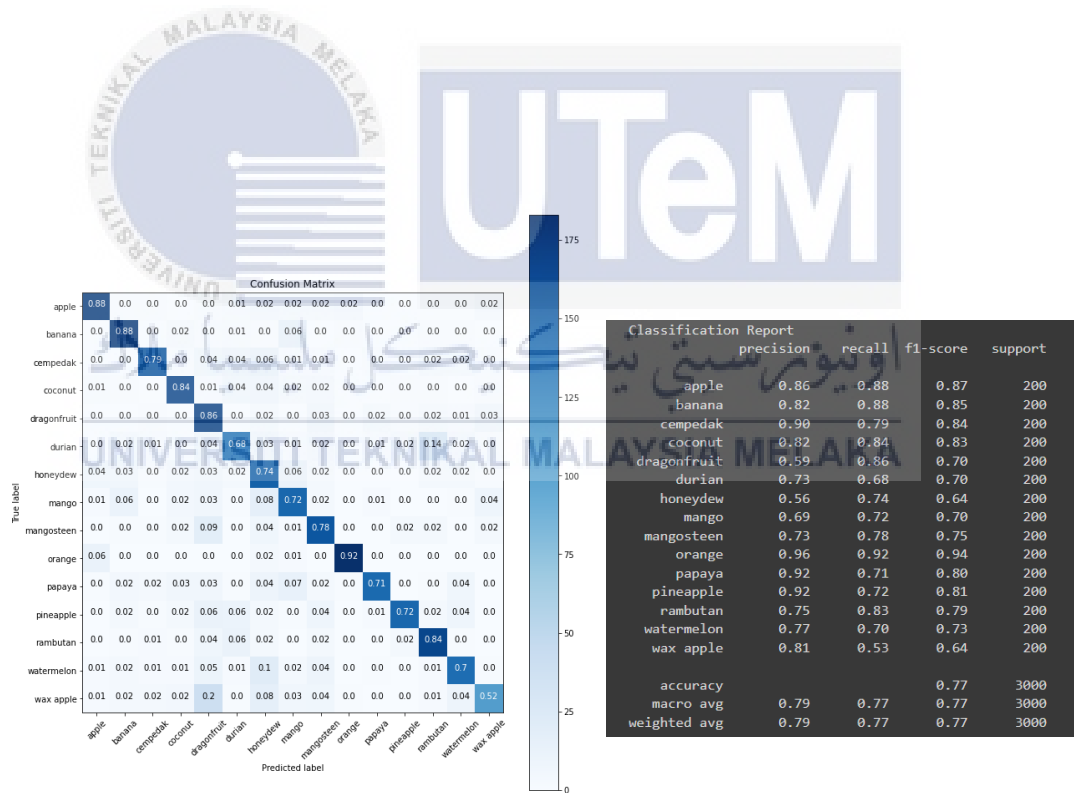


Figure 4.15: MobileNetV2 confusion matrix

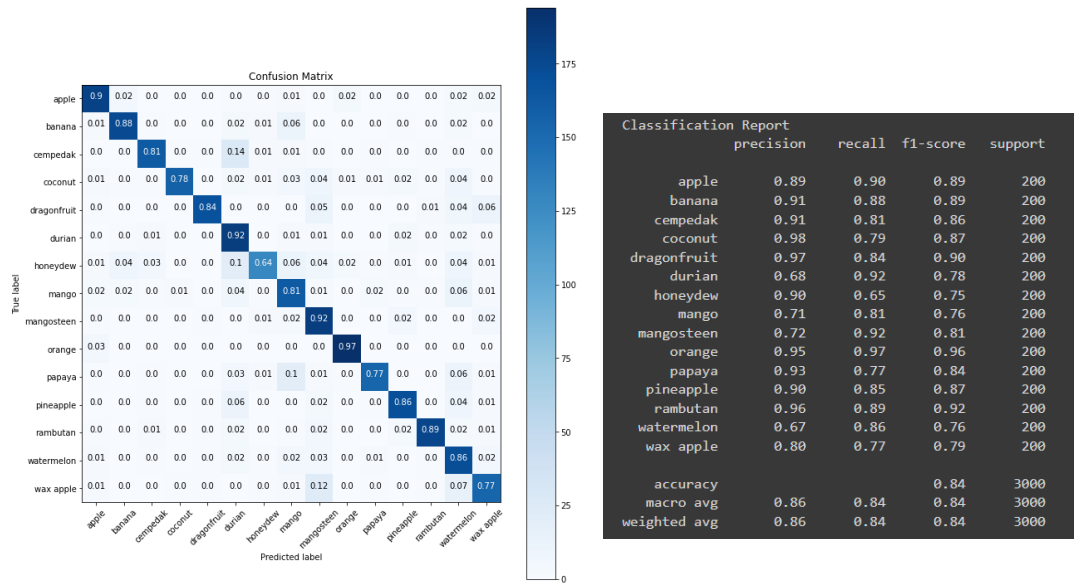


Figure 4.16: DenseNet121 confusion matrix

4.4 Environment and sustainability

To comply with sustainable design criteria, this project fulfils one of the 17 sustainable development goals set by United Nations as shown in Figure 4.17, the 9th goal, Industry, Innovation, and Infrastructure. This goal aims to build resilient infrastructure, promote sustainable industrialization, and foster innovation.



Figure 4.17: Sustainable development goals [22]

According to United Nations in Figure 4.18, global manufacturing growth has been decreasing way before the COVID-19 pandemic hits the world. Then, the pandemic came and make situations worst that causing a trouble for a global supply chain. Thus, technological and innovation progress are key to finding lasting solutions to both economic and environmental challenges, such as increased resource and