SARM: SYNTHETIC DATA ANNOTATION FOR ENHANCING THE EXPERIENCES OF AUGMENTED REALITY APPLICATION BASED ON MACHINE LEARNING

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Supervisor
Assoc. Prof. Minh Nguyen
Assoc. Prof. Wei Qi Yan

30th May 2022

By
Robert (Huy) Le
School of Engineering, Computer and Mathematical Sciences

"I'm excited about Augmented Reality because unlike Virtual Reality, which closes the world out, AR allows individuals to be present in the world but hopefully allows an improvement on what's happening presently."

Tim Cook, CEO Apple Inc.

Abstract

Background and Objective: Augmented Reality is one of the fastest-growing fields, increasing funding for the last few years, as people realise the potential benefits of rendering virtual information in the real world. As the equipment gets more commercialised, the cost would get lowered while the performance also goes up. However, most of today's Augmented Reality marker-based applications would use local features detection and tracking techniques. The disadvantages of applying these techniques are that the markers must be modified to match the unique classified algorithms or suffer from lower detection accuracy. Machine learning is a perfect solution to overcome the current drawbacks of image processing in Augmented Reality applications.

Methods: This thesis is split into two investigation directions. The **first investigation** is to implement new Augmented Reality markers with concealed information such as bar-code or quick response code while keeping most of the visual information of the original texture. The **second investigation** demonstrates the Augmented Reality marker without using any embedded codes and original texture modification required by immersing the machine learning technology into the marker detection process. The new approach incorporated Machine Learning using deep neural networks to detect and track the Augmented Reality application's marker targets. The research implemented the auto-generated dataset tool, which uses for the Machine Learning dataset preparation step. The final iOS prototype application was developed to incorporate object detection, object tracking and Augmented Reality. The Machine Learning model was taught to recognise the differences between targets using YOLO's most famous object detection methods. The model was trained by either Pytorch, and the final product uses a valuable toolkit for developing the Augmented Reality application called ARKit.

Results: Several different experimental exercises have been conducted to qualify the proposed methods on technical performances. The experimental outcomes indicated that the object detection model could achieve over 80% precision, over 90% recall, and over 70% mean average precision using proposed synthetic datasets. The proposed method significantly improves object detection accuracy where it could achieve at least 18% higher than the real-world dataset. The iOS prototype can detect the target markers and display the augmented objects under different lighting conditions at an average rate of 50 frames per second.

Contents

List of Tables

List of Figures

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Robert Huy Le

Auckland, New Zealand
30th May 2022

Publications

This thesis is based on the following original publications which are referred to in the number as Paper 1 - Paper 14. The publications are reproduced with kind permission from the publishers.

1. **Le, H.**, Nguyen, M., Yan, W. Q, S. Lo. (2021, December). Training object detection neural network with synthetic dataset for transportation signs. In *2021 36th International Conference on Image and Vision Computing New Zealand (IVCNZ)* (**Accepted**)

2. **Le, H.**, Nguyen, M., Yan, W. Q., Nguyen, H. (2021). Augmented Reality and Machine Learning Incorporation Using YOLOv3 and ARKit. *Applied Sciences, 11*(13), 6006. https://doi.org/10.3390/app11136006 (**Published**)

3. Nguyen, Q., Nguyen, M., Bowen Sun, **H. L**. (2021). New Zealand Shellfish Detection, Recognition and Counting: A Deep Learning Approach on Mobile Devices. *Geometry and Vision, 1386*, 119. https://doi.org/10.1007/978-3-030-72073-5_10 (**Published**)

4. **Le, H.**, Nguyen, M., Yan, W. Q. (2020, November). Machine Learning with Synthetic Data–a New Way to Learn and Classify the Pictorial Augmented Reality Markers in Real-Time. In *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)* (pp. 1-6). IEEE. https://doi.org/10.1109/IVCNZ 51579.2020.9290606 (**Published**)

5. **Le, H.**, Nguyen, M., Nguyen, Q., Nguyen, H., Yan, W. Q. (2020, October). Automatic Data Generation for Deep Learning Model Training of Image Classification used for Augmented Reality on Pre-school Books. In *2020 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)* (pp. 1-5). IEEE. https://doi.org/10.1109/MAPR49794.2020.9237760 (**Published**)

6. Nguyen, H., Nguyen, M., Nguyen, Q., Yang, S., Le, H. (2020, October). Web-based object detection and sound feedback system for visually impaired people. In *2020 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)* (pp. 1-6). IEEE. https://doi.org/10.1109/MAPR49794.2020.9237770 (**Published**)

7. Nguyen, M., **Le, H.**, Yan, W. Q. (2020, February). Red-Green-Blue Augmented Reality Tags for Retail Stores. In *International Conference on Advanced Concepts for Intelligent Vision Systems* (pp. 467-479). Springer, Cham. https://doi.org/10.1007/978-3-030-40605-9_40 (**Published**)

8. **Le, H.**, Nguyen, M. (2020). An Online Platform for Enhancing Learning Experiences with Web-Based Augmented Reality and Pictorial Bar Code. In *Augmented Reality in Education* (pp. 45-57). Springer, Cham. https://doi.org/10.1007/978-3-030-42156-4_3 (**Published**)

9. **Le, H.**, Nguyen, M., Yan, W. Q. (2019, November). A Web-Based Augmented Reality Approach to Instantly View and Display 4D Medical Images. In *ACPR (2)* (pp. 691-704). https://doi.org/10.1007/978-3-030-41299-9_54 (**Published**)

10. Nguyen, M., Lai, M. P., **Le, H.**, Yan, W. Q. (2019, November). A web-based augmented reality plat-form using pictorial QR code for educational purposes and beyond. In *25th ACM symposium on virtual reality software and technology* (pp. 1-2). https://doi.org/10.1145/3359996.3364793 (**Published**)

11. Wang, I., Nguyen, M., **Le, H.**, Yan, W., Hooper, S. (2018, November). Enhancing visualisation of anatomical presentation and education using marker-based augmented reality technology on web-based platform. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)* (pp. 1-6). IEEE. https://doi.org/10.1109/AVSS.2018.8639147 (**Published**)

12. Nguyen, M., **Le, H.**, Yan, W. Q., Dawda, A. (2018, November). A vision aid for the visually impaired using commodity dual-rear-camera smartphones. In *2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)* (pp. 1-6). IEEE. https://doi.org/10.1109/M2VIP.2018.8600857 (**Published**)

13. **Le, H.**, Nguyen, M., Yan, W. Q. (2018, November). CSPM: a Novel Curtain Style Pictorial Marker for Enhancing Augmented Reality Experiences. In *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)* (pp. 1-6). IEEE. https://doi.org/10.1109/IVCNZ.2018.8634697 (**Published**)

14. Vo, T. V., Nguyen, M., **Le, H.** (2018, November). Augmented Reality on Mobile Platform: A New Way to Instantly View and Display Foreign Currency Exchange Rate. In *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)* (pp. 1-5). IEEE. https://doi.org/10.1109/IVCNZ.2018.8634773 (**Published**)

Acknowledgements

Firstly, to my two supervisors: **Associate Professor Minh Nguyen**, for his continuous support throughout this research. Assoc. Prof. Nguyen is not only my primary supervisor, my longtime friend, but he is also like my older brother, who has encouraged and guided me into the academic world. He was among the first and maybe the only person who believed in me and gave me this fantastic opportunity to expand my knowledge when I came back to study in 2016. So thank you again, brother, for allowing me to grow and be reborn again. To my second supervisor, **Associate Professor Wei Qi Yan**, for his kind guidance and mentorship throughout this PhD's academic article writing stages.

To **Barry Dowdeswell** for his excellent thesis proofreading service. He has been working so hard and trying to give me tips on writing the best thesis I could. So thank you again, Barry for your outstanding works, this thesis would not be perfect without your help.

I have received significant financial and technical support throughout my PhD. I would like to thank the AUT technical team for providing me with all the research equipment and tools that I needed; and to the School of Engineering, Computer and Mathematical Sciences for awarding me the **doctoral scholarships** that have provided excellent support over the last three years.

And finally, to my beloved family for their love and support, especially to **my parents** for their sacrifices and for providing moral and financial support for my decision to go back to study. I am grateful for having such a great family. I am so proud of all of them. Most of all, I thank **my dear wife - Vy Nguyen**, who has supported me more than anyone. She has always encouraged and reassured me. She has taken great responsibly for housework and let me both work and rest. I thank her with all my heart.

Robert Huy Le

Auckland, New Zealand
30th May 2022

Chapter 1

Introduction

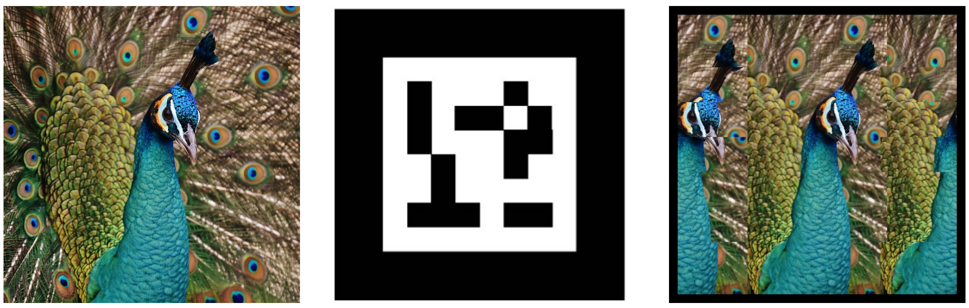
Parts of this Chapter have been published in ***paper 2*** *listed in the publication list*

The combination of natural and un-real environments sounds like a fiction story in the mid 20th century, but it is now real after a long history of research and failure. The result is “Augmented Reality”, where the virtual information is rendered upon the natural objects in real-time. However, due to the detection accuracy and usability limitations, Augmented Reality is still far from reality. The Machine Learning’s advancement in computer vision on another hand significantly increased object detection accuracy, and the interest in computer vision applications in many diverse areas have gained more attention. One clear example is that Augmented Reality could be a valuable and handy knowledge delivery tool for both students and teachers to replace the original hard copy materials in the education sector. The development of vision-based Machine Learning to distinguish objects is crucial in the computer vision field. Therefore, this thesis aims to improve the Augmented Reality object detection capability by incorporating Machine Learning using deep neural networks.

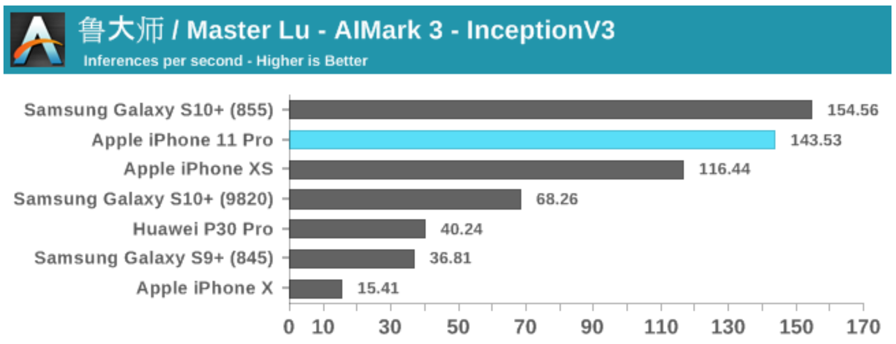
1.1 Inspiration

Augmented Reality has a great potential to provide information and direct assistance virtually for daily activities, and there has been significant research and commercial interested in this field. Marker-based is the most used type of Augmented Reality where the application is implemented to recognise and track the targeted marker to active the experience. However, there are still many challenging issues that are waiting to be discovered and improved. One of the significant issues is that there are many different Augmented Reality markers with unique encoded information algorithms available in the market (Rekimoto & Ayatsuka, 2000; Kan, Teng & Chou, 2009; Liu, Tan & Chu, 2010; Le & Nguyen, 2017; Nguyen, Tran, Le & Yan, 2017), such as template (Figure 1.1a), bar-code (Figure 1.1b) or pictorial bar-code (Figure 1.1c) markers. Most of them usually require the users to modify either partially or entirely their contents.

However, the users are not always happy to change their existing materials to match the Augmented Reality markers requirements (Koch, Neges, König & Abramovici, 2014). The marker recognition process could be another issue where the traditional computer vision-based techniques, such as scale-invariant feature transformations (Lindeberg, 2012) or histograms of oriented gradients (Lowe, 1999) are used for the classification task. These mathematical methods are vulnerable to unanticipated real-world lighting (Wu et al., 2013), marker orientation (Cheon, Lee, Hyun & Park, 2011), and unexpected noises (Bobeshko, 2017).



(a) Template marker (b) Bar-code marker (c) Pictorial bar-code marker



(d)

Figure 1.1: A few of Augmented Reality markers usually present unuseful information that requires users to modify the original pictorial either partially or fully (a, b, and c). The Machine Learning inference performance among different phone models using InceptionV3 (d).

The revolutionary of Machine Learning using deep learning convolutional neural networks hopes to overcome the traditional computer vision issues after AlexNet won the ImageNet challenge in 2012 (Krizhevsky, Sutskever & Hinton, 2012). Later improvement in the Machine Learning field has achieved human-like accuracy in object recognition (Acharya, Hayes & Kanan, 2020) and real-time data processing (Tan, Pang & Le, 2020), making the ideas of Augmented Reality and Machine Learning combination more and more possible than ever.

Another inspiration is that the Apple bionic computing processing unit is getting more powerful (Figure 1.1d). Since the release of the A11 chip on the iPhone X model, the new neural engine accelerator's combination gives nine times faster Machine Learning calculations than a regular CPU [1]. It means that the tiny mobile devices are now easier and more efficient in executing the heavy deep learning-Augmented Reality-based applications. However, deep learning requires typically massive data sets to train on, and it could be non-beneficial and have fewer quality data, which can reduce the accuracy rate. The high time-consuming of trainable data preparation is another limitation of deep learning, where it is usually done manually (Mamoshina, Vieira, Putin & Zhavoronkov, 2016).

1.2 Thesis goal and research questions

This research's purpose is to **enhance Augmented Reality experiences by developing a robust vision-based application**. The ultimate goal was to use the meaningful images as the markers within the Augmented Reality applications. Furthermore, the research is split into two investigation directions:

[1]Chandra, H. (2018). Hardware acceleration for Machine Learning on apple and android devices. Retrieved from https://heartbeat.fritz.ai/hardware-acceleration-for-machine-learning-on-apple-and-android-f3e6ca85bda6

The first investigation is the implementation of the new Augmented Reality markers with concealed information such as bar-code or quick response code while keeping most of the visual information of the original texture. The details are demonstrated in **Section 3.4**.

The second investigation demonstrates the Augmented Reality marker without using any embedded codes and original texture modification required by immersing the Machine Learning technology into the marker detection process. This investigation direction proposed a new way to generate the Machine Learning dataset synthetically to overcome the extensive time consuming during the manual data annotation process. The implementation plan will be described in **Section 1.3**, and details demonstrated in **Chapter 5**.

The following research questions **(RQs)** have been addressed in order to accomplish the described research goals:

RQ 1 (described in detail in Chapter 3): *How well do the new proposed markers perform comparing the original Augmented Reality markers? If so, what disadvantages could the new proposed markers have?*

RQ 2: (described in detail in Section 6.2) *How well does the synthetic dataset perform during the neural network training and testing process?*

RQ 3 (described in detail in Section 6.3): *What type of natural environments could be complex for the deep learning model?*

RQ 4 (described in detail in Section 6.4): *How well does the implemented application perform under real-world situations?*

1.3 SARM implementation planning

The second investigation of this thesis aims to use Machine Learning techniques to enhance Augmented Reality experiences by using the original texture as the target Augmented Reality marker. The proposed method can classify any images as Augmented Reality markers without the need for users to modify the contents. The process also requires a minimal amount of time to produce the marker as quickly as possible. The proposed system is called "**S**ynthetic data annotation system for **A**ugmented **R**eality **M**achine learning-based application" or **SARM**. The practical implementation is divided into three different modules to present the proposed system in more detail:

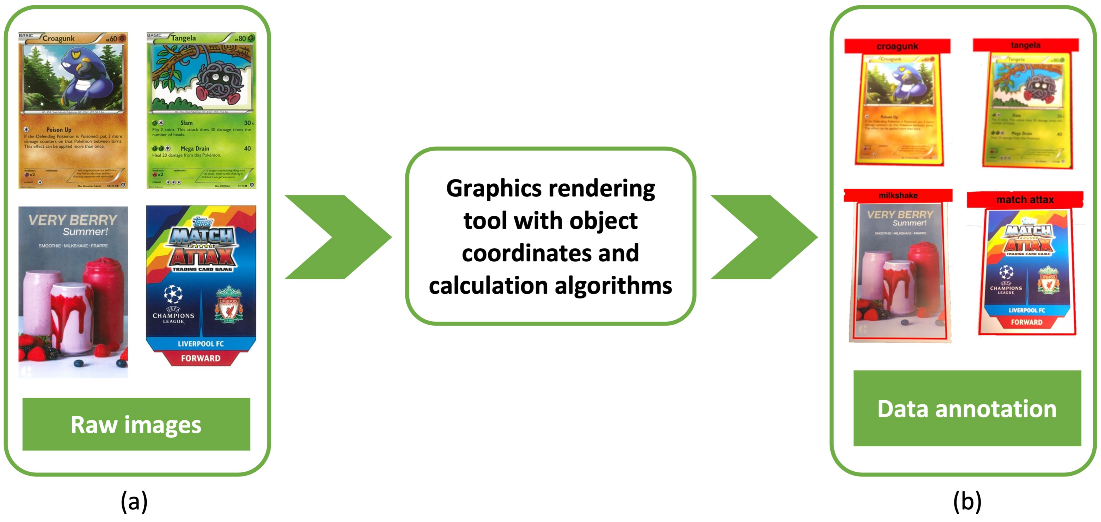

Raw images
Graphics rendering tool with object coordinates and calculation algorithms
Data annotation
croagunk
tangela
milkshake
match attax
(a)
(b)

Figure 1.2: The raw images (a) are processed through the graphic rendering unit and augmentation algorithms to output the class name and coordinates for further deep learning neural network training (b).

Module 1 - Synthetic data annotation is a critical step during the deep learning neural network training process (as shown in Figure 1.2). The main principle of this process applies domain-specific techniques to the original data and generates new data in different forms and conditions using available graphic rendering software. This new method is quicker than traditional data annotation, which can produce up to 20 training dataset images per second on a graphic rendering unit. It also gives an excellent opportunity to regenerate or modify the dataset faster for an additional training class or improve the deep learning model quality.

Module 2 - Deep neural network training provides the learning capability for the system to determine the potential marker from the scene. Many different convolutional neural networks, including image classification and object detection models such as AlexNet, ResNet, or YOLO, were successfully implemented and gave outstanding results. This uses a suitable deep learning model to train with the previous module's dataset and evaluate the training and test outputs.

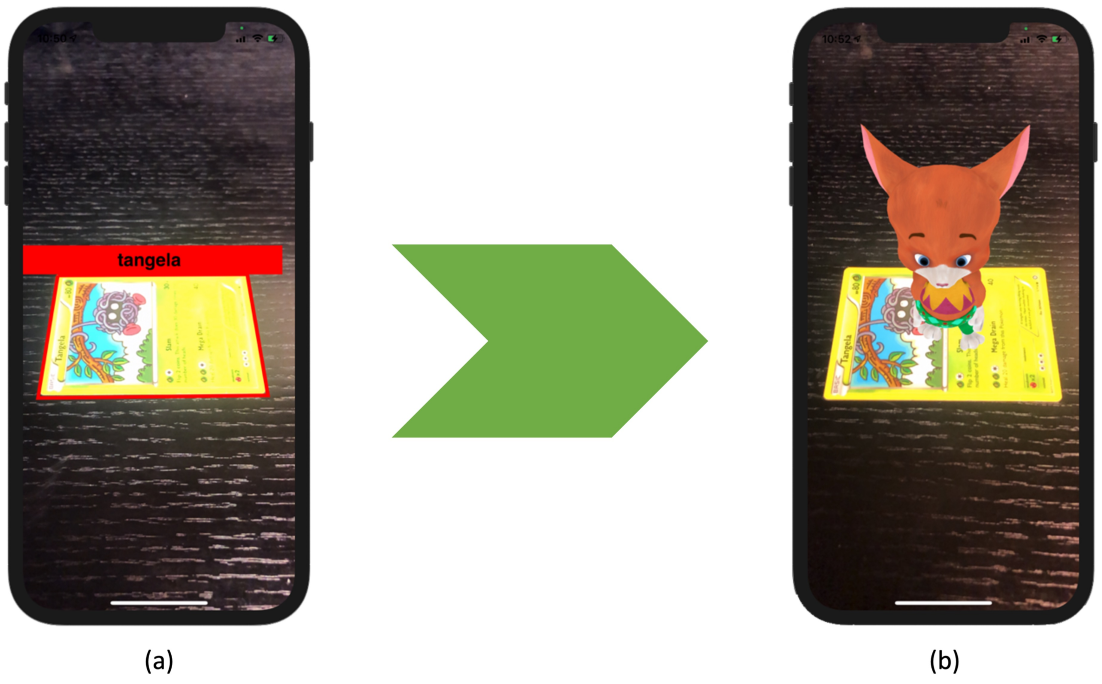


Figure 1.3: Machine learning and Augmented Reality incorporation module where the object detection algorithm using the deep neural network (a) can easily find the marker in the scene and produce the Augmented Reality experience (b).

Module 3 - Machine learning and Augmented Reality incorporation allows the system to combine object prediction and project the 2D coordinates to the natural world 3D coordinates in the Augmented Reality scene (as shown in Figure 1.3). The system then renders the virtual information, such as a 3D model, on the top of the predicted marker based on its identity. In short, the markers and new methods proposed in this thesis demonstrated four primary contributions, which will be presented in the next Section.

1.4 Principal contributions of this thesis

This thesis purpose is to examine the use and performance of **newly proposed Augmented Reality markers that can conceal the encrypted information** and another marker that does not require the original texture modification **(SARM)** by using the Machine Learning techniques. In short, this thesis makes four main contributions that are listed below, along with their related publications:

1. **Investigation of new Augmented Reality markers that could conceal the hidden codes and provide minimal original texture modification.** In many Augmented Reality applications, visual content markers (also known as template markers) are often used to provide meaningful pictorial information (Lepetit, Lagger & Fua, 2005; Lepetit & Fua, 2006). However, they required manual image registration and a collection of registered images for comparison processing. On the other hand, data marker such as a bar-code or quick response (QR) code provides a capability to improve template markers drawbacks. They are designed with many black and white lines, bars, or patterns. These usually present binary numbers of "0" or "1" and take relatively little time to decode computationally (Rekimoto & Ayatsuka, 2000; Mohan, Woo, Hiura, Smithwick & Raskar, 2009; Olson, 2011). However, these data markers generally do not present much helpful information to users unless extra graphical contents are added to the sides.

Consequently, both template and data markers have their pros and cons. The proposed markers aim to fill the missing gap to consider both makers' technical advantages in providing a proper graphical content presentation and improving detectability performance. More details on the proposed markers are presented in **Section 3.4**.

Related publications:

- Nguyen, M., **Le, H.**, Yan, W. Q. (2020, February). Red-Green-Blue Augmented Reality Tags for Retail Stores. In *International Conference on Advanced Concepts for Intelligent Vision Systems* (pp. 467-479). Springer, Cham. https://doi.org/10.1007/978-3-030-40605-9_40 (**Published**)
- Nguyen, M., Lai, M. P., **Le, H.**, Yan, W. Q. (2019, November). A web-based augmented reality plat-form using pictorial QR code for educational purposes and beyond. In *25th ACM symposium on virtual reality software and technology* (pp. 1-2). https://doi.org/10.1145/3359996.3364793 (**Published**)
- **Le, H.**, Nguyen, M., Yan, W. Q. (2018, November). CSPM: a Novel Curtain Style Pictorial Marker for Enhancing Augmented Reality Experiences. In *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)* (pp. 1-6). IEEE. https://doi.org/10.1109/IVCNZ.2018.8634697 (**Published**)

2. **Investigation of the performance of object detection deep neural network training and detectability using the synthetic dataset**. The image recognition accuracy using computer vision-based features extraction techniques such as histogram of oriented gradients (Lowe, 1999) or scale-invariant feature transform (Lindeberg, 2012) usually come with some issues. These mathematical algorithms are susceptible to unexpected real-world illumination (Wu et al., 2013)

or orientation (Cheon et al., 2011) and could be unreliable when the noise objects partially cover the markers (Bobeshko, 2017). The revolution of Machine Learning and convolutional neural networks gives a promising solution of overcoming the traditional computer vision issues (Krizhevsky et al., 2012). However, as described in Section 1.1, it typically requires massive datasets for the Machine Learning training process, and it could be non-beneficial and have fewer quality data, which can reduce the detection accuracy rate. The trainable data preparation time requirements are another limitation of Machine Learning where this step is usually done manually. The newly proposed method generates a synthetic dataset and labels each data item automatically using various computer graphics rendering techniques to overcome this issue. The figures of chosen Machine Learning trainable classes are randomly blended into different natural-looking backgrounds and other ambient lighting conditions and mixed camera orientation. The trained model using the proposed synthetic dataset could achieve over 80% precision, over 90% recall, and over 70% mean average precision against the real-world test dataset.

Related publications:

- **Le, H.**, Nguyen, M., Yan, W. Q, S. Lo. (2021, December). Training object detection neural network with synthetic dataset for transportation signs. In *2021 36th International Conference on Image and Vision Computing New Zealand (IVCNZ)* (**Accepted**)
- **Le, H.**, Nguyen, M., Yan, W. Q., Nguyen, H. (2021). Augmented Reality and Machine Learning Incorporation Using YOLOv3 and ARKit. *Applied Sciences, 11*(13), 6006. https://doi.org/10.3390/app11136006 (**Published**)
- **Le, H.**, Nguyen, M., Yan, W. Q. (2020, November). Machine Learning

with Synthetic Data–a New Way to Learn and Classify the Pictorial Augmented Reality Markers in Real-Time. In *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)* (pp. 1-6). IEEE. https://doi.org/10.1109/IVCNZ51579.2020.9290606 (**Published**)

- **Le, H.**, Nguyen, M., Nguyen, Q., Nguyen, H., Yan, W. Q. (2020, October). Automatic Data Generation for Deep Learning Model Training of Image Classification used for Augmented Reality on Pre-school Books. In *2020 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)* (pp. 1-5). IEEE. https://doi.org/10.1109/MAPR49794.2020.9237760 (**Published**)

Related research project:

- The international collaborative computer vision and artificial intelligence related project between Auckland University of Technology (New Zealand) and Pontificia Universidad Católica de Valparaíso (Chile), Jan. 2021

3. **Investigation of the detectability performance of different Augmented Reality markers without modifying the original figure contents**. This investigation presents a new method for incorporating Machine Learning to detect and track Augmented Reality applications' marker targets using deep neural networks. The synthetic dataset described in the previous contribution was used as the training dataset for the deep neural network model. The YOLOv3 model (Redmon & Farhadi, 2018) is used as the main object detection model and ARKit (Linowes & Babilinski, 2017) as the primary software tool for developing the application prototype. The proposed method achieved an 80% accuracy rate with an average of 60 frames per second for real-time detection on a mobile device. The results indicated that the detection process is effective in poor lighting conditions with

an acceptable detection accuracy rate. This means that a synthetic dataset can produce a similar result for Augmented Reality marker detection without modifying the original content.

Related publications:

- **Le, H.**, Nguyen, M., Yan, W. Q., Nguyen, H. (2021). Augmented Reality and Machine Learning Incorporation Using YOLOv3 and ARKit. *Applied Sciences, 11*(13), 6006. https://doi.org/10.3390/app11136006 (**Published**)

4. **Investigation of a low-cost platform that could immerse the Augmented Reality experiences to enhance learning efforts**. Lacking teaching materials is one of the common issues of the education system, especially in developing countries. The teachers attempt to give as much theoretical knowledge as possible to students while forgetting to train them with practical activities and ways of self-thinking. This way of teaching has been producing the generations of students with theory rather than practical. Augmented Reality technology can overlay virtual information onto educational textbooks, making them more attractive, motivating students to learn. An online learning platform is proposed that uses the advantages of Augmented Reality to enhance students' imagination. A redesigned transparent QR code sticker will be added to the existing education book figure. The students then can use their mobile devices to scan the QR code that contains an encrypted URL which allows them to render 3D graphics or animations virtually. That virtual information will allow the students to learn and understand the teaching theory much more intuitively. Moreover, the proposed method is implemented as an online platform; thus, no specific software installation is needed. It provides a stress-free, low-cost, portable, and promising solution to be used in school textbooks of all grades, to enhance the teaching and learning experiences.

Related publications:

- **Le, H.**, Nguyen, M. (2020). An Online Platform for Enhancing Learning Experiences with Web-Based Augmented Reality and Pictorial Bar Code. In *Augmented Reality in Education* (pp. 45-57). Springer, Cham. https://doi.org/10.1007/978-3-030-42156-4_3 (**Published**)
- **Le, H.**, Nguyen, M., Yan, W. Q. (2019, November). A Web-Based Augmented Reality Approach to Instantly View and Display 4D Medical Images. In *ACPR (2)* (pp. 691-704). https://doi.org/10.1007/978-3-030-41299-9_54 (**Published**)
- Nguyen, M., Lai, M. P., **Le, H.**, Yan, W. Q. (2019, November). A web-based augmented reality plat-form using pictorial QR code for educational purposes and beyond. In *25th ACM symposium on virtual reality software and technology* (pp. 1-2). https://doi.org/10.1145/3359996.3364793 (**Published**)
- Wang, I., Nguyen, M., **Le, H.**, Yan, W., Hooper, S. (2018, November). Enhancing visualisation of anatomical presentation and education using marker-based augmented reality technology on web-based platform. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)* (pp. 1-6). IEEE. https://doi.org/10.1109/AVSS.2018.8639147 (**Published**)

1.5 Thesis structure

This thesis consists of seven Chapters. The first Chapter, which is the current Chapter, introduces the thesis idea and answers the question about why this thesis project is needed. **Chapter 2** describes the conceptual background of Augmented Reality, including its history, application domains and demands for the future. The differences between Augmented Reality and other technologies such as Virtual Reality and Mixed Reality are also presented. Following in **Chapter 3** introduces different Augmented Reality marker-based techniques and the mathematical framework or image processing techniques used to encrypt and decrypt information. **Chapter 4** primarily focuses on Machine Learning and how it inspires the goal of this thesis. **Chapter 5** then describes the methods to answer the research questions, which are stated in **Chapter 1**. Moreover, it includes the synthetic data generation techniques, deep learning model used, and software implementation to merge Machine Learning and Augmented Reality. The experimental results and the iOS prototype are presented and evaluated in **Chapter 6**. Finally, the **Chapter 7** concludes the thesis and its limitations and future work.

Chapter 2

Background: The Rise of Augmented Reality

Parts of this Chapter have been published in ***paper 2****,* ***paper 4****,* ***paper 5****,* ***paper 7****,* ***paper 8****,* ***paper 9****,* ***paper 10****, and* ***paper 11*** *listed in the publication list*

Have you ever thought that the line between your imagination and the natural world does not exist? It is possible with Augmented Reality and easily accessible by most modern information devices (Figure 2.1). Augmented Reality is a relatively new technology that can overlay virtual digital information onto a physical environment (Craig, 2013). It is a perfect solution for visualising events that are impossible or impractical to see. However, this does not mean it is impossible to make further contributions in this exciting field. This Chapter reviews the literature on defining Augmented Reality, its historical background and related information.

Figure 2.1: Augmented Reality applications are widely used around the world in different domains such as: games, navigation, and education.

2.1 What is Augmented Reality?

Augmented Reality is a computer vision field where the physical environment is immersed and overlayed with computer-generated information to create an interactive space. Its main principle is to augment parts of natural objects with virtual information in real time (Carmigniani & Furht, 2011). Milgram and Kishino stated that "display of an otherwise real environment is augmented by means of virtual (computer graphic) objects" (Milgram & Kishino, 1994). Another meaning is that the Augmented Reality aims to present to the users the virtual contents and keep them staying in the real world simultaneously. In general, it is defined as a system that includes the following characteristics (Azuma, 1997):

- Capability to combine the natural and the virtual world.
- Presents the natural and virtual interactable environment in real-time.
- Ability to view virtual information in three-dimensional spaces.

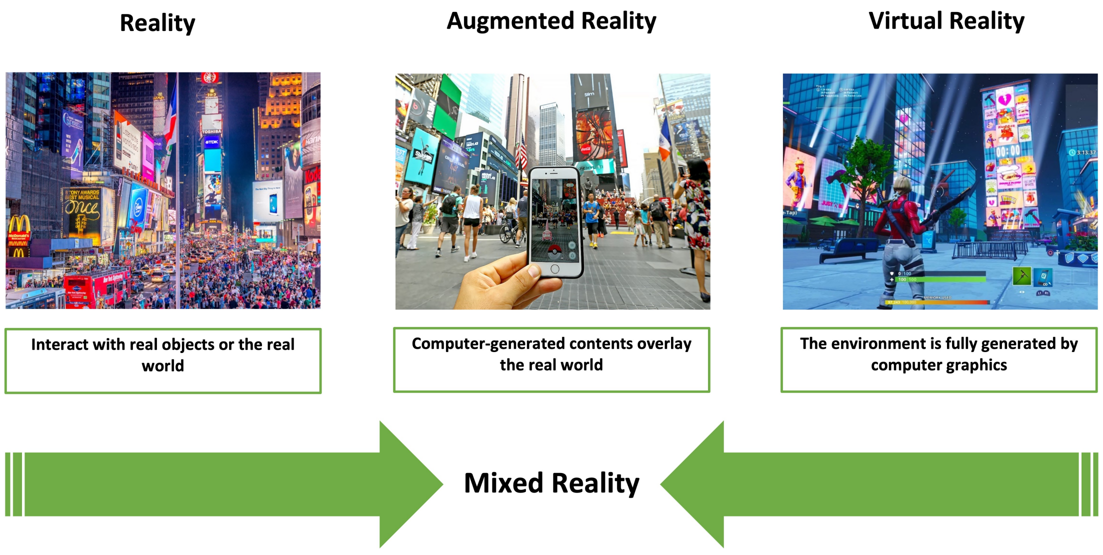
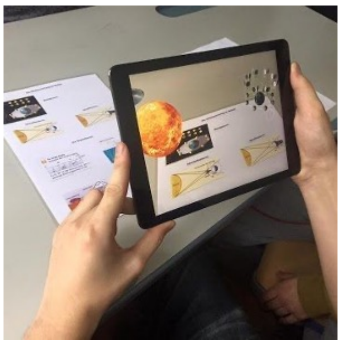

Reality
Augmented Reality
Virtual Reality
Interact with real objects or the real world
Computer-generated contents overlay the real world
The environment is fully generated by computer graphics
Mixed Reality

Figure 2.2: Milgram and Kishino’s Mixed Reality Continuum.

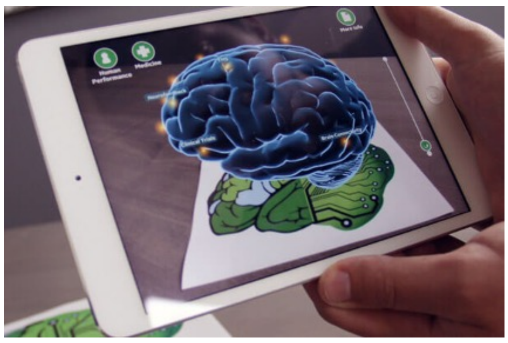
Virtual Reality and Mixed Reality are mentioned most of the time but how they are related to Augmented Reality? Figure 2.2 presents the Mixed Reality continuum that shows the Mixed Reality as a convergence of the virtual world and the natural environment along a digital information continuum. The Virtual Reality environment is entirely generated by computer graphics, which disconnects users from the real world and transfer them to an artificial digital environment. These immersive experiences are usually applied in training, education, and video games. Augmented Reality lies between Reality or the natural world and Virtual Reality to integrate virtual information into the actual physical world’s live view. By combining the physical world and the additional digital information, Augmented Reality has successfully created a new experience to allow users to interact and gain knowledge much more efficiently (Fisher & Baird, 2006; Narzt et al., 2006). Developers use different solutions and tools to implement Augmented Reality experiences but, in general, the Augmented Reality related applications are usually designed in one of two ways:

1 - Augmented Reality marker-based is the most commonly applied method that uses a physical object as a trigger. The trigger needs to be detected by the device camera; it then decodes the trigger's input confidential information and displays the corresponding virtual information. In these applications, the users will use a hard-copy marker as the trigger and view the overlay as the computer graphics generated. This method is usually suitable for indoor use such as education or product instruction. An example of this is shown in Figure 2.3 in which the images in a textbook can be used as the Augmented Reality markers to display virtual content, such as 3D models used in geography application (Figure 2.3a) or in the biology application (Figure 2.3b). More details on the Augmented Reality marker-based method is discussed in Chapter 3.

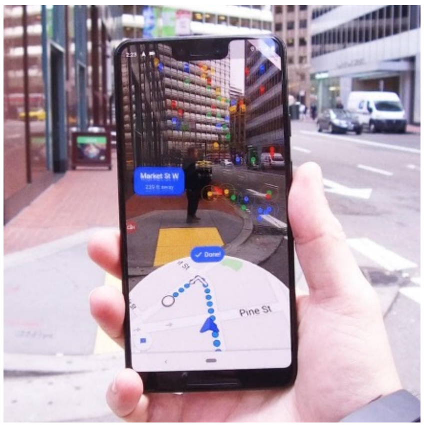
2 - Markerless Augmented Reality is different from the marker-based Augmented Reality application since it does not require the physical marker. The markerless Augmented Reality uses non-physical trigger tools like Global Positioning System (GPS), on-device compasses or the internet to recognise the user's location. The system updates the Augmented Reality presentation whenever the users move or their device changes orientation. For example in Figure 2.3, the users can use built-in GPS and compass functions to display directions to a particular location on a street (Figure 2.3c) or display a game character (Figure 2.3d).



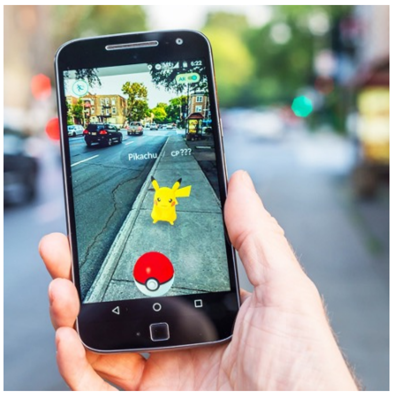
(a) Geography application [1]



(b) Biology application [2]



(c) Augmented Reality Google map application [3]



(d) Pokemon Go game [4]

Figure 2.3: Augmented Reality applications can be either marker-based (a and d) using a hard-copy printout as the trigger or markerless (b and c) using GPS or other location-tracking tools as the trigger.

2.2 How does it work?

A basic Augmented Reality system will consist of three main layers, as shown in Figure 2.4.

[1] Retrieved from https://i.ytimg.com/vi/Qw7HJPol8ZQ/maxresdefault.jpg

[2] Retrieved from https://trainingindustry.com/content/uploads/2017/09/12.7.15-Health-Care-Gamification.jpg

[3] Retrieved from https://geektech.me/wp-content/uploads/2020/10/ed3e8d559e2da1d45e03f5441c985ab5.jpeg

[4] Retrieved from http://www.technokraft.co/assets/img/resize-ar.jpg

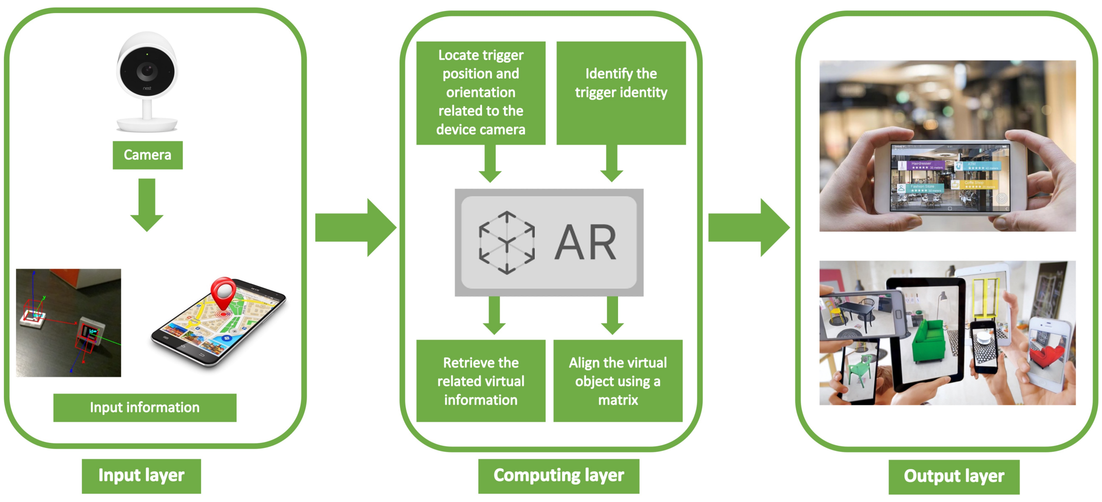

Camera
Input information
Input layer
Locate trigger position and orientation related to the device camera
Identify the trigger identity
AR
Retrieve the related virtual information
Align the virtual object using a matrix
Computing layer
Output layer

Figure 2.4: Augmented Reality systems consist of three different layers: input layer, computing layer, and output layer.

1 - Input layer is where the device camera captures the natural scene and other properties such as GPS information if a markerless approach is being used. This layer also records the object distance and orientations related to the device camera. The device needs to understand the environment around the user based on the content captured on the camera feed. This helpful information allows the application to present the digital content relevant to what the user is looking at.

2 - Computing layer is used to process and track the target information from the input layer. The system usually uses template matching (Brunelli, 2009) if it is a marker-based application or Simultaneous Localisation and Mapping (SLAM) (Mur-Artal, Montiel & Tardos, 2015; Mur-Artal & Tardós, 2017) if a markerless approach is being used to distinguish the target identity. The template matching technique tracks potential 2D images with unique properties, such as black borders, after quantising the image to a specified resolution matrix (Kato & Billinghurst, 1999). More details on marker-based detection and tracking is presented in Section 3.1. Markerless applications use a built-in sensor and SLAM to remove natural features and textures and reach real-time performance with sufficient accuracy. However, this method usually requires a high volume of data flow over the internet; hence, it is not excellent for low-cost and un-professional users. The system then gets the digital content related to the target's identity ready for rendering in the output layer. To instruct the output layer on rendering the virtual contents correctly, Augmented Reality uses a six degree-of-freedom matrix (6DOF) (Van Krevelen & Poelman, 2007), consisting of three in translation and three in orientation. This unique characteristic makes it different from Virtual Reality, where three degree-of-freedom (3DOF) are used (K.-M. Lee & Shah, 1988). The 6DOF matrix includes three perpendicular axes (x,y,z) for the positions combined with orientation changes via rotation (pitch, yaw, and roll). At the same time, the 3DOF only tracks the position changes (Figure 2.5).