energy efficiency [23]. This project might be one of the keys to solve these problems with a low emission and high efficiency output without harming our planet.

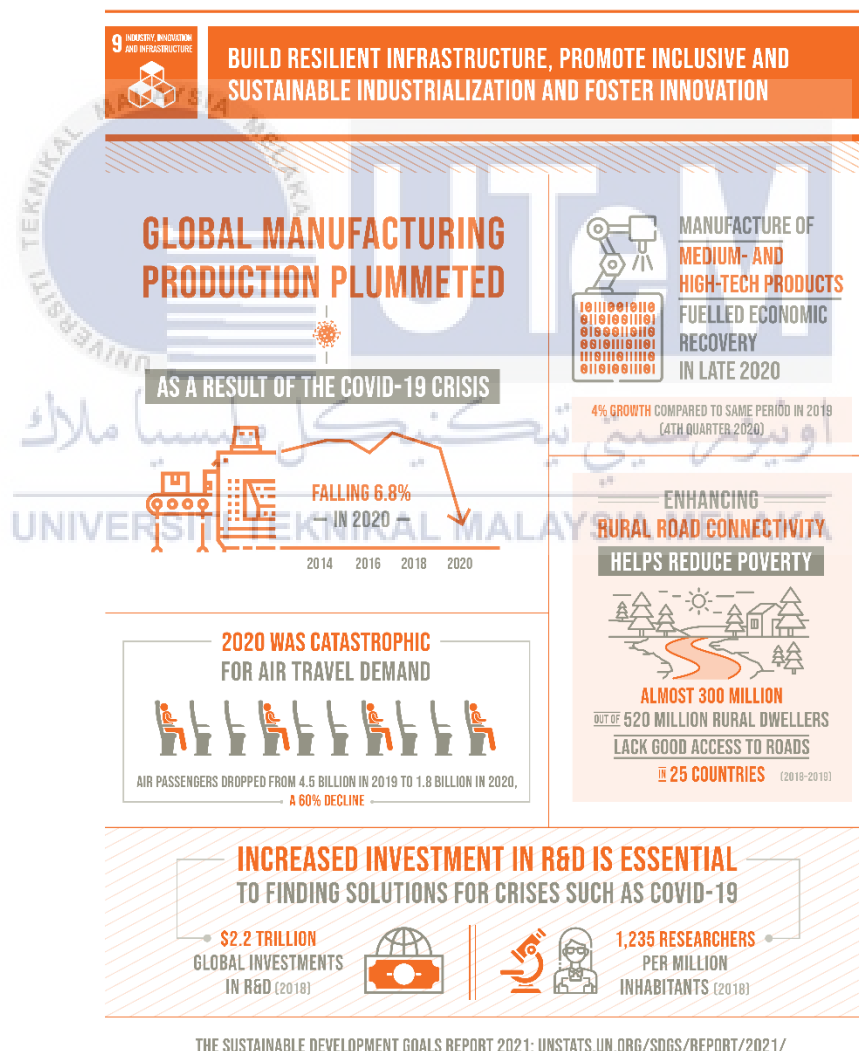


Figure 4.18: 9th goal infographic [24]

CHAPTER 5

CONCLUSION AND FUTURE WORKS



To conclude all the works done while completing this project, it is fortunately achieved all the main objectives. First, the H-CNN model architecture was developed on Google Colab using Python's Keras and TensorFlow library. Secondly, the result from H-CNN was compared thoroughly with pre-trained models such as ResNet50, VGG16 and MobileNetV2 using transfer learning method with same datasets for all models. Lastly, the model successfully inferenced on software and hardware to validate the functionality of the model. After completing this project, it is massively giving an impact of impression on how the computer vision's AI works behind the scenes. It is completely complex and interesting.

Even though this thesis ends here, the project subject still lives on with rapidly advancing days by days. There are still a lot of room for CNN architecture

improvement with many hyperparameters embedded inside the CNN architectures. As hardware's are getting faster, a software also must be in the same trend to ensure the technology beyond these two always in parallel. In the future, there might be a model that can learn as efficiently as this but with a lower amount of dataset. Hopes for future studies on this topic could be more details than this thesis by investigating each layer and their effects on the model's accuracy so that even a simple CNN layer can be utilized as much as it can to be on par with well-established pre-trained models. More studies also suggested on inferencing the CNN model with a small devices such as mobile, embedded, and Internet of Things (IoT) devices and computers like how TensorFlow Lite does. This will surely bring an AI model to a new era where it is light, optimized, and friendly user for cheap microcontroller that are in common right now such as Raspberry Pi. Besides, this studies do not limited to a fruit classification application, it can be implement into any application that requires a similar functionality such as to classify a faulty product in a factory line.

After done a hectic year of work, this project proves that a small simple CNN model can be as good as well-established pre-trained models with a proper development. A dataset remains one of the most important parts to achieve a high accuracy. With a high quality of dataset, it can beat a model with a much more but a low-quality dataset. Then, by tweaking the hyperparameters will help the model to reach a true potential of the developed model.

REFERENCES

- [1] P. Wang, E. Fan, and P. Wang, “Comparative analysis of image classification algorithms based on traditional machine learning and deep learning,” *Pattern Recognition Letters*, vol. 141, pp. 61–67, 2021, doi: 10.1016/j.patrec.2020.07.042.
- [2] J. Janzen, “CNN: TRANSFER LEARNING VS BUILD FROM SCRATCH,” *JOSH JANZEN’S DATA SCIENCE BLOG*, 2018. <https://joshjanzen.com/cnn-transfer-learning-vs-build-from-scratch/> (accessed Jan. 05, 2022).
- [3] “7 APPLICATIONS OF CONVOLUTIONAL NEURAL NETWORKS,” *Flatworld Solutions*. <https://www.flatworldsolutions.com/data-science/articles/7-applications-of-convolutional-neural-networks.php> (accessed Jun. 08, 2022).
- [4] A. Intelligence, “What Is AI ?,” pp. 5–16, 2008, doi: 10.1007/978-3-030-51110-4.
- [5] T. Eprs and E. Parliamentary, *Panel for the Future of Science and Technology EPRS / European Parliamentary Research Service*, no. June. 2020.
- [6] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

- [7] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, and D. De, *Fundamental Concepts of Convolutional Neural Network*, no. June. 2020. doi: 10.1007/978-3-030-32644-9.
- [8] C. C. Chatterjee, “Basics of the Classic CNN,” *towards data science*, 2019. <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add> (accessed Jan. 05, 2022).
- [9] “What do the terms Resolution and Aspect Ratio mean for my television?,” *Sony Support*, 2019. <https://www.sony.com/electronics/support/articles/00105548> (accessed Jan. 05, 2022).
- [10] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,” *towards data science*, 2018. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> (accessed Jan. 05, 2022).
- [11] J. Brownlee, “A Gentle Introduction to the Rectified Linear Unit (ReLU),” *machinelearningmastery*, 2019. <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/> (accessed Jan. 06, 2022).
- [12] A. Kulkarni, D. Chong, and F. A. Batarseh, *Foundations of data imbalance and solutions for a data democracy*. Elsevier Inc., 2020. doi: 10.1016/B978-0-12-818366-3.00005-8.
- [13] scikit, “API Reference,” *scikit-learn*, 2022. <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics> (accessed May 30, 2022).