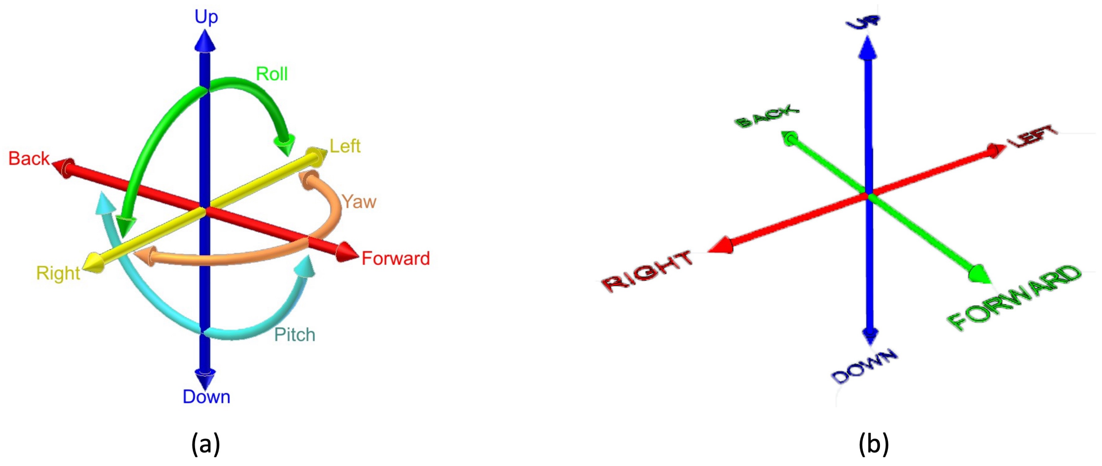


Figure 2.5: Augmented Reality uses six degree-of-freedom (a) to present the changes in the virtual content's position and orientation, whereas virtual reality uses three degree-of-freedom (b).

3 - Output layer renders the virtual content at the location in the natural scene. However, making this step fast and realistic is a challenging task. Augmented Reality systems consider computer vision as the inverse rendering, which means that computer vision recognises and understands 3D objects from their 2D images. The system then can present the 3D content that is rendered onto the 2D display screen. This transformation is called camera pose estimation (Quan & Lan, 1999) which is described in more detail in Section 3.2. The generated content's shapes and orientations are changed when the device or target situation changes to give the users a full view of the 3D experience.

2.3 The History of Augmented Reality

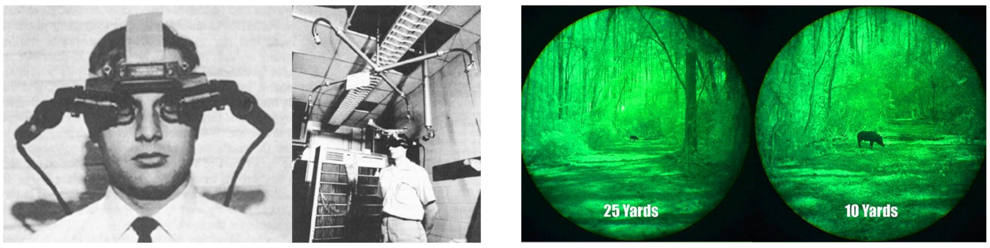


Figure 2.6: The first HMD prototype was introduced by Ivan Sutherland's research team (a) [1]. The night vision device introduced by the US Army during the Vietnam war allows viewing targets in low light and range estimation (b).

The Augmented Reality idea started in the 1960s when the first workable prototype was introduced by Ivan Sutherland and his colleagues (Sutherland, 1968), as shown in Figure 2.6a. This prototype is one of the first head-mount display (HMD) that allows users to view 3D computer-generated graphics via its display optics. Later, during the peak of the Vietnam War, the US Army introduced a night vision device (GEN 1,2,3 - NVD) (Braybrook, 1998). The GEN system was designed to mount on weapons allowing soldiers to view targets in levels of light approaching total darkness together with range estimation (as shown in Figure 2.6b). While Virtual Reality became more popular with investments during the 1970s and 1980s, Augmented Reality was forgotten due to hardware limitations and the lack of potential application ideas. However, people did not need to wait for so long for Augmented Reality to reappear again due to the significant growth of technology in the early 1990s (Caudell & Mizell, 1992; Bajura,

[1]Retrieved from https://glassdevelopment.files.wordpress.com/2014/04/cascosutherland.jpg

Fuchs & Ohbuchi, 1992). This is the first time Boeing researcher Tom Caudell used the word "Augmented Reality" in his research on mounting cables in airplanes (Janin, Mizell & Caudell, 1993). In the late 1990s, the Columbia University Computer Science research group demonstrated the prototype of an interaction wearable device (S. Feiner, MacIntyre, Hollerer & Webster, 1997). Since the expansion of the smartphone market in the beginning of the 21st century, Augmented Reality concepts have been gaining more public attention, increasing the number of supported technologies and research. In 2016, Goldman Sachs' annual investment banking report predicted that Augmented Reality revenue would reach 80 billion USD in 2025 (Figure 2.7) [1].

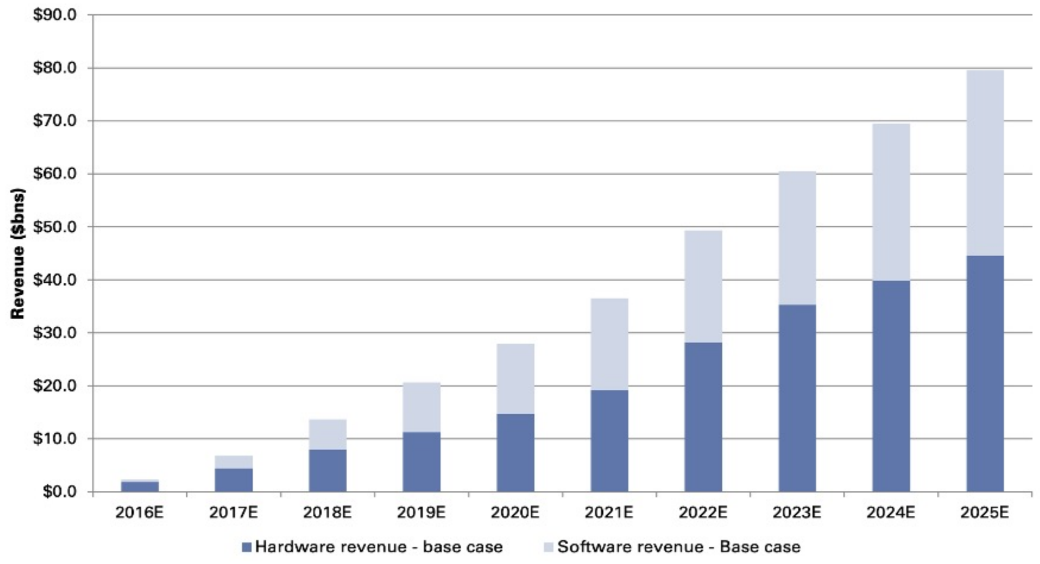
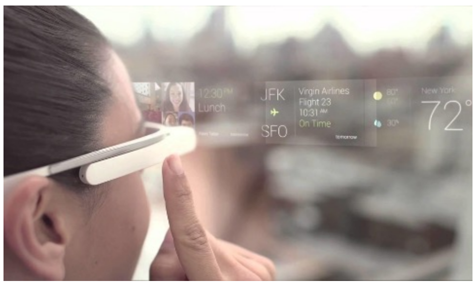


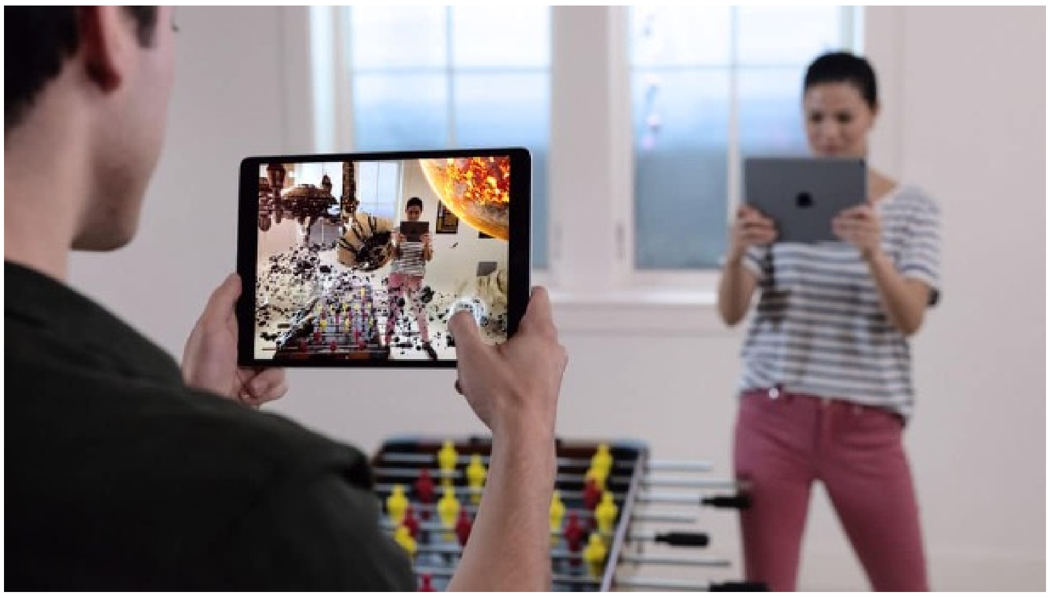
Figure 2.7: The Goldman Sachs annual report on Augmented Reality future estimated revenue in US dollars.

[1]Heather Bellini, M. S. M. S. S. A. D. T., Wei Chen. (2016). Profiles in innovation virtual and augmented reality. Retrieved from http://www.goldmansachs.com/our-thinking/pages/technology-driving-innovation-folder/virtual-and-augmented-reality/report.pdf



(a) Google Glass [1]

(b) HoloLens [2]



(c) Game developed with Apple's ARKit [3]

Figure 2.8: The devices allow users to experience Augmented Reality (a) or interact with the virtual world (b). The software development kit helps developers implement Augmented Reality-related applications on mobile devices quicker and easier (c).

In the last ten years, many major technology companies have made the significant investments in both software and hardware for this high-potential profit industry. Google is one of the first competitors to introduce to the public their own designed of Augmented Reality equipment called "Google Glass" (Figure 2.8a) (Rauschnabel, Brem & Ro, 2015). Google Glass is intended to be a mini wearable HMD that allows the users

[1]Retrieved from http://nanoday.com/blog_cover_photo/myglass.png

[2]Retrieved from https://cc-prod.scene7.com/is/image/CCProdAuthor/augmented-reality_P1_900x420?$pjpeg$jpegSize=200wid=900

[3]Retrieved from https://www.techrepublic.com/article/apple-ios-12-cheat-sheet/

to experience Augmented Reality via the glass optics. In 2016, Microsoft announced HoloLens (Furlan, 2016) that enables the users to experience Augmented Reality and interact with the virtual environment with their own hands simultaneously (Figure 2.8b). Apple went one step further in 2017 by re-configuring their iPhone processing chip [2] and introducing the software development kit (SDK) called "ARKit" to enable Augmented Reality experiences on mobile devices (Figure 2.8c).

2.4 Augmented Reality Applications

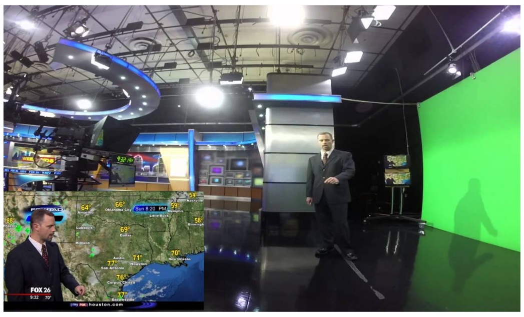
Augmented Reality can be found as the supported technologies in many different application domains such as advertising, navigation, entertainment and education (K. Lee, 2012). It is becoming an irreplaceable technology of daily lives, and it could be the future method of how humans collect, process, and interact with data (S. K. Feiner, 2002).

2.4.1 Broadcasting and Entertainment

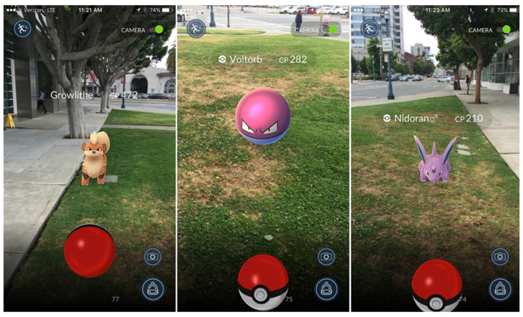
Each of us has watched the weather forecast on TV at least once, where the reporters are standing in front of changing weather backgrounds. However, there is no dynamic weather background in the TV studio. The reporters always stand in front of a single colour background (which is usually green, as shown in Figure 2.9a) while the images of weather are generated by digital graphics, which are sometimes called chroma-keying techniques. The green background acts as the marker which can be overlayed with different pictures or videos. This technique allows us to re-use the television station for other TV programs and reduces setup time. An example of an Augmented Reality entertainment application is Pokémon GO, launched in 2016 (Figure 2.9b) (Serino,

[2]Caughill, P. (2017). Here's why apple's custom gpu and a11 bionic chip are utterly revolutionary. Retrieved from https://futurism.com/heres-why-apples-custom-gpu-and-a11-bionic-chip-are-utterly-revolutionary/

Cordrey, McLaughlin & Milanaik, 2016). The game uses a markerless-based technique to locate virtual Pokémon characters in natural scenes and simultaneously interact with other nearby players.



(a)



(b)

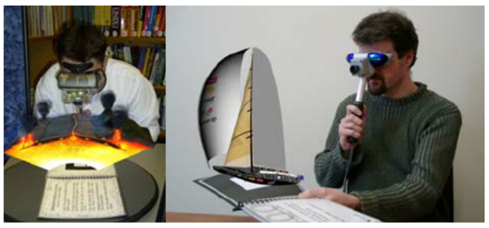
Figure 2.9: Examples of Augmented Reality in the broadcasting industry (a) and mobile gaming (b) [1].

2.4.2 Education

In recent years, the demand for Augmented Reality applications in the education sector has increased. Educators and students believe that Augmented Reality-related technology can help them improve the teaching and learning process. People usually struggle with the imagination of 3D objects when most graphical educational content presents them as 2D images from one angle. Augmented Reality can visualise spatial relationships as virtual 3D views without any extra physical teaching tools (Chang, Wu & Hsu, 2013; Billinghurst & Duenser, 2012). Magic Book was one of the first marker-based applications for educational books developed by Mark Billinghurst at Hiroshima City University in 2001 (Figure 2.10a) (Billinghurst, Kato & Poupyrev, 2001). At the same time, Rainer Malaka and their team designed a storytelling Augmented Reality-based application to allow visitors to view historic site ruins virtually at the exact location

[1]Retrieved from https://thewebappmarket.com/wp-content/uploads/2020/08/AR-VR-Technology-1.png

they were (Figure 2.10b) (Malaka, Schneider & Kretschmer, 2004; Kretschmer et al., 2001). By viewing virtually reconstructed landmarks, visitors could understand more about the site's history. This idea allows the users to learn the past and allows them to live and breathe with the history.


(a)

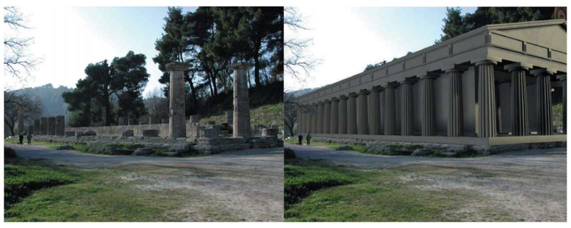

(b)

Figure 2.10: Examples of Augmented Reality applications in education include a marker-based educational book (a) and a historic site reconstruction (b).

2.4.3 Navigation

Navigation is currently the most common commercial use of Augmented Reality such as the navigation and reverse system in vehicles (Figure 2.11a). This is an example of markerless Augmented Reality using GPS data to provide drivers with useful information such as vehicle speed, virtual road lanes and virtual directions via the head-up display (HUD) located in the front of the vehicle. The HUD allows the driver to keep their eyes on driving direction or navigate the vehicle easier while reversing to minimise the chance of an accident. Some restaurant suggestion applications like Yelp (Figure 2.11b) embed an Augmented Reality function to provide users the most convenient method to find somewhere to eat, drink or shop via virtual directions in real-time (Kipper & Rampolla, 2013).

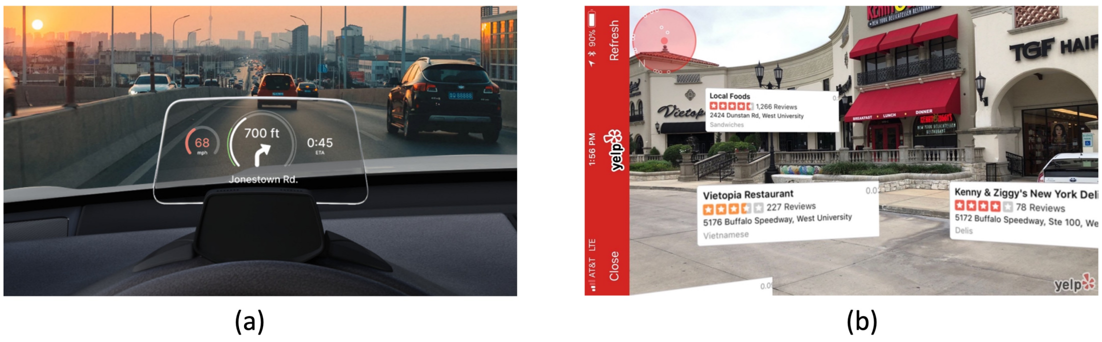

68
700 ft
0:45
Jonestown Rd.
(a)
Local Foods
1,266 Reviews
2424 Dunstan Rd, West University
Vietopia Restaurant
227 Reviews
5176 Buffalo Speedway, West University
Vietnamese
Kenny & Ziggy's New York Deli
78 Reviews
5172 Buffalo Speedway, Ste 100
Refresh
Close
yelp
(b)

Figure 2.11: Augmented Reality is used as a supported technology in automobile navigation systems (a) [1] or in applications for shopping store recommendation (b) [2]

2.4.4 Military

Augmented Reality research in the military started almost simultaneously as the first workable prototype introduced by Sutherland's team (Sutherland, 1968). However, most of their research information remained forbidden due to the national security policy. The head-up display (HUD) now seen in modern cars was proposed by the US Navy in 1955 (Previc & Ercoline, 2004). This early HUD concept consisted of three main components: (1) a projector unit, (2) a combiner, and (3) a digital-generated video display. This design did not require the fighter pilot focusing on the outside view after looking at the nearer instruments (Figure 2.12a). Later in the 1980s, the US Air Force launched the Integrated Helmet and Display Sighting System (IHADSS) for AH-64 Apache pilots (Hiatt, Rash, Harris & Gilberry, 2004). This gives the pilots a wider field of view and does not require them to turn their helicopter toward the direction of the target. The new system automatically locks the selected target using the information captured by the symbology monocular display located in the front of the pilot's eye (Figure 2.12b). Hence, increasing the pilot's survival chance and combat performance.

[1] Retrieved from https://miro.medium.com/max/1400/1*1D9xe8JKGHMNCPTUoniPBQ.jpeg
[2] Retrieved from https://s.hdnux.com/photos/67/30/40/14518276/3/1200x0.jpg

Figure 2.12: Augmented Reality is used as the military combat equipment such as in the head-up display on fighter jets (a) [1] or in integrated target engagement helmets (b) [2]

2.5 Summary

This Chapter presents the literature to understand Augmented Reality, where it came from, and how it got into today's stage. It clearly showed that Augmented Reality is a very high-demand and exciting field for both kinds of research and commercialisation. There are different ways to implement Augmented Reality technology, such as markerless and marker-based. However, due to the goals of this thesis, the marker-based technique is only considered. The next Chapter will describe different types of Augmented Reality markers and the decoding principles behind them.

[1]Retrieved from https://i.pinimg.com/736x/71/e4/86/71e48638792c55f36f8cb1967affd797–fighter-aircraft-fighter-jets.jpg

[2]Retrieved from https://i.pinimg.com/564x/c6/82/62/c68262156a85bd3eb6b717313c6a13f3.jpg

Chapter 3

Theory and Practice of Existing and Our Newly-Designed Augmented Reality Markers

Parts of this Chapter have been published in ***paper 2****,* ***paper 4****,* ***paper 5****,* ***paper 7****,* ***paper 8****,* ***paper 9****,* ***paper 10****,* ***paper 11****,* ***paper 13****, and* ***paper 14*** *listed in the publication list*

Markers are the critical component of marker-based Augmented Reality applications. There are many different types of Augmented Reality markers designed for various purposes and development-supported tools. However, all of them can be organised in one of three main categories: (1) barcode markers, (2) template markers, and (3) barcode pictorial markers. The rest of Chapter will focus on the following discussions:

- The significant steps to encrypt and decrypt the markers hidden information *(Section 3.1)*.
- The principle of pose estimation that is used to display the virtual 3D objects *(Section 3.2)*.
- The differences between some popular Augmented Reality markers ***(Section 3.3)***.
- The advantages of newly-designed Augmented Reality markers over the original markers ***(Section 3.4)***.

3.1 Marker detection procedure

The marker-based detection process includes several different steps to identify the hidden marker information. Firstly, the system needs to find the outlines of potential markers and determine their boundaries. Many image processing techniques could quickly do marker boundary detection. The system should also be capable of identifying the marker information within the confirmed boundary. In addition, there will be different algorithms that are used for the marker information identification based on the marker type. In theory, the marker-based detection process consists of the following two main steps: (1) Marker boundary detection, (2) Marker information identification

3.1.1 Marker boundary detection

The intensity image (grey-scale image) needs to be obtained first in the marker boundary detection. The system must convert the captured image to grey-scale format from RGB format using the formula as shown in the following Equation.

$$Y = 0.2126R + 0.7152G + 0.0722B \tag{3.1}$$

The system will then use the thresholding technique (Chowdhury & Little, 1995) to search for the potential marker from the binary image and the edge detection method (Canny, 1986) to identify the marker boundary. The thresholding technique usually uses the adaptive thresholding method to determine the illumination changes in the picture (Bradley & Roth, 2007), as shown in Figure 3.1. As long as the illumination changes in the image are identified, the system can classify which objects are most likely to be the marker area.

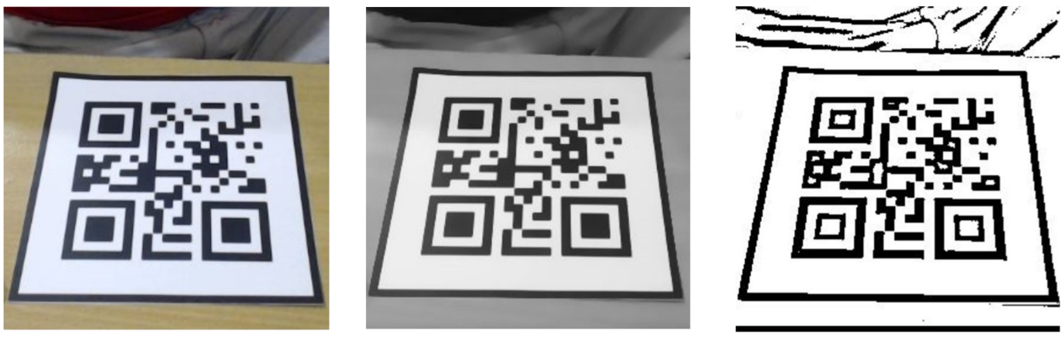

(a) (b) (c)

Figure 3.1: The adaptive thresholding image can be obtained (c) from the originally captured image (a) by converting it into a grey-scale image (b).

Canny edge detection is a widely used method to detect a wide range of edges in an image, as shown in Figure 3.2. The algorithm processes the result from the thresholding step to determine the figure boundary and this can be broken down into five smaller steps:

1. Use a Gaussian filter to remove high-frequency noise.
2. Compute the image intensity (Figure 3.2a).
3. Apply non-maximum suppression to remove "false" responses to edge detection.
4. Apply thresholding using a lower and upper boundary on the gradient values (Figure 3.2b).
5. Use hysteresis to track edges; the weak edges that are not connected to strong edges will be suppressed (Figure 3.2c).

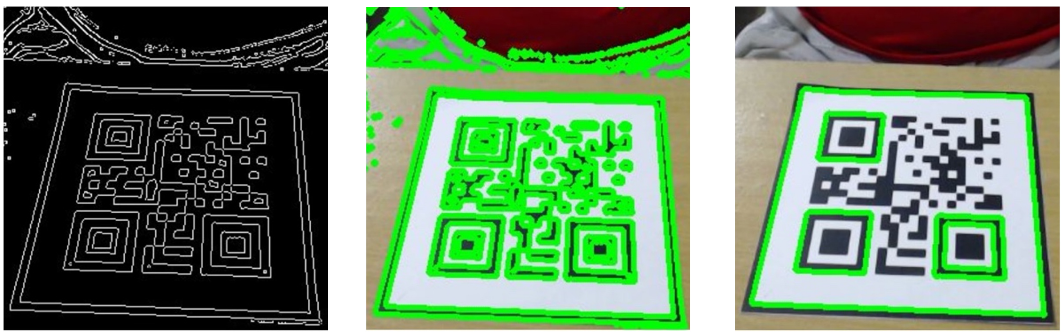

(a) (b) (c)

Figure 3.2: The edge detection process (a) uses a threshold image (Figure 3.1) with superimposed contours onto the initially captured image (b). It then produces the remaining detected edges after applying the noise removal algorithm (c).

3.1.2 Marker information identification

As described in the previous Sections, many different types of markers are available on the market, but they fall into two main categories: template markers and data markers. Each marker category provides various ways to store and encrypt information. The information will be encrypted using either the template matching technique (template marker) or the data decoding method (data marker). The difference between these data encryption methods will be described in detail in the following subsections.

Template matching

The template matching technique identifies the detected marker identity by comparing it with each sample image stored in the database (Brunelli, 2009). However, the detected marker's size, location and orientation are unknown, as the detected marker is not warped. The system then scales the detected marker to the exact size of the sample image and examines it in four different positions according to four possible orientations. The sample image that gives the Highest Similarity Value (HSV) is the correct marker. The orientation of the detected marker is also defined as the same as the position of the matching sample image. This orientation information can be used for future display purposes, such as the orientation of virtual objects. However, if the HSV is lower than the threshold, the detected marker is rejected. The HSV can be easily calculated based on either the Sum of Squared Differences (SSD) or cross-correlation (Lewis, 1995).

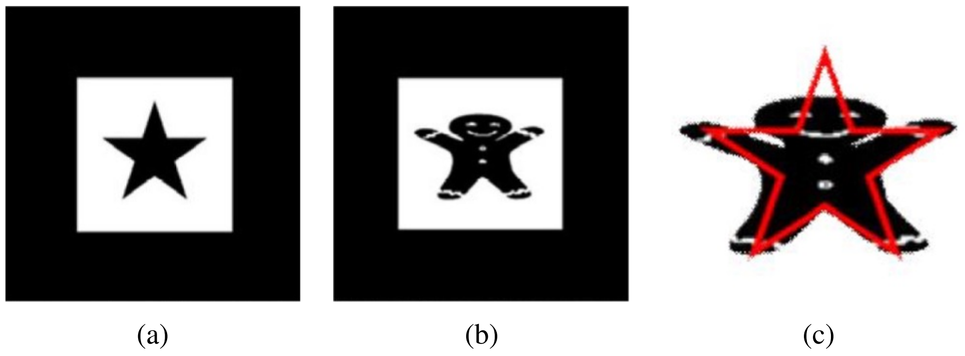


Figure 3.3: The example of the undesired similarity between markers where they look entirely different to human eyes (a) and (b), but the template matching process may confuse them as their presentation areas are almost overlapping (c).

As the system needs to match the detected marker against each of the dataset sample images four times, it is clear that more time is required for the marker identification. Therefore, it would be inefficient in practice if there was a large data set. Another disadvantage of template matching is that the detection process can produce a poor outcome due to the undesired similarity (Tikanmäki & Röning, 2011) of images despite looking entirely different to the human eye, as shown in Figure 3.3.

Data decoding

The data decoding method is generally used for data markers consisting of black and white data cells. Black colour represents as "1" in binary code and white is "0". The system uses this principle to obtain a series of binary values, which can be represented as the marker data. As the binary series are unique, this decoded binary number is the same as a marker ID or identity, as shown in Figure 3.4. This method gives an advantage over template matching in terms of processing time and undesired similarity of markers. Another advantage of data decoding is that it also provides error detection and correction capability besides encoding information. This feature cannot be used on template markers without altering the graphical area. The Hamming codes (Hamming, 1950) and Reed—Solomon codes (Reed & Solomon, 1960) are the two most used error detection and correction methods.

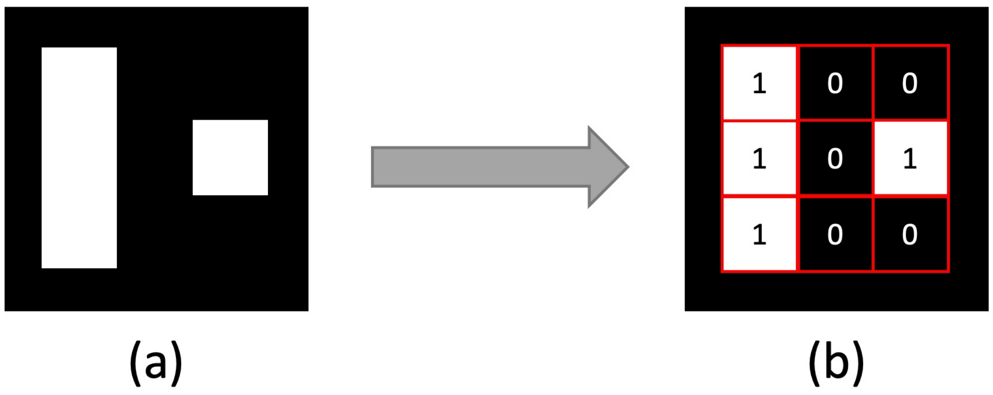


Figure 3.4: The data decoding technique will decode the cells of the original data code marker (a) to a binary (100101100) number (b) and then to a decimal (300) number.

The Hamming codes algorithm uses parity bits that determine whether the number

"1" in a particular binary string is even or odd. There are two types of bit parity: odd parity and even parity. The odd parity equals one when the number "1" is even and zero if otherwise. The even parity equals one when the number "1" is odd and zero if otherwise. The added parity bit to data can detect the error of one single bit in that binary string. One parity bit can only reveal one single-bit error of the entire binary series. Therefore, more than one parity bit is generally added for binary string error detection and correction. The binary string or data is usually divided into blocks (e.g., four bits), and each block is encoded separately. The Hamming (7,4) adds three additional parity bits to every block of four data bits in a binary string. It can detect all single-bit errors and correct any of them. However, there will not always be a single bit of error in practice. The Hamming (8,4) is an extended version of Hamming (7,4) suitable for both single error correction and double error detection. In other words, the more parity bits that are added, the more errors can be detected and corrected. Irving Stoy Reed and Gustave Solomon invented the Reed-Solomon codes in 1960. The codes operate on m-bit symbols, whereas the Hamming codes operate on the individual bit. The codes are defined as polynomials operating over finite fields. They are designed to detect and correct multiple symbol errors. The number of t check symbols will be added to the original data; the codes can detect up to t erroneous symbols or correct up to $\frac{t}{2}$ symbols. The Reed-Solomon codes are usually used in DVDs, CDs, satellite communications and complex barcodes, such as quick response code where a high degree of data reliability is required (Plank et al., 1997). Therefore, the Hamming codes are often preferred for error detection and correction in simple data markers.

3.2 Marker pose estimation

To render virtual information on a physical marker, the system needs to find the marker pose estimation. This step begins with determining the camera position related to the marker. The system then can use this information to blend the virtual information into the real-world environment (physical marker). It means that the Augmented Reality system needs to transfer the 3D coordinates of the virtual object in the real world to 2D coordinates and present them on the display screen. In most Augmented Reality systems, there are three different coordinate systems or transformations:

World coordinates (M_W): define the location of the trackable physical marker or virtual information that is rendered in the real-world scene. The world coordinate system is presented as X_w, Y_w, and Z_w in 3D space. This process is sometimes called pinhole camera transformation which will be described in Section 3.2.1.

Camera coordinates (M_C): define the position and orientation (pose) of the video camera that is currently in use to view the natural world scene. All the physical points of the natural world (including the virtual information) scene are defined relative to the camera in this transformation. The details of camera coordinates will be discussed in Section 3.2.2.

Display screen coordinates (M_S): define a projection from the real-world coordinates (3D) to the 2D coordinates which are used to render the pixels on the digital display device. However, to maintain the realism of the augmented scene, a high level of accuracy for all three geometric transformations (M_W, M_C, and M_S) is required.

3.2.1 Pinhole camera model

The pinhole camera model (Sturm, 2014) projects 3D points (3D coordinates of **P**) into the camera screen using a perspective transformation as shown in Figure 3.5. The *optical axis* is presented as the line through the centre of focus of the camera **O** and is perpendicular to the camera screen at point **C** (**Z** axis). The *focal length (f)* is the distance between the centre of focus of the camera and the camera screen. Firstly, the 3D coordinate of point **P** where **P** = $[U, V, W]$ on the camera screen needs to be projected. Then the 2D projection of point **P** is the intersection point between the camera screen and the line which goes through point **P** and the centre of focus of the camera; donated by **p'** = $[x, y]$. The values of x, y can be expressed by the following formulas:

$$x = f\frac{X}{Z} \tag{3.2}$$

$$y = f\frac{Y}{Z} \tag{3.3}$$

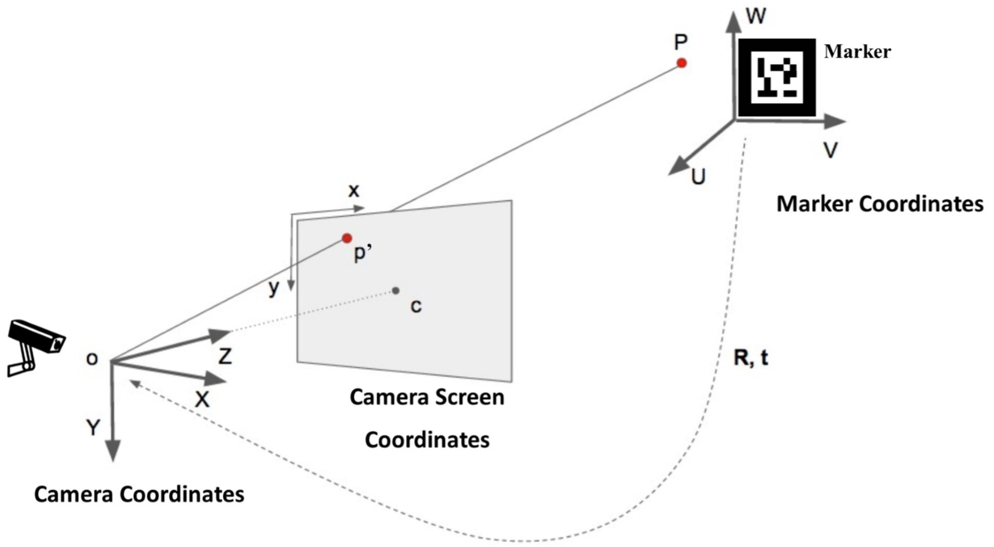


Figure 3.5: Relationship between the marker coordinates and the camera coordinates using the pinhole camera model.

3.2.2 Camera parameters

There are two different camera parameters used to define the relationships between the coordinate systems in most Augmented Reality applications: extrinsic camera parameters and intrinsic camera parameters. Figure 3.6 shows how the camera parameters are used within the virtual information rendering process.

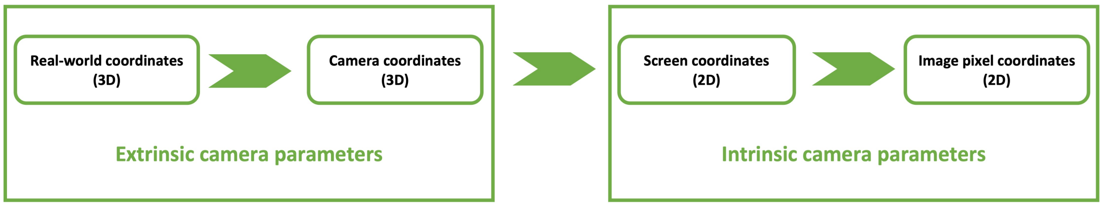

Real-world coordinates (3D)
Camera coordinates (3D)
Extrinsic camera parameters
Screen coordinates (2D)
Image pixel coordinates (2D)
Intrinsic camera parameters

Figure 3.6: The camera coordinates consist of two different types of camera parameters: extrinsic camera parameters and intrinsic camera parameters.

Extrinsic camera parameters

The extrinsic camera parameters identify the transformation between the unknown camera coordinate (M_C) and the world coordinate (M_W), including the coordinates of the rendered virtual information. The extrinsic camera parameters are external to the camera and may change with respect to the world frame. Augmented Reality technology uses six degrees of freedom (6DOF) (Van Krevelen & Poelman, 2007) for rendering and updating virtual information orientation and location. This means that the virtual information has only two kinds of transformation forms on a static camera: translation and rotation. The translation motion occurs when the camera is moved from its current location (X, Y, Z) to a new location (X', Y', Z') in 3D space, as shown in Figure 3.7a. The rotation motion is absorbed when the camera is rotated about the X, Y and Z axes. The camera rotation motion is often represented by using Euler angles (Diebel, 2006) (roll, pitch and yaw), or the direction of rotation angles (α, β, γ) as shown in Figure 3.7b.

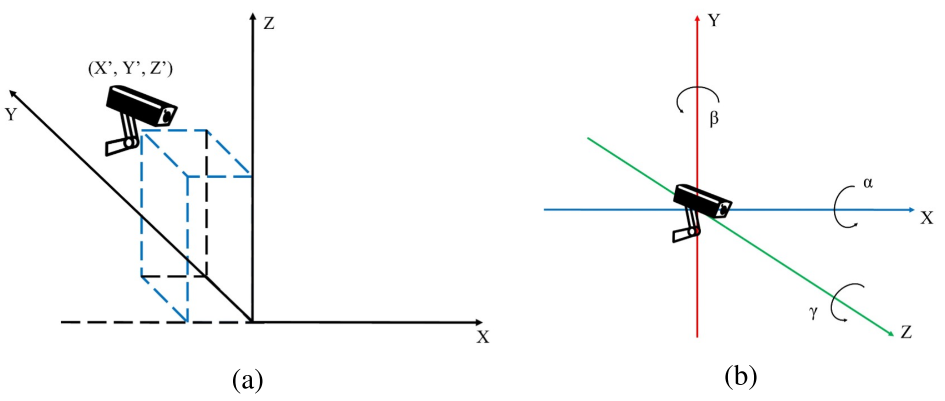


Figure 3.7: The orientation matrix can be expressed with rotation (b) angles (α, β, γ) around axes (X, Y, Z) and the new location (X', Y', Z') in 3D space is defined by translations (a) along the axis (X, Y, Z)
.

The translation motion occurs when the camera is moved from its current location (X, Y, Z) to a new place (X', Y', Z') in 3D space. It has three degrees of freedom and represented by vector t which can be calculated as in the followed Equation:

$$t = (X' - X, Y' - Y, Z' - Z) = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{3.4}$$

The rotation motion is absorbed when the camera rotates about the X, Y and Z axes. Camera rotation motion is often represented by using Euler angles (Diebel, 2006) (roll, pitch and yaw), a 3×3 matrix R or a direction of rotation and angle as shown in Equation below:

$$R = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} \tag{3.5}$$

or can be expressed in homogeneous coordinates:

$$T = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \tag{3.6}$$

Intrinsic camera parameters

The intrinsic camera parameters are internal and fixed to individual camera properties or setup. Most of the time, the intrinsic camera calibration matrix K (Hartley & Zisserman, 2003) needs to be calculated (Equation 3.7) first before starting the tracking process.

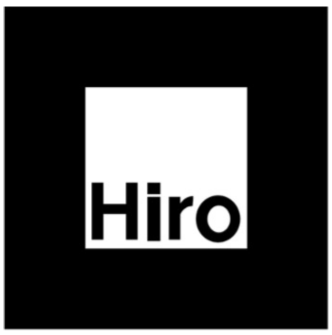
$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

Where f_x and f_y are the camera focal length in the x and y directions. The coordinates of image centre point C is donated by (c_x, c_y). s is the axis skew due to projected image distortion. However, in most modern cameras, pixels are often square and columns and rows are straight. Thus, the value of s can be discarded; thus, $s = 0$ and $f_x = f_y$.

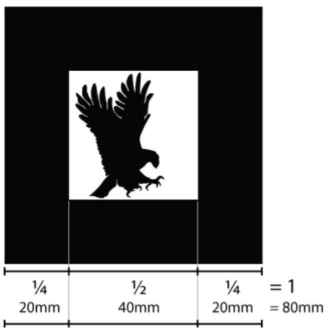
$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

3.3 Augmented Reality Markers

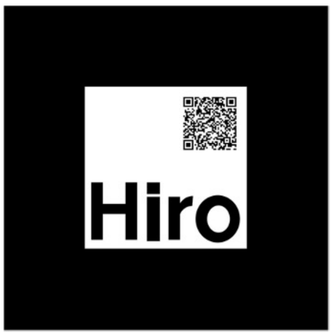
There are different types of Augmented Reality markers used for varying purposes. These markers provide various methods to encrypt and decrypt confidential information and have unique pros and cons. This section will demonstrate different examples of Augmented Reality markers with their unique encryption/decryption algorithms.



(a) Original marker



(b) Marker structure



(c) QR code embedded

Figure 3.8: There are many different Augmented Reality markers available in the market, such as: the original tag (a) could be embedded with a QR code to improve the detectability accuracy (b); ARToolkit marker designed with thick black border where the border area occupies over 60% of the marker content (c).

3.3.1 ARToolkit marker

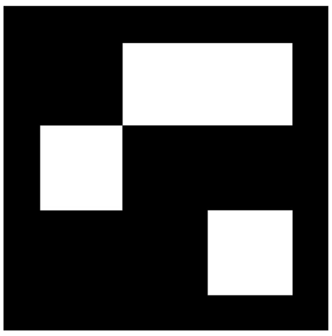
The ARToolkit marker was one of the first programmable markers and was designed by Hirokazu Kato (Kato, Billinghurst, Weghorst & Furness, 1999) in 1999 (Figure 3.8). This marker is the default optical input of the open-source computer tracking library, also referred to as ARToolkit. The design of the ARToolkit marker has few tricky rules that the marker designers must follow:

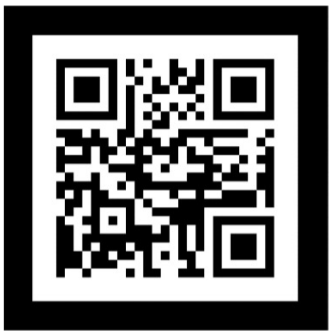
- The marker must be a square shape.
- The border must be a single colour in either black or white.
- The border thickness must be 25% of the marker edge length by default.
- The edge dimension of the inner graphical content must be 50% of the marker edge, and the colour must be significantly different from the border.

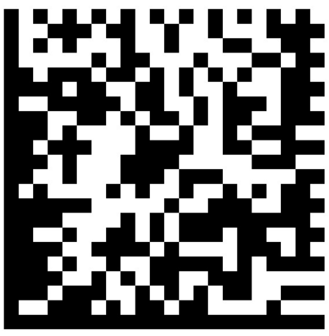
The marker uses a Quick Response (QR) code as an embedded corner image to increase the detection accuracy, as shown in Figure 3.8c. This marker design is compelling, as it can be identified in many different lighting conditions and is not too difficult to implement on digital devices. However, it conveys less meaning to users due to limitations of the displayable content area.

3.3.2 Data Marker

The data markers, also known as binary markers (Figure 3.9a), are usually designed as the black and white data cells inside thick borders. Each type of these binary markers has its own decoded binary string representing the black and white cell patterns to present the unique identity (Figure 3.9). The more encoded data, the larger the marker size; it means almost no limitation of the number of encrypted messages. The most significant advantage of this marker design is built-in error detection and correction, which is discussed in Section 3.1.2. There are many data markers, such as QR (Soon, 2008; Kato & Tan, 2007) (Figure 3.9b), data matrix (Stevenson, 2005) (Figure 3.9c), or PDF147 (Wang, 1993) (Figure 3.9d). Data matrix markers can hold up to 3,116 encoded characters with extensions and recover 30% of faulty content using Reed-Solomon codes. The QR code is the most famous member of the data marker family, introduced by the Japanese corporation Denso-Wave in 1994. The advantage of the QR code over other binary codes is that it can store non-alphabet characters such as Kanji, one of the essential scripts in Japanese writing. Ynjiun Wang invented the PDF417 code at Symbol Technology in 1991. It usually appears in a rectangle shape and is primarily used in transport or entertainment tickets. A single PDF417 code consists of four data bars that can hold up to 1850 alphanumeric characters or 1108 bytes of data. These data marker designs are easy to implement on any digital system, making them the most used recognition symbol globally. However, they mainly appear with uninteresting colours and provide non-meaningful information to users.


(a) Binary marker


(b) QR code marker


(c) Data matrix marker

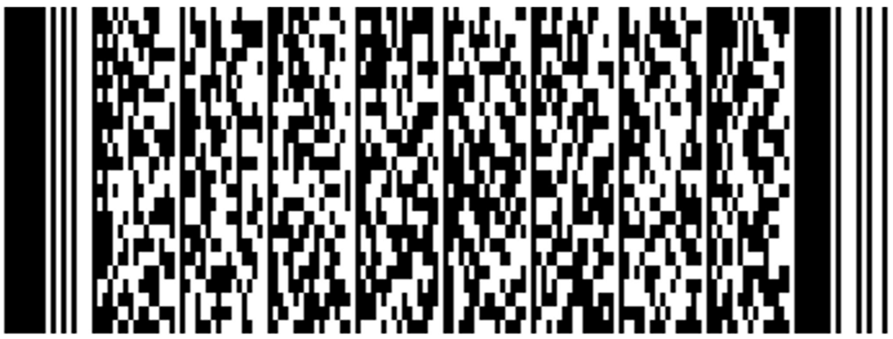
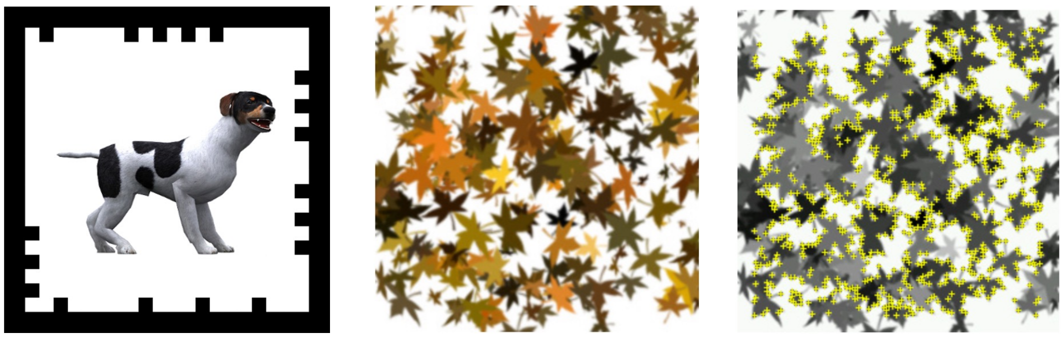

(d) PDF147 code

Figure 3.9: Data marker examples.

3.3.3 Vuforia marker

The Vuforia marker is one of the most well-known Augmented Reality marker types (Figure 3.10), designed by Qualcomm Technologies (*Vuforia*, 2021). Compared to data markers, Vuforia markers can provide more meaningful information to the users while keeping the image content almost unchanged. This design mainly uses the template matching techniques (as described in Section 3.1.2) to recognise the potential marker target in the real world. Users can generate Vuforia markers in two ways: (1) raw image or (2) frame marker. As shown in Figure 3.10a, the frame marker is similar to other barcodes or ARToolkit markers, which use the computer vision techniques to identify the potential marker target. The template marker (Figure 3.10b) does not require the black boundary and can be applied to any coloured figures such as photos, books, posters, etc. The system will retrieve the natural features available on each registered image target and use them as the unique identity of the image itself (Figure 3.10c).



(a) Frame marker (b) Original natural image (c) Image features

Figure 3.10: Vuforia marker designs give users the option to use the marker frame design as another version of the barcode tag (a) or register natural images as the template marker (b,c).

However, this design has a significant drawback when the markers are detected under different unexpected lighting conditions. When users' register their image targets, the images are usually prepared with ideal lighting conditions. However, when the system scans for the marker in the natural environment, different lighting conditions can lead to poor detection accuracy. Hence, Vuforia recommends the following features in image targets to help improve the detection accuracy:

- The marker should be designed with rich details with good contrast graphical content.
- The marker should not include repetitive patterns.

The Vuforia marker is always the top choice for Augmented Reality marker-based applications due to its usability. However, in practice, it is difficult to control natural factors such as lighting, noise, faulty graphical content, or similarity which could impact the accuracy of the final detection values.

3.4 Newly-designed Augmented Reality Markers

During the research of this thesis, several newly designed Augmented Reality markers have been developed to overcome the disadvantages of the aforementioned Augmented Reality markers in the previous Section. These markers are designed to keep the original pictorial information as much as possible and improve detection accuracy. In short, the newly proposed markers are:

1. Pictorial marker with hidden bar-code ***(Section 3.4.1)***.
2. Curtain style pictorial marker ***(Section 3.4.2)***.
3. Tile-based Quick Response code marker ***(Section 3.4.3)***.
4. Removable transparent Quick Response code marker ***(Section 3.4.4)***.
5. Red-green-blue Augmented Reality marker ***(Section 3.4.5)***.

3.4.1 Marker 1: Pictorial marker with hidden bar-code (PMBC)

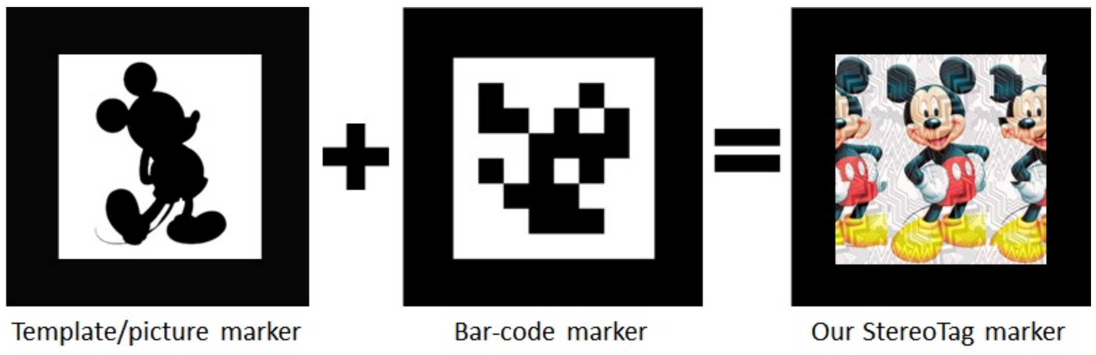
Pictorial marker with hidden bar-code (PMBC) uses the idea of stereogram to conceal a multi-level bar-code optically. The demo video of PMBC concealed code decryption process can be reachable via **`https://www.youtube.com/watch?v=fOwu318KYOw`**. The design is demonstrated in Figure 3.11 which presents some notable advantages over others:

- **Extensive range of data:** The multi-level bar-code can hold L^N different numbers, with L being the number of levels in each bar, and N, the number of bars.

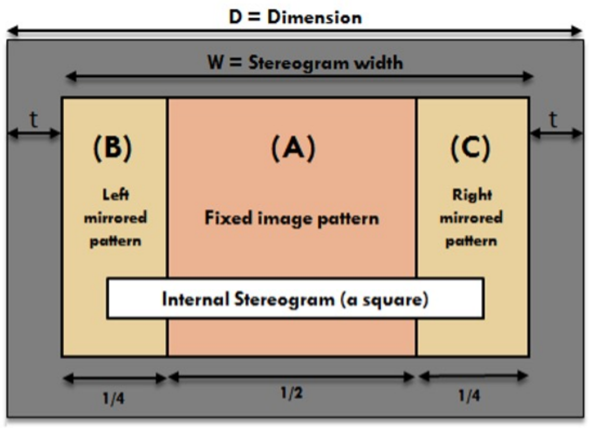
- **Virtually Pictorial:** The image inside each PMBC marker is made from meaningful illustrations rather than black bars, squares, or dots.

- **The flexibility of Pattern:** The decoded information is independent of image patterns; a broad range of images can be used to encode the same bar-code.

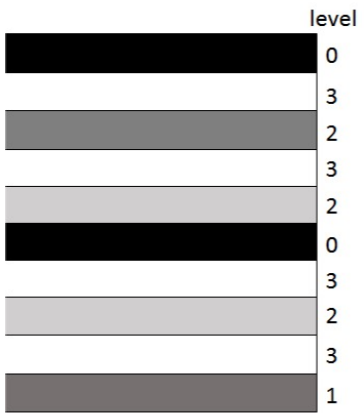
Each PMBC marker is a rectangle with a dimension $D = M \times N$ measured in pixels or millimeters; border thickness t is relatively small ($t = 4\%$ of D). The quadrilateral property of the rectangles can be used to detect their four straight lines and four corners; these are used for detecting the marker. The internal image is a stereogram (size $W \times H$) made of three regions. The central area is a fixed image (region A that fills up $\geq 50\%$ of the stereogram) and two repeated patterns on both sides of the region A (region B and region C with $\leq 25\%$ of the stereogram each). Hidden inside each stereogram is a bar-code with many horizontal bars of the same thickness. Each bar is coated with different grey levels between black and white; these levels represent different depth levels inside the stereogram. Figure 3.11c displays an example of 4-level binary bar-code with 10 horizontal bars. Each bar can hold four levels: 0, 1, 2, 3, corresponding to black, dark grey, light grey, and white. Thus, this barcode can store as many as $4^{10} = 1,048,576$ different numbers.



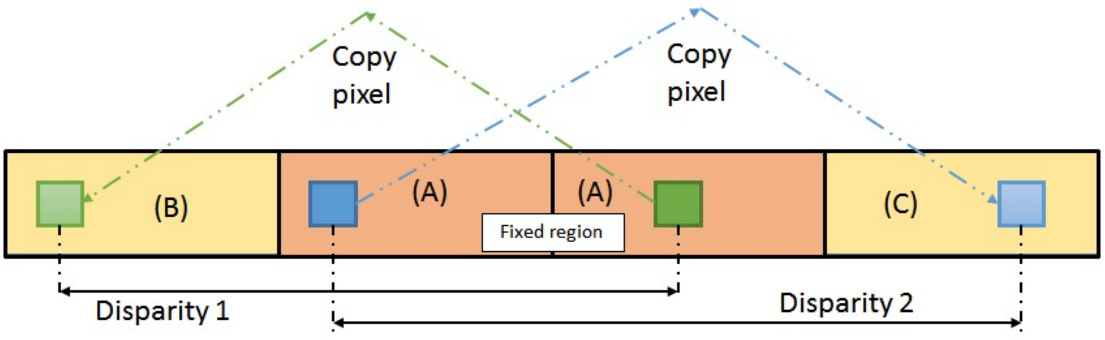
(a) Pictorial marker with hidden bar-code proposed idea.



(b) Marker design.



(c) Multi-level barcode.



(d) Pictorial marker with hidden bar-code information encrypted principle.

Figure 3.11: The design of Pictorial marker with hidden bar-code that optically hides a multi-level encrypted bar-code.

Figure 3.12 demonstrates the necessary steps of creating PMBC marker. As described, the PMBC marker has a black border to quickly and reliably detectable under various circumstances. In theory, the internal stereogram of the PMBC marker can encode any 1D bar-code such as Code11, Code 32, Code 49, Code 93, Code 128, EAN-8, and EAN-13.

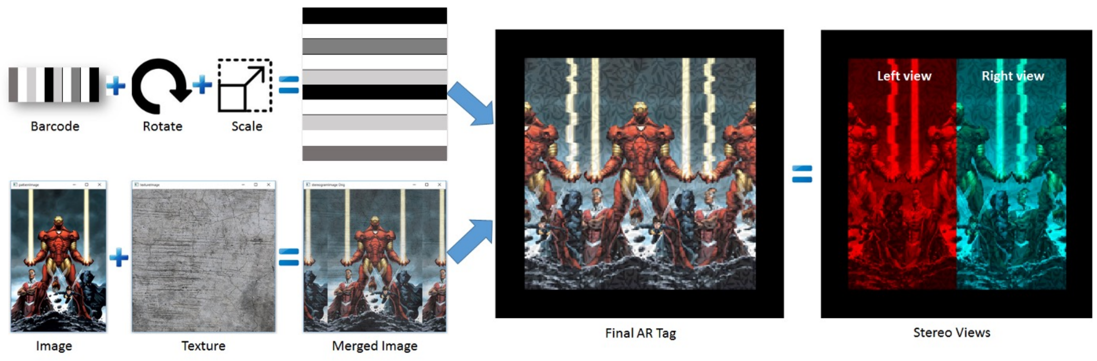


Figure 3.12: The proposed marker information encrypted process from a multi-level bar-code and a picture (Ironman). The middle 50% of the marker is a fixed illustration of the Ironman figure. The marker is also a side-by-side stereo image pair (red/cyan painted).

In order to decrypt the concealed information, we could use the internal stereogram image of the detected PMBC to rebuild the hidden multi-level bar-code. This step is equivalent to a stereo reconstruction process applied on two stereo images C_1 and C_2; C_1 is the left half of the stereogram and C_2 is the right half of the stereogram. The disparity levels (ranged between 0, d_{MAX}) are known from the width of the internal stereogram. The intensity-based stereo matching is a complex algorithm; they can be categorized into at least three families below:

- **Local matching algorithms:** Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), and Cross-correlation based block matching stereo algorithm (BMS) (Konolige, 1997).

- **Global stereo matching algorithms in 1D (one scan-line at a time):** Dynamic Programming Stereo (DPS) (Birchfield & Tomasi, 1999), Symmetric Dynamic Programming Stereo (SDPS) (Gimel'farb, 1999), and Belief Propagation - 1D version (1DBP) (Gong, 2011).

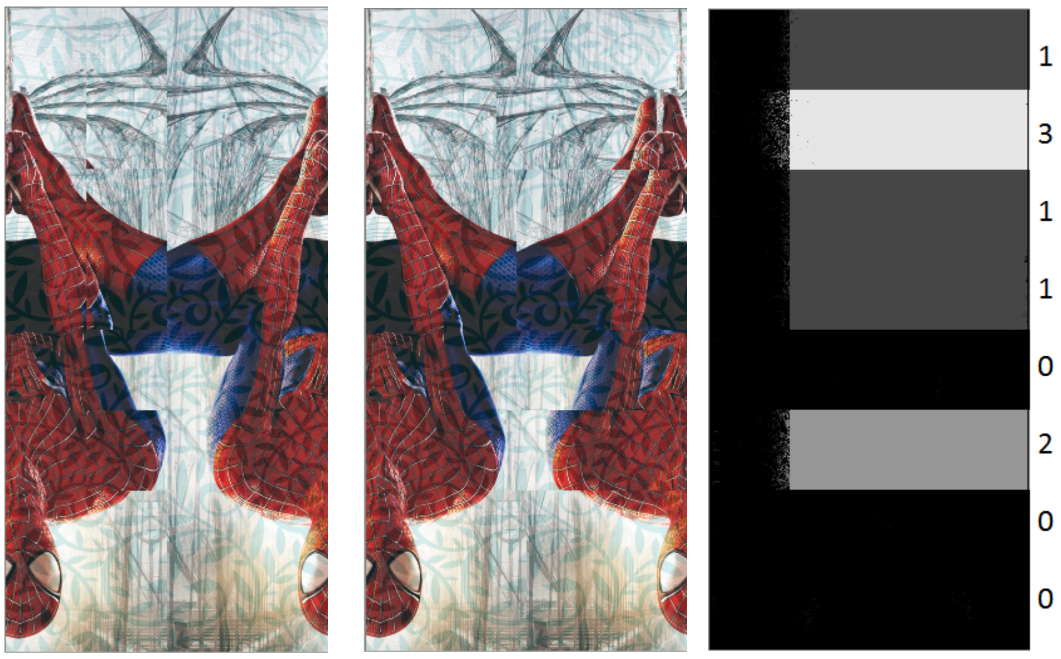
- **Global stereo matching algorithms in 2D:** Graph Cuts Stereo (GCS) (Kolmogorov & Zabih, 2002).

We use a technique that guides DPS using pre-computed BMS depth map, thus restricting and guiding the DPS search for optimal profile related to signals in 2D. Previously being evaluated in (Nguyen, Chan, Delmas & Gimel'farb, 2013), **both DPS and BMS are fast but GCS is not**; therefore, to achieve a good speed, we combine BMS and DPS for the guidance method: Stereo SGBM - the semi block matching algorithm. The profiles made from BMS are used as a guiding profile. In addition to its general dynamic programming calculation, quantitative guiding scores are added to DPS's structure so that the method can follow these guiding profiles to establish a more accurate stereo matching. The algorithm uses Dynamic Programming described by Birchfield and Tomasi (Birchfield & Tomasi, 1999), to make the stereo matching more robust as the computation of disparities is cast as an energy minimisation problem. BMS uses windows to match correspondences; its results are influenced by pixels in multiple scanlines. The cooperation of BMS and DPS should eliminate the effect of the straight strikes made by single scanline reconstruction of DPS. On Middlebury Stereo Evaluation website (*Middlebury Stereo Vision webpage*, 2001), this SGBM algorithm obtains approx. 76% matching accuracy.

Figure 3.13 displays a disparity map results of some sample AR markers. The disparity range d is set to be between -32 and +31 pixels (64 levels). We calculate the SAD value at each disparity d:

$$SAD_d(x,y) = \sum_{a=-k}^{k} \sum_{a=-k}^{k} |I_l(x+a, y+a) - I_r(x+a+d, y+a)| \tag{3.9}$$

The disparity map is normalised to 0 and 255 and display to users along with the left and right stereo images. Mainly, the brighter the pixel, the closer the 3D point is, and vice-versa. If dynamic programming fails to obtain a disparity value for a point, it returns a value of -1.



(a) Left stereo image (b) Right stereo image (c) Resulted disparity

Figure 3.13: Left and right stereo images extracted from Spider man marker and its disparity map, read from top down: $13110200_4 = 29984_{10}$.

This barcode can be recognised using basic image processing methods, depth level of each bar is averaged, and a unique number can be calculated accordingly using the following Equation:

$$A = b_{N-1}L^{N-1} + b_{N-1}L^{N-2} + .. + b_1 L^1 + b_{k-1}L^0 \tag{3.10}$$

Where A is the number presented by the hidden bar-code, b_i is depth level at i^{th} bar, L is number of depth levels at each bar, and N is the number of bars.

The controlled zone is a horizontal bar at the bottom of the bar-code; it is used to self-check the validity of the bar-code and specifies the bar-code version. The controlled bar has the same thickness as a vertical bar, and always have the lowest depth level (black colour). Currently, we define five different versions of this multi-level bar-code as shown in Table 3.1. The Higher version has more vertical bars and more levels presenting in each bar; thus storing a larger range of numbers. For instance, version 0 can only represent numbers between 0 and 63 using 6 bars and two levels each. On the other hand, version 6 may store up to 7.95×10^{25} different numbers using 24 bars and 12 levels each.

Version	**Controlled Zone**	**Number of bars (N)**	**Depth levels (L)**	**Presentation Range**
0	$\frac{1}{4}$ width	6	2	0 – 63
1	$\frac{1}{8}$ width	8	4	0 – 6.55×10^4
2	$\frac{1}{12}$ width	12	6	0 – 2.18×10^9
3	$\frac{1}{16}$ width	16	8	0 – 2.81×10^{14}
4	$\frac{1}{24}$ width	24	12	0 – 7.95×10^{25}

Table 3.1: Specification of different versions of multi-level bar-code

3.4.2 Marker 2: Curtain style pictorial marker (CSPM)

The Curtain Style Pictorial Marker (CSPM) aims to conceal the more incredible amount of encrypted information by using the quick-response code (QR) encoding (Figure 3.14). The proposed marker provides a proper graphical content presentation and improves detection performance and security issues. The demo video of CSPM concealed code decryption process can be reachable via **`https://www.youtube.com/watch?v=`**

`ciYq0Ke8i1U` and sample 3D pose estimation at `https://www.youtube.com/watch?v=DYCltnvkyYg`. This proposed marker has been successfully published at the **33rd International Conference on Image and Vision Computing New Zealand (IVCNZ 2018)**, which presents the following novel contributions:

- **Large data storage capacity:** The concealed QR code could hold at least 133 encoded data items with the lowest version and up to 23,648 encoded data items with the highest version (Kieseberg et al., 2010).

- **The flexibility of Pattern:** The images are approximately similar to each other that can be used without worrying about the undesired similarity as the decoded information and the image patterns are independent.

- **Virtually Pictorial:** The image inside CSPM is made from meaningful illustrations rather than black and white patterns as in data marker, which provides no real pictorial information.

- **Error detection and correction capability:** The capability to restore at approx. 30% of codewords.

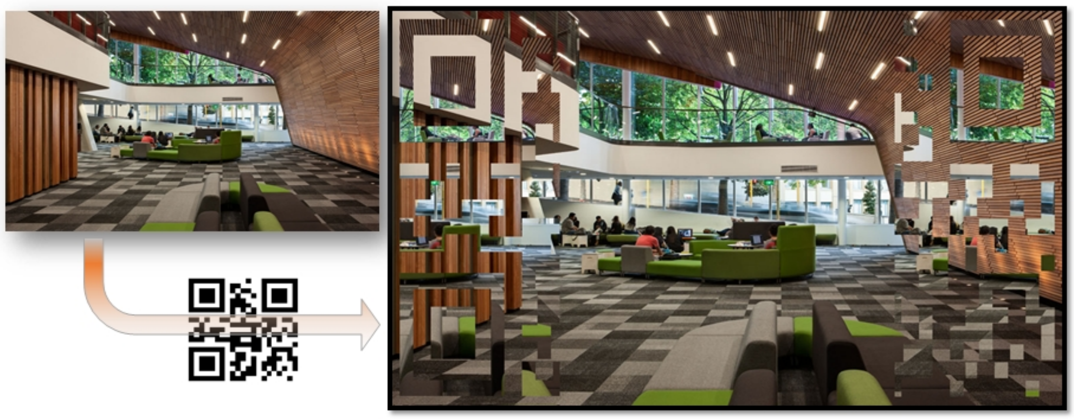


Figure 3.14: The curtain style pictorial marker design allows the QR code to be embedded on the edges of the original figure.

The encryption process of CSPM is shown in Figure 3.15; it will have a black colour border which is used to ease the marker recognition, detection and segmentation. Given an image with the dimension of $H \times W$ where H is the height and W is the width. The border thickness t is equal to 2% of the image longest edge or $t = max(H, W) \times 2\%$. The central area *A* is a fixed original image pattern that occupies 50% of the CSPM area. Other two repeated patterns on both sides of region *A* (region *B* and region *C* with 25% each). Assume that a unique string: "Peacock" needs to be encrypted into a QR code. Thus, there is a QR code being generated, called I_{QR}, and resized it to $\frac{W}{2} \times H$ pixels. Then the two curtain-edged regions (*B* and *C*) will be used to hold this binary QR code.

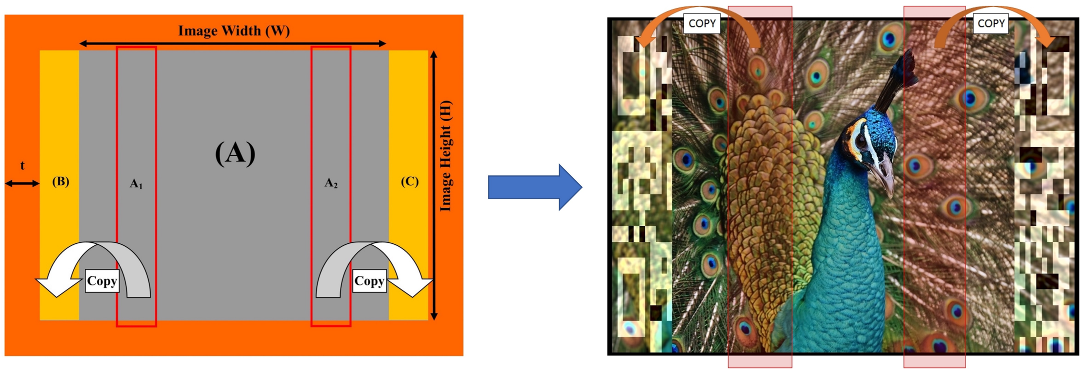


Figure 3.15: Hidden QR code encryption process.

The region **B** is read the left half of the QR Code I_{QL}; if the QR block is black, the 1-to-1 from the central colour image to the curtain-like region is copied. If the block is white, a block with the highest illuminant difference in the curtain is generated. Region *C* is done similarly, using the right half of the QR Code I_{QR}. The brief idea of CSPM is that when the left region is subtracted from the right region, it will produce a binary QR Code. The concealed data decryption process is demonstrated in Figure 3.16. The procedure includes three steps: (1) internal image detection, (2) QR code regions detection, and (3) QR code reconstruction and decoding.

Step 1 - Internal image detection. The marker border needs to be identified first to retrieve the internal image. This is a typical image processing-related problem which could be solved effectively by using the contour approximation method (Kim, 1998). The steps to detect marker border are outlined below:

- Convert the input image from RGB to greyscale
- Perform an adaptive binary thresholding method to detect contours in the image.
- If there are four vertices in the contour, it should be identified as a quadrilateral.
- Apply Perspective Transform (OpenCV, 2008) to retrieve the internal image of the marker as shown in Equation 3.11.

$$\overset{I'}{\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix}} = \overset{\substack{map \\ matrix}}{M} . \overset{I}{\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}} \tag{3.11}$$

where M is a map matrix, I' is the internal image created from the four corners $g(x_i, y_i), i = 0, 1, 2, 3$ of the original image I. Once the internal image is identified, it could be used to obtain further decoding.

Step 2 - QR code regions detection. After the internal image is discovered, the image is cut into halves vertically: A and B. For half A, it is also cut into halve: A_L and A_R. To achieve the first half of the QR Code, the image subtraction between the two A_L and A_R is performed:

$$A_{diff} = ||A_L - A_R|| \tag{3.12}$$

Repeat the same steps with half B of the image to get B_{diff}. In the end, the two A_{diff} and B_{diff} are combined to reconstruct the entire QR code image. Some further histogram equalisations would be used to compensate for the illumination difference and some thresholding.

Step 3 - QR code reconstruction and decryption. The complete detected QR code can now be used to extract the hidden information and present the marker's orientation. There are many available open-source barcode reading libraries such as Zbar [1] or ZXing. Each of those libraries can read an image frame and automatically detect the QR code. The meaningful text can be then decoded and the code orientation by defining the four corners of the QR code. The system then can use this information to render computer-generated graphics on the marker.

3.4.3 Marker 3: Tile-based Quick Response code marker

The tile-based Augmented Reality marker is another proposed marker that can conceal the encrypted quick response (QR) code information. This marker is motivated by the 3D illusion created by looking at a kitchen or bathroom tiled wall, as shown in Figure 3.17. If both eyes are paralleled and virtually looked at a point behind that wall, the tiles appear to float at different layers of depth. The effect is very similar to autostereogram, or magic eye pictures described in (Tyler & Clarke, 1990). The irregular gaps between the tiles create the various depths of individual tiles. If a QR code $M \times M$ is considered as the tiled wall, and the black and white dots are simply many tiles lying at different depth levels, then a wall that optically hides a QR code can be built using this principle. The tiles are all having the same size, but the gaps between them are different. However, there are only two sizes of gaps for a row of tiles: larger and smaller for binary values of '1' and '0', respectively. The marker is thus pictorial and yet robust enough to be detected under various lighting conditions. Moreover, gaps are small compared to tiles; on average, 80-90% of the original picture is retained.

[1]Brown, J. (2007). Zbar bar code reader. Retrieved from http://zbar.sourceforge.net/

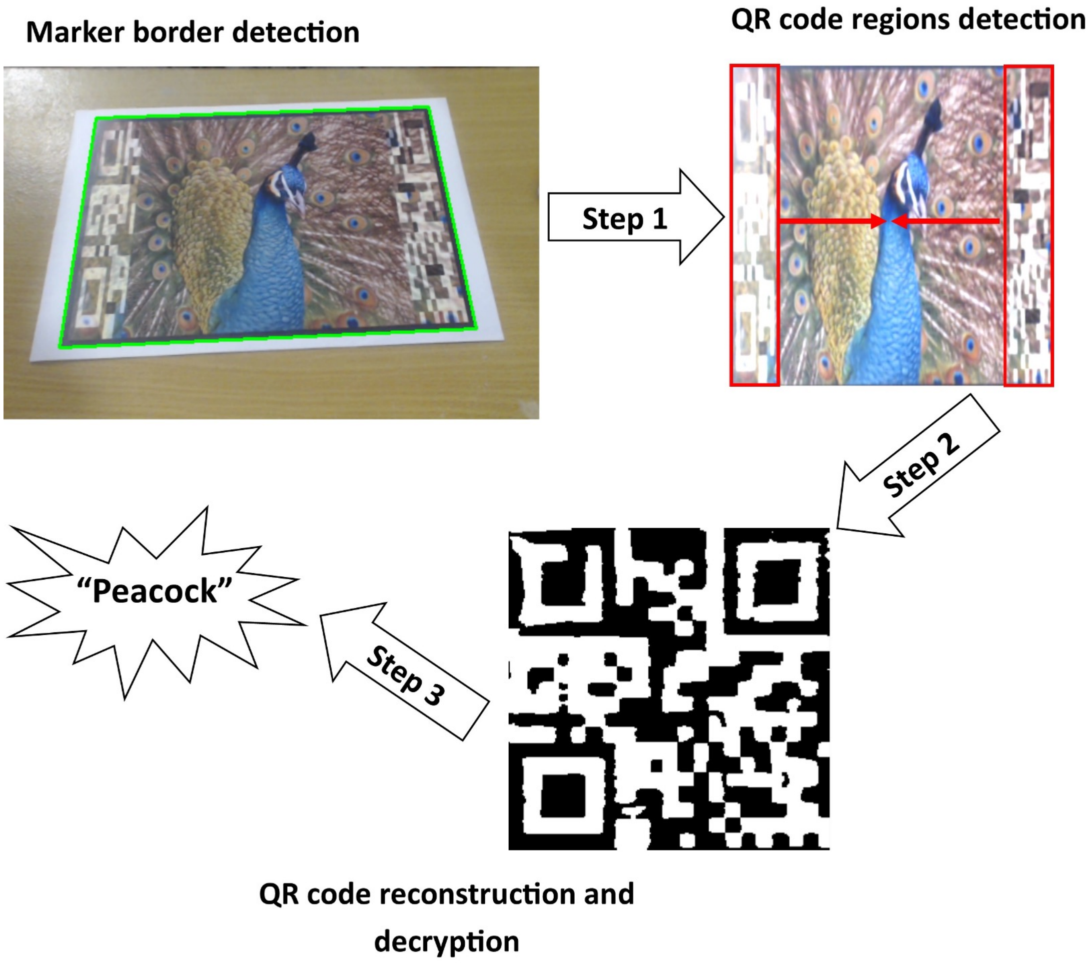

Marker border detection
QR code regions detection
Step 1
Step 2
Step 3
"Peacock"
QR code reconstruction and decryption

Figure 3.16: Hidden QR code decryption process.

There are only two components are needed for the creation: a colour picture and a QR code. QR codes are now easily obtained by either an online tool such as `www.qr-code-generator.com` or a public library such as QRcode python library at `pypi.python.org/pypi/qrcode`. Assume a QR code with dimension $M \times M$, mapped on a colour picture of dimension $W \times H$ pixels. In order to build a tiled walled, that neatly fit in the image, $(M + 1) \times M$ rectangular tiles are required. Each tile should have the same size of w_t, h_t pixels. The components of the vertical cap (g_y) need to be

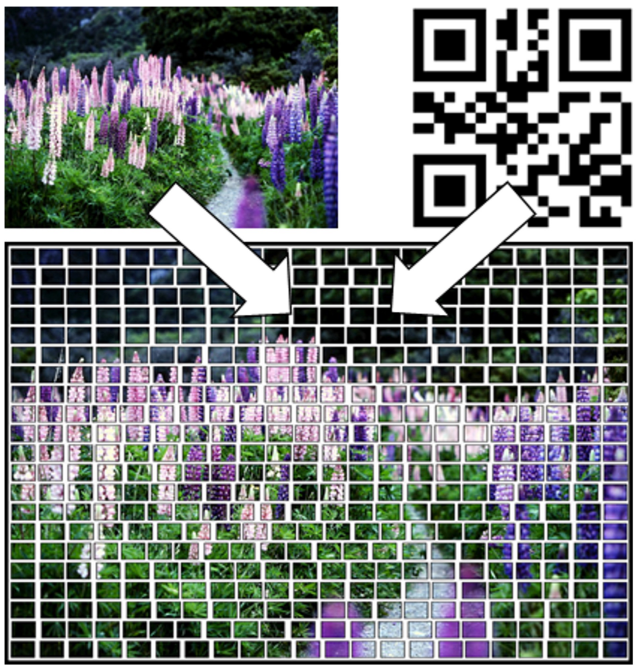


Figure 3.17: The tile-based Augmented Reality marker design allows the QR code to be concealed as the tile blocks on the indoor walls.

calculated first by using the following Equation:

$$g_y = \frac{H - M \times h_t}{M} \tag{3.13}$$

The horizontal big gap $g_x b$ for black QR dot and horizontal small gap $g_x s$ for white QR dot are calculated differently for each horizontal line i^{th} of the QR code. Assume that a QR scan-line i has a black dots and b white dots: $a + b = M$, and the big gap $g_x b$ is n times the small gap $g_x s$. They are calculated as follow:

$$g_x s(i) = \frac{W - (M + 1) \times w_t}{n + 1} \tag{3.14}$$

$$g_x b(i) = n \times g_x s(i) \tag{3.15}$$

When all the components (tiles and gaps) are defined, they can be placed accordingly on top of the provided picture. Each tile is a transparent glass with a black frame; the gaps are filled with white cement. After that, a black border is placed with width = 1% of the image on top. This rectangular border is used for marker recognition, detection, and segmentation (the quadrilateral property of the squares can be used to detect their four straight lines and four corners).

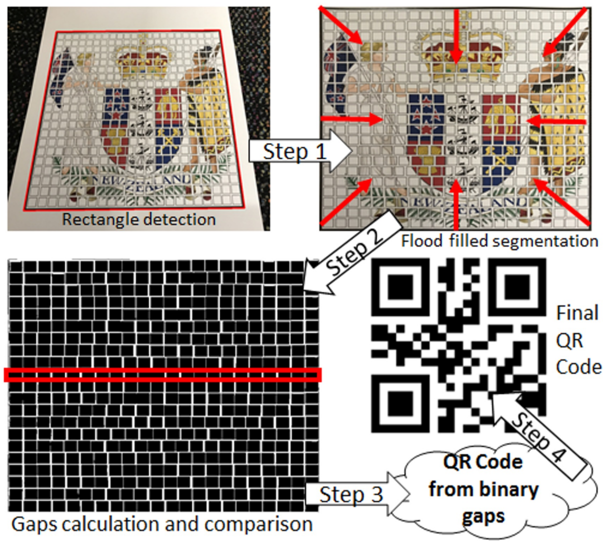


Figure 3.18: The tile-based Augmented Reality marker concealed information decryption process.

Figure 3.18 shows four steps of the proposed marker decryption steps: (1) rectangular marker detection, (2) gap region segmentation, (3) binary gap classification, and (4) QR Code reconstruction from gap's distances. Steps 1 and 4 are similar to steps 1 and 3 of the curtain style pictorial marker decryption process described in Section 3.4.2.

Step 2 - Gap Region Segmentation. After the internal tiled image is detected, it will be resized to a suitable dimension (1024 × 1024 pixels in this case). A simple flood filling algorithm (Nosal, 2008) can be applied with the knowledge of the white colour gaps between tiles, and each tile has a thin black frame. This is also called seed fill algorithm, which determines the area connected to a given node in a multi-dimensional array. Shadows and lights can change the appearance of the photo; for a robust segmentation, eight seed points at the corners and boundaries of the image are set, as seen in Figure 3.18 top-right. At each seed point, the below flood fill algorithm is applied:

```
def floodFill(x, y, fillColor, interiorColor):
 getPixel(x,y,colour)
 if colour is similar to interiorColor:
  setPixel(x, y, fillColour)
  floodFill(x+1, y, fillColor, interiorColor)
  floodFill(x−1, y, fillColor, interiorColor)
  floodFill(x, y+1, fillColor, interiorColor)
  floodFill(x, y−1, fillColor, interiorColor)
```

Step 3 - Binary Gap Classification. The Flood fill segmentation is used to separate gaps and tiles as shown in Figure 3.18 bottom-left. In general, it does not expect all the gaps are detected due to many constraints such as noises and lighting conditions. However, there is an assumption that a majority of the gaps are segmented. The tiles can be reconstructed to find the marker orientation. All the components such as the dimension $M \times M$ of QR code, the size of each tile w_t, h_t pixels, the vertical cap (g_y), the big horizontal gap $g_x b$ and small horizontal gap $g_x s$; can also be estimated statistically. After segmentation, the result is similar to what shown in Figure 3.18 bottom-left, every horizontal and vertical scan line are analysed to create collections of:

- Number of horizontal and vertical black and white gaps.
- Sizes of horizontal and vertical black and white gaps.

If the statistics mode value (the data value that appears most often) of horizontal white gaps for all scan lines is one value higher than the mode value of vertical white gaps for all scan lines, then the marker is at correct orientation or upside-down orientation. If not, a rotation needs to apply to the image. The mode value of all horizontal black gaps is the estimate of the width of each tile w_t. Thus, the mode value of all vertical black gaps is the height h_t of each tile. The vertical gap g_y is found from the statistics mode of all vertical white gaps. The gap sizes are harvested horizontally, the known width w_t of each tile is used to control the quality to ensure no gaps are missing. The tiles can be separated from the collection into two groups: big with size $g_x b$ and small gaps with size $g_x s$. The big gaps represent black dots, and small gaps represent white dots of the QR code.

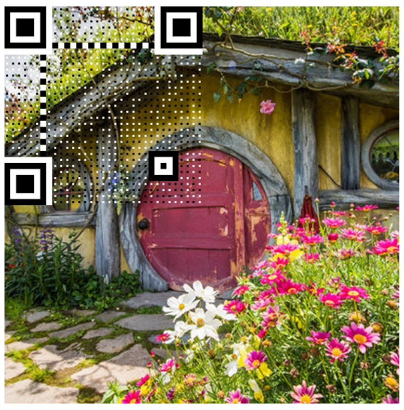
3.4.4 Marker 4: Removable transparent Quick Response code marker

The removable transparent quick response (QR) code marker is designed for real-life production to keep the original texture information as much as possible and could be applicable on any texture with the minimum cost (Figure 3.19a). The proposed marker is a part of an online deployment platform powered by web-based AR.js that allows real-time display of virtual information such as 3D models on the top of the given texture. This idea was successfully published at the **25th ACM Symposium on Virtual Reality Software and Technology (VRST 2019)** and at the **5th Asian Conference on Pattern Recognition (ACPR 2019)**, where they presented the uses of the proposed marker for the medical application domain. The application demo video can be reachable via `https://www.youtube.com/watch?v=fOwu318KY0w`. In short, this marker and online deployment system produce the following advantages:

- **All in one Augmented Reality marker.** The proposed system does not require the installation of specific software like other Augmented Reality applications. The user can scan the QR code to access the website, and the 3D model is displayed right on the marker.

- **Light-weight and portable application.** There will be no pre-loaded process for virtual information such as 3D models. The information will be loaded when the marker identity is detected; hence less disk space is required.

- **Transparent QR code.** The website URL is encrypted in a QR code located on the left top of the tag. The code sticker is primarily transparent to preserve most of the original information of the given texture. QR codes allow us to reuse the same code on different textures and display various models by using the same code (Figure 3.20).


(a)

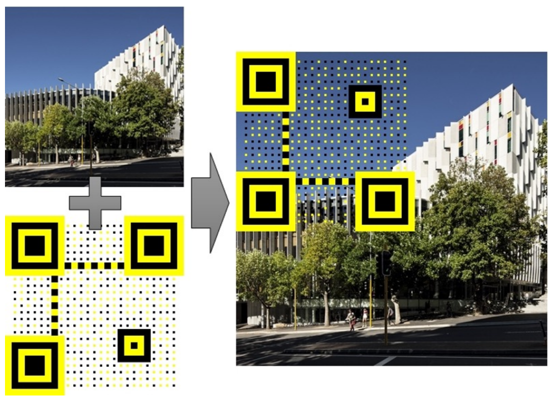

(b)

Figure 3.19: The example of transparent QR marker (a) and its encryption/decryption steps (b).

The proposed marker encryption/decryption steps are presented in Figure 3.19b. The four black boxes and their surrounding (yellow boxes) are preserved for reliable detection of the QR Tag. All other dots are squeezed into 20% of their original sizes. All other spaces are made transparent to achieve the QR Code. This transparent QR code can be pasted on top of a colour image digitally as seen on the figure or physically by sticking this on a textbook page. Alternatively, the QR code can be printed on adhesive plastic paper using a printer. Notes: Most printers do not have white ink (as they print on white paper; therefore, white ink is unnecessary). This is why the yellow colour is used for the white square in the above transparent QR code. This use of transparent QR codes has some benefits compared to the traditional one. For instance, it does not fully cover the pictorial feature of the RGB images; users can stick it to any region of the image. Moreover, from the experiment, the performance of this transparent QR code is also equivalent to the original one (approx. 90% of detection and decryption rate).