- [14] G. Zeng, "Fruit and vegetables classification system using image saliency and convolutional neural network," *Proceedings of 2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference, ITOEC 2017*, vol. 2017-Janua, pp. 613–617, 2017, doi: 10.1109/ITOEC.2017.8122370.
- [15] Y. D. Zhang *et al.*, "Image based fruit category classification by 13-layer deep convolutional neural network and data augmentation," *Multimedia Tools and Applications*, vol. 78, no. 3, pp. 3613–3632, 2019, doi: 10.1007/s11042-017-5243-3.
- [16] Y. Miki *et al.*, "Classification of teeth in cone-beam CT using deep convolutional neural network," *Computers in Biology and Medicine*, vol. 80, no. September 2016, pp. 24–29, 2017, doi: 10.1016/j.combiomed.2016.11.003.
- [17] N. Saranya, K. Srinivasan, S. K. P. Kumar, V. Rukkumani, and R. Ramya, "Fruit Classification Using Traditional Machine Learning and Deep Learning Approach," in *Advances in Intelligent Systems and Computing*, ICCVBIC 20., vol. 186 AISC, Springer Nature Switzerland, 2020, pp. 79–89. doi: 10.1007/978-3-030-37218-7.
- [18] W. Koehrsen, "Transfer Learning with Convolutional Neural Networks in PyTorch," *towardsdatascience*, 2018. <https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce> (accessed Jan. 12, 2022).
- [19] I. E. T. Davv, "CLASSIFICATION OF FRUITS USING CONVOLUTIONAL NEURAL NETWORK AND TRANSFER LEARNING MODELS," vol. 24, no. 3, pp. 1–12, 2021.

- [20] A. Patil and M. Rane, “Convolutional Neural Networks: An Overview and Its Applications in Pattern Recognition,” *Smart Innovation, Systems and Technologies*, vol. 195, pp. 21–30, 2021, doi: 10.1007/978-981-15-7078-0_3.
- [21] L. Alzubaidi *et al.*, *Review of deep learning: concepts, CNN architectures, challenges, applications, future directions*, vol. 8, no. 1. Springer International Publishing, 2021. doi: 10.1186/s40537-021-00444-8.
- [22] “#Envision2030: 17 goals to transform the world for persons with disabilities,” *United Nations*, 2016. <https://www.un.org/development/desa/disabilities/envision2030.html> (accessed May 29, 2022).
- [23] “Goal 9: Build resilient infrastructure, promote sustainable industrialization and foster innovation,” *United Nations*, 2021. <https://www.un.org/sustainabledevelopment/infrastructure-industrialization/> (accessed May 29, 2022).
- [24] “Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation,” *United Nations*, 2021. <https://sdgs.un.org/goals/goal9> (accessed May 29, 2022).

APPENDICES

5.1 Appendix A: H-CNN's model architecture script

```

#Import Libraries
import sys
import numpy as np
import tensorflow as tf
from keras import backend as K
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix
from keras.layers import Conv2D, Flatten, Dense, MaxPool2D, MaxPooling2D, Activation, Dropout, BatchNormalization, Input
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
import itertools
from keras.regularizers import l2
import os
import matplotlib.pyplot as plt
from PIL import Image
import math
from keras.preprocessing import image
import random

#Model Creation / Sequential
model = Sequential()

model.add(Conv2D(filters = 64, kernel_size = 2, input_shape=(208, 256, 3), padding='same', name='input_tensor'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=3))

model.add(Conv2D(filters = 128, kernel_size = 2, activation= 'relu', padding='same'))
model.add(MaxPooling2D(pool_size=3))

model.add(Conv2D(filters = 256, kernel_size = 2, activation= 'relu', padding='same'))
model.add(MaxPooling2D(pool_size=3))

model.add(Conv2D(filters = 512, kernel_size = 2, activation= 'relu', padding='same'))
model.add(MaxPooling2D(pool_size=3))

# specifying parameters for fully connected layer
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(150))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Dense(15, activation = 'softmax', name='output_tensor'))

#Get summary of the model
model.summary()

#Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

```

5.2 Appendix B: H-CNN's model architecture summary

Layer (type)	Output Shape	Param #
input_tensor (Conv2D)	(None, 208, 256, 64)	832
activation (Activation)	(None, 208, 256, 64)	0
max_pooling2d (MaxPooling2D)	(None, 69, 85, 64)	0
conv2d (Conv2D)	(None, 69, 85, 128)	32896
max_pooling2d_1 (MaxPooling2D)	(None, 23, 28, 128)	0
conv2d_1 (Conv2D)	(None, 23, 28, 256)	131328
max_pooling2d_2 (MaxPooling2D)	(None, 7, 9, 256)	0
conv2d_2 (Conv2D)	(None, 7, 9, 512)	524800
max_pooling2d_3 (MaxPooling2D)	(None, 2, 3, 512)	0
dropout (Dropout)	(None, 2, 3, 512)	0
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 150)	460950
activation_1 (Activation)	(None, 150)	0
dropout_1 (Dropout)	(None, 150)	0
output_tensor (Dense)	(None, 15)	2265

Total params: 1,153,071
 Trainable params: 1,153,071
 Non-trainable params: 0