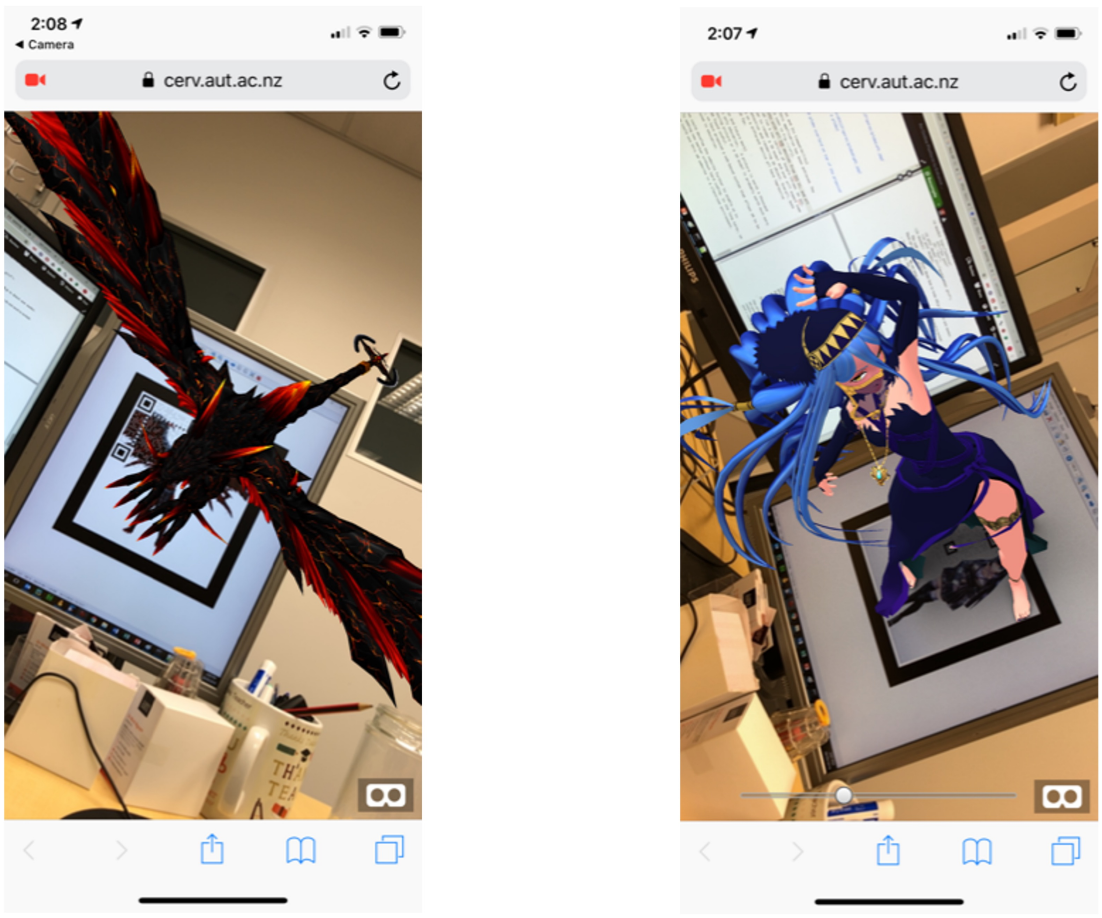
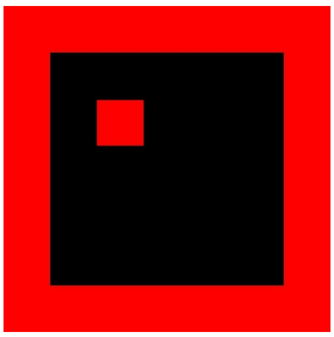


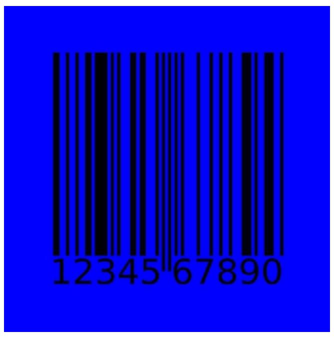
Figure 3.20: Virtual 3D models displayed on the top of proposed removable transparent QR code marker.

3.4.5 Marker 5: Red-green-blue Augmented Reality marker

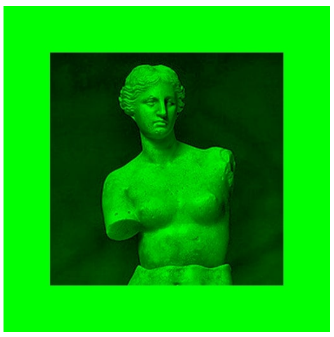
The red-green-blue (RGB) Augmented Reality marker aims to encode three unique pictures (1) an oriented marker, (2) a bar-code and (3) a graphic image into three respective three colour channels: green, blue and red. This proposed marker has been successfully published at the **Advanced Concepts for Intelligent Vision Systems: 20th International Conference (ACIVS 2020)**.



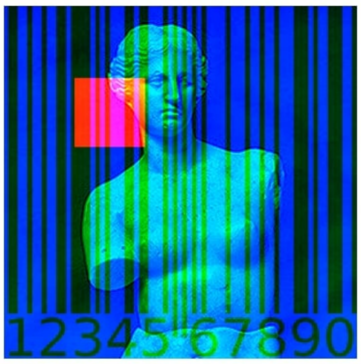
(a) Red oriented marker



(b) Blue bar-code



(c) Green texture



(d) Result after RGB channels combined

Figure 3.21: The proposed Red-green-blue Augmented Reality marker design.

The marker design idea is demonstrated in Figure 3.21. The red channel is used to store an orientation robust squared marker (Figure 3.21a); the blue channel holds the product bar code (Figure 3.21b); and the green channel is used to display the original image of the product (Figure 3.21c). The choice of the colour is decided carefully as the human eyes are most sensitive to yellowish-green colour, then to red, and not very sensitive to blue colour. However, to computers, they are relatively the same. The green colour is considered to present the product photo because it is the most important visual information for retailers and shoppers. Blue is for the bar-code because it is complex and distracting. Red is then kept for the orientation figure.

Red Oriented Tag

To blend between the real and virtual environments harmoniously, it is essential that the 3D graphics are well aligned with the real-world objects. The bordered marker is stored in the blue channel for tracking relative position and orientation to the camera. Our marker is made of two parts: solid black square and inner white square. The position of the black square determines the 3D graphics position, while the white square is for orientation. Like other computer vision projects, there is a primary pipeline for marker detection as well: Thresholding, Labeling, Extract contours, Find four corners, and Get information and verification. The converted grayscale image is segmented (the black square is separated from the rest of the image) and a binary image created by thresholding technique. The principle of the thresholding technique is to separate the light and dark regions. Each pixel below a certain threshold is turned to zero, and each pixel above that threshold is transformed to one.

Labelling seeks groups of connected pixels and ultimately identifies the closed area in the image. Each pixel that satisfies the thresholding will obey the following algorithms: First, scan all pixels row by row and assign a preliminary label; Second, merge the equivalent regions that have different labels into one single label. Next is finding the contours of the objects for further determination of the curve, which stands for the solid border of a marker. Future extraction of the corner markers will be by this curve. Also, the edge detection algorithms can draw out the contour. Canny edge detection is a multi-stage edge detection algorithm with noise suppressed at the same time. The first stage is using a Gaussian filter to smooth the image for the noise and unwanted details and textures reduction. The process of smoothing the image *I* with a Gaussian filter *G* can be written as

$$G(x) = exp(-x^2/2\sigma^2)/2\pi\sigma^2 \tag{3.16}$$

$$I(x,y) = [G(x)G(y)] * f(x,y) \tag{3.17}$$

In the above equation, σ is the Gaussian filter spread, the smoothing degree is determined by it. The second stage is the calculation of the gradient direction and magnitude. A 2×2 neighboring area is often selected in the Canny edge detection algorithm to acquire the corresponding magnitude and direction gradient image. Through the following formulas, the first order partial derivatives in the directions of x and y can be gained:

$$M(x,y) = \sqrt{N_x^2(x,y) + N_y^2(x,y))} \tag{3.18}$$

$$D(x,y) = arctan[N_y(x,y)/N_x(x,y)] \tag{3.19}$$

$$N_x = (-S_1 + S_2 - S_3 + S_4)/2 \tag{3.20}$$

$$N_x = (S_1 + S_2 - S_3 - S_4)/2 \tag{3.21}$$

Here M(x, y) represents the gradient magnitude of the image, and D(x,y) stands for the gradient direction of the image, also the pixel values of the image (x,y), $(x,y+1)$, $(x+1,y)$, $(x+1,y+1)$ are stated by S_1, S_2, S_3, S_4 respectively. After gaining the gradient magnitude image M and gradient direction image D, we need to perform non-maximum with the goal to detect the edge and avoid the occurrence of false edges efficiently. A 3×3 adjacent area is chosen in the Canny algorithm to compare a pixel with its two adjacent pixels along the gradient direction. If the magnitude $M(i, j)$ is more significant than the two interpolation results on the gradient direction, a candidate edge point will be arranged to the pixel. Otherwise, a non-edge point will be assigned to it (Neubeck & Van Gool, 2006). After applying non-maxima suppression, canny methods employ double thresholds, which consist of low and high thresholds to detect and connect edge points. If a pixel has a gradient magnitude that is bigger than the high threshold, an edge point will be assigned to the pixel, while a pixel with a smaller gradient will be assigned with a non-edge point. For those gradient magnitudes between high and low thresholds, if there is a point around the pixel more significant than the high threshold, then it is the edge point. Following these steps, the edge image is acquired.

After finding the contours of the marker, we need to find further the polygon approximations of the contours for the marker square using the polygon approximation algorithm (Wikipedia, 2017). This algorithm can reduce the number of points in a curve, which is approximated by a series of points. The functions of this algorithm are to find the distance dimension on the line for each point and conduct the simplified curve reconstruction. Finally, the points of the simplified square can be seen as the marker corners. After identifying the marker corner coordinates, which should be vertical, the 2D and 3D coordinates need to be mapped in the real world so that the 3D objects appear as if they are positioned on top of the marker. The camera is positioned in the origin of the coordinate system of the camera, looking along the Z-axis. Two axes of the coordinate marker system are parallel to the sides of the squared marker. To establish the mapping relationship between the marker coordinates and screen coordinates, we need to get transformation between them.

In the circumstance of coding, all these transformations are a matrix. Also, the transformation is represented by a matrix in linear algebra. Here, we can use homogeneous coordinates (Li, Hestenes & Rockwood, 2001) to perform the matrix. The scientific name of matrix C is the camera internal reference matrix, and matrix T_m is referred to as the camera external reference matrix, where camera calibration obtains the internal reference matrix in advance, and the external reference matrix is unknown. Thus, we need to define it according to the screen coordinates (x_c, y_c) in advance according to the marker's coordinate system and internal parameter matrix. Then, based on T_m, we can draw the graphic on it. After marker detection and verification, it is possible to display the 3D model on the AR marker. We want to keep track of the marker's orientation so that the 3D graphics can rotate consequently as the marker rotates from the camera. We will use the small inner square to orient. The detection steps are almost the same as the previous processes to find the large square. Once we finish detecting the solid outer border, next step is to detect the small square that designates the vertex of the large square closest to the small square as "first".

Blue Product Bar Code

Most products are printed with barcodes for the ease of detecting and calculating the final prices. The commonly used barcode formats in commodity products are EAN-8, EAN-13, and UPC. They are one-dimensional barcodes which can be readable by most Laser scanners used in retails. The same 1D barcode of a particular product can be used for this Blue Product Bar Code. Nowadays, 2D barcodes are more and more frequently used, such as QR code, due to its higher storing capacity and better error correction rates. QR code was introduced by Denso Wave in 1994 (Kieseberg et al., 2010). Since then, the QR code has obtained wide popularity in many diverse industries, like construction, marketing, healthcare, tourism, life sciences, and education. The reason for its popularity due to the data stored in QR code is of a higher density than the information in a barcode. Another reason is that it is convenient and straightforward to download and install a code detector onto a mobile, enabling data to be quickly retrieved. Furthermore, it has the advantage that either electronic or static media efficiently distribute data in QR code. Today, QR code can act as either an identifier or a database itself. Likewise, our RGB tag is also capable to store 2D QR code in its blue channel.

Green channel Texture Image

The texture image is the photo of the product, which is stored in the marker's green channel so that users can have an essential first sight of what the 3D graphics will be. As stated, human's eyes are most sensitive to green colour. Also, contrast sensitivity of the human visual system plays another important part (Barten, 1999). If the contrast of the image is high, this provides a better perception of human vision. Contrast refers to the separation of the bright and dark areas present in an image. The difference between the colour and brightness of the object and its background determines the image contrast. Contrast enhancement aims to enhance the visibility of the object by improving the difference between the brightness of the object and its background.

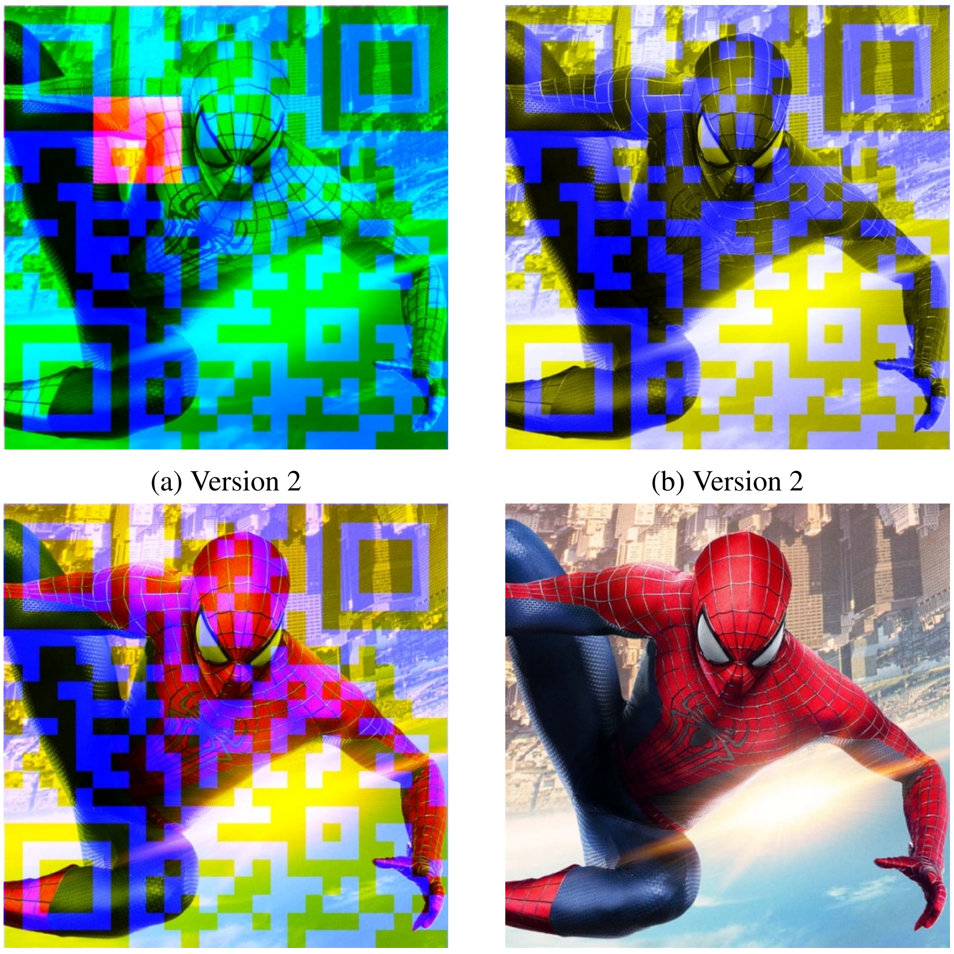
The most widely used technique for increasing the contrast of images is the grey-histogram method. This is a graph displaying the number at which each grey level occurs in an image, plotting the number of pixels for each tonal value (Bora & Gupta, 2016). The contrast of an image can be enhanced by changing its grey histogram: distributing the image value to cover a wide range on the graph. Here, we use a method called "Contrast Limited Adaptive Histogram Equalisation (CLAHE)", which is optimised based on the adaptive histogram equalisation method (AHE). The difference between AHE and CLAHE is the contrast limit as the CLAHE introduced clipping limit to overcome the noise amplification problem (Zuiderveld, 1994). CLAHE was firstly introduced to process the medical image and has proven to be an effective method for enhancing a low-contrast image. The principle behind this method is that the image is segmented into several non-overlapping regions with nearly equivalent sizes (Sasi & Jayasree, 2013). The steps in this approach are described below:

- First, calculate the histogram of each region.
- Then, based on the required restrictions for contrast expansion, acquire a limit of the clip for clipping histograms.
- Next, for that height, does not exceed the clip restriction, distribute each histogram in the same way again.
- Finally, with the aim of grayscale mapping, determine cumulative distribution functions (CDF) of the generated contrast limited histograms. In the CLAHE technique, pixel mapping can be achieved by integrating the outcomes resulting from the mappings of the four nearest regions linearly.

Combination of the three channels

After identifying the components (1) an oriented marker, (2) a bar-code and (3) a graphic image; we can store them in red, green and blue channels of the final image. Figure 3.22a display such a RGB tag (Version 1) created from the original picture of the Spiderman in Figure 3.22d. As we are not very sensitive to blue colour, the QR code is not too visible; however, the red square of the oriented marker is quite obvious and distracting. The Spiderman is relatively visible with low contrast due to the use of only one channel.

When we use QR Code as the embedded bar-code, it is known that the QR codes themselves include position detection patterns that could be used to identify the orientation for the tag. Thus, the oriented marker is needed anymore if QR code is used. Thus, one red channel is saved and we may use it for presenting the texture image, to increase the contrast. Figure 3.22b shows the effect of elimination of red orientation marker (Version 2). The contrast definitely increase and the figure of the Spiderman is relatively visible to most people; however, the figure is greyscaled. We slightly improve the look of it by retaining the original red and green colour channels of the Spiderman figure to achieve its version 3 (Figure 3.22c). Comparing this tag with the original images in Figure 3.22d; we do not see much difference; most pictorial information still remains correctly and the viewer can very quickly recognise that it is a Spiderman figure.



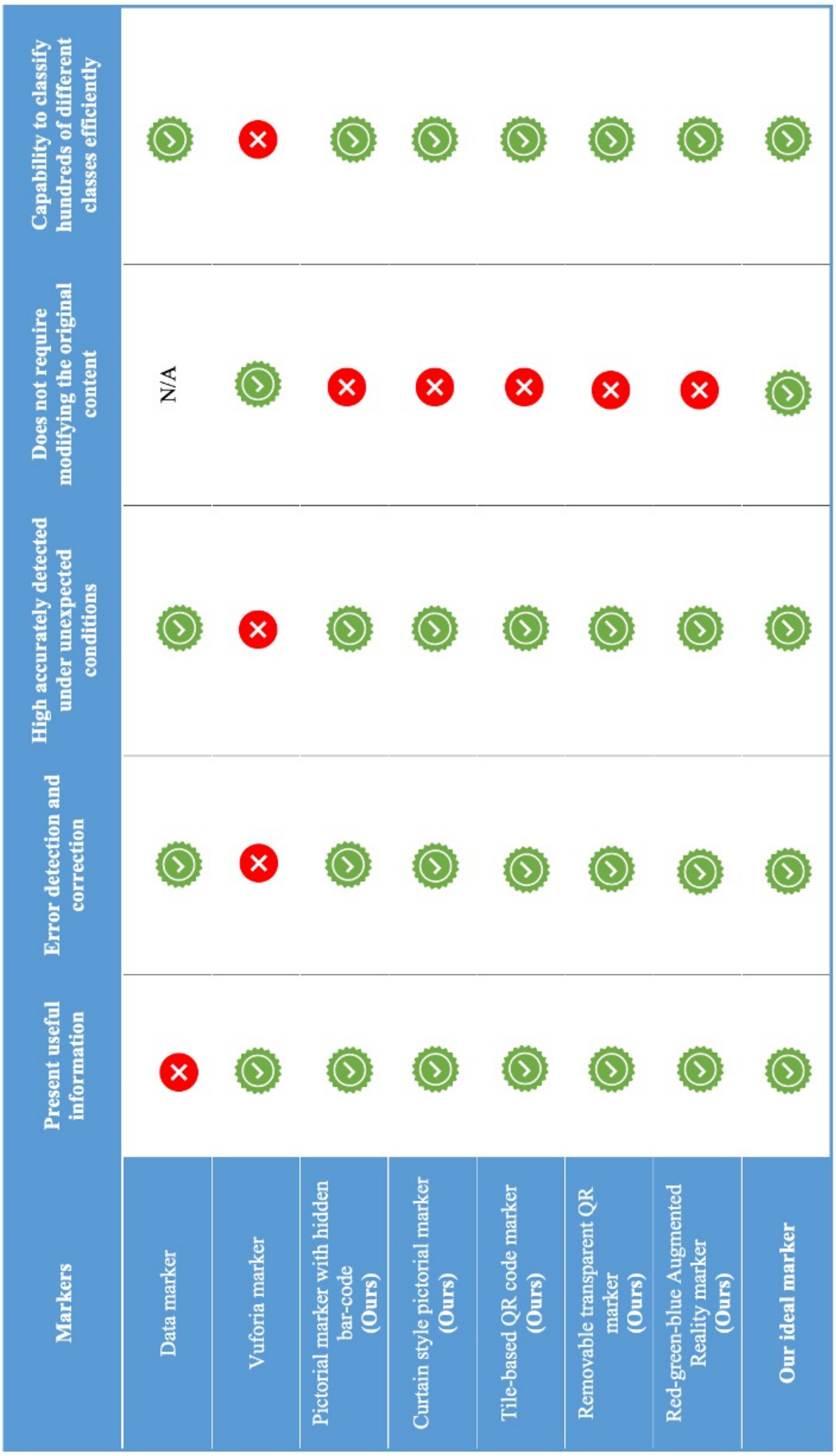
(a) Version 2 (b) Version 2

(c) Version 3 (d) Original image

Figure 3.22: The proposed Red-green-blue Augmented Reality marker with QR code as the concealed information.

3.5 Summary

This Chapter has demonstrated different types of Augmented Reality markers, their encrypting and decrypting techniques. The original Augmented Reality markers have their strong points and drawbacks. It also demonstrated the newly-designed markers which aim to overcome the original marker disadvantages. In short, the proposed markers have remarkably improved encrypting, decrypting, and visualising information compared to the other markers as summarised in Figure 3.23. They can easily overcome the drawbacks of the original markers where they can provide meaningful information and give the high accuracy detected under different unexpected conditions to avoid the undesired similarity between template markers. However, another significant issue for the newly-designed markers is that they often require the users to modify their original graphical material, either partial or complete, which is usually not acceptable. Hence, a novel solution that does not require changes to the original texture and could keep the reliability at the acceptance level is needed; as shown as **our ideal marker** in Figure 3.23. This is the overall of this thesis that will be described in detail in the following Chapters. In the next Chapter, another interesting topic – Machine Learning is demonstrated that is the critical solution to help overcome the issues of the Augmented Reality markers using traditional computer vision techniques.



Markers	Present useful information	Error detection and correction	High accurately detected under unexpected conditions	Does not require modifying the original content	Capability to classify hundreds of different classes efficiently
Data marker	✗	✓	✓	N/A	✓
Vuforia marker	✓	✗	✗	✓	✗
Pictorial marker with hidden bar-code (Ours)	✓	✓	✓	✗	✓
Curtain style pictorial marker (Ours)	✓	✓	✓	✗	✓
Tile-based QR code marker (Ours)	✓	✓	✓	✗	✓
Removable transparent QR marker (Ours)	✓	✓	✓	✗	✓
Red-green-blue Augmented Reality marker (Ours)	✓	✓	✓	✗	✓
Our ideal marker	✓	✓	✓	✓	✓

Figure 3.23: The performance comparison between newly-designed markers and the original markers.

Chapter 4

Contemporary Machine Learning for AR, its Pros and Cons, and the Proposed Idea of using Synthetic Dataset Generation

Parts of this Chapter have been published in ***paper 1****,* ***paper 2****,* ***paper 3****,* ***paper 4****, and* ***paper 5*** *listed in the publication list*

The term of Machine Learning was started in 1959 by Arthur Samuel (Samuel, 1959; Provost & Kohavi, 1998), but the outcome only stops at the proposal or prototype level due to the low public demand and lacking supported tools. However, with the storm of new technologies in both hardware and software today, the application of Machine Learning can easily be found in many sectors of daily life. The implementation of Machine Learning has become more accessible than before with many affordable training devices available in the market. Therefore, it is one of the critical inspirations in this research. The rest of this Chapter primarily focuses on the following aspects:

- The theoretical background surrounding Machine Learning in general, its functions, and training styles ***(Section 4.1, 4.2 , 4.3, and 4.4)***.
- Introduction of the most suitable Machine Learning model for the proposed Augmented Reality system in this thesis ***(Section 4.5)***.
- Identifying the primary drawbacks during the Machine Learning training process and propose the potential solution to overcome them ***(Section 4.6)***.

4.1 What is Machine Learning?

The Machine Learning is defined as a sub-field of artificial intelligence (Mitchell et al., 1997) that focuses on building the applications that learn from sample data to make decisions or predictions without being programmed to do so (Koza, Bennett, Andre & Keane, 1996). The principle theory of Machine Learning is the same as how the humans learn new concepts. The Machine Learning algorithm is a sequence of statistical processing steps trained to find the specific patterns or features in a massive amount of data to predict the new or unseen sample accurately later (Bishop, 2006; Schapire & Freund, 2012). The approaches of Machine Learning are generally divided into three main types of learning:

Supervised learning is when the system is fed with sample inputs and their desired outputs (also called labelled outputs) (Russel & Norvig, 2003), and the goal is studying the general pattern that maps the inputs with outputs.

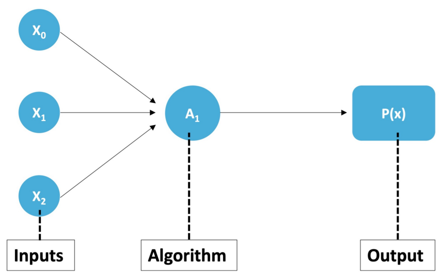
Unsupervised learning is slightly different from supervised learning when there is no output given to match the inputs. The system needs to find its structure based on its given inputs (Hinton, Sejnowski et al., 1999).

Reinforcement learning is different from supervised and unsupervised learning, where there is a specific goal that the system must perform under a dynamic environment without providing inputs and outputs (Van Otterlo & Wiering, 2012).

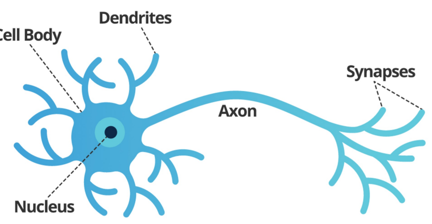
The supervised learning is considered in this thesis where a massive amount of sample data could feed into the system to recognise unseen images. In supervised learning, the system estimates the probability $P(y|x)$ where y is the output of a given input x. A wide range of supervised learning algorithms are available, but the artificial neural network is highly considered due to its effectiveness and relevance.

4.2 Artificial neural network

The artificial intelligence neural networks, usually called neural networks, are the Machine Learning algorithm inspired by the biological neural networks of the human brains, as shown in Figure 4.1. The neural network consists of many artificial neurons that take the inputs from the user or previous layers, sum them together and pass them through an activation function to give the predicted outputs.



(a) Artificial neural network



(b) Biological neural network

Figure 4.1: The artificial neural network (a) is inspired by the biological neural network of the human brain (b).

4.2.1 Artificial neuron

Warren McCulloch and Walter Pitts introduced the idea of the artificial neuron in 1943 with the first form called Linear Threshold Unit (Anthony, 2001). An artificial neuron is an elementary unit of the neural network that sums up the inputs after being multiplied with the corresponding weights and sent through the activation function, as shown in Figure 4.2. For a given artificial neuron Y, let there be n inputs with the signals started from x_1 to x_n and the corresponding weights from w_1 to w_n. The following Equation can describe the output Y:

$$Y = \varphi\left(b + \sum_{i=1}^{n} w_i * x_i\right) \tag{4.1}$$

where φ is the activation function and b is the bias. The term of bias first appeared in the research title, "The need for biases in learning generalizations", by Tom Michell in 1980 (Mitchell, 1980). The bias occurs when Machine Learning prediction results are systemically prejudiced due to erroneous assumptions during the training process.

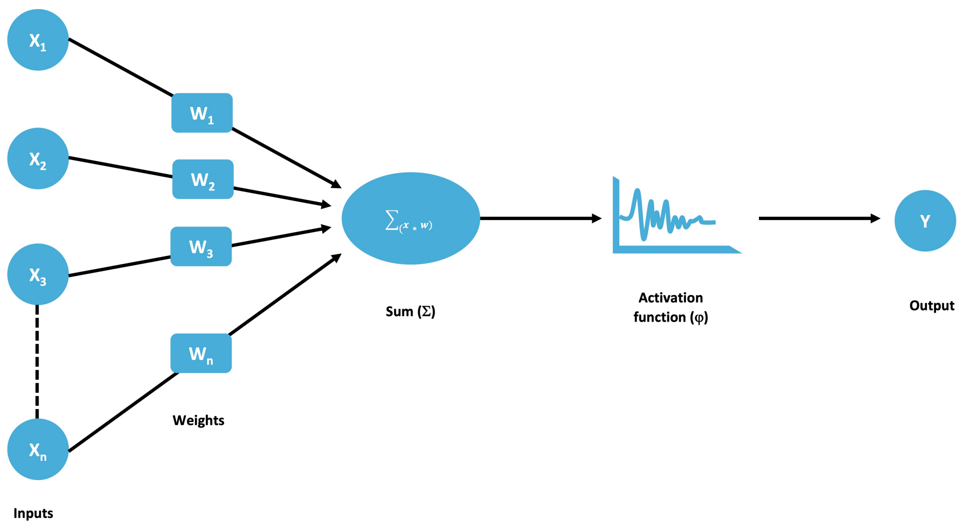


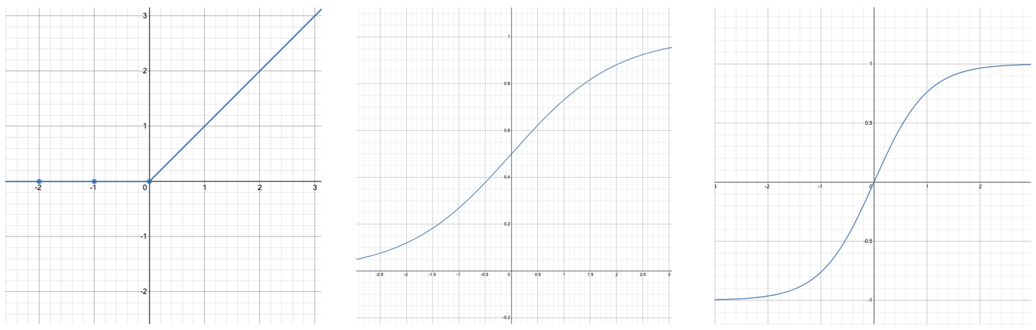
Figure 4.2: The overview of the Machine Learning perceptron structure.

4.2.2 Activation functions

The activation function φ takes the sum of the neurons to define the output by deciding whether they should be fired (activated). The concept of activation functions started in the late 1960s (Fukushima, 1969). Since Machine Learning gets more public attention, the activation functions enable better training of the deeper networks (Glorot, Bordes & Bengio, 2011). Today, there are many activation function options to choose from, as shown in Figure 4.3:

The **rectified linear unit (ReLU)** is the activation function that will ignore all the negative input values by shifting them to zero; otherwise, output the input values directly. It is useful when the negative values generally do not make sense, as shown in Figure 4.3a. It is the default activation function of many neural networks to produce the probability of identifying whether the input belongs to a specific class. The ReLU is described as:

$$y = \max(0, x) \tag{4.2}$$



(a) ReLU (b) Sigmoid (c) Tanh

Figure 4.3: The example of few popular activation functions.

The **Sigmoid** activation function is the most frequently used in neural networks, as shown in Figure 4.3b. The Sigmoid function aims to solve the non-linear nature problems by keeping the input values between 0 and 1. The advantage of using Sigmoid is that it can present the infinite values in the range (0,1) to avoid the vanishing of activation value to allow the learning to happen. However, the gradient vanishes when it gets smaller and converges to 0 or nothing to learn. The optimisation algorithm that reduces the training errors now can be attached to the local minimum values, which cannot maximise the model's performance. The Sigmoid function can be calculated by using the following formula:

$$y = \frac{1}{1 + e^{-x}} \tag{4.3}$$

The **Tanh** activation function is the hyperbolic tangent function that shifts the values to either -1 or 1, as shown in Figure 4.3c. Its structure is very similar to Sigmoid, but the range is defined as $(-1, +1)$ instead of $(0, +1)$. The most significant advantage of using Tanh over Sigmoid is that its derivative can cover more values to make the model learning and grading faster. The Tanh function can be calculated by using the following formula:

$$y = \frac{2}{1 + e^{-2x}} - 1 \tag{4.4}$$

The **Softmax** activation function, also called softargmax (Goodfellow, Bengio & Courville, 2016), is often used to normalise the neural network's output to the probability distribution over the predicted classes. Its structure is very similar to the Sigmoid function; hence, it performs well when used as a classifier. Its difference from other functions is preferred in the output layer, where the model needs to classify more than one class. The following Equation shows how Softmax function is implemented:

$$\sigma(\vec{v})_i = \frac{e^{v_i}}{\sum_{j=1}^{C} e^{v_j}} \tag{4.5}$$

Where σ is the Softmax value, $\vec{v}$ is the input vector, e^{v_i} is the standard exponential function of the input vector, C is the number of classes, and e^{v_j} is the standard exponential function of the output vector.

4.2.3 Loss functions

Humans make mistakes, so it is possible for machines to also make mistakes during the training process. In the beginning, the training process typically outputs the wrong prediction because it depends on the initial random weight values. It then needs to identify how far it was from the actual output values and recalculate the direction in the network to make the errors as small as possible. For this reason, the loss function is the primary method to handle neural network errors during the training process. There are different loss functions for various applications and training purposes. **Mean Square Error (MSE)** and **Mean Absolute Error (MAE)** are the most common loss functions. The MSE measures the average of square differences of predictions and the actual observations (shown in Equation 4.6). This loss function always gives the positive values, and the values closer to zero are better (Lehmann & Casella, 2006). It means that the predictions that give the values further away from the actual values are mostly to be ignored due to the consequence of the squaring.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{4.6}$$

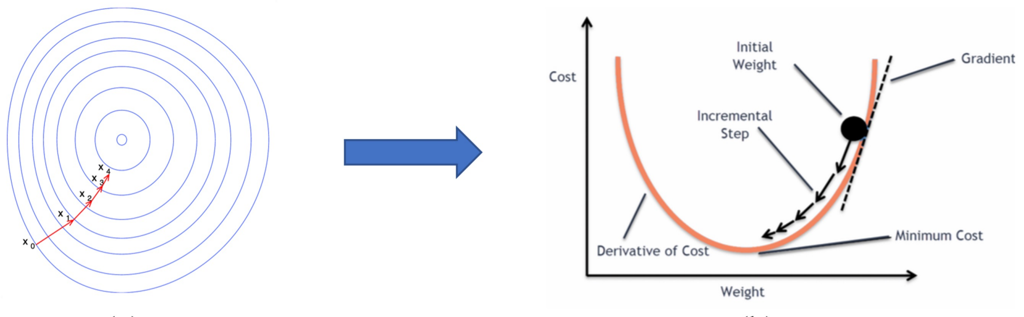
Where n is the number of training samples, i is the position of the training sample in the dataset, y_i is the ground truth value of the i^{th} training sample (observation), and the $\hat{y}_i$ is the prediction of the training sample at the position i^{th}.

The MAE, on the other hand, measures the average sum of absolute differences of predictions and the actual observations (shown in Equation 4.7). This loss function is more robust to the outliers as it will fit the data based on the median. However, it requires more complicated tools such as linear programming to compute the gradients, whereas the MSE is easier to implement. Hence, the MEA is less common than MSE, where the MSE is usually implemented as the default function to calculate the loss.

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{4.7}$$

4.2.4 Optimisers

After the loss is calculated, the neural network will make changes to itself to improve the performance. It is when the actual learning taking place, also called the **Back-propagation** process. The principle behind backpropagation is to go backwards in the network from the output to compute the gradient of the loss function corresponding to the weights. Another meaning is that the system will track back to the respected weights and change their values to make the error values as small as possible. The backpropagation idea was born in the 1970s as the general optimisation method of the complex nested automatic differentiation functions (Werbos, 1994). It was then stated to be the essential algorithm of the neural networks in the 1980s by Rumelhart, Hinton Williams (Rumelhart, Hinton & Williams, 1986). The popular method for minimising the error is **Gradient Descent (GD)** to find the local minimum of a differentiable function. It simply finds the coefficient values that minimise the function cost as far as possible, as shown in Figure 4.4.



(a) (b)

Figure 4.4: The illustration of gradient descent on a series of level sets (a) and how it could be used to update the weight values in the neural network (b).

The GD Δw_i of of the weight w at position i in the dataset can be calculated with the following Equation:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \tag{4.8}$$

Where ∂E is the derivative of the differential loss function E which could be one of the loss functions described in Section 4.2.3, ∂w_i is the derivative of the weight w at location i, and η is the constant called the **learning rate**. In the backpropagation process, picking a suitable learning rate is very important. The learning rate needs to be an appropriate value that is neither too high nor small. If it is too high, the system takes greater steps, and it may not reach the local minima, as shown in Figure 4.5a. On the other hand, if the learning rate value is too small, it may take much longer to reach the local minima or high resources consumption during training, as shown in Figure 4.5b.

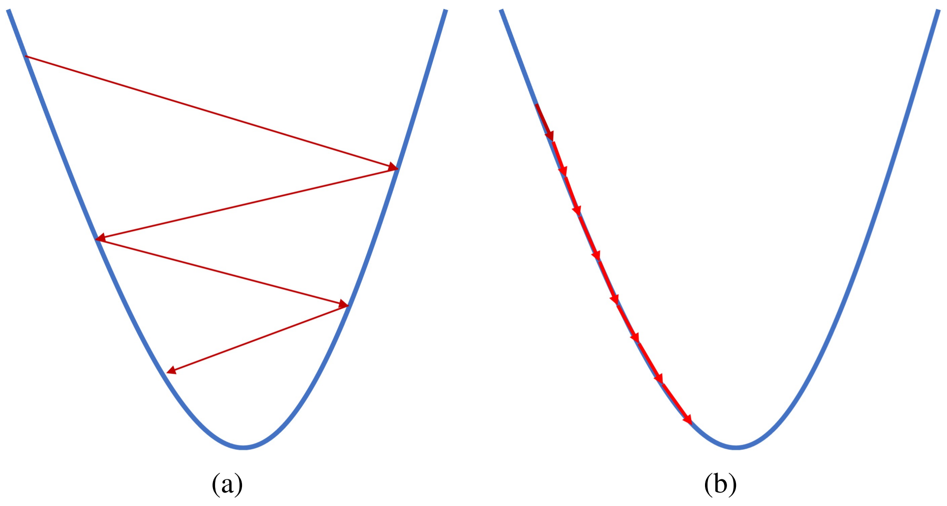


Figure 4.5: The outcome differences between using large value (a) and small value (b) for learning rate.

The **Stochastic gradient descent (SGD)** is the extension of the GD aiming to overcome the disadvantage of consuming higher memory during the training process when using the GD. In SGD, the derivative is computed by taking a point per time instead of loading the entire dataset of n points in GD. The most significant advantage is that the system requires less memory and reduces missing local minima. Hence, it is almost infeasible to use GD when the training dataset size is vast. The following Equation can calculate the value of SGD $f(x)$:

$$f(x) = \frac{\sum_{i=1}^{n} E_i(x)}{n} \tag{4.9}$$

Where $E_i(x)$ is the loss function of the training data at the location i in the dataset with the size n.

The **Adaptive moment estimation (Adam)** is one of the most popular optimiser functions, seen as the combination of SGD and RMSprop. The RMSprop keeps an exponentially decaying average of past squared gradients s_t, whereas the SGD keeps an exponentially decaying average of past gradients p_t. The Adam holds a running average of both gradients and can be expressed by using the following equations:

$$p_t = \beta_1 p_{t-1} + (1 - \beta_1) g(t) \tag{4.10}$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) g^2(t) \tag{4.11}$$

Where $\beta_1, \beta_2 \varepsilon [0, 1]$

4.2.5 Hidden layers

In reality, a neural network usually has more than one layer. It usually consists of the input layer that connects to each neuron in the following layers, and throughout many different layers, it will reach the final output layer. The layers between input and output are called hidden layers, where each neuron connects to all the neurons in the previous and the following layers, as shown in Figure 4.6. The networks with multi hidden layers are considered as the deep neural network or another famous term Deep Learning.

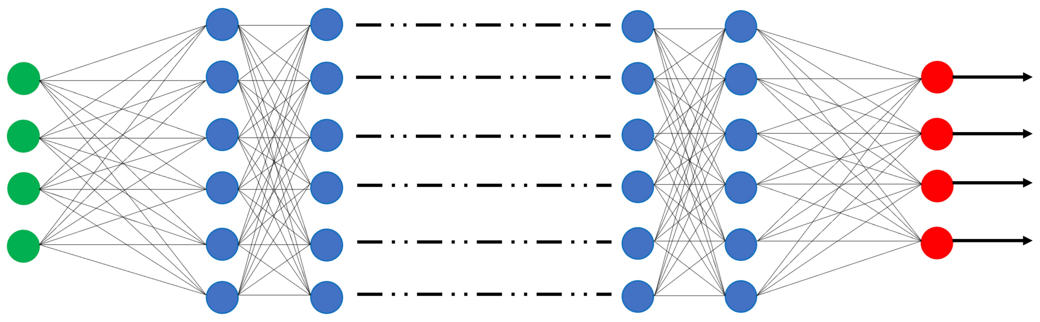
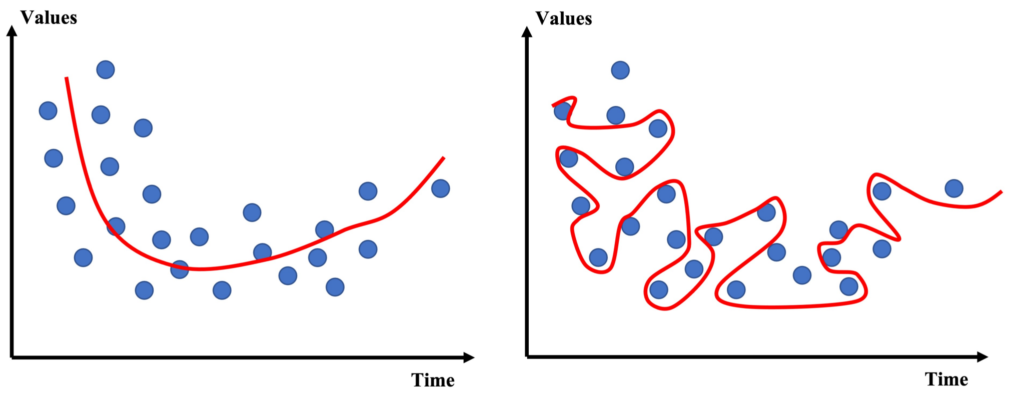


Figure 4.6: The example of a deep neural network with each neurons in the input layer (green) connecting with all neurons in the hidden layers (blue) to lead to the output layer (red).

4.2.6 Overfitting

In the deep neural network, less time for training can lead to the low performance of the model. However, it does not mean that a longer training time produces higher performance, but sometimes it could worsen. This situation is called overfitting (Figure 4.7b), where the model can perform exceptionally well in the training dataset but fails to predict the incoming samples or in the validation/test dataset (Hawkins, 2004). The overfitting occurs when the model has been over-trained and could learn the data noise in the training dataset, negatively impacting the unseen samples.



(a) Good fit (b) Overfitted

Figure 4.7: The good fit model (a) with the line follows the data structure, whereas the overfitted model (b) is too dependent on the data. Hence, it is likely to have a higher error rate on the new unseen samples.

The overfitting situation can be prevented by randomly cutting off the layers during the training process; this method is called the **dropout** (Srivastava, Hinton, Krizhevsky, Sutskever & Salakhutdinov, 2014). The dropout may be implemented in few or all hidden layers and the visible layers but not in the output layer. Another powerful method is **cross-validation**, where the initial training dataset is split into the smaller train-test set to tune the model faster (Stone, 1974). **Data augmentation** is another efficient method to reduce overfitting when the available dataset appears diverse (Shorten & Khoshgoftaar, 2019). The model can look at a single data sample from a different perspective to prevent the model from learning the dataset's characteristics.

4.3 Convolutional neural networks

Computer vision is the most popular applied field of Machine Learning and neural network. Computer vision deals with digital graphical content to gain a high-level understanding of the underlying meaning. The **convolutional neural network (CNN)** is the standard approach to most image processing tasks in the computer vision field. Since the revolutionary of Machine Learning and CNN (Krizhevsky et al., 2012), the CNN models can achieve nearly human-like accuracy in real-time (Acharya et al., 2020; Tan et al., 2020). The CNN uses a deep neural network structure by feeding the data through the hidden and visible layers, but some perform the convolutions (as shown in Figure 4.8).

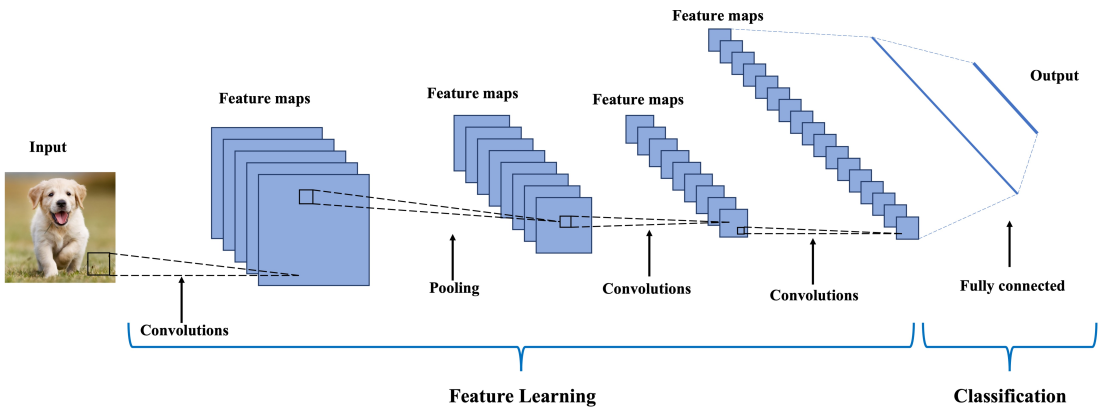


Figure 4.8: The data of a convolutional neural network flows from left to right. The network is divided into two different sections. The feature learning usually includes several convolution layers and pooling functions to discover the classification from the raw input data. The classification contains the fully connected layers (hidden layers) to predict a class label for a given example of input data.

4.3.1 Convolutional layers

The convolutional layers make CNN more efficient compared to the original neural networks in terms of computational power. For example, the network would need to process more than two million inputs with the standard high definition image of size 1280 × 720. So instead of using every single pixel as the input, the network will use the convolutional layers to slide over the input image and convolve the texture information. Each convolutional layer uses the filters or kernels to filter the characteristics of the current input texture (Shawe-Taylor, Cristianini et al., 2004). Each kernel has a corresponding feature map in a 3D feature maps stack of the output. The kernel can be seen as a single 3 × 3 or 5 × 5 matrix used for edge detection, blurring, sharpen, or other specific image processing tasks. The convolution matrix is different from the matrix multiplication, where it multiplies the value of each image pixel in the matrix and its corresponding value in the kernel. Then it will add the results together to present a single value corresponding to one cell in the feature map (Figure 4.9). The system will repeat the same procedure until it slides over the whole input image and produces a completed feature map.

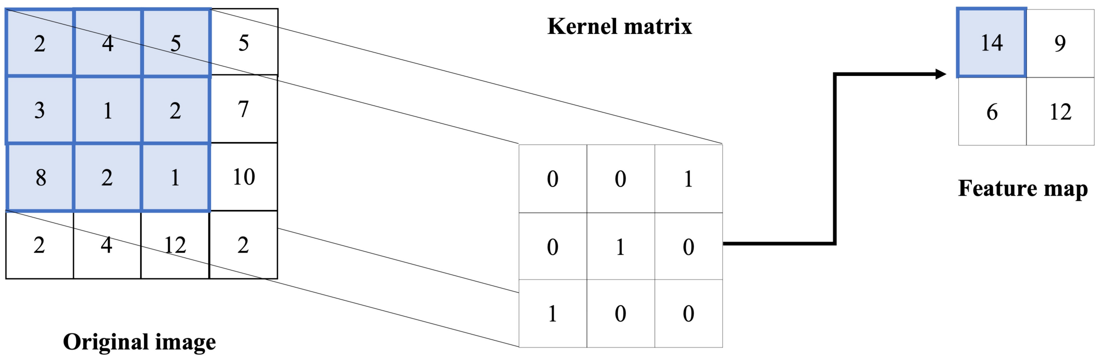


Figure 4.9: An example of convolutional operation.

However, in practice, it is essential not to change the size of the feature map. For example, if the size of a single input image is 4 × 4 and the kernel size of 3 × 3, then the output feature map would be 2 × 2. To keep the dimension of the feature map remaining unchanged, the sizes of the original input can be padded with zeros (Figure 4.10). This procedure is called “zero-padding”.

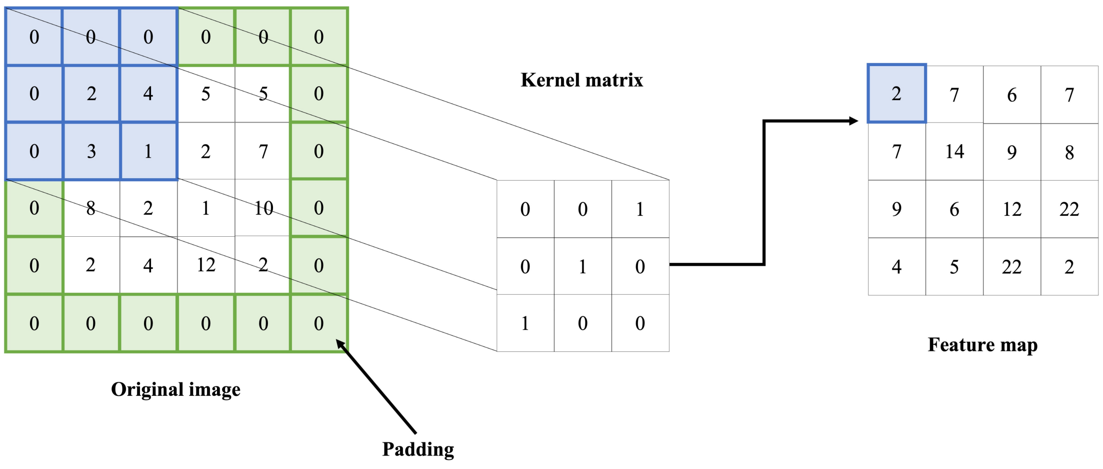


Figure 4.10: An example of convolutional operation with zero-padding.

4.3.2 Pooling layers

A single pooling layer is usually used to connect one convolutional layer with other. Its job is transferring the input to a lower dimension without losing the critical information; this procedure is called "sub-sampling". The most common pooling operator is max pooling, where it calculates the maximum value of a particular spatial region and discard the rest (Figure 4.11). Other popular pooling methods are average (Figure 4.12) and sum (Figure 4.13) pooling, where they take the average or the total of all values within the spatial region, respectively.

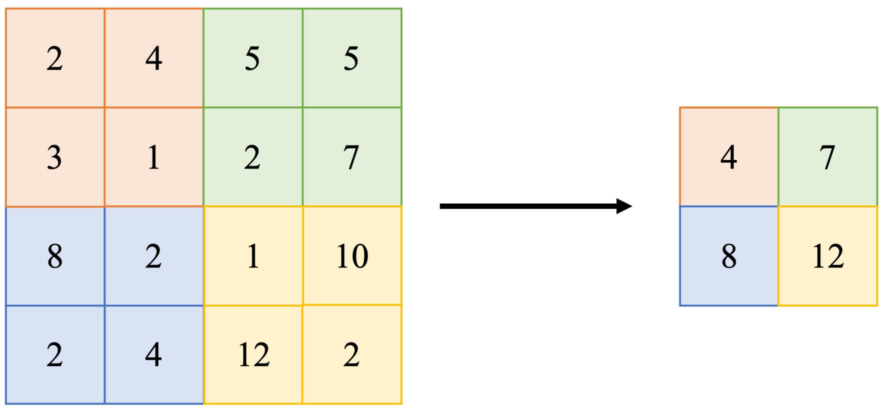


Figure 4.11: An example of max pooling with 2 × 2 layer and a stride of 2.

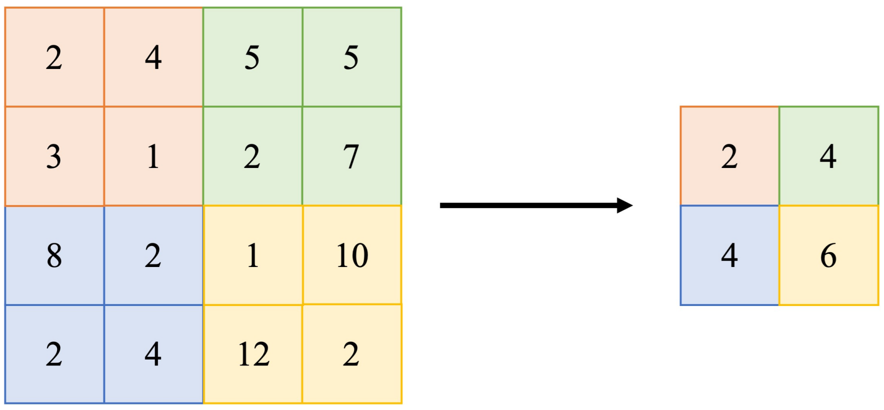


Figure 4.12: An example of average pooling with 2 × 2 layer and a stride of 2.

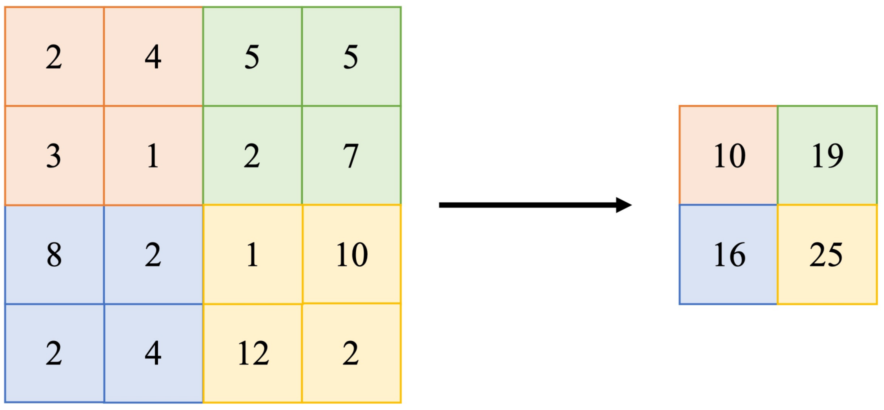


Figure 4.13: An example of sum pooling with 2 × 2 layer and a stride of 2.

4.3.3 Fully connected layers

The fully connected layers are usually sitting at the very end of the neural network. The shape of these layers is flattened or can be seen as a long vector instead of a matrix form. The role of the fully connected layers is to define the meaning of the previous layers to come up with a potential prediction. However, a neural network usually has more than one fully connected layer stacked next to each other, with the last layer acting as the prediction, which has multiple output neurons. Each neuron in the prediction layer represents a single class that model would like to predict.

4.4 Transfer Learning

Training an extensive deep neural network entirely from scratch is costly in time and computing power consumption. Especially for a smaller project where it is nearly impossible to have enough data required for training. Therefore, using a pre-trained network for training is more efficient in practice. The neural network can re-use the universal characteristics like edge, colour, and shape without retraining them again since the convolutional layers in the pre-trained network can recognise them easily. It means that the neural network only needs to run the backpropagation on the last fully connected layer to fit the purpose of the model. For example, the CNN models usually use the ImageNet (Krizhevsky et al., 2012) dataset for training, and it contains 1,000 different categories, or the model should have 1,000 neurons at the last layer. Hence, the final task is to modify the last layer to match the number of classes that the model needs (Figure 4.14).

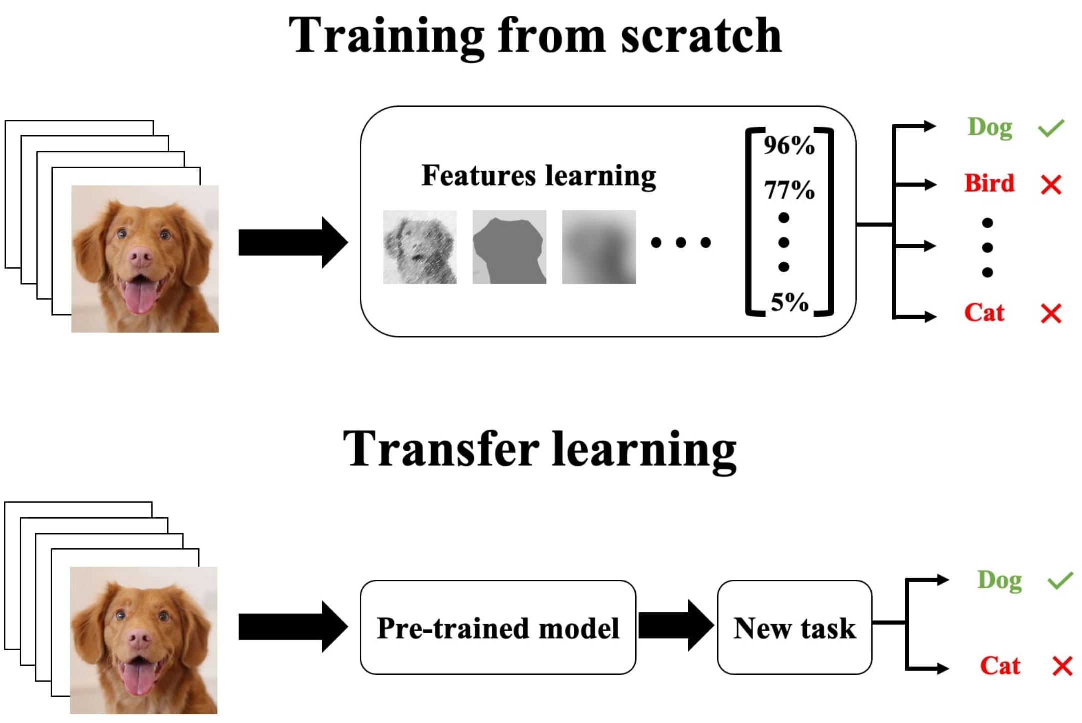


Figure 4.14: The comparison between transfer learning using a pre-trained model and model training from scratch.

4.5 Real-time object detection

Object detection using a CNN can be categorised into two different types: region nomination and regression. Region nominations such as R-CNN (Girshick, Donahue, Darrell & Malik, 2014), SPP-Net (He, Zhang, Ren & Sun, 2015), Fast R-CNN (Girshick, 2015), and Faster R-CNN (Ren, He, Girshick & Sun, 2015) use step-by-step detection strategy algorithms. They first extract the proposal regions on the image using selective search and then classify the image within the proposal regions. The output accuracies of these models are consistently above 80%. However, the frame per second (FPS) rate reduces dramatically. Only seven FPS is possible for Faster R-CNN, which is one of the fastest models but is still far from the real-time FPS standard. On the other hand, the YOLO model can reach 45 FPS, which is suitable for real-time detection tasks, especially for the Augmented Reality applications. It means that the YOLO model is a strong candidate for this thesis deep neural network backbone. YOLO (Redmon, Divvala, Girshick & Farhadi, 2016) uses the regression method to predict the object bounding box and class name instead of using the proposed region method. However, due to the simpler network architecture, the detection accuracy reduces when the frame rate increases.

4.5.1 You Only Look Once (YOLO)

The main principle of YOLO is using the entire image as the input to the network and directly returning the bounding box coordinates and corresponding class name. YOLOv3 (Redmon & Farhadi, 2018) is the next generation of YOLOv2 (Redmon & Farhadi, 2017) and contains significant improvements. YOLOv2 uses Darknet-19 as its backbone and an additional 11 object detection layers. However, YOLOv2 struggles with detecting small objects whereas the YOLOv3 can perform the state-of-the-art by using residual blocks, skip connections and upsampling, as shown in Figure 4.15. It uses Darknet-53 as the backbone, which is reported to be more efficient than Darknet–19, ResNet-101 and ResNet-152 (He, Zhang, Ren & Sun, 2016).

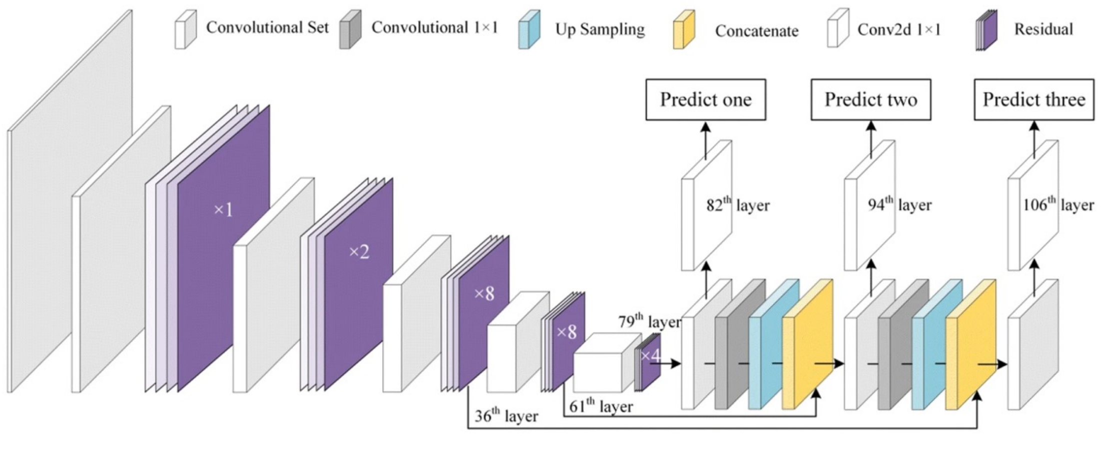


Figure 4.15: YOLOv3 neural network architecture.

The main difference of YOLOv3 to its predecessors is predicting three different scale levels. Each of the input images is downsampled by 32, 16 and 8, respectively. The detection first made at 82^{nd} layer after the downsampling process of the original input from the previous 81 layers. The 81^{st} layer has a stride of 32, meaning that if the image size is 416×416, then the resultant feature map would be $13 \times 13 \times depth$. Then the feature map from 79^{th} layer upsampled by 2 to dimensions of $26 \times 26 \times depth$ is depth concatenated with the feature map from 61^{st} layer. The combination feature maps are subjected through a few convolutional layers before reaching the 94^{th} layer, where the second detection occurs. The same procedure is executed again where the feature map at 94^{th} layer is subjected through a few convolutional layers and depth concatenated with the feature map from the 36^{th} layer. The final detection occurs at 106^{th} layer yielding the feature map of $52 \times 52 \times depth$. Detections in three different scale levels help address the limitations around detecting small objects in YOLOv2. The 82^{nd} prediction layer is responsible for detecting large-scale objects. The last prediction layer is responsible for detecting the small-scale objects, whereas the 94^{th} prediction layer is suitable for medium-scale objects. The predictions are made by applying 1 x 1 detection kernels on the feature map, as shown in Figure 4.16. At the tensor procedure level, the YOLOv3 network divides the input image into an $S \times S$ grid of cells. Each cell is responsible for predicting bounding boxes B and class probabilities C of the potential objects whose centres are inside the grid cell. Each bounding box has five attributes: four bounding box coordinates (t_x, t_y, t_h, t_w) and object confidence score P_o. The confidence score represents the probability of a box containing an object and usually falls between 0 and 1.

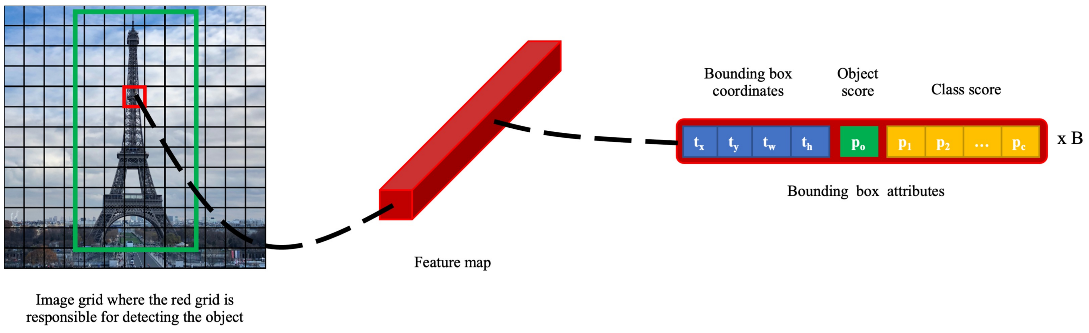


Figure 4.16: YOLOv3 detection procedure at tensor level.

In YOLOv3, the loss function L can be calculated using the following Equation:

$$L = Error_{boxes} - Error_{confidence} - Error_{classes} \tag{4.12}$$

Where the $Error_{boxes}$ is the bounding box coordinate regression, which can be defined as follows:

$$Error_{boxes} = \lambda_{coordinate} \sum_{i=0}^{S^2} \sum_{j=0}^{N} l_{ij}^{obj} \left[(t_x - t'_x)^2 + \left(t_y - t'_y\right)^2 + (t_w - t'_w)^2 + (t_h - t'_h)^2 \right] \tag{4.13}$$

Where $S^2 = S \times S$ cells, N is the number of bounding boxes in each predicted cell with the corresponding coordinates (t_x, t_y, t_h, t_w), and $\lambda_{coordinate}$ is the coordinate error weight. The $Error_{confidence}$ is used to calculate the loss of confidence of the existing object in the bounding box which can be defined as follows:

$$\begin{aligned} Error_{confidence} = \sum_{i=0}^{S^2} \sum_{j=0}^{N} l_{ij}^{obj} \left[c'_i log(c_i) + (1 - c'_i) \, log \, (1 - c_i) \right] + \\ \lambda_{confidence} \sum_{i=0}^{S^2} \sum_{j=0}^{N} l_{ij}^{confidence} \left[c'_i log(c_i) + (1 - c'_i) \, log \, (1 - c_i) \right] \end{aligned} \tag{4.14}$$

Where c is the number of classes, $\lambda_{confidence}$ is the confience error weight. The final $Error_{classes}$ can be calculated using the following Equation:

$$Error_{classes} = \sum_{i=0}^{S^2} l_i^{obj} \sum_{c\epsilon classes} [p_i'(c)log(p_i(c)) + (1 - p_i'(c))log(1 - p_i(c))] \quad (4.15)$$

4.5.2 Model evaluation metrics

The following metrics used in MS COCO (Lin et al., 2014) and Pascal VOC (Everingham, Van Gool, Williams, Winn & Zisserman, 2010) dataset are beneficial to measure the accuracy of the object detection model:

The **recall** metric evaluates how well the objects have been detected (Equation 4.16) and can be calculated by obtaining the intersection-over-union (IoU) value. The IoU determines the overlap between the predicted bounding and the ground truth over their area union (Figure 4.17). If the IoU value is greater than a threshold value (generally set to 0.5), it is considered as true-positive (TP), otherwise false-positive (FP). The false-negative (FN) is made when no prediction is made for a particular ground truth bounding box.

$$recall = \frac{TP}{TP + FN} \quad (4.16)$$

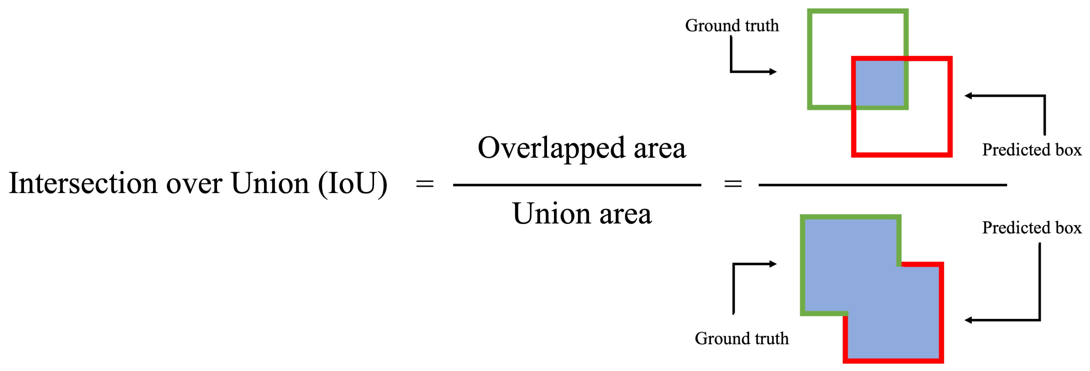

Ground truth
Predicted box
Intersection over Union (IoU) = Overlapped area / Union area =
Predicted box
Ground truth

Figure 4.17: The Intersection Over Union can be calculated by dividing the area of overlap between the bounding boxes by the area of the union.

The mean average precision (mAP) metric calculates the mean value of average precision over the IoU thresholds, where the precision can be obtained using Equation 4.17. In precision equation, the value of false-negative is replaced by false-positive (FP).

$$precision = \frac{TP}{TP + FP} \tag{4.17}$$

The average precision AP can then be defined by finding the area under the precision-recall curve, as shown in the following Equation:

$$AP = \int_0^1 Precision(recall)dr. \tag{4.18}$$

The final mAP can be calculated as:

$$mAP = \frac{1}{n}\sum_{c=1}^{c=n} AP_c \tag{4.19}$$

where AP_c is the AP value of the class c and n is the number of classes. The YOLOv3 model also uses $mAP0.5$ or mean average precision 50 with a static threshold of 0.5 for IoU. Another mAP metric is $mAP[0.5 : 0.95]$, where the threshold is from 0.5 to 0.95.

4.6 The issue of Machine Learning and the proposed method

Machine learning is potent for solving classification problems; however, there are also many limitations of standard algorithms. One is the lack of understandings of the exact underlying laws of the input data. In other words, to estimate the best output, a large number of input data must be needed that covers as many test cases as possible. For instance, to use the Machine Learning method to efficiently recognise one particular book page (out of a hundred pages - a classification problem), the model would need to train with hundreds or thousands of pictures of each book page. The training takes time, but it can be automated; however, the acquisition of the dataset is not. Hundreds of pictures still need to be acquired manually. This is very time-consuming and labour-intensive. If the dataset could be generated synthetically, the data preparation process could be much more efficient, and the results could be improvable.

Task	**Synthetic**	**Manual**
Preparing 30 raw images	30 minutes	30 minutes
Preparing 60 backgrounds	60 minutes	60 minutes
Preparing 10k of trainable images	15 minutes	350 hours

Table 4.1: Time requirements for synthetic and manual data generation

The newly method purpose is to achieve Augmented Reality marker recognition and increase the marker detection rate by using the power of Machine Learning while reducing the limitation of manual data preparation (data annotation). A Machine Learning model only becomes stable after being trained by thousands of manually labelled images. This proposed method aims to achieve a stable Machine Learning detection model with minimal effort. The details of the proposed method implementation is presented in Chapter 5, and it requires two key steps:

Synthetic dataset generation. This newly proposed method aims to generate a synthetic dataset and label them automatically using various computer graphics rendering techniques. The figures of chosen deep neural network trainable classes are randomly blended into different natural-looking backgrounds and other ambient lighting conditions and mixed camera orientation. This new way of data-generating is much quicker than traditional data annotation, where it could produce up to 20 training dataset images per second on a graphics processing unit (as shown in Table 4.1). It also gives an excellent opportunity to regenerate or modify the dataset faster for an additional training class or improve model quality.

Deep neural network training with synthetic dataset. The synthetic dataset generated in the previous step could be trained through an object recognition deep neural network model to boost the Augmented Reality marker detection performance.

4.7 Summary

This Chapter reviews the principles behind deep learning using neural networks. It also nominated the potential object detection model for this project's main deep learning training backbone. The YOLO model clearly shows that it could process and classify the input images in real-time, which is suitable for Augmented Reality applications. However, the time-consuming and labour-intensive issues of data annotation are the primary drawbacks during the Machine Learning training process. Hence, the new method that uses digital graphics to produce the data synthetically is proposed to overcome the current data annotation issues and improve the model training performance. The next Chapter focuses on the design and implementation, such as generating data, what training parameters the model uses to train the model, and the application prototype implementation.

Chapter 5

Design and Implementation of Synthetic Data Annotation Method (SARM)

Parts of this Chapter have been published in ***paper 1***, ***paper 2***, *and* ***paper 4*** *listed in the publication list*

This Chapter will cover the methods that are used to achieve the goals of this thesis that described in the **Section 1.2**. Furthermore, it will focus on the following key aspects:

- The proposed dataset generation technique ***(Section 5.1)***.
- The discussion about the object detection deep neural network training process ***(Section 5.2)***.
- Describe how the deep neural network outcome would be implemented onto the Augmented Reality - iOS application ***(Section 5.3)***.
- Describe the frameworks and software requirements are used within this thesis ***(Section 5.4)***.

5.1 Synthetic data generation

Developing an accurate object detection requires training data that has good quality and can represent the real-world environment well is a challenging task—especially when developing a classifier where the data and environments are unknown. One example is when the book images are the target markers for an AR application which occurs infrequently and could be existing in many unexpected environments. Therefore, there are not many existed data that could be used to train the deep neural network model. In these rare scenarios, synthetic data appears to be a helpful method for generating high-quality and diverse training data in the minimum amount of time. To generate the synthetic dataset, it is necessary to collect all possible images used as the target markers in the AR application as the first step. Identifying all possible complex natural environment situations such as lighting, orientation, and backgrounds are the most significant task that helps to improve the quality of the training dataset. For example, if the application will be used primarily by students and teachers, the backgrounds should be indoor, and the light conditions are warm and medium-level. Once all necessary environment components are identified, they can be imported into a game engine such as Unity (Haas, 2014) to generate the image dataset used for deep neural network training and testing, as shown in Figure 5.1. This proposed method aims to give a significant improvement in dataset quality and data collection time.

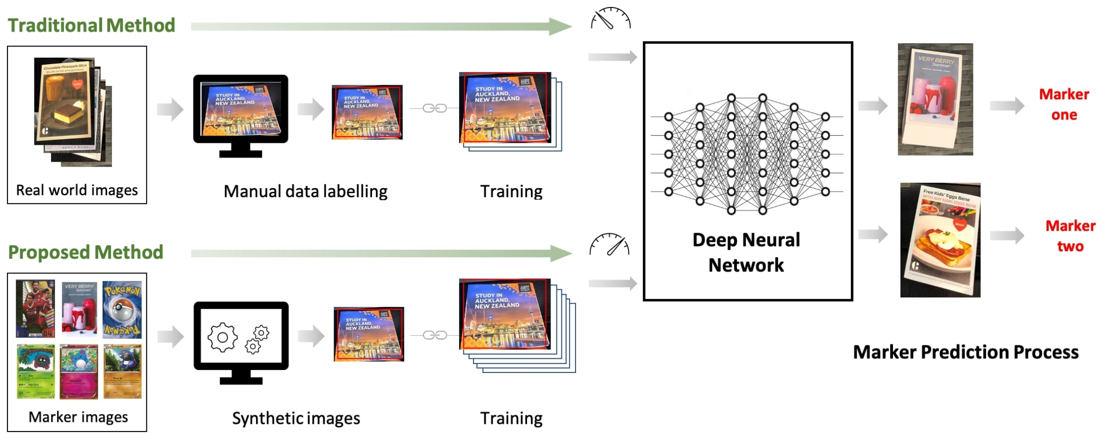


Figure 5.1: **Overview of the comparison of the proposed method and traditional data labelling technique**. The traditional method is done with manual data labelling that requires more time and labour cost to complete, while the proposed method can reduce these drawbacks by using the auto rendered computer graphics to generate the synthetic dataset for the deep neural network model.

The marker image pool contains the AR marker images, saved as individual image files. Each image width will be cropped to a size of 1280. For the background images, over 1,400 indoor images, including day and night time, are used as the "background images pool", and they are cropped to the sized of 1,500 × 2,250. The process of synthetic data generation is described as follows (Figure 5.2).

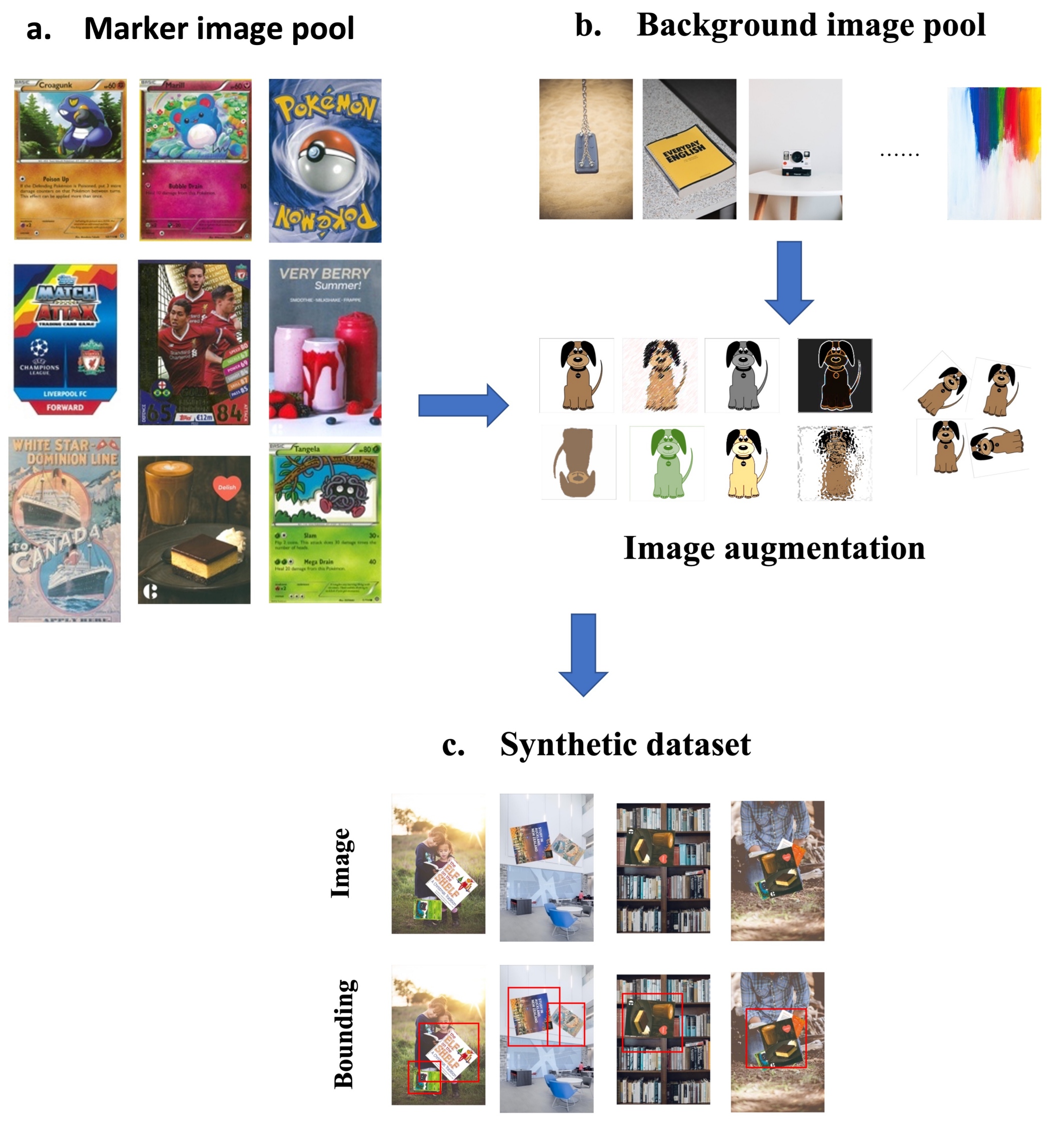


Figure 5.2: **Synthetic dataset preparation process.** **(a)** The real-world images of different Augmented Reality markers stored in the marker image pool. **(c)** Synthetic dataset generation where the images are generated by combining the AR markers with random background images **(b)** together with various augmentation filters. The details of generated ground-truth label (bounding box) are shown at the bottom.

Firstly, one or multiple images will be randomly selected from the "marker images pool" and carefully pasted to the transparent canvas of size 1,500 × 2,250. The probability of displaying single or multiple images on the same background is set to 50.

Secondly, another image of the background is selected from the "background images pool". The selected marker images then will be augmented by applying random filters such as rotation, the x and y coordinates, scaling, blur, noise. However, the coordinate values are restricted to a specific range so that the signs do not exceed the background size. They are dependent on the size of the marker image after scaling and its rotation angle.

Lastly, the AR marker is pasted to the transparent canvas according to the described parameters, and the canvas will be pasted on the top of the selected background. During the synthetic dataset generation, the bounding box with the same size as the marker image is created along with its label name. The size and the coordinates of the bounding boxes are normalised between 0 and 1 by dividing by the width and height of the background. This procedure can generate image size of 1,500 × 2,250 with the AR markers randomly oriented inside a random indoor background. It takes three hours to produce over 40,000 trainable images and their corresponding labels files. While in the real world, it would take weeks or months to complete a similar process.

5.1.1 Augmented reality marker images

There are approximately 22 different rectangular-shaped images integrated as the marker images. Each of these images has a different design, and some of them have a very similar colour. This setup helps to qualify how good the dataset is in identifying the similarity objects. The images were from four different categories: (1) trading/business cards, (2) posters, (3) children's educational books, and (4) food advertisements. The details of target markers are presented in Table 5.1 and Figure 5.3.

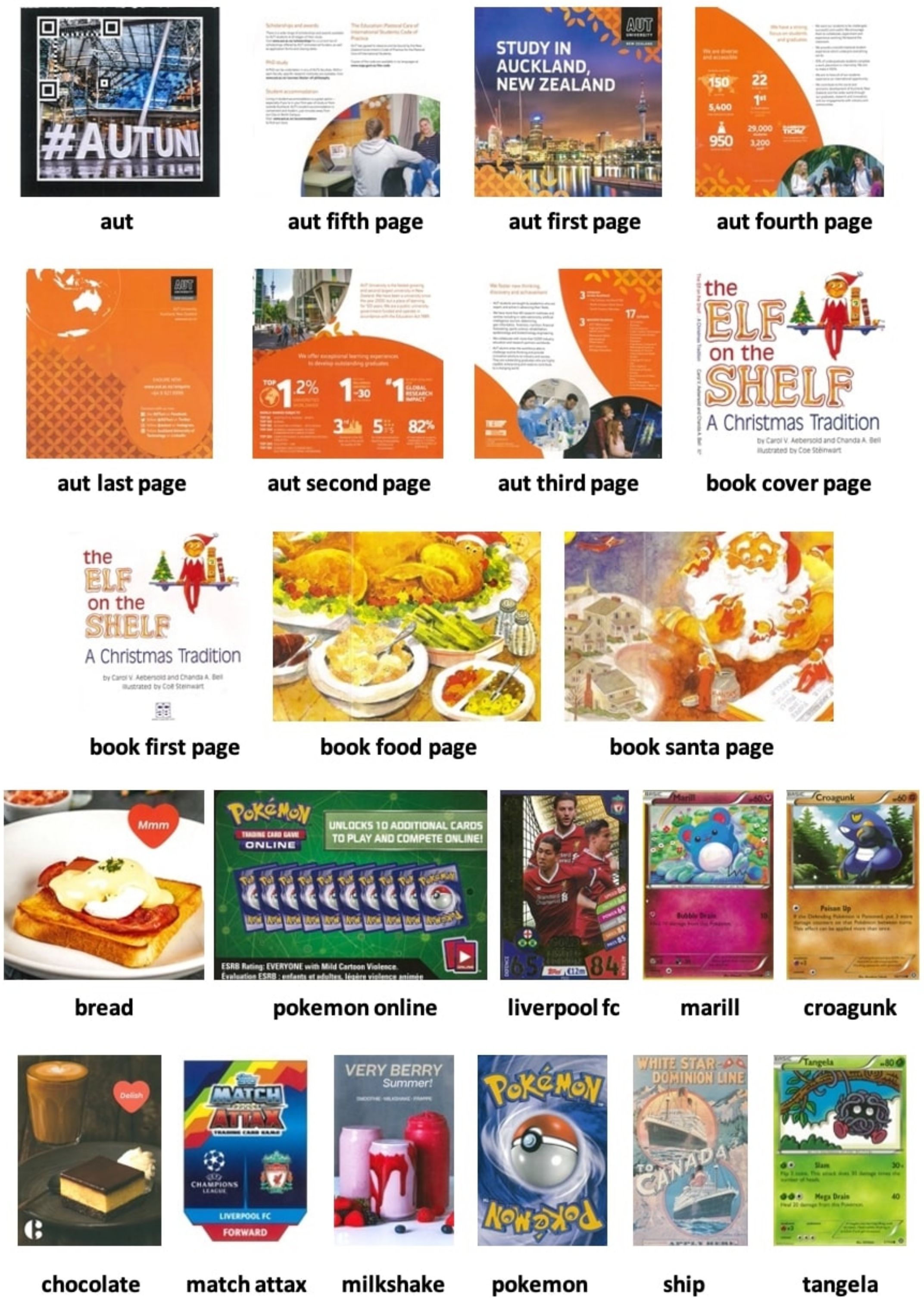


Figure 5.3: The Augmented Reality marker images with their label name are described below each image.

Label name	Width	Height	Sample generated
aut	9 cm	5.5 cm	3,890
aut fifth page	21 cm	21 cm	3,309
aut first page	21 cm	21 cm	3,179
aut fourth page	21 cm	21 cm	3,319
aut last page	21 cm	21 cm	3,324
aut second page	21 cm	21 cm	3,285
aut third page	21 cm	21 cm	3,427
book cover	21 cm	21 cm	3,205
book first page	21 cm	21 cm	3,192
book food page	21 cm	21 cm	3,853
tangela	6 cm	8.5 cm	2,514
ships	35 cm	60 cm	2,258
pokemon online	8.5 cm	6 cm	3,869
pokemon	6 cm	8.5 cm	2,542
milkshake	7 cm	11 cm	2,308
match attax	6 cm	8.5 cm	2,518
marill	6 cm	8.5 cm	2,552
liverpool fc	6 cm	8.5 cm	2,356
coragunk	6 cm	8.5 cm	2,558
chocolate	13 cm	17 cm	2,763
bread	13 cm	17 cm	2,870
book santa page	21 cm	21 cm	3,877

Table 5.1: Details of target markers and their label name, width, height and number of generated samples in the dataset.

5.1.2 Data labelling

YOLO is used as the primary deep neural network for this thesis and is described in detail in Section 4.5.1. In YOLO, the bounding box has four values, **[object_id,**

x_centre, y_centre, width, height], as shown in Figure 5.4. **[object_id]** represents the number corresponding to the object index in the class names list. **[x_centre, y_centre]** represents the centre point coordinate of the bounding box, which is normalised to between 0 and 1 by dividing by the width and height of the image. **[width, height]** represents the width and height of the bounding box, which is normalised to between 0 and 1 by dividing by the width and height of the image.

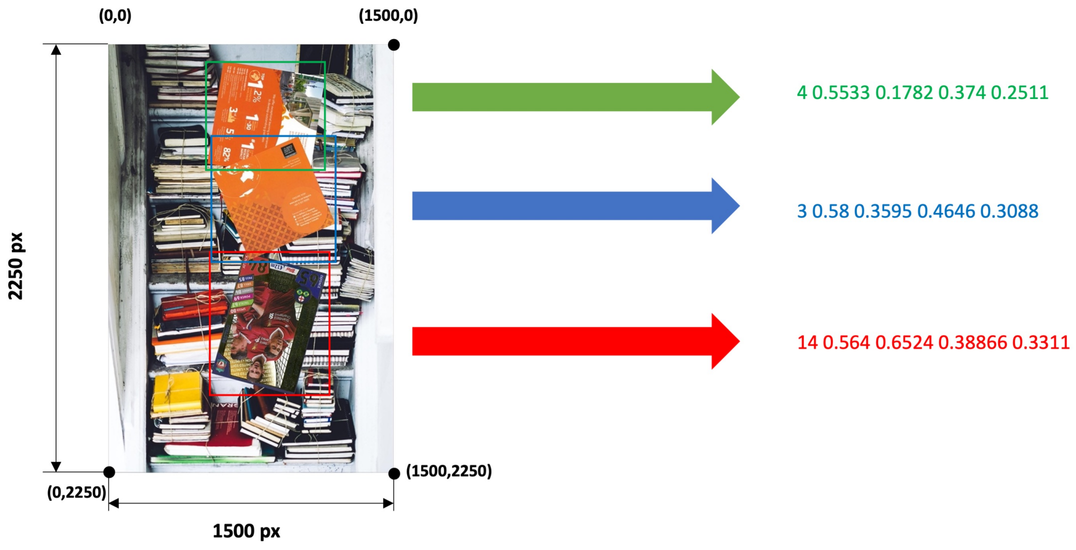


Figure 5.4: Examples of formats representing coordinates of a bounding box in YOLO training.

5.2 Deep neural network training

The original YOLOv3 was evaluated on the Microsoft COCO (Common Objects in Context) dataset (Lin et al., 2014). Most of the training parameters remain unchanged, as described in the original YOLO paper (Redmon & Farhadi, 2018). However, the training data is replaced by the proposed synthetic data, as described in Section 4.5.1. This could be done by changing the number of filters and classes in each yolo layer in the **.yaml** file. The number of filters can be calculated using the following Equation:

$$filters = (4 + 1 + C) \times 3 \tag{5.1}$$

Where C is the number of classes, or the $filters$ would be 81 if there are 22 different classes available for classification. The network is trained for 80 iterations using 80% of the dataset for training and the rest for validation. The batch size of 16 and sub-division of 8 matched the training hardware requirement for Nvidia RTX 2080 Super. Pytorch (Paszke et al., 2019) is used as the training framework instead of Darknet due to its Core ML conversion capability for implementing iOS applications. The full details of the training parameters are presented in Table 5.2.

Hyper parameter	**Value**
Activation	Linear, Leaky
Backbone	Darknet-53
Batch size	8
Decay	0.0005
Epochs	80
Filter size	64, 128, 256, 512, 1024
Learning rate (final)	0.2
Learning rate (initial)	0.01
Momentum	0.937
Number of classes	22
Optimiser	SGD
Sub-division	4

Table 5.2: Training parameters and their values for the deep neural network.

The in-training data augmentation techniques are also applied to increase available data during the training process. They are applied randomly to the images in the dataset. The augmentation techniques are used during the model training process includes:

Reflection is the technique to flip the image vertically or horizontally. The reflection probability value is set to 50% to allow the images are flipped in both horizontal and vertical ways. Hence, the reflection points of an image (x, y) over the $x - axis$ and $y - axis$ to the new point (x', y').

Saturation is the famous value to change the saturation value of the image or defines the portion of grey in a unique colour. The probability for changing the value of saturation is set to $\pm 70\%$.

Brightness is the component that works in conjunction with saturation to control the image's brightness or intensity. The brightness value usually falls from 0 to 100, where 0 is the most darkness and 100 is the brightest level of colour. The probability for changing the value of brightness is set to $\pm 40\%$.

Hue is the value that defines the portion of red, yellow, green, cyan, blue, and magenta as a number from 0 to 360 degrees. The probability for changing the value of the hue is set to $\pm 1.5\%$.

Scale is another technique to rescale the image by a particular portion value of the width or height of the image. This implementation's percentage change of scaling is set to $\pm 50\%$.

The **translation** is one of the most common used augmentation techniques to relocate the image along the $x - axis$ or $y - axis$.

5.3 iOS application implementation

The iOS application was built using Xcode and Swift 5.0 and developed for iOS 14.0 or above. The application was tested on the iPhone X, but it should also work on other recent iPhone models that support ARKit 4.0 and have the Bionic A11 chip or later built-in.

5.3.1 Integrating the pre-trained model to Xcode

The neural network model (as discussed in Section 5.2) usually takes much computational power. However, smaller computational devices such as smartphones can efficiently run most pre-trained neural network models. CoreML format is used for deploying Machine Learning models into Apple built products such as iPhone, iPad, Apple Watch, and Mac applications. The model can be integrated and accessed using the CoreML framework installed on macOS devices. Apple also allows training the neural network model directly on macOS devices using the Create ML application. However, Create ML has limitations on the number of functionalities that can be accessed and modified. The most approachable way is training the neural network using PyTorch, then convert the PyTorch pre-trained model into CoreML format. This conversion can be broken down into the following steps.

Step 1 - PyTorch to ONNX. The ONNX is an open cross-platform deep learning model that helps developers move their trained models into different training frameworks. The converter then takes the pre-trained PyTorch model (**.pth**) as the input, and instead of running on the actual neural net, it will identify **torch.onnx.export** as the built-in PyTorch API to export to an ONNX formatted model (Figure 5.5).

Exporting the ONNX format from PyTorch is essentially tracing the network, and the system internally runs the network on ’dummy data’ to generate the graph, as shown in

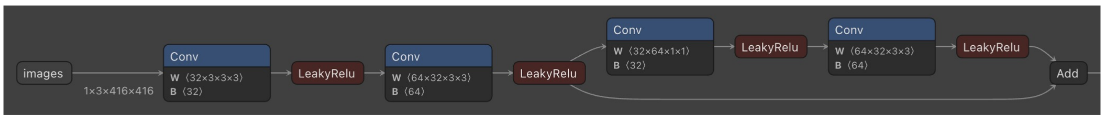


Figure 5.5: The example of ONNX exported model visualisation.

the example below:

```python
# Load PyTorch model
model = attempt_load("input.pt", map_location=torch.device('cpu'))
model.eval()
labels = model.names

# Checks
gs = int(max(model.stride))
opt.img_size = [check_img_size(x, gs) for x in opt.img_size]

# Input
img = torch.zeros(1, 3, *opt.img_size)

# Update model
for k, m in model.named_modules():
    m._non_persistent_buffers_set = set()
    if isinstance(m, models.common.Conv):
        if isinstance(m.act, nn.Hardswish):
            m.act = Hardswish()
        elif isinstance(m.act, nn.SiLU):
            m.act = SiLU()

model.model[-1].export = True
y = model(img)

# TorchScript export
```

```
    try:
        # filename
        f =  'input.torchscript.pt'
        ts = torch.jit.trace(model, img)
        ts.save(f)
    except Exception as e:
        print('TorchScript export failure: %s' % e)

# ONNX export
    try:
        import onnx

         # filename
        f = 'input.onnx'
        torch.onnx.export(model, img, f, verbose=False, opset_version=12, input_names=['images'],
                                      output_names=['classes', 'boxes'] if y is None else ['output'])

        # Checks
        # load onnx model
        onnx_model = onnx.load(f)
        # check onnx model
        onnx.checker.check_model(onnx_model)
    except Exception as e:
        print('ONNX export failure: %s' % e)
```

Listing 5.1: Python code to convert Pytorch model to ONNX format.

Step 2 - ONNX to CoreML. The ONNX can then be converted to the CoreML model format by using the built-in **onnx_coreml** package, as shown in the example below:

```python
import sys
from onnx import onnx_pb
from onnx_coreml import convert

model_in = "input.onnx"
model_out = "output.coreml"

model_file = open(model_in, 'rb')
model_proto = onnx_pb.ModelProto()
model_proto.ParseFromString(model_file.read())
coreml_model = convert(model_proto, image_input_names=['0'],
image_output_names=['186'])
coreml_model.save(model_out)
```

Listing 5.2: Python code to convert ONNX format to CoreML format.

The CoreML model then can be imported by dragging and dropping the model file into the Xcode project. Xcode will generate a new Swift class for the newly imported model, and it can easily be accessed by creating an instance of this class.

5.3.2 Class diagrams

The class diagram in Figure 5.6 presents the relationship between different Swift classes and how they fit and work together under the application system. The diagram is divided into three parts: (1) **ViewController**, (2) **Yolo**, and (3) **AlteredImage**. The **ViewController** class (code detailed shown in Appendix A.1) is the main controller of the application that holds the **ARSCNView**. This class is responsible for rendering the virtual objects and updating the application states. Every instruction is executed on the main UI thread as the system does not capture the camera's input frame in this class. Hence, there is no main thread blocking issue due to the video feedback execution.

The **Yolo** class (code detailed shown in Appendix A.2) is where the actual deep neural network predicts the incoming frame from the camera. The prediction time interval is set to 0.03 seconds, which means the system will run the prediction process every 30 milliseconds if no prediction process is currently running. This set-up helps to reduce the system workload by classifying a single frame for every given interval of time instead of all of the frames. Each predicted frame is converted to **MLMultiArray** and fed into the deep neural network model. The advantage of working directly with MLMultiArray's memory is that this speeds up the CoreML prediction performance significantly. Every process within this class is run on the background thread to avoid blocking the main UI thread. After the bounding box of the input frame is found, the system starts searching for the closest rectangle in the 3D world that matches the predicted bounding box.

After identifying the matched rectangle in the natural scene, the **AlteredImage** class (code detailed shown in Appendix A.3) will keep a copy of the rectangle as the reference image and create a 3D plane of the rectangle with its exact dimension in the real world. The virtual object is retrieved based on the predicted identification and rendered in the scene using **SCNNode** class. Each virtual object node is grouped under one root node, meaning all nodes are defined relative to the transformation or orientation of the same root node. The reference image tracking time interval is set to 0.03 seconds so that the system will start searching for the saved reference image every 30 milliseconds, and it will begin the actual YOLO prediction process if the reference image cannot be found via the **AlteredImageDelegate** protocol. Hence, this reduces the workload for the neural engine.

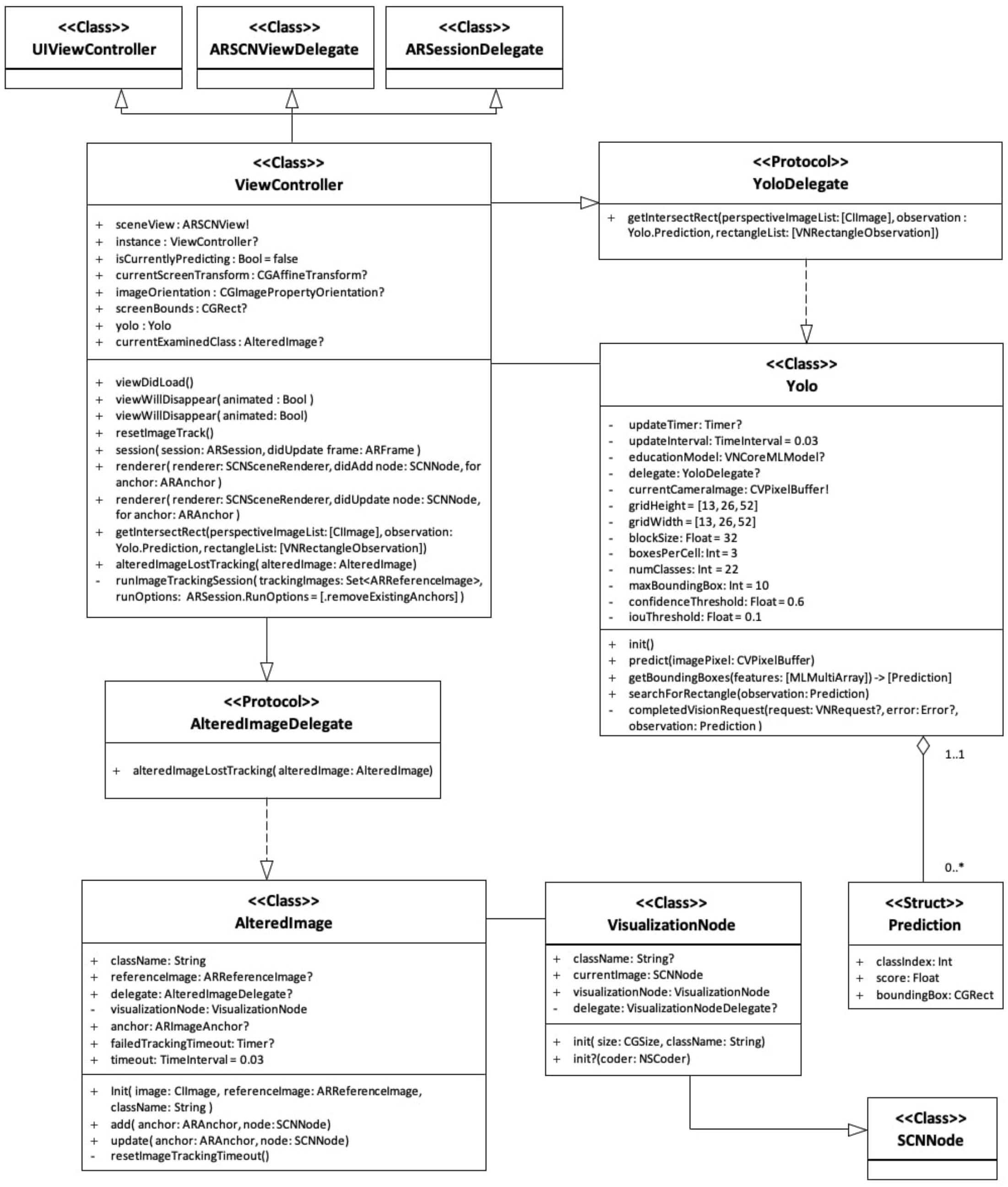
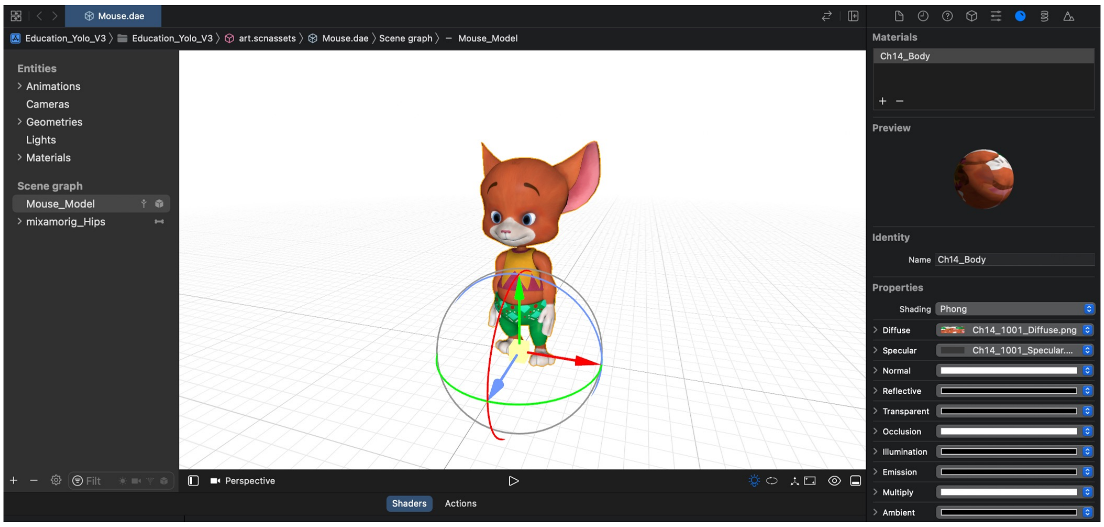

<<Class>>
UIViewController
<<Class>>
ARSCNViewDelegate
<<Class>>
ARSessionDelegate
<<Class>>
ViewController
+ sceneView : ARSCNView!
+ instance : ViewController?
+ isCurrentlyPredicting : Bool = false
+ currentScreenTransform : CGAffineTransform?
+ imageOrientation : CGImagePropertyOrientation?
+ screenBounds : CGRect?
+ yolo : Yolo
+ currentExaminedClass : AlteredImage?
+ viewDidLoad()
+ viewWillDisappear(animated : Bool)
+ viewWillDisappear(animated: Bool)
+ resetImageTrack()
+ session(session: ARSession, didUpdate frame: ARFrame)
+ renderer(renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor)
+ renderer(renderer: SCNSceneRenderer, didUpdate node: SCNNode, for anchor: ARAnchor)
+ getIntersectRect(perspectiveImageList: [CIImage], observation: Yolo.Prediction, rectangleList: [VNRectangleObservation])
+ alteredImageLostTracking(alteredImage: AlteredImage)
- runImageTrackingSession(trackingImages: Set<ARReferenceImage>, runOptions: ARSession.RunOptions = [.removeExistingAnchors])
<<Protocol>>
YoloDelegate
+ getIntersectRect(perspectiveImageList: [CIImage], observation : Yolo.Prediction, rectangleList: [VNRectangleObservation])
<<Class>>
Yolo
- updateTimer: Timer?
- updateInterval: TimeInterval = 0.03
- educationModel: VNCoreMLModel?
- delegate: YoloDelegate?
- currentCameraImage: CVPixelBuffer!
- gridHeight = [13, 26, 52]
- gridWidth = [13, 26, 52]
- blockSize: Float = 32
- boxesPerCell: Int = 3
- numClasses: Int = 22
- maxBoundingBox: Int = 10
- confidenceThreshold: Float = 0.6
- iouThreshold: Float = 0.1
+ init()
+ predict(imagePixel: CVPixelBuffer)
+ getBoundingBoxes(features: [MLMultiArray]) -> [Prediction]
+ searchForRectangle(observation: Prediction)
- completedVisionRequest(request: VNRequest?, error: Error?, observation: Prediction)
1..1
0..*
<<Protocol>>
AlteredImageDelegate
+ alteredImageLostTracking(alteredImage: AlteredImage)
<<Class>>
AlteredImage
+ className: String
+ referenceImage: ARReferenceImage?
+ delegate: AlteredImageDelegate?
- visualizationNode: VisualizationNode
+ anchor: ARImageAnchor?
+ failedTrackingTimeout: Timer?
+ timeout: TimeInterval = 0.03
+ Init(image: CIImage, referenceImage: ARReferenceImage, className: String)
+ add(anchor: ARAnchor, node: SCNNode)
+ update(anchor: ARAnchor, node: SCNNode)
- resetImageTrackingTimeout()
<<Class>>
VisualizationNode
+ className: String?
+ currentImage: SCNNode
+ visualizationNode: VisualizationNode
- delegate: VisualizationNodeDelegate?
+ init(size: CGSize, className: String)
+ init?(coder: NSCoder)
<<Struct>>
Prediction
+ classIndex: Int
+ score: Float
+ boundingBox: CGRect
<<Class>>
SCNNode

Figure 5.6: The class diagram of the iOS application.

5.3.3 Virtual 3D objects in ARKit

ARKit 4 itself does not read any 3D formats, it instead uses the rendering engine to read and render the 3D object onto the scene. The rendering engine only accepts four 3D model formats: (1) Collada's Digital Asset Exchange **.dae**, (2) Pixar's Zipped Universal

Scene Description **.usdz**, (3) Native Scene Format **.scn**, and (4) Reality Composer Format **.rcproject** or **.reality**. The built-in 3D scene inspector can be used to adjust the model characteristics such as lighting, animation or music background (Figure 5.7).



Name Ch14_Body
Shading Phong
Diffuse Ch14_1001_Diffuse.png
Specular Ch14_1001_Specular....

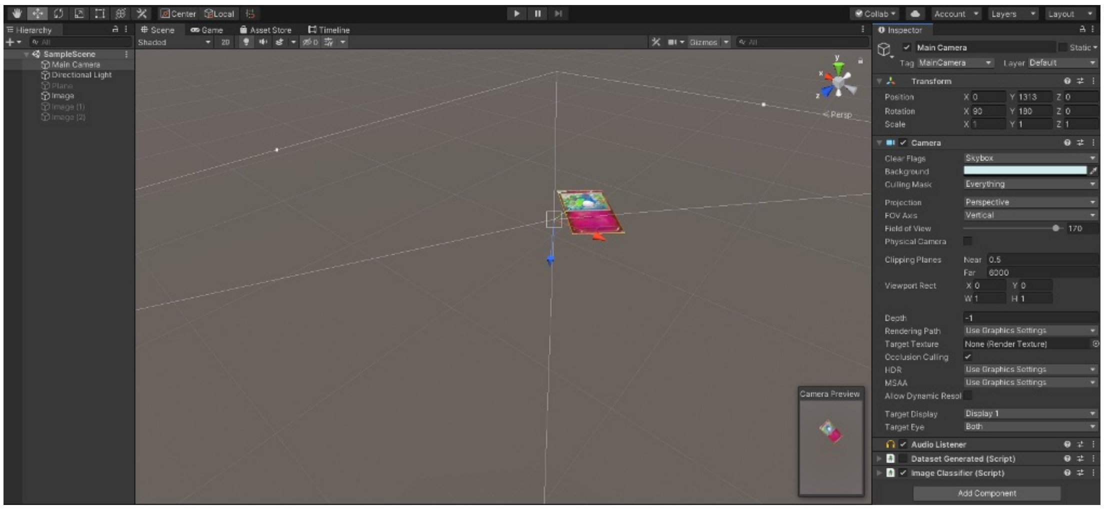
Figure 5.7: The view of 3D scene inspector in Xcode.

5.4 Frameworks and software

The deep neural network training process or prototype application implementation described in this thesis can never be done without the supported tools. This section will focus on the software and development frameworks used to support this project's implementation and testing step.

5.4.1 Unity and C#

Every game developer or gamer might have heard the term of the game engine at least one. It provides the developers with different ways to add and control the game components such as physics, rendering, scripting, etc. Unity is considered as the primary game engine to render and produce the synthetic dataset for this thesis (Figure 5.8). Unity is not the only game engine that can provide excellent rendering capability, but it has the unique standouts that could be the best candidate for this thesis.



```
Main Camera                          Static
Tag MainCamera        Layer Default
Transform
Position        X 0     Y 1313   Z 0
Rotation        X 90    Y 180    Z 0
Scale           X 1     Y 1      Z 1
Camera
Clear Flags          Skybox
Background
Culling Mask         Everything
Projection           Perspective
FOV Axis             Vertical
Field of View        170
Physical Camera
Clipping Planes      Near 0.5
                     Far  6000
Viewport Rect        X 0   Y 0
                     W 1   H 1
Depth                -1
Rendering Path       Use Graphics Settings
Target Texture       None (Render Texture)
Occlusion Culling    ✓
HDR                  Use Graphics Settings
MSAA                 Use Graphics Settings
Allow Dynamic Resol
Target Display       Display 1
Target Eye           Both
Audio Listener
Dataset Generated (Script)
Image Classifier (Script)
```

Figure 5.8: An example of the project view in Unity.

Simplicity. Unity engine architecture is simpler to understand and work with. It provides the capability to enable on-device graphics processing unit (GPU), which is very important for texture rendering. It is written in C#, which is not as difficult to use and learn as C++ used in the Unreal game engine (Pv, 2021).

Flexibility and scalability. Using C# is another advantage over using blueprints as Unreal, where the user has the greater flexibility to control and customise the game components. It is also easier when the size of the project is required to expand.

Community and supports. The Unity and C# community are also massive, which is very helpful for seeking questions and help. Most Unity game assets are free to use, which is very handy for researchers when the budget is one of limitations.

5.4.2 PyTorch and Python

Many different deep learning frameworks are available on the market, such as Tensorflow (Abadi et al., 2016), PyTorch (Paszke et al., 2019), Caffe (Jia et al., 2014), etc. However, PyTorch is indicated as the future of the deep learning framework. It is chosen as the primary deep neural network framework due to its unique standouts.

Easy to learn. PyTorch has the same structure as the famous Python programming language. Hence, it is not too difficult to get started and learn due to its simplicity, and well-documented developer supported community.

Productivity. PyTorch is designed to incorporate Python and many powerful built-in APIs that could be implemented in both Windows and Linux OS. It can also enable the NVIDIA tensor cores, which could speed up to 10X in AI training.

Data Parallelism. PyTorch can efficiently distribute the computational tasks among multiple CPUs and GPU at the same time to accelerate the neural network training process.

5.4.3 CoreML framework

As mentioned in the introduction, the thesis goal is to implement an iOS application to run the pre-trained deep neural network model. Hence, CoreML is the most suitable Machine Learning framework that can be used across different Apple products (macOS, iOS or iPadOS). The CoreML framework is built for quick prediction performance and low inference that allows the application to perform the real-time prediction of live images or video (Figure 5.9).

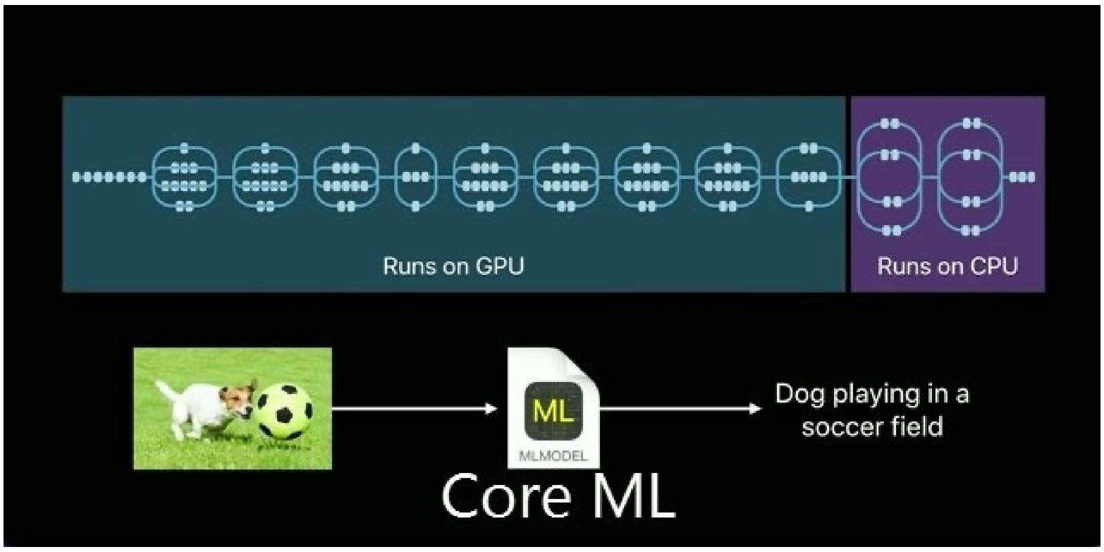
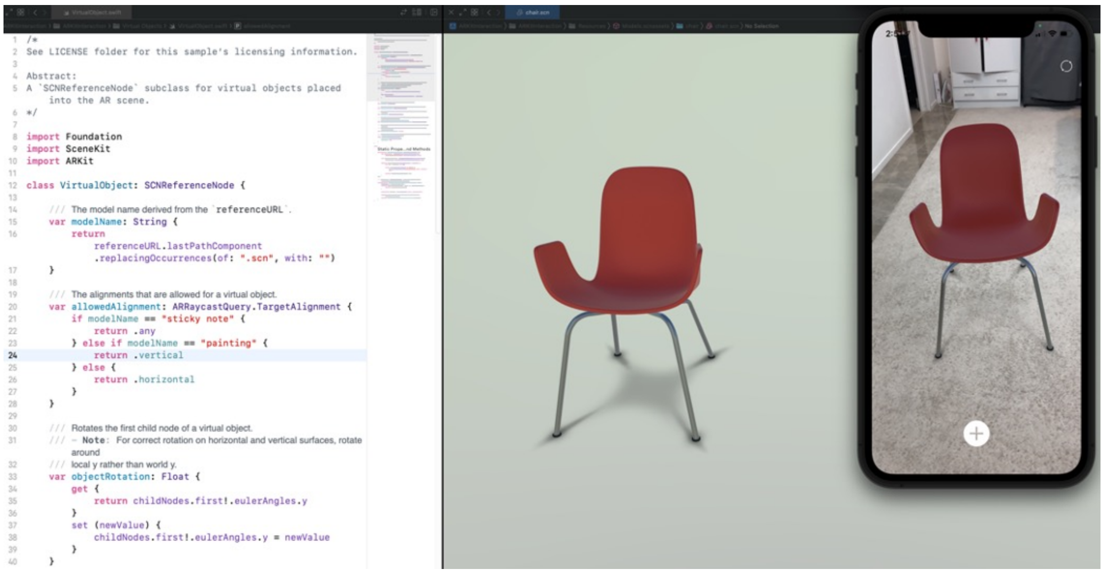


Figure 5.9: An example of deep neural network execution process using CoreML framework.

As Apple keeps optimising its hardware and software to give the best user experience and system performance, CoreML is not an exception. The CoreML works efficiently with the bionic chip, which gives the best prediction performance with low energy consumption. Since the first generation of the bionic chip (A11), Apple has integrated and increased the neural processing unit (NPU) cores used to execute and accelerate the Machine Learning tasks. The latest bionic chip can perform at least 600 billion calculations per second or 600 GFLOPS, where it requires approximate 160 GFLOPS for an average YOLO model to be executed. The partial deep neural network execution process now can be run on NPU instead of CPU to minimise the system work-loaded and improve the performance. Low latency is the most considered crucial factor for the application prototype. In general, it does not need to make a network or external API call to send the data and wait for the response as every process will be done within the device and built-in APIs. Another advantage of using CoreML is privacy. If privacy is a big concern for most of today artificial intelligence-based applications, it is not a problem with CoreML. All of the input data will be process offline and within the user's device, and no external APIs called means the private data would never leave the device.

5.4.4 ARKit



```
/*
See LICENSE folder for this sample's licensing information.

Abstract:
A `SCNReferenceNode` subclass for virtual objects placed
    into the AR scene.
*/

import Foundation
import SceneKit
import ARKit

class VirtualObject: SCNReferenceNode {

    /// The model name derived from the `referenceURL`.
    var modelName: String {
        return
            referenceURL.lastPathComponent
            .replacingOccurrences(of: ".scn", with: "")
    }

    /// The alignments that are allowed for a virtual object.
    var allowedAlignment: ARRaycastQuery.TargetAlignment {
        if modelName == "sticky note" {
            return .any
        } else if modelName == "painting" {
            return .vertical
        } else {
            return .horizontal
        }
    }

    /// Rotates the first child node of a virtual object.
    /// - Note:  For correct rotation on horizontal and vertical surfaces, rotate
        around
    /// local y rather than world y.
    var objectRotation: Float {
        get {
            return childNodes.first!.eulerAngles.y
        }
        set (newValue) {
            childNodes.first!.eulerAngles.y = newValue
        }
    }
```

Figure 5.10: An Xcode setup of the iOS project using ARKit and its runnable application on real-time input images.

ARKit is the AR development framework designed by Apple to quickly and easily build AR experiences into games and applications (Figure 5.10). It uses the device's camera, processor, and built-in sensors to create immersive interactions. According to Apple, ARKit is suitable for any devices that run on iOS 11 or later, which means it could be runnable on most Apple build smartphone and iPad products nowadays. As the product is fully designed and supported by Apple, the processing time on each frame is expected to be shorter and generates better AR algorithm performance. Less time on frame processing means more CPU/ GPU available for more realistic rendering graphics and virtual scenes. Another advantage of using ARKit is simplicity and integration capability. ARKit can be easily accessed and modified using Xcode and Swift. It also can work efficiently with other Apple-build frameworks or APIs such as CoreML and Vision to give the best performance with low latency.

Chapter 6

Experimental Designs and Results

Parts of this Chapter have been published in ***paper 1***, ***paper 2***, *and* ***paper 4*** *listed in the publication list*

Conducting several experiments against different scenarios is the primary step to qualify the proposed method. As described in Section 1.2, one of the thesis investigations aims to immerse the Machine Learning technology into the Augmented Reality marker detection process. Therefore, the following Sections of this Chapter will analyse the performance of both Machine Learning training using the proposed synthetic dataset and the Augmented Reality application. In short, these experiments will focus on the following aspects:

- **Synthetic datasets generation** is one of the primary proposed features to improve the detectability of Augmented Reality markers without modifying the original texture. This aspect introduces several different synthetic datasets that use the proposed method described in **Section 5.1**. These datasets are generated based on real-world purposes such as education, business, entertainment, or transportation (more details will be demonstrated in **Section 6.1**).

- **Deep neural network training outcome using the synthetic datasets** is the most crucial experiment to qualify the performance of the proposed synthetic dataset generation method. This experiment compares the performances between the synthetic datasets with the real-world datasets using the model evaluation metrics described in **Section 4.5.2**, such as recall, precision, or mAP (more details will be demonstrated in **Section 6.2**).

- **Prediction performance under natural conditions** gives a broader picture of how well the deep neural network model trained by the proposed synthetic datasets when it shifted from the experimental environment to real-world situations (more details will be demonstrated in **Section 6.3**).

- Lastly, **the performance of the pre-trained deep neural network model on the implemented iOS application** will be analysed. This experiment reviews the overall performance of the proposed idea and gives the chance to indicate future improvements to overcome any potential limitations (more details will be demonstrated in **Section 6.4**).

6.1 Synthetic datasets

The first synthetic dataset (**40K**) contains 22 different book and poster related categories that consist of over 40,000 trainable images and their corresponding labels files (Figure 6.1). It is the primary dataset that will be used to train the deep neural network model for the iOS application described in **Section 5.3**. The full details of the 40K dataset image categories and generation steps are described in **Section 5.1**. However, the result of one single synthetic dataset does not provide enough information to qualify the proposed method performance. Thus, several other synthetic datasets are generated using the same principle but for different application sectors to optimal the findings.

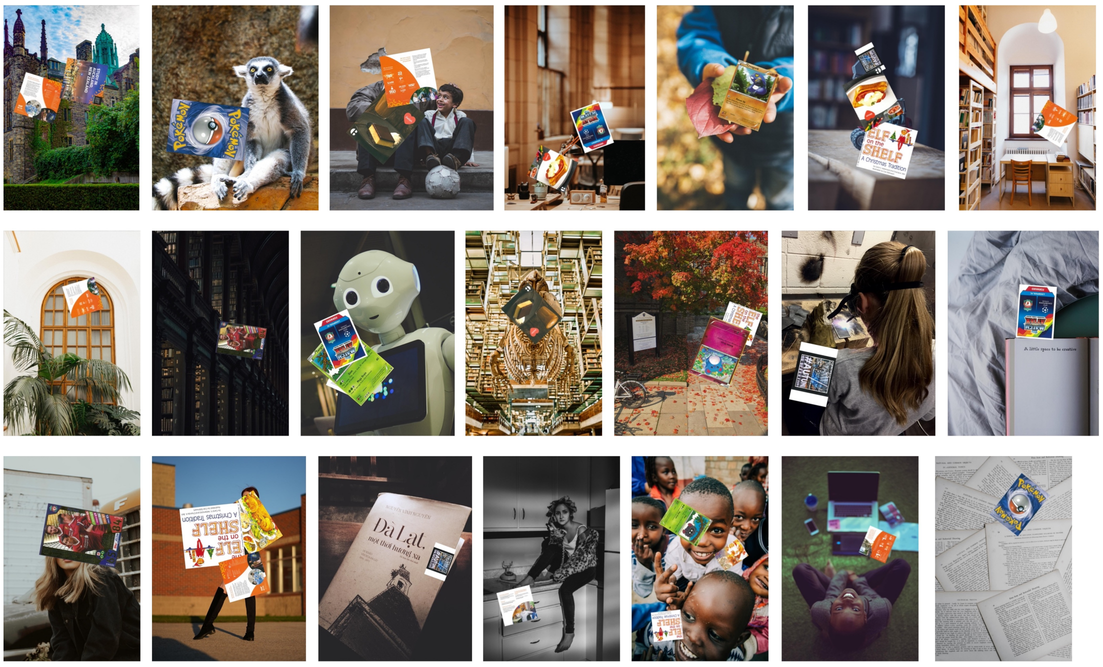


Figure 6.1: The examples of different images that the **40K dataset** contains.

The first related application sector is the standard 52-card pack (McGuigan, 2020) commonly used for magic tricks, card games, etc (Figure 6.2). The original deck contains 52 standard white backgrounds of playing cards. However, the number of training classes are minimised to 13 by grouping 52 playing cards by their ranks, and this dataset is called **S13**. The S13 dataset consists of 26,000 training images, and each image could contain one or multiple playing cards with random locations and orientations. **S6** dataset is a smaller version of the S13 dataset where it consists of only 12,000 training images, and it contains only six different classes (nine, ten, jack, queen, king, and ace) instead of 13 classes.

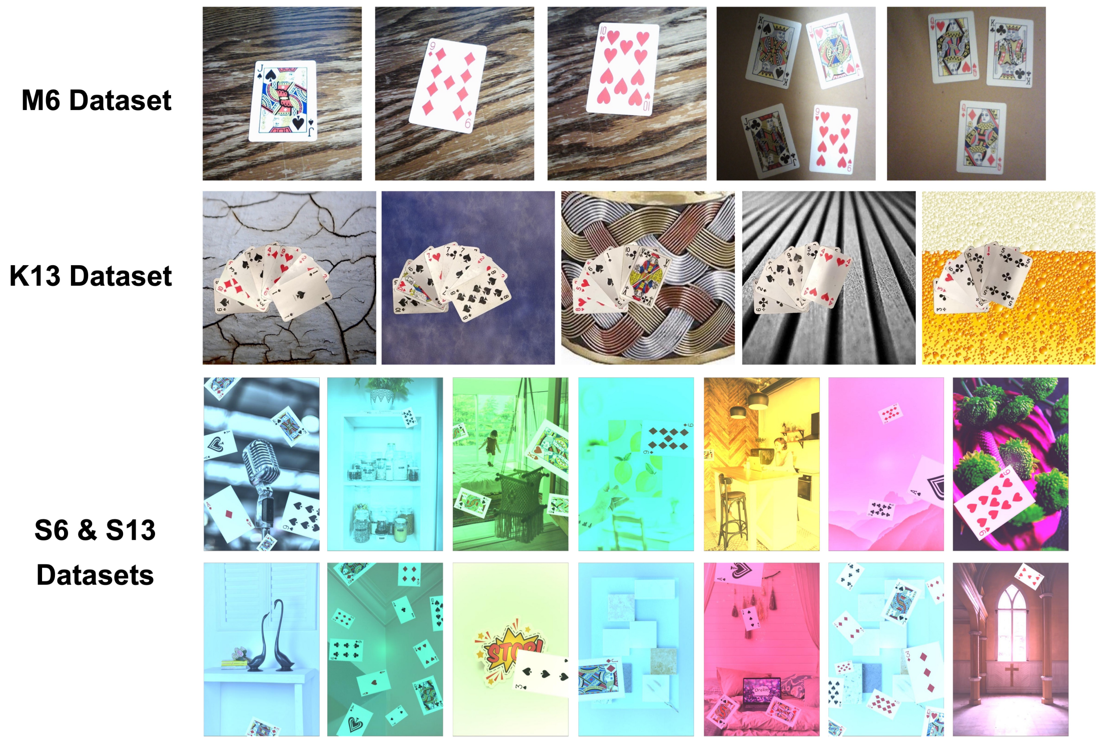


Figure 6.2: The examples of different images that the **S6, S13, K13, and M6 datasets** contain.

In order to evaluate the proposed synthetic dataset, this experiment also uses a synthetically made Kaggle open-source playing cards 13 classes dataset (Hugo Paigneau, 2020) and another six classes manually made dataset. The Kaggle 13 classes dataset **(K13)** initially consists of 52 different cards, but the cards are grouped into their according ranks to minimise the number of training classes to 13. The K13 dataset was generated using OpenCV and Python by taking the images of each playing card in 10-second videos. The whole process was done with a CPU instead of using GPU. The manually made 6 classes dataset **(M6)** contains real-world images of animals, indoor/outdoor objects with different lighting conditions, and a random number of cards and their orientations. The M6 dataset was generated using the iPhone camera by taking hundreds of playing cards, including individual and overlapping items.

The other related application sector is the transportation sign. This experiment uses the New Zealand standard road signs that are categorised into three different types: (1) compulsory, (2) warning, (3) information. All the details of the transportation signs can be obtained at the NZ Transport Agency official website [1]. In general, it has hundreds of different transportation signs in New Zealand; but only the top 50 common road signs are used for this experiment (Figure 6.3) : Speed limit (R1-1, R1-1.1, R1-1.2); Speed Limit Derestriction (R1-3); Stop (R2-1); No Stopping (R6-10.1); Disabled Parking (R6-55); Bus Parking (R6-53); Motorcycle Parking (R6-51); No Parking: Bus Stop (R6-71); No Parking (R6-70); No Parking: Taxi Stand (R6-72); Attention (TW-2); Road Works (T1A); Turn Left or Right (R3-11); Turn Left (R3-8); Turn Right (R3-10); Go Straight (R3-9); Priority Over Oncoming Vehicles (R2-8); Wrong Way (R3-7); No Turn Left (R3-1); No Turn Right (R3-2); No U-Turn (R3-3); Road Closed (R3-6); Traffic lights ahead (W10-4); Must Turn Left/Right (R4-1); May Proceed Straight or Turn Left/Right (R4-3); Must Proceed Straight (R4-2); Give Way (R2-2); Give Way at Roundabout (R2-3); Give Way to Oncoming Vehicles (R2-7); Except Bus (R3-5.1); Bus Lane (R4-7); Buses Only (R4-7.1); No Entry (R3-4); School; No Exit (A40-1);

[1]https://www.nzta.govt.nz/roadcode/

One-way traffic (R3-12); Temporary (R1-8).

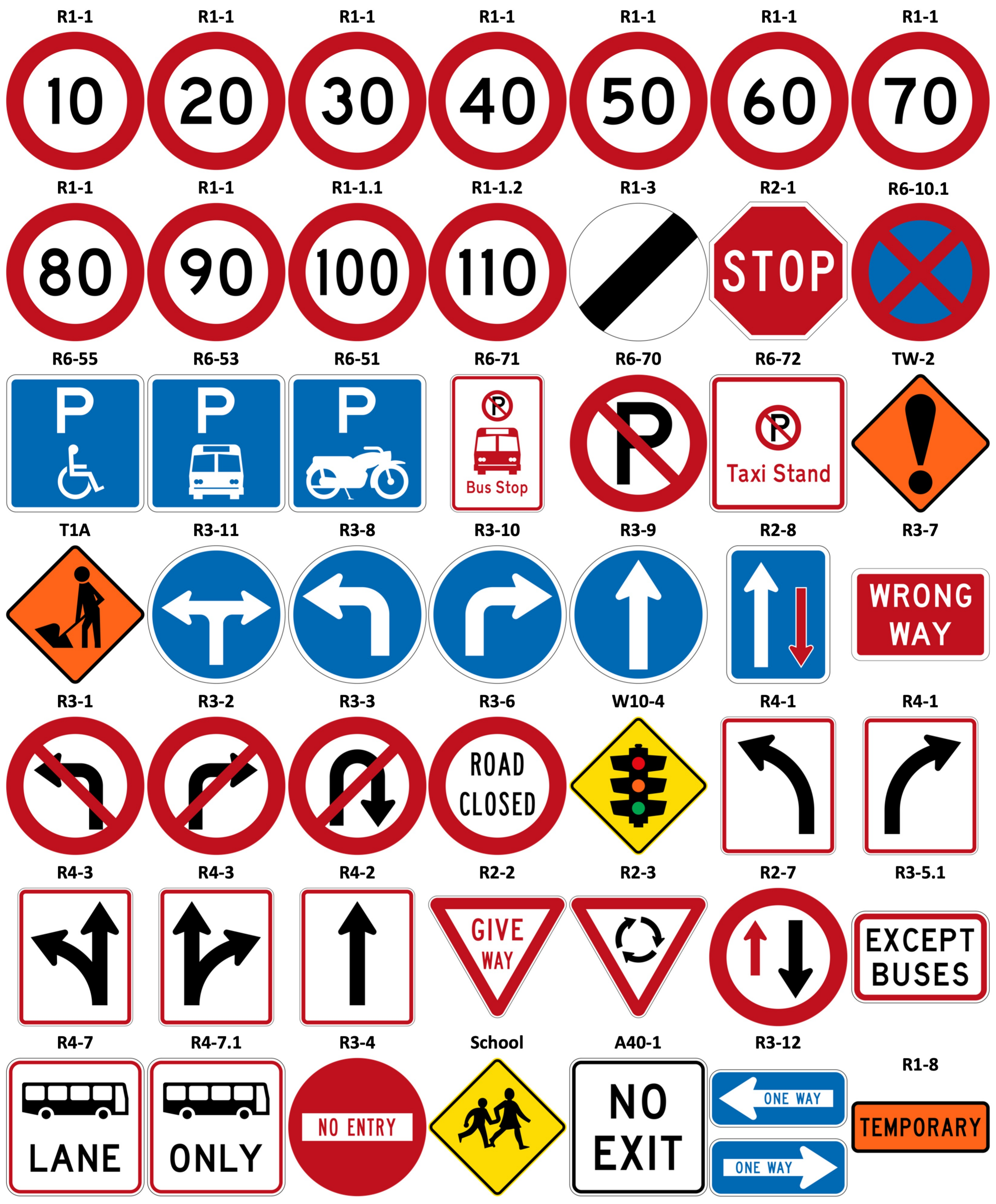


Figure 6.3: The real-world images of top 50 common New Zealand road signs with their codes described above each image.

There are two synthetic datasets generated using the proposed method; one has 50 different class names (**T100K**), and the other consists of 35 class names (**T350**). A real-world dataset (**M350**) is also prepared that consists of 350 images of transportation signs taken using different photography equipment such as digital or build-in mobile cameras (Figure 6.4). Each image has a size of 1,500 x 2,250 and contains single or multiple images of the transportation signs. The labelling process is done manually with the online open-source RoboFlow (https://blog.roboflow.com/labelme/). This manual process takes more than two weeks to complete with the help of over 30 volunteers. The software and hardware requirements keep the same where a single Radeon Pro 560X GPU and Unity version 2019.4.18f1 were used to generate all of the described datasets.

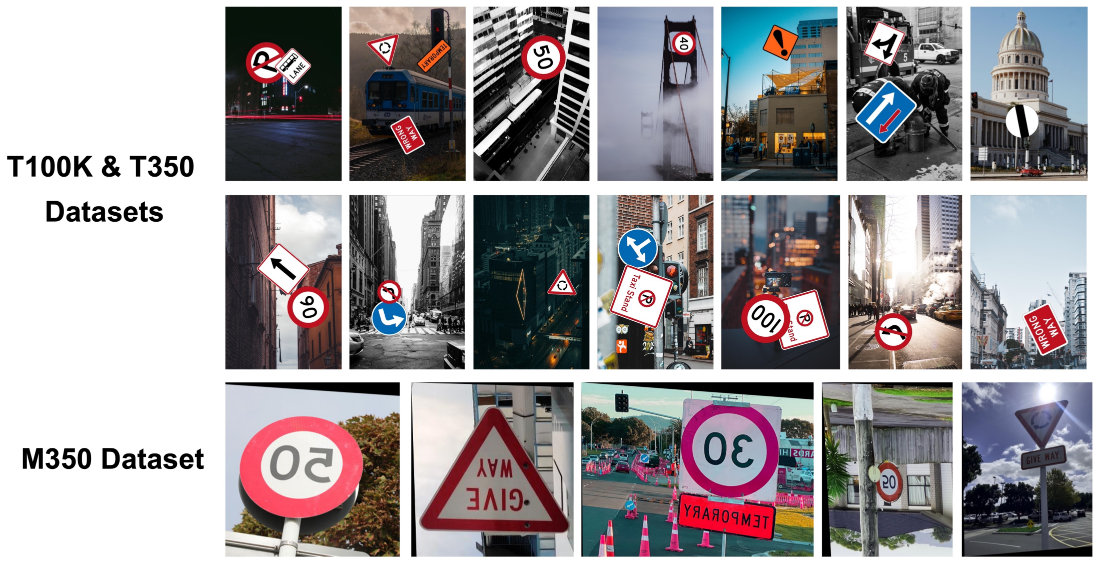


Figure 6.4: The examples of different images that the **T100K, T350, and M350 datasets** contain.

Table 6.1 indicates the primary differences between the synthetic datasets using the proposed method and the datasets that were generated manually. The results show that synthetic datasets generated using GPU took remarkably less time than other datasets using different methods. The proposed method took **four times faster** than the synthetic dataset generated using CPU and could reach approximately **200 times faster** than the manually made dataset. Regarding the amount of manual processing work needed, the proposed synthetic datasets require fewer people, whereas others might require more. This concrete evidence confirms that the GPU synthetic dataset will produce a much more efficient dataset quantity and labour cost than the manually made version.

Dataset	**Generated method**	**Dataset szie**	**Time consumed**	**No of labour**
40K **(Ours)**	Synthetic (GPU)	44,000	3 hours	1
S6 **(Ours)**	Synthetic (GPU)	12,000	50 minutes	1
S13 **(Ours)**	Synthetic (GPU)	26,000	1 hour 48 minutes	1
T100K **(Ours)**	Synthetic (GPU)	100,000	6 hours 56 minutes	1
T350 **(Ours)**	Synthetic (GPU)	350	1 minute 30 seconds	1
K13	Synthetic (CPU)	6,000	1 hour 30 minutes	1
M6	Manual	364	5 hours	1
M350	Manual	350	Two weeks	30

Table 6.1: The comparison details between the synthetic datasets using the proposed method and manually made datasets.

6.2 Deep neural network training performance

The synthetic and manual made datasets are trained on the same Linux system that used Python 3.8 with Intel i7-9700F 3.0 GHZ CPU and Nvidia RTX 2080 Super (8 GB memory). The training parameters were kept the same for all datasets as described in Section 5.2. The open-source online Machine Learning platform Weight & Biases [2] is used as the primary tool to collect and analyse the results. This platform uses the number of steps to track and visualise the results instead of using epochs, and there are

[2]https://docs.wandb.ai/

1,034 steps approximately for 80 training epochs.

6.2.1 Learning curves

Analysing the learning curves during training and validation is the most common method to diagnose the model behaviour. There are three common learning curves categories:

1. **Overfitting** happens when the model has learnt the training dataset too well, including noises described in detail in Section 4.2.6. The model is then likely to perform poorly when the new data is being fed through the model.

2. **Underfitting** is another example of poor model training performance. It happens when the model has not adequately learnt the training dataset to score a sufficiently low loss value. The underfitting can be easily identified when the loss continues to decrease at the end of the training period. Not having enough training data is the most common reason for this situation to occur.

3. **Good fitting** is the goal of the training process. The curves are identified as good fitting curves when they decrease to the point of stability, and their gap is minimal.

There are two essential loss parameters that are used to examine the results of all learning curves:

The first parameter is **classification loss** value which indicates how well the model is to classify the object's class correctly from other classes. Figure 6.5 presents the classification loss differences between training and validation of 52-card pack related datasets. The result indicates that the proposed synthetic datasets (**S6 and S13**) can efficiently perform the good fitting behaviour with the minimal gap is approximate

0.001 on average at the final step. On the other hand, the synthetic **K13** that used a different algorithm created approximate five times greater learning curves gap differences with more noise occurring. The manual made dataset (**M6**) exhibits overfitting learning curves since the training loss value decreases while the validation loss value increases.

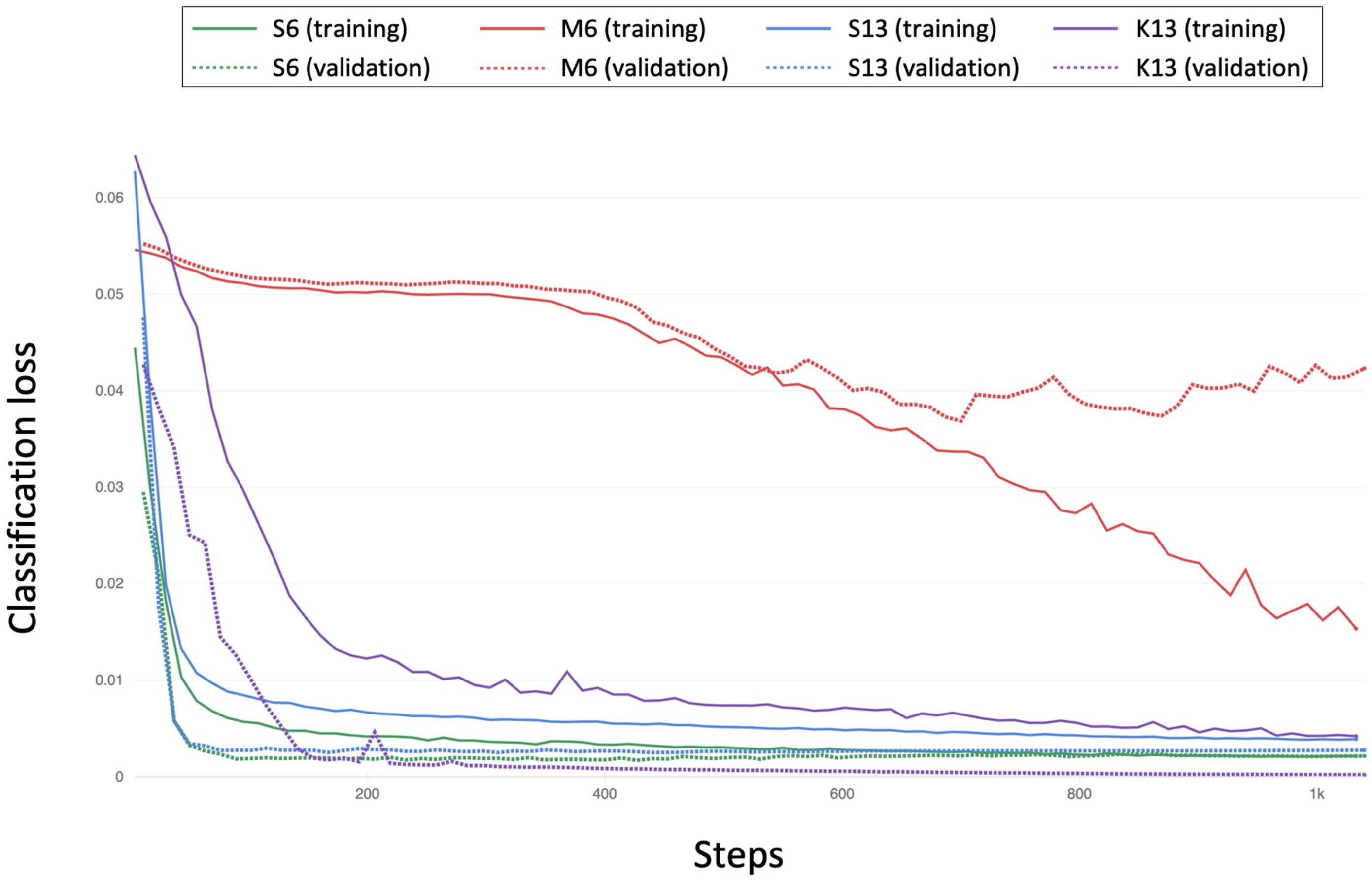


Figure 6.5: The classification loss comparison during model training (solid curves) and validation (dashed curves) using 52-card pack related datasets **(S6, M6, S13, and K13)**.

Figure 6.6 presents the other classification loss differences between training and validation of New Zealand transportation sign related datasets. The result shows that the proposed synthetic dataset (**T100K**) performs the lowest score on the training and validation loss differences, approximate 0.006. The T100K dataset also gives less noise than others, especially the manually made dataset (**M350**) which creates the most remarkable learning gap differences. The proposed synthetic dataset reached the stabilise point at 80% of training time (T100K), whereas the manually made dataset could not.

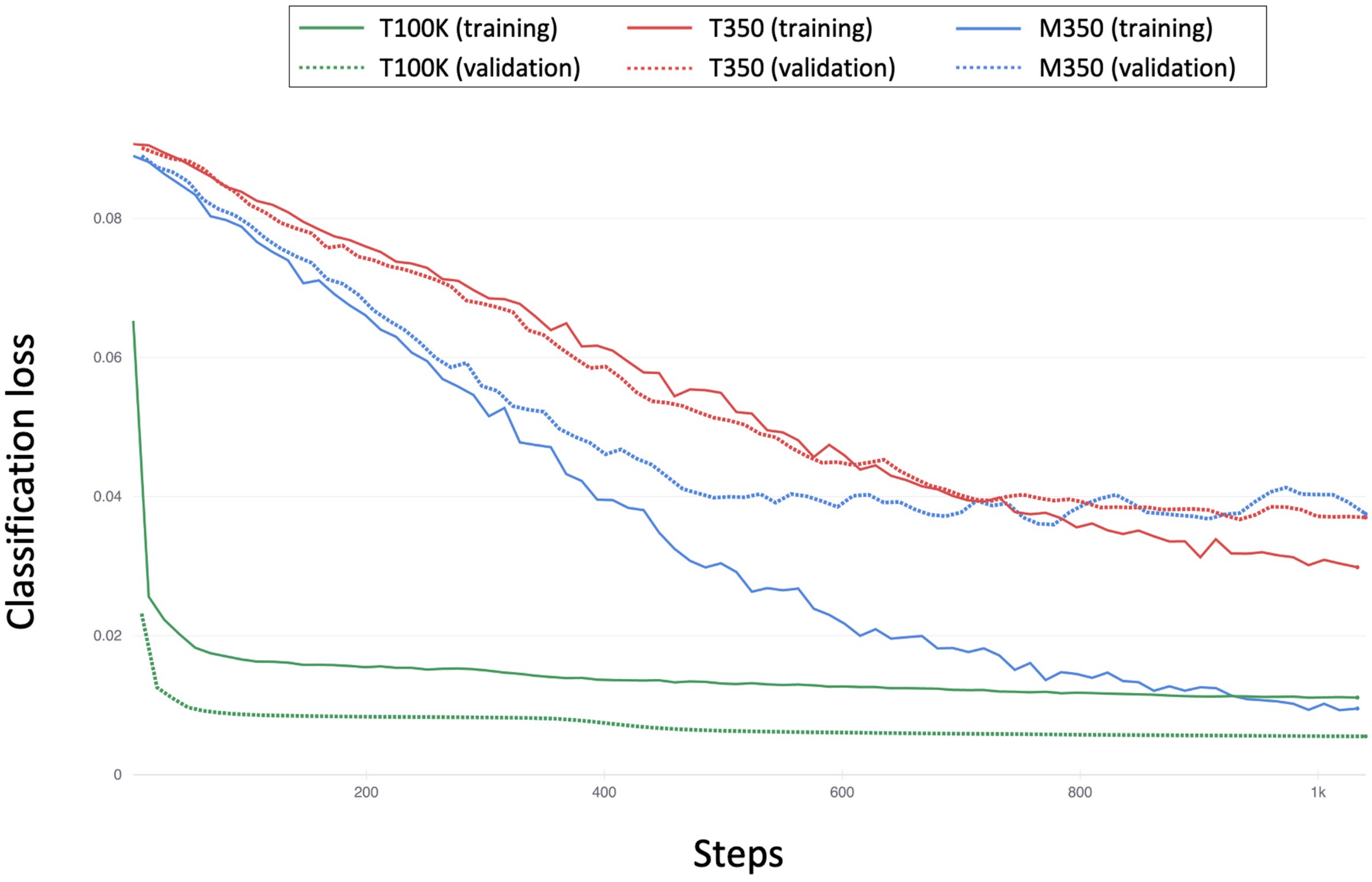


Figure 6.6: The classification loss comparison during model training (solid curves) and validation (dashed curves) using New Zealand transportation sign related datasets **(T100K, T350, and M350)**.

The other loss parameter is the **bounding box** value that indicates how well the predicted bounding box can cover an object and how well the predicted centre coordinate would be compared to the ground truth values. Figure 6.7 presents the bounding box loss differences between training and validation of 52-card pack related datasets. The proposed synthetic datasets (**S6 and S13**) gave the best results on stabilisation since they both could reach their stability points at 80% of training time, whereas the other datasets could not. However, their gap difference values were the greatest and produced less noise than other datasets.

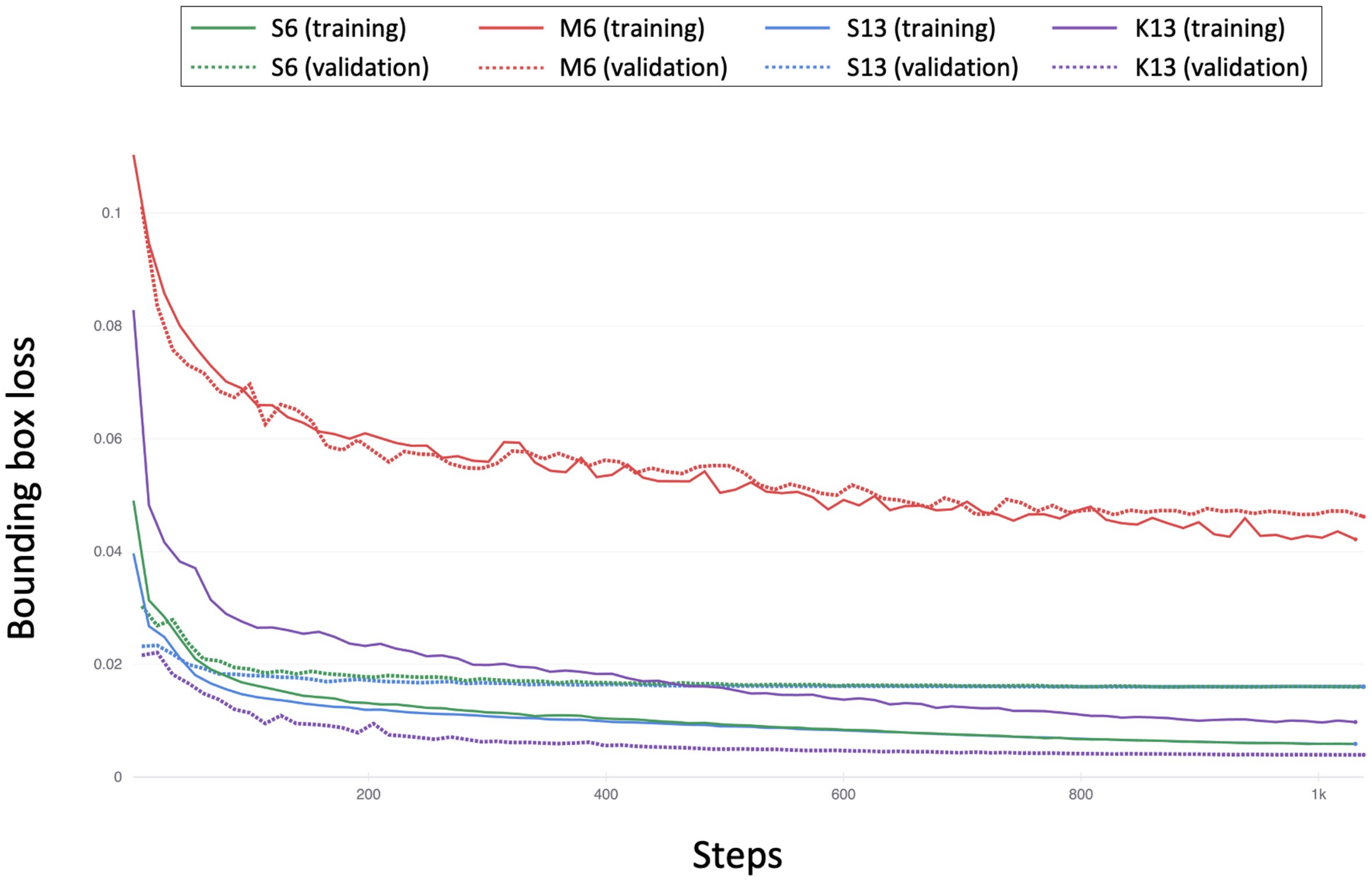


Figure 6.7: The bounding box loss comparison during model training (solid curves) and validation (dashed curves) using 52-card pack related datasets **(S6, M6, S13, and K13)**.

The other comparison is on New Zealand transportation signs related datasets (Figure 6.8). This comparison demonstrated a clear difference since the proposed synthetic dataset with the high number of training data (**T100K**) could produce the minor gap differences between training and validation loss at approximate 0.002 at the final step. It could also reach the stability point very quickly at approximate 10% of training time. The two comparison results confirm that it is possible to improve the learning performance and get close to the good fitting training behaviour by increasing the size of the training dataset.

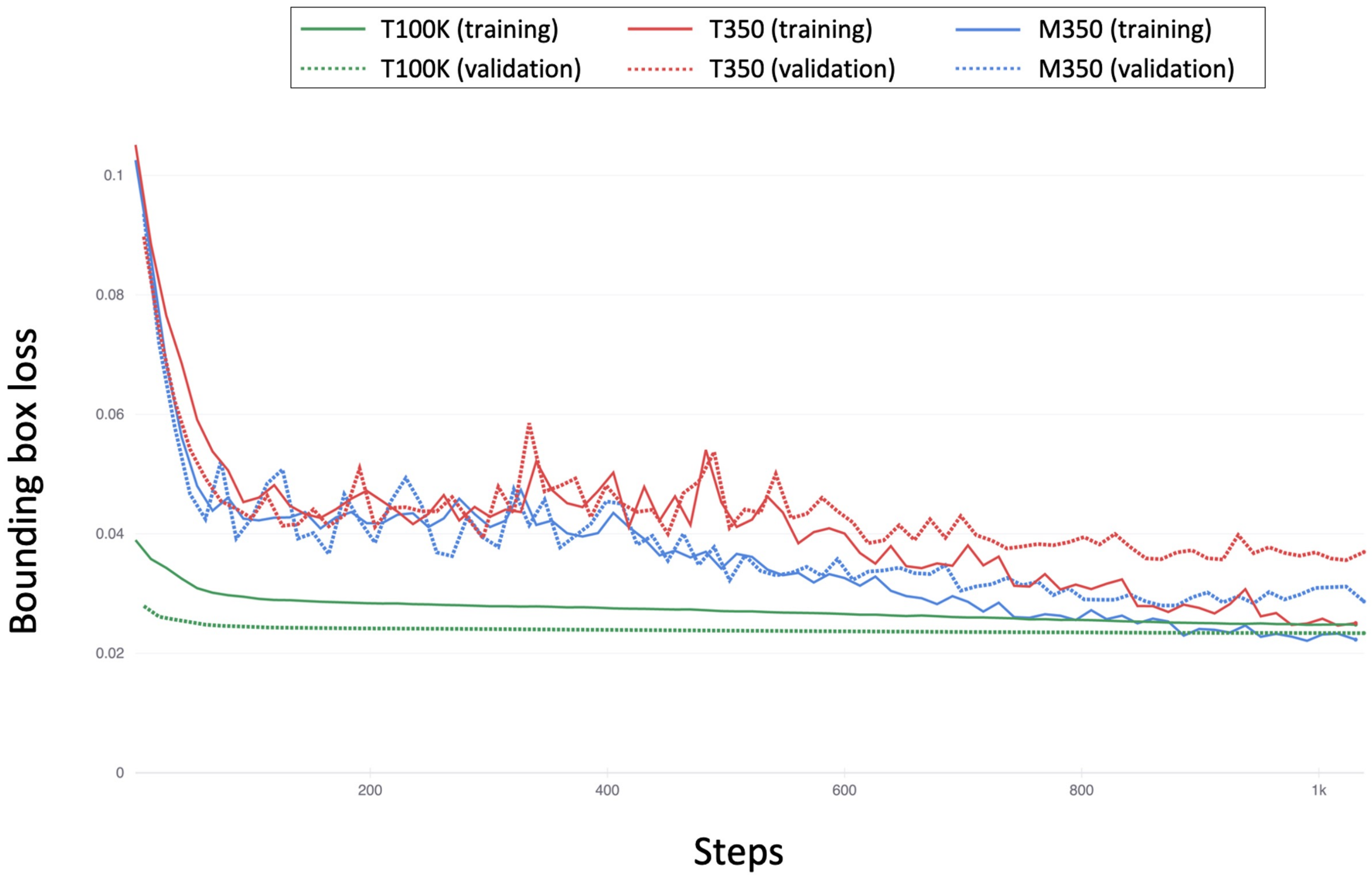


Figure 6.8: The bounding box loss comparison during model training (solid curves) and validation (dashed curves) using New Zealand transportation sign related datasets (**T100K, T350, and M350**).

Overall, the learning curves results indicated that the proposed synthetic datasets are most likely to achieve good fitting training behaviour. It means that they provide less noise, reduce errors during training and improve the model learning performance.

6.2.2 Model evaluation

This subsection indicated how well the model would be using different datasets. The model evaluation metrics (described in Section 4.5.2) are used in the following comparisons.

Dataset	**Recall**	**Precision**	**mAP0.5**	**mAP[0.5:0.95]**
40K (Ours)	**0.993**	**0.987**	**0.993**	**0.605**
S6 (Ours)	**0.999**	**0.998**	**0.995**	**0.994**
S13 (Ours)	**0.999**	**0.996**	**0.995**	**0.994**
T100K (Ours)	**0.982**	**0.873**	**0.907**	**0.576**
T350 (Ours)	**0.842**	**0.359**	**0.735**	**0.432**
K13	0.999	0.998	0.995	0.955
M6	0.365	0.365	0.314	0.152
M350	0.6	0.356	0.538	0.349

Table 6.2: Full model evaluation. The table describes the evaluation results of the proposed synthetic datasets (**presented in bold**) and the other three datasets conducted using the manual data annotation process and other different synthetic algorithm. The recall and precision values are set at the IoU threshold of 0.5. The mean average precision (**mAP**) values are set at the IoU threshold of 0.5 (**mAP0.5**) and from 0.5 to 0.95 (**mAP[0.5:0.95]**).

Table 6.2 summarised the quantitative evaluation using recall, precision, and mean average precision (mAP) values. The synthetic datasets using the proposed method could achieve **over 80% of recall values**. Significantly, the synthetic dataset could produce an average of **30% higher** than the real-world dataset for the New Zealand transportation signs category and almost **three times better** for the 52-card pack category. It means that the synthetic dataset can predict the bounding box position with the least false-negative rate. There is the same result with the false-positive value found in the precision records (precision values) where the proposed synthetic dataset could achieve **over 87%** and an average of **18.7% higher** than the real-world dataset for the New Zealand transportation signs category and **2.5 times better** for 52-card pack category approximately. The mAP values, which were calculated based on the IoU thresholds, comparable mAP0.5 values scored **over 70%** for the synthetic dataset. The higher IoU threshold (mAP[0.5:0.95]) scored **over 43%** for the synthetic dataset. The experiment results suggest that the model's ability to predict the objects and their bounding box is better in the case of synthetic than real-world images.

6.3 Predictions under natural conditions

A sample image of a **52-card pack** with random lighting conditions, orientation, and card position is chosen for this experiment (Figure 6.9). However, the top and bottom rows of the original image were cropped to visualise the situation of the partially covered object. Figure 6.10 presents the prediction results on the pre-trained weights using different training datasets. The proposed datasets (**S6 and S13**) are the winners in this experiment since they can correctly classify over **80% of the objects**, whereas other datasets can only achieve a few or no number of correct predictions. Significantly, the proposed synthetic dataset can give the correct prediction with an average of **74%** confidence for the partially covered cards and **76%** confidence for the unexpected dazzling lighting condition. Correctly detection under different orientations is another advantage for the proposed synthetic datasets since they could achieve **84%** confidence whereas other datasets could not. It presents that the proposed method is capable of providing higher quality datasets for object recognition. The synthetic datasets using the proposed method gave better accuracy results under different natural lighting and other unexpected conditions.

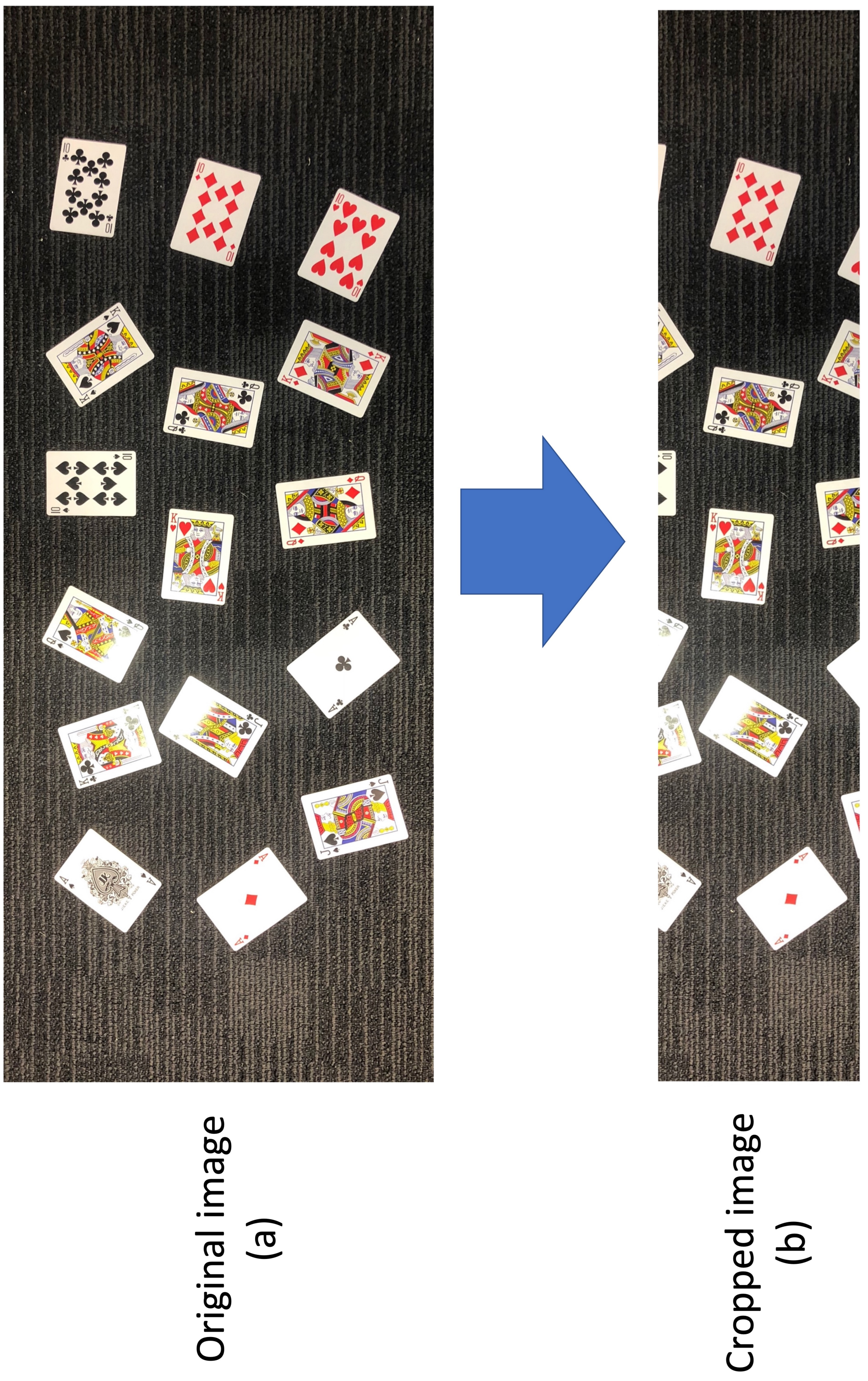


Figure 6.9: The original random images of playing card (**a**) orientations and lighting conditions are cropped (**b**) in order to present other unexpected natural conditions.

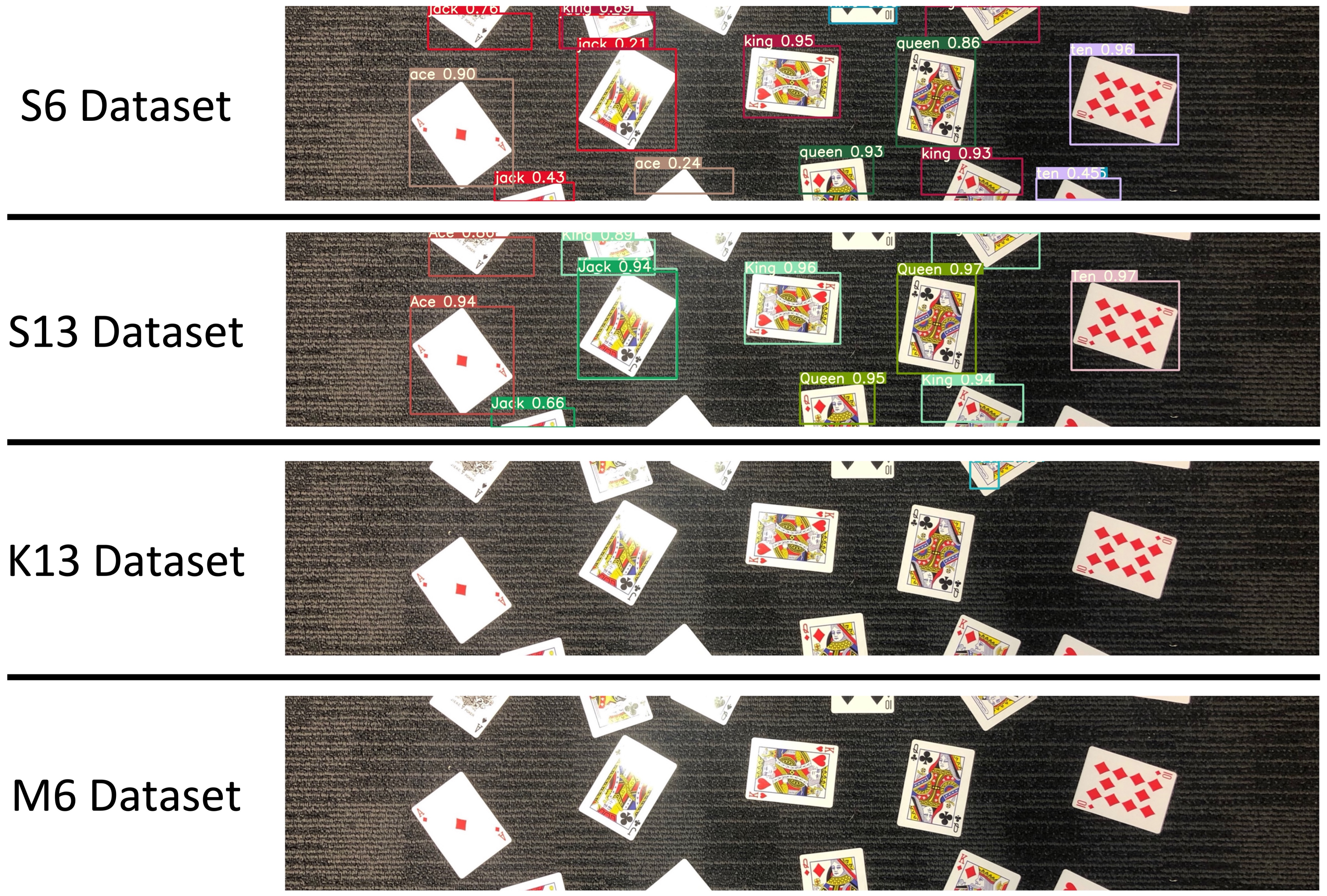


Figure 6.10: The model prediction comparison under random indoor conditions (lighting, orientation, and noise) that used different 52-card pack related datasets (**S6, S13, K13, and M6**) for training.

Different experiments were conducted to qualify the model prediction performances using the **New Zealand transportation signs** related datasets (Figure 6.11). Several natural lighting and weather conditions are applied in these experiments, such as night, daylight, cloudy or foggy. The experimental results indicate that the model trained by the proposed synthetic datasets (**T100K and T350**) gave the most outstanding accuracy outcomes. The model trained by the proposed synthetic datasets could correctly classify **over 85%** of the available road signs, whereas **less than 50%** for the manual-made dataset (**M350**). The main reason for this difference is that the proposed synthetic datasets contain a much higher number of training data representing almost all of the common real-world scenarios. Significantly, the T100K dataset could predict unexpected conditions such as low lighting or foggy with approximately **70%** confidence value, approximately **30% higher** than the manually made dataset (M350). The T350 dataset gave an incomparable result to the M350 dataset under unexpected conditions since the synthetic dataset was failed to predict the road signs in some situations such as foggy, whereas the M350 dataset could. However, it was dominant under other brighter lighting conditions such as daylight or cloudy. The following YouTube short clip demos were conducted using the 100K dataset pre-train model, which presents how well the model could detect in real-time:

- **https://www.youtube.com/watch?v=kkjydYdgI90** was captured by using the bike helmet dash camera that recorded the daylight urban riding activity footage.
- **https://www.youtube.com/watch?v=wVaxdjnSwr8** was captured by using the high-definition mobile camera that recorded the daylight urban walking activity footage.
- **https://www.youtube.com/shorts/ebXNoEp_w4s** was captured by using the low-definition mobile camera that recorded the sunset rural walking activity footage.

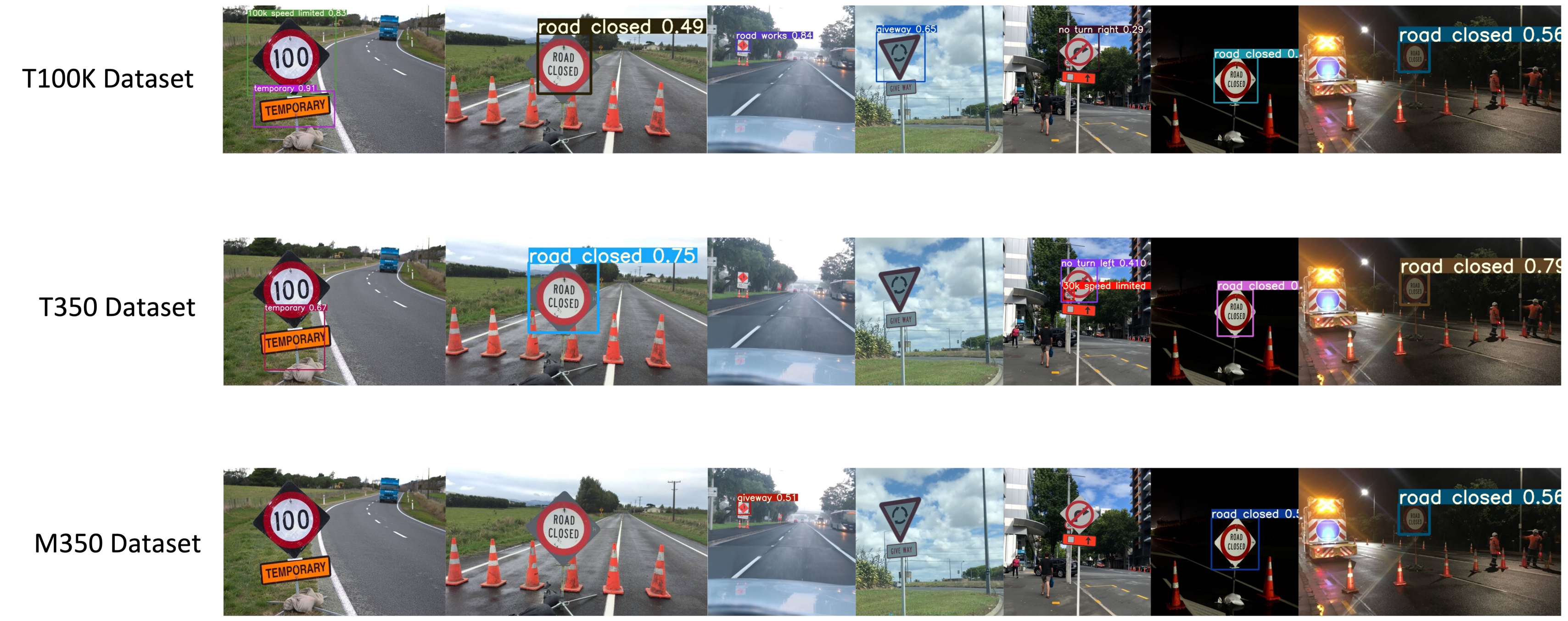


Figure 6.11: The model prediction comparison under different natural lighting and weather conditions (**day, night, raining, and foggy**) that used different New Zealand transportation signs related datasets (**T100K, T350, and M350**) for training.

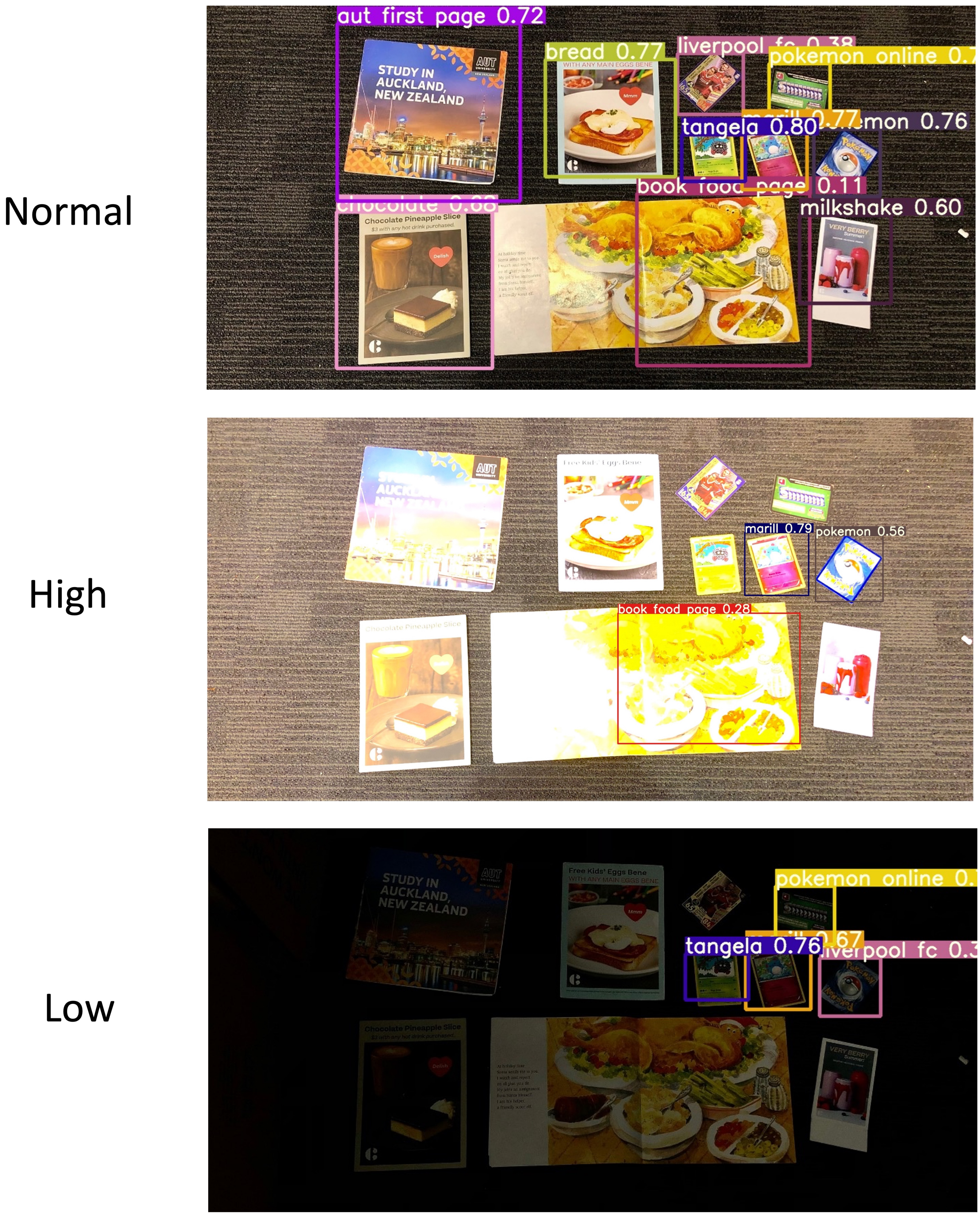


Figure 6.12: The model prediction results under different natural lighting conditions (**normal, high, and low**) using the **40K** dataset for training.

The final experiment was conducted using the **40K synthetic dataset**, the primary dataset for this research. There are three typical indoor lighting contrast levels (normal, high, and low) used for this experiment, as shown in Figure 6.12. The result indicates that the trained model could detect approximate **100% of markers correctly** with an average of **63%** prediction confidence for the normal contrast level. On the other hand, it could achieve only a **30%** accuracy rate and approximate **56%** prediction confidence for the low and high contrast levels. In the end, all synthetic datasets using the proposed method could achieve **at least 60%** for prediction accuracy rate at the standard natural condition and over **58%** on average for the poor natural condition. Again, these records indicate that the model can achieve the same or even better prediction accuracy rate by using the proposed synthetic dataset for training.

6.4 Augmented Reality application

The implemented Augmented Reality iOS application successfully used the trained YOLOv3 model using the proposed synthetic dataset to detect the target markers without modifying their original pictorial contents (Figure 6.13). The application performance test was conducted primarily on the iPhone X model, with 3 GB of RAM and the Apple-designed A11 Bionic chip. The results showed that the iOS application could detect the markers under different lighting conditions at **an average rate of 60 frames per second (FPS)**. The added animations work efficiently at an average rate of **above 50 FPS**. Significantly, the application can detect and display the virtual object on the top of the marker under poor lighting conditions. However, due to hardware limitations, the frame rate drops to 30 FPS after 30 minutes of running. This is a known issue with current iOS devices where the neural engine (ANE) inside the CPU is responsible for Machine Learning tasks. The CPU will have thermal throttling after an extended period using the ANE and forces the system to slow down CPU performance to protect the device's components. Therefore, it could lead to low Augmented Reality experiences and detection accuracy rates. The following YouTube clip demos demonstrate in detail how the implemented iOS application works in real-life:

- **https://www.youtube.com/watch?v=YcS_5pym9xM** demonstrates the 3D animation under standard indoor lighting conditions.
- **https://www.youtube.com/shorts/oFFRx-IQdRk** demonstrates the 3D models under low indoor lighting conditions.
- **https://www.youtube.com/watch?v=xLJCvuiKFwg** demonstrates the 3D animation under standard indoor lighting conditions with sound effects (short clip).
- **https://www.youtube.com/watch?v=aLfQqJQZoUUt=12s** demonstrates the 3D animation under standard indoor lighting conditions with sound effects (long clip).

Overall, the results indicated that the implemented application detection process is effective in poor lighting conditions with an acceptable detection accuracy rate. This means that the deep neural network trained by the proposed synthetic dataset can produce a similar result for object detection tasks, which requires less time and labour. Moreover, this approach could be helpful in education where the textbook figure contents require to be unchanged, and high detection accuracy is required.

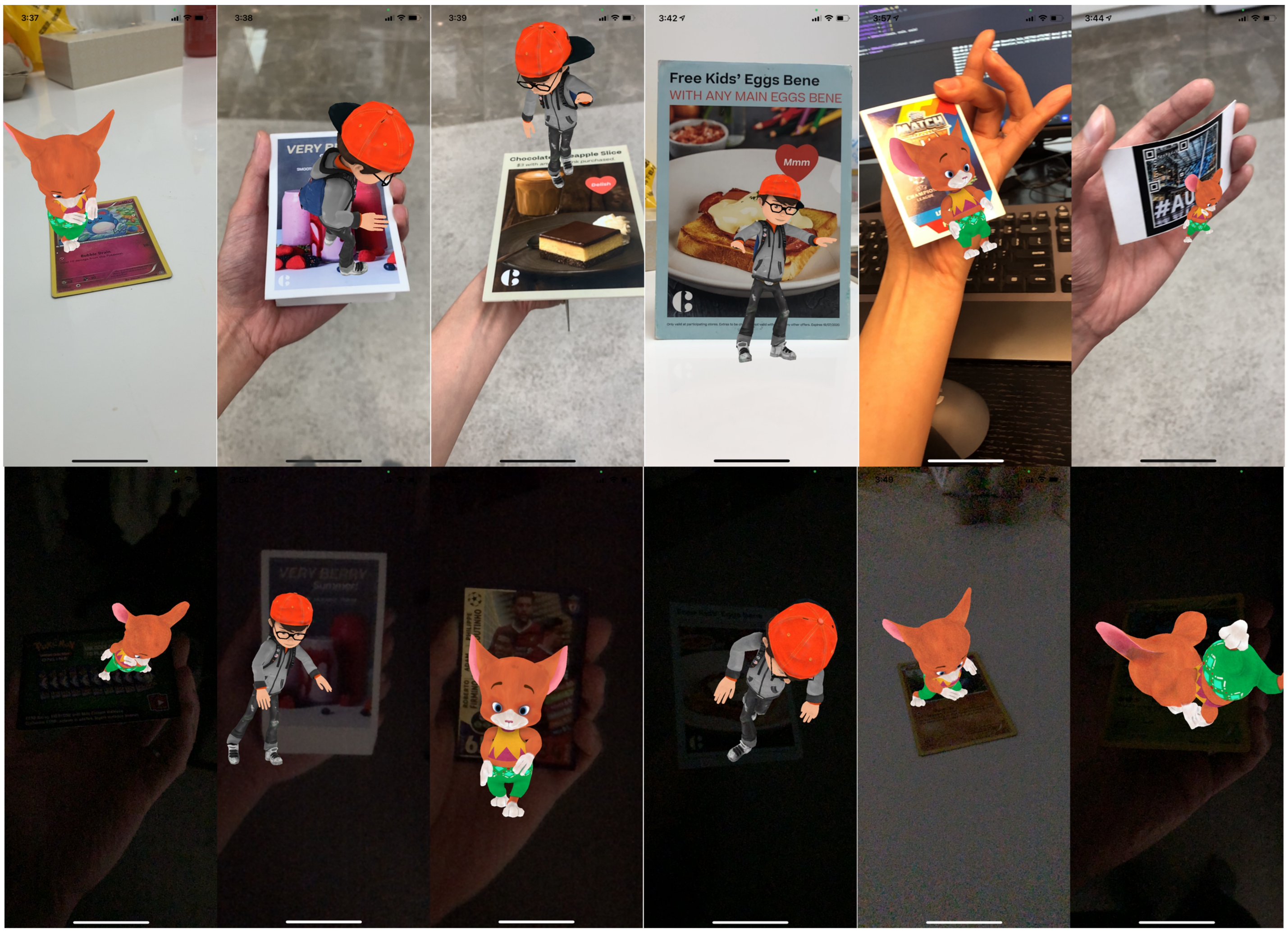


Figure 6.13: The prototype iOS application shows that deep neural network model trained by the proposed synthetic dataset can predict the markers under different lighting conditions and successfully render the corresponding virtual models.

Chapter 7

Conclusion and Future Research Directions

The research purpose of this thesis is divided into two different distinct directions. One is investigating the new Augmented Reality (AR) markers that conceal hidden information such as encrypted code into the graphical content with the minimal modification required. Another is investigating using the deep neural network trained by the synthetic dataset to classify the marker identity without modifying the original content. The proposed system introduced in the second investigation is called “**S**ynthetic data annotation system for **A**ugmented **R**eality **M**achine learning-based application” or **SARM**.

The newly designed AR markers introduced in the first investigation direction provide a higher detection accuracy rate under unexpected natural conditions and better error detection and correction capabilities. They also present meaningful information to the audiences with minimal modification of the original graphical contents. The commercial perspective also indicated that the newly designed AR markers have great usage potential in a few application sectors such as education and medicine. However, it also indicated that the original contents modification requirement is a primary disadvantage that pulls the proposed ideas further away from the daily commercialised application level.

Another research direction ultimately ruled out the physical texture content modification method by applying the object detection capability using the pre-trained deep neural network model to classify the given markers. This proposed idea combines the two most promising modern technologies, Machine Learning and Augmented Reality, into one product. The experimental results have shown that it is possible to use the digital generated synthetic data to train the deep neural network model and achieve equivalent outcomes when trained using the real data. This novel solution helps to improve the training performance, real-world object detection accuracy and minimises labour cost during the data annotation process. In the end, a fully functional prototype was implemented and detecting the AR markers efficiently on a 2017 built iPhone X device.

7.1 Limitations

Nothing is perfect, and this thesis also has some unique limitations. The **first major limitation** is the negative impact of the COVID 19 pandemic on the research progress. In the beginning, this project was planned to be completed within three years, starting from July 2018. However, due to the unexpected national lockdown incidents, the progress has been delayed for another five months since all the accessible research facilities and tools had been frozen. However, in the end, the project continued with the self-funding research equipment and online tools. This limitation somehow gave the advantage for us as the thesis author to critically proceed with the article proofreading process.

The **second limitation** came from the technical disadvantage aspect. The experimental results and the implemented application used only 2017 built iPhone X as the testing device. This mobile model contains the A11 bionic chip that is outdated, and it could lead to poor processing power issues. This chip was designed primarily for facial recognition and a few other Machine Learning-related tasks and minimal support for the Augmented Reality feature. Therefore, the performance of the prototype application started falling after a short period, and the virtual object presentation was not accurate in texture updating or recalculating the new physical location. However, these technical issues would be to be addressed readily by future hardware and architectural software designs.

7.2 Future research directions

The research clearly showed that Augmented Reality and Machine Learning are the most demanding technologies for the next couple of decades; hence, our future research directions will focus on these technical aspects. The **first future research direction** could be expanding the current project into other mobile operating systems such as iPadOS or Android OS since the current implemented application is only runnable on iOS devices. At the current stage, the proposed system works efficiently well with any flat 2D objects such as cards, papers, or books. In the future, the **other research direction** could be on detecting 3D objects such as 3D spheres, cubes, or cylinder-shaped objects using the latest built-in LiDAR (Rosell et al., 2009) scanning technology on most modern iPhone devices. This technology uses the light reflection time latency to determine how far the individual object is standing away from the phone location and creates a depth map of the surrounding space. The depth map helps the system to understand the surrounding environment instantly and accurately deliver the best Augmented Reality experience.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... others (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265–283).

Acharya, M., Hayes, T. L. & Kanan, C. (2020). Rodeo: Replay for online object detection. *arXiv preprint arXiv:2008.06439*.

Anthony, M. (2001). *Discrete mathematics of neural networks: selected topics*. SIAM.

Azuma, R. T. (1997). A survey of augmented reality. *Presence: Teleoperators and virtual environments*, *6*(4), 355–385.

Bajura, M., Fuchs, H. & Ohbuchi, R. (1992). Merging virtual objects with the real world: Seeing ultrasound imagery within the patient. In *Acm siggraph computer graphics* (Vol. 26, pp. 203–210).

Barten, P. G. (1999). *Contrast sensitivity of the human eye and its effects on image quality* (Vol. 21). Spie optical engineering press Bellingham, WA.

Billinghurst, M. & Duenser, A. (2012). Augmented reality in the classroom. *Computer*, *45*(7), 56–63.

Billinghurst, M., Kato, H. & Poupyrev, I. (2001). The magicbook-moving seamlessly between reality and virtuality. *Computer Graphics and Applications, IEEE*, *21*(3), 6–8.

Birchfield, S. & Tomasi, C. (1999). Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, *35*(3), 269–293.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bobeshko, A. (2017). Object recognition in augmented reality [Computer software manual]. (`https://virtualrealitypop.com/object-recognition-in-augmented-reality-8f7f17127a7a`)

Bora, D. J. & Gupta, A. K. (2016). Aerascis: An efficient and robust approach for satellite color image segmentation. In *Electrical power and energy systems (icepes), international conference on* (pp. 549–556).

Bradley, D. & Roth, G. (2007). Adaptive thresholding using the integral image. *Journal of Graphics Tools*, *12*(2), 13–21.

Braybrook, R. (1998). Looks can kill. *ARMADA INTERNATIONAL-ENGLISH EDITION-*, *22*, 44–49.

Brunelli, R. (2009). *Template matching techniques in computer vision: theory and practice*. John Wiley & Sons.

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*(6), 679–698.

Carmigniani, J. & Furht, B. (2011). Augmented reality: an overview. In *Handbook of augmented reality* (pp. 3–46). Springer.

Caudell, T. P. & Mizell, D. W. (1992). Augmented reality: An application of heads-up display technology to manual manufacturing processes. In *System sciences, 1992. proceedings of the twenty-fifth hawaii international conference on* (Vol. 2, pp. 659–669).

Chang, H.-Y., Wu, H.-K. & Hsu, Y.-S. (2013). Integrating a mobile augmented reality activity to contextualize student learning of a socioscientific issue. *British Journal of Educational Technology*, *44*(3), E95–E99.

Cheon, M., Lee, W., Hyun, C.-H. & Park, M. (2011). Rotation invariant histogram of oriented gradients. *International Journal of Fuzzy Logic and Intelligent Systems*, *11*(4), 293–298.

Chowdhury, M. H. & Little, W. D. (1995). Image thresholding techniques. In *Communications, computers, and signal processing, 1995. proceedings., ieee pacific rim conference on* (pp. 585–589).

Craig, A. B. (2013). *Understanding augmented reality: Concepts and applications*. Newnes.

Diebel, J. (2006). Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, *58*(15-16), 1–35.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J. & Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, *88*(2), 303–338.

Feiner, S., MacIntyre, B., Hollerer, T. & Webster, A. (1997). A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. In *Wearable computers, 1997. digest of papers., first international symposium on* (pp. 74–81).

Feiner, S. K. (2002). Augmented reality: A new way of seeing. *Scientific American*, *286*(4), 48–55.

Fisher, M. & Baird, D. E. (2006). Making mlearning work: Utilizing mobile technology for active exploration, collaboration, assessment, and reflection in higher education. *Journal of Educational Technology Systems*, *35*(1), 3–30.

Fukushima, K. (1969). Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, *5*(4), 322–333.

Furlan, R. (2016, June). The future of augmented reality: Hololens - microsoft's ar headset shines despite rough edges. *IEEE Spectrum*, *53*(6), 21-21. doi: 10.1109/MSPEC.2016.7473143

Gimel'farb, G. (1999). Stereo terrain reconstruction by dynamic programming. *Handbook of Computer Vision and Applications. Signal Processing and Pattern Recognition.*, *2*, 505-530.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the ieee international conference on computer vision* (pp. 1440–1448).

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 580–587).

Glorot, X., Bordes, A. & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 315–323).

Gong, R. (2011). *Belief Propagation Based Stereo Matching with Due Account of Visibility Conditions* (Unpublished master's thesis). The University of Auckland, New Zealand, Computer Science department.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). 6.2. 2.3 softmax units for multinoulli output distributions. *Deep learning*, 180–184.

Haas, J. (2014). A history of the unity game engine. *Diss. WORCESTER POLYTECHNIC INSTITUTE*.

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell Labs Technical Journal*, *29*(2), 147–160.

Hartley, R. & Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge university press.

Hawkins, D. M. (2004). The problem of overfitting. *Journal of chemical information and computer sciences*, *44*(1), 1–12.

He, K., Zhang, X., Ren, S. & Sun, J. (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, *37*(9), 1904–1916.

He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Hiatt, K. L., Rash, C. E., Harris, E. S. & Gilberry, W. H. (2004). *Apache aviator visual experiences with the ihadss helmet-mounted display in operation iraqi freedom* (Tech. Rep.). ARMY AEROMEDICAL RESEARCH LAB FORT RUCKER AL.

Hinton, G. E., Sejnowski, T. J. et al. (1999). *Unsupervised learning: foundations of neural computation*. MIT press.

Hugo Paigneau, D. A. (2020). *Playing cards lebelized dataset.* `https://www.kaggle.com/hugopaigneau/playing-cards-dataset`.

Janin, A. L., Mizell, D. W. & Caudell, T. P. (1993). Calibration of head-mounted displays for augmented reality applications. In *Virtual reality annual international symposium, 1993., 1993 ieee* (pp. 246–255).

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd acm international conference on multimedia* (pp. 675–678).

Kan, T.-W., Teng, C.-H. & Chou, W.-S. (2009). Applying qr code in augmented reality applications. In *Proceedings of the 8th international conference on virtual reality continuum and its applications in industry* (pp. 253–257).

Kato, H. & Billinghurst, M. (1999). Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings 2nd ieee and acm* *international workshop on augmented reality (iwar'99)* (pp. 85–94).

Kato, H., Billinghurst, M., Weghorst, S. & Furness, T. (1999). A mixed reality 3d conferencing application. *Human Interface Technology Laboratory*.

Kato, H. & Tan, K. T. (2007). Pervasive 2d barcodes for camera phone applications. *IEEE Pervasive Computing*, *6*(4).

Kieseberg, P., Leithner, M., Mulazzani, M., Munroe, L., Schrittwieser, S., Sinha, M. & Weippl, E. (2010). Qr code security. In *Proceedings of the 8th international conference on advances in mobile computing and multimedia* (pp. 430–435).

Kim, J.-H. (1998, June 30). *Contour approximation method for representing a contour of an object.* Google Patents. (US Patent 5,774,595)

Kipper, G. & Rampolla, J. (2013). Augmented reality, an emerging technology guide to ar. *Waltham MA: Elsevier Inc*.

Koch, C., Neges, M., König, M. & Abramovici, M. (2014). Natural markers for augmented reality-based indoor navigation and facility maintenance. *Automation in Construction*, *48*, 18–30.

Kolmogorov, V. & Zabih, R. (2002, 27 May - 2 Jun). Multi-camera Scene Reconstruction via Graph Cuts. In *Proceedings of the european conference on computer vision* (Vol. 2352, pp. 82–96). Copenhagen, Germany.

Konolige, K. (1997, Oct). Small vision systems: Hardware and implementation. In *Proceedings of the international symposium on robotics research* (Vol. 8, pp. 203–212). Kanagawa, Japan.

Koza, J. R., Bennett, F. H., Andre, D. & Keane, M. A. (1996). Automated design of both the topology and sizing of analog electrical circuits using genetic programming. In *Artificial intelligence in design'96* (pp. 151–170). Springer.

Kretschmer, U., Coors, V., Spierling, U., Grasbon, D., Schneider, K., Rojas, I. & Malaka, R. (2001). Meeting the spirit of history. In *Proceedings of the 2001 conference on virtual reality, archeology, and cultural heritage* (pp. 141–152).

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Le, H. & Nguyen, M. (2017). Enhancing textbook study experiences with pictorial barcodes and augmented reality. In *International conference on computer analysis of images and patterns* (pp. 139–150).

Lee, K. (2012). Augmented reality in education and training. *TechTrends*, *56*(2), 13–21.

Lee, K.-M. & Shah, D. K. (1988). Kinematic analysis of a three-degrees-of-freedom in-parallel actuated manipulator. *IEEE Journal on Robotics and Automation*, *4*(3), 354–360.

Lehmann, E. L. & Casella, G. (2006). *Theory of point estimation*. Springer Science & Business Media.

Lepetit, V. & Fua, P. (2006). Keypoint recognition using randomized trees. *IEEE transactions on pattern analysis and machine intelligence*, *28*(9), 1465–1479.

Lepetit, V., Lagger, P. & Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *2005 ieee computer society conference on computer vision and pattern recognition (cvpr'05)* (Vol. 2, pp. 775–781).

Lewis, J. P. (1995). Fast normalized cross-correlation. In *Vision interface* (Vol. 10, pp. 120–123).

Li, H., Hestenes, D. & Rockwood, A. (2001). Generalized homogeneous coordinates for computational geometry. In *Geometric computing with clifford algebras* (pp. 27–59). Springer.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., … Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740–755).

Lindeberg, T. (2012). Scale invariant feature transform.

Linowes, J. & Babilinski, K. (2017). *Augmented reality for developers: Build practical augmented reality applications with unity, arcore, arkit, and vuforia*. Packt Publishing Ltd.

Liu, T.-Y., Tan, T.-H. & Chu, Y.-L. (2010). Qr code and augmented reality-supported mobile english learning system. In *Mobile multimedia processing* (pp. 37–52). Springer.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh ieee international conference on computer vision* (Vol. 2, pp. 1150–1157).

Malaka, R., Schneider, K. & Kretschmer, U. (2004). Stage-based augmented edutainment. In *International symposium on smart graphics* (pp. 54–65).

Mamoshina, P., Vieira, A., Putin, E. & Zhavoronkov, A. (2016). Applications of deep learning in biomedicine. *Molecular pharmaceutics*, *13*(5), 1445–1454.

McGuigan, J. (2020). *52 playing cards.* `https://www.kaggle.com/jamesmcguigan/playingcards/discussion`.

Middlebury stereo vision webpage. (2001). Retrieved from `http://vision.middlebury.edu/stereo/`

Milgram, P. & Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, *77*(12), 1321–1329.

Mitchell, T. M. (1980). *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research ….

Mitchell, T. M. et al. (1997). Machine learning.

Mohan, A., Woo, G., Hiura, S., Smithwick, Q. & Raskar, R. (2009). Bokode: imperceptible visual tags for camera based interaction from a distance. In *Acm transactions on graphics (tog)* (Vol. 28, p. 98).

Mur-Artal, R., Montiel, J. M. M. & Tardos, J. D. (2015). Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, *31*(5), 1147–1163.

Mur-Artal, R. & Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, *33*(5), 1255–1262.

Narzt, W., Pomberger, G., Ferscha, A., Kolb, D., Müller, R., Wieghardt, J., … Lindinger, C. (2006). Augmented reality navigation systems. *Universal Access in the Information Society*, *4*(3), 177–187.

Neubeck, A. & Van Gool, L. (2006). Efficient non-maximum suppression. In *Pattern* *recognition, 2006. icpr 2006. 18th international conference on* (Vol. 3, pp. 850–855).

Nguyen, M., Chan, Y. H., Delmas, P. & Gimel'farb, G. (2013). Symmetric dynamic programming stereo using block matching guidance. In *Image and vision computing new zealand (ivcnz), 2013 28th international conference of* (pp. 88–93).

Nguyen, M., Tran, H., Le, H. & Yan, W. Q. (2017). A tile based colour picture with hidden qr code for augmented reality and beyond. In *Proceedings of the 23rd acm symposium on virtual reality software and technology* (p. 8).

Nosal, E.-M. (2008). Flood-fill algorithms used for passive acoustic detection and tracking. In *2008 new trends for environmental monitoring using passive systems* (pp. 1–5).

Olson, E. (2011). Apriltag: A robust and flexible visual fiducial system. In *Robotics and automation (icra), 2011 ieee international conference on* (pp. 3400–3407).

OpenCV, L. (2008). Computer vision with the opencv library. *GaryBradski & Adrian Kaebler-O'Reilly*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... others (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Plank, J. S. et al. (1997). A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Softw., Pract. Exper.*, *27*(9), 995–1012.

Previc, F. H. & Ercoline, W. R. (2004). *Spatial disorientation in aviation* (Vol. 203). Aiaa.

Provost, F. & Kohavi, R. (1998). Glossary of terms. *Journal of Machine Learning*, *30*(2-3), 271–274.

Pv, S. (2021). C++ and unreal engine 4. In *Beginning unreal engine 4 blueprints visual scripting* (pp. 33–54). Springer.

Quan, L. & Lan, Z. (1999). Linear n-point camera pose determination. *IEEE Transactions on pattern analysis and machine intelligence*, *21*(8), 774–780.

Rauschnabel, P. A., Brem, A. & Ro, Y. (2015). Augmented reality smart glasses: definition, conceptual insights, and managerial importance. *Unpublished Working Paper, The University of Michigan-Dearborn, College of Business*.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 779–788).

Redmon, J. & Farhadi, A. (2017). Yolo9000: better, faster, stronger. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 7263–7271).

Redmon, J. & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Reed, I. S. & Solomon, G. (1960). Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, *8*(2), 300–304.

Rekimoto, J. & Ayatsuka, Y. (2000). Cybercode: designing augmented reality environments with visual tags. In *Proceedings of dare 2000 on designing augmented reality environments* (pp. 1–10).

Ren, S., He, K., Girshick, R. & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*.

Rosell, J. R., Llorens, J., Sanz, R., Arno, J., Ribes-Dasi, M., Masip, J., … others (2009). Obtaining the three-dimensional structure of tree orchards from remote 2d terrestrial lidar scanning. *Agricultural and Forest Meteorology*, *149*(9), 1505–1515.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Russel, S. & Norvig, P. (2003). Artificial intelligence: A modern approach, 2003. *EUA: Prentice Hall*, *178*.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, *3*(3), 210–229.

Sasi, N. M. & Jayasree, V. (2013). Contrast limited adaptive histogram equalization for qualitative enhancement of myocardial perfusion images. *Engineering*, *5*(10), 326.

Schapire, R. E. & Freund, Y. (2012). Foundations of machine learning.

Serino, M., Cordrey, K., McLaughlin, L. & Milanaik, R. L. (2016). Pokémon go and augmented virtual reality games: a cautionary commentary for parents and pediatricians. *Current opinion in pediatrics*, *28*(5), 673–677.

Shawe-Taylor, J., Cristianini, N. et al. (2004). *Kernel methods for pattern analysis*. Cambridge university press.

Shorten, C. & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, *6*(1), 1–48.

Soon, T. J. (2008). Qr code. *Synthesis Journal*, *2008*, 59–78.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Stevenson, R. (2005). Laser marking matrix codes on pcbs. *Printed Circuit Design and Manufacture*, *22*(12), 32.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, *36*(2), 111–133.

Sturm, P. (2014). Pinhole camera model. In *Computer vision* (pp. 610–613). Springer.

Sutherland, I. E. (1968). A head-mounted three dimensional display. In *Proceedings of the december 9-11, 1968, fall joint computer conference, part i* (pp. 757–764).

Tan, M., Pang, R. & Le, Q. V. (2020). Efficientdet: Scalable and efficient object detection. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 10781–10790).

Tikanmäki, A. & Röning, J. (2011). Markers–toward general purpose information representation. In *Iros2011 workshop on knowledge representation for autonomous robots*.

Tyler, C. W. & Clarke, M. B. (1990). Autostereogram. In *Sc-dl tentative* (pp. 182–197).

Van Krevelen, D. & Poelman, R. (2007). Augmented reality: Technologies, applications, and limitations.

Van Otterlo, M. & Wiering, M. (2012). Reinforcement learning and markov decision processes. In *Reinforcement learning* (pp. 3–42). Springer.

Vuforia. (2021). Retrieved from `https://www.vuforia.com/`

Wang, Y. P. (1993, September 7). *System for encoding and decoding data in machine readable graphic form.* Google Patents. (US Patent 5,243,655)

Werbos, P. J. (1994). *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting* (Vol. 1). John Wiley & Sons.

Wikipedia. (2017). *Ramer–douglas–peucker algorithm.internet.* `https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm`. ([Online; accessed 10-May-2018])

Wu, J., Cui, Z., Sheng, V. S., Zhao, P., Su, D. & Gong, S. (2013). A comparative study of sift and its variants. *Measurement science review*, *13*(3), 122–131.

Zuiderveld, K. (1994). Contrast limited adaptive histogram equalization. *Graphics gems*, 474–485.

Appendix A

Code for implementing the iOS Application Using Swift pogramming language

A.1 View Controller Class

```swift
import UIKit
import SceneKit
import ARKit
import AVFoundation


class ViewController: UIViewController, ARSCNViewDelegate,
    ARSessionDelegate, YoloDelegate, AlteredImageDelegate{

    // MARK: - Initialisation

    @IBOutlet var sceneView: ARSCNView!

    static var instance: ViewController?

    var isCurrentlyPredicting: Bool = false

    var currentExaminedClass: AlteredImage?

    var currentClassName: String = ""

    let yolo = Yolo()

    var currentScreenTransform: CGAffineTransform?

    var imageOrientation: CGImagePropertyOrientation?

    var screenBounds : CGRect?


    override func viewDidLoad() {
        super.viewDidLoad()

        yolo.delegate = self

        // Set the view's delegate
        sceneView.preferredFramesPerSecond = 60

        sceneView.delegate = self
        sceneView.session.delegate = self


        let soundSession = AVAudioSession.sharedInstance()
        try? soundSession.setActive(false)
        try! soundSession.setCategory(.playAndRecord, options: [.
defaultToSpeaker,
                                                                         .
allowBluetooth,
                                                                         .
allowAirPlay])
        try! soundSession.setActive(true)

    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        ViewController.instance = self

        let configuartion = ARWorldTrackingConfiguration()
        configuartion.environmentTexturing = .automatic

        sceneView.session.run(configuartion)
        resetImageTrack()
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)

        // Pause the view's session
        sceneView.session.pause()
    }

    /// This method used when the program first started or when the system couldn't track any detected images
    func resetImageTrack(){
        currentExaminedClass?.delegate = nil
        currentExaminedClass = nil
        currentClassName = ""
        /// Restart the session and remove any image anchors that may have been detected previously.
        runImageTrackingSession(with: [], runOptions: [.removeExistingAnchors, .resetTracking])
    }

    /// Reset the image reference tracking if current image lost
    private func runImageTrackingSession(with trackingImages: Set<ARReferenceImage>,
                                         runOptions: ARSession.RunOptions = [.removeExistingAnchors]){
        let configuration = ARImageTrackingConfiguration()
        configuration.trackingImages = trackingImages
        configuration.maximumNumberOfTrackedImages = 1

        sceneView.session.run(configuration, options: runOptions)

    }

    // MARK: - Methods


    func session(_ session: ARSession, didUpdate frame: ARFrame) {
        screenBounds = self.sceneView.bounds
        self.currentScreenTransform = frame.
   displayTransformCorrected(
            for: interfaceOrientation,
            viewportSize: screenBounds!.size
        )

        imageOrientation = .up
    }


    func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode
   , for anchor: ARAnchor) {

        currentExaminedClass?.add(anchor, node: node)
    }

    func renderer(_ renderer: SCNSceneRenderer, didUpdate node:
   SCNNode, for anchor: ARAnchor) {
        currentExaminedClass?.update(anchor)
    }



```

```swift
 func getIntersectRect(perspectiveImageList: [CIImage], observation: Yolo.Prediction,
                            rectangleList: [VNRectangleObservation]) {

     /// get iou value of yolo bounding boxes and the rectangles
     let iouIndexList: [Int] = calculateIOU(observation: observation,
                                                         rectangleList: rectangleList,
                                                         currentScreenTransform: self.currentScreenTransform!,
                                                         screenSize: screenBounds!)

     DispatchQueue.main.async { [self] in
         for iouIndex in iouIndexList{

             /// Ignore when the iou index is -1 --> means that the iou value is too small to be considered as a match.
             if iouIndex > -1{
                 guard self.currentExaminedClass == nil else {
                     return
                 }

                 guard let referenceImagePixelBuffer =
                         perspectiveImageList[iouIndex].toPixelBuffer(pixelFormat: kCVPixelFormatType_32BGRA) else {
                     print("Error: Could not convert rectangle content into an ARReferenceImage.")
                     return
                 }

```

```
                /*
                 Set a default physical width of 50 centimeters
 for the new reference image.
                 While this estimate is likely incorrect, that's
  fine for the purpose of the
                 app. The content will still appear in the
 correct location and at the correct
                 scale relative to the image that's being
 tracked.
                 */

                let possibleReferenceImage = ARReferenceImage(
 referenceImagePixelBuffer,

 orientation: .up,

 physicalWidth: CGFloat(0.07))
                possibleReferenceImage.validate { [self] (error)
  in
                    if let error = error {
                        print("Reference image validation failed
 : \(error.localizedDescription)")
                        return
                    }


                    guard let newAlteredImage = AlteredImage(
 perspectiveImageList[iouIndex],

 referenceImage: possibleReferenceImage,

 className: labels[observation.classIndex]) else {
```

```
                                        return
                                    }

                                    newAlteredImage.delegate = self
                                    self.currentExaminedClass = newAlteredImage
                                    currentClassName = self.currentExaminedClass
    !.className

                                    self.runImageTrackingSession(with: [
    newAlteredImage.referenceImage!])
                                }

                            }
                        }
                    }
            }

    func alteredImageLostTracking(_ alteredImage: AlteredImage) {
        resetImageTrack()
    }
}
```

A.2 Yolo Class

```
import Foundation
import UIKit
import CoreML
import Vision
import UIKit
```

```swift

class Yolo{

    // MARK: - Initialisation
    // Perform the prediction each interval of time
    private var updateTimer: Timer?
    private var updateInterval: TimeInterval = 0.03

    private var educationModel: VNCoreMLModel?

    weak var delegate: YoloDelegate?

    private var currentCameraImage: CVPixelBuffer!

    let gridHeight = [13, 26, 52]
    let gridWidth = [13, 26, 52]
    let blockSize: Float = 32
    let boxesPerCell = 3
    let numClasses = 22


    public static let maxBoundingBoxes = 10
    let confidenceThreshold: Float = 0.3
    let iouThreshold: Float = 0.1

    struct Prediction {
      let classIndex: Int
      let score: Float
      let rect: CGRect
    }

    // MARK: - Methods
```

```
 public init() {
     educationModel = {
                 let modelConfig = MLModelConfiguration()
             modelConfig.computeUnits = .all
             let model = try? VNCoreMLModel(for: Education_416(configuration: modelConfig).model)
             return model
     }()

     self.updateTimer = Timer.scheduledTimer(withTimeInterval: updateInterval, repeats: true) { [weak self] _ in
             if let capturedImage = ViewController.instance?.sceneView.session.currentFrame?.capturedImage {
                 self?.predict(imagePixel: capturedImage)
             }
     }
 }

 public func predict(imagePixel: CVPixelBuffer) {



     // If the system is currently searching for the rectangles --> STOP
     guard !(ViewController.instance!.isCurrentlyPredicting) else {
             return
     }

     guard ViewController.instance!.currentClassName == ""  else {
             return
     }
```

```

        ViewController.instance!.isCurrentlyPredicting = true

        currentCameraImage = imagePixel

        let request = VNCoreMLRequest(model: educationModel!,
    completionHandler: { (request, error) in

            guard let observations = request.results as? [
    VNCoreMLFeatureValueObservation] else{
                ViewController.instance!.isCurrentlyPredicting =
    false
                ViewController.instance!.currentClassName = ""
                return
            }

            let predictions = self.getBoundingBoxes(features: [
    observations[2].featureValue.multiArrayValue!,

    observations[1].featureValue.multiArrayValue!,

    observations[0].featureValue.multiArrayValue!])

            if predictions.count > 0{
                let observation = predictions[0]



                self.searchForRectangle(observation: observation)
            }else{
                ViewController.instance!.isCurrentlyPredicting =
    false
```

```
                    ViewController.instance!.currentClassName = ""
                    return
                }

        })

        request.imageCropAndScaleOption = .scaleFill

        let imageRequestHandler = VNImageRequestHandler(
    cvPixelBuffer: currentCameraImage,
                                                            orientation:
     (ViewController.instance?.imageOrientation)!,
                                                        options: [:])

        // Perform request on background thread
        DispatchQueue.global(qos: .background).async {

            do {

                    try imageRequestHandler.perform([request])
                } catch {
                    print("Error: Vision request failed ")
                    ViewController.instance!.isCurrentlyPredicting =
     false
                    ViewController.instance!.currentClassName = ""
                }
        }

    }

    public func getBoundingBoxes(features: [MLMultiArray]) -> [
    Prediction] {
```

```swift


        assert(features[0].count == (numClasses+5)*3*13*13)
        assert(features[1].count == (numClasses+5)*3*26*26)
        //assert(features[2].count == (numClasses+5)*3*52*52)

        var predictions = [Prediction]()

        var featurePointer = UnsafeMutablePointer<Float32>(
    OpaquePointer(features[0].dataPointer))
        var channelStride = features[0].strides[0].intValue
        var yStride = features[0].strides[1].intValue
        var xStride = features[0].strides[2].intValue

        func offset(_ channel: Int, _ x: Int, _ y: Int) -> Int {
          return channel*channelStride + y*yStride + x*xStride
        }

        for i in 0..<2 {
            featurePointer = UnsafeMutablePointer<Float32>(
    OpaquePointer(features[i].dataPointer))
            channelStride = features[i].strides[0].intValue
            yStride = features[i].strides[1].intValue
            xStride = features[i].strides[2].intValue

            for cy in 0..<gridHeight[i] {
                for cx in 0..<gridWidth[i] {
                    for b in 0..<boxesPerCell {
                        let channel = b*(numClasses + 5)

                        // The fast way:
```

```
                let tx = Float(featurePointer[offset(channel, cx, cy)])
                let ty = Float(featurePointer[offset(channel + 1, cx, cy)])
                let tw = Float(featurePointer[offset(channel + 2, cx, cy)])
                let th = Float(featurePointer[offset(channel + 3, cx, cy)])
                let tc = Float(featurePointer[offset(channel + 4, cx, cy)])



                let scale = powf(2.0,Float(i)) // scale pos by 2^i where i is the scale pyramid level
                let x = (Float(cx) * blockSize + sigmoid(tx))/scale
                let y = (Float(cy) * blockSize + sigmoid(ty))/scale

                let w = exp(tw) * anchors[i][2*b    ]
                let h = exp(th) * anchors[i][2*b + 1]

                let confidence = sigmoid(tc)

                var classes = [Float](repeating: 0, count: numClasses)
                for c in 0..<numClasses {

                    // The fast way:
                    classes[c] = Float(featurePointer[offset(channel + 5 + c, cx, cy)])
                }
                classes = softmax(classes)
                let (detectedClass, bestClassScore) = classes.argmax()

                let confidenceInClass = bestClassScore * confidence



                if confidenceInClass > confidenceThreshold {
                    let rect = CGRect(x: CGFloat(x - w/2), y: CGFloat(y - h/2),
                                      width: CGFloat(w), height: CGFloat(h))

                    let prediction = Prediction(classIndex: detectedClass,
                                                score: confidenceInClass,
                                                rect: rect)

                    predictions.append(prediction)
                }

            }
        }
    }

}
return nonMaxSuppression(boxes: predictions, limit: Yolo.maxBoundingBoxes, threshold: iouThreshold)
}


public func searchForRectangle(observation : Prediction) {

    // Note that the pixel buffer's orientation doesn't change even when the device rotates.
    let handler = VNImageRequestHandler(cvPixelBuffer: currentCameraImage, orientation: .up)

    // Create a Vision rectangle detection request for running on the GPU.
    let request = VNDetectRectanglesRequest { request, error in
        self.completedVisionRequest(request, error: error, observation: observation)
    }

    // Look only for one rectangle at a time.
    request.maximumObservations = 5

    // Require rectangles to be reasonably large.
    request.minimumSize = 0.2

    request.minimumConfidence = 0.0

    // Ignore rectangles with a too uneven aspect ratio.
    request.minimumAspectRatio = 0.3

    // You leverage the `usesCPUOnly` flag of `VNRequest` to decide whether your Vision requests are processed on the CPU or GPU.
    // This sample disables ‘usesCPUOnly‘ because rectangle detection isn't very taxing on the GPU. You may benefit by enabling
    // ‘usesCPUOnly‘ if your app does a lot of rendering, or runs a complicated neural network.
    request.usesCPUOnly = true
    try? handler.perform([request])
}

private func completedVisionRequest(_ request: VNRequest?, error: Error?,
                                    observation :Prediction) {
    defer {
        ViewController.instance!.isCurrentlyPredicting = false
        self.currentCameraImage = nil
    }
    do{
        guard let rectangles = request?.results as? [VNRectangleObservation] else {
            ViewController.instance!.isCurrentlyPredicting = false
            ViewController.instance!.currentClassName = ""
            return
        }

        guard let filter = CIFilter(name: "CIPerspectiveCorrection") else {
            ViewController.instance!.isCurrentlyPredicting = false
            ViewController.instance!.currentClassName = ""
            return
        }

        //Only determine the rectangles corrdinates if the total is at least 1
        if rectangles.count > 0 {
            var perspectiveImageList: [CIImage] = []
            var rectangleList: [VNRectangleObservation] = []

            for rectangle in rectangles{
                let width = CGFloat(CVPixelBufferGetWidth(currentCameraImage))
                let height = CGFloat(CVPixelBufferGetHeight(currentCameraImage))
                let topLeft = CGPoint(x: rectangle.topLeft.x * width, y: rectangle.topLeft.y * height)
                let topRight = CGPoint(x: rectangle.topRight.x * width, y: rectangle.topRight.y * height)
                let bottomLeft = CGPoint(x: rectangle.bottomLeft.x * width, y: rectangle.bottomLeft.y * height)
                let bottomRight = CGPoint(x: rectangle.bottomRight.x * width, y: rectangle.bottomRight.y * height)

                filter.setValue(CIVector(cgPoint: topLeft), forKey: "inputTopLeft")
                filter.setValue(CIVector(cgPoint: topRight), forKey: "inputTopRight")
                filter.setValue(CIVector(cgPoint: bottomLeft), forKey: "inputBottomLeft")
                filter.setValue(CIVector(cgPoint: bottomRight), forKey: "inputBottomRight")

                let ciImage = CIImage(cvPixelBuffer: currentCameraImage).oriented(.up)
                filter.setValue(ciImage, forKey: kCIInputImageKey)

                guard let perspectiveImage: CIImage = filter.value(forKey: kCIOutputImageKey) as? CIImage else {
                    print("Error: Rectangle detection failed - perspective correction filter has no output image.")
                    ViewController.instance!.isCurrentlyPredicting = false
                    ViewController.instance!.currentClassName = ""
                    return
                }

                rectangleList.append(rectangle)
                perspectiveImageList.append(perspectiveImage)
            }

            delegate?.getIntersectRect(perspectiveImageList: perspectiveImageList,
                                       observation: observation, rectangleList: rectangleList)

        }else{
            ViewController.instance!.isCurrentlyPredicting = false
            ViewController.instance!.currentClassName = ""
            return
        }
    }
}
}
```

```

// MARK: - Protocol
protocol YoloDelegate: class {
    func getIntersectRect(perspectiveImageList: [CIImage],
                                  observation : Yolo.Prediction,
                                  rectangleList: [VNRectangleObservation])
}
```

A.3 Altered Image Class

```
import Foundation
import ARKit
import CoreML

/// - Tag: AlteredImage
class AlteredImage  {
    // MARK: - Initialisation
    /// A delegate to tell when image tracking fails.
    weak var delegate: AlteredImageDelegate?

    public var className: String = ""

    public var referenceImage: ARReferenceImage?

    /// A SceneKit node that animates images of varying style.
    private let visualizationNode: VisualizationNode

    /// A handle to the anchor ARKit assigned the tracked image.
    private(set) var anchor: ARImageAnchor?

    /// A timer start every second checking whether the
imageReference still detectable or lost
    private var failedTrackingTimeout: Timer?

    private var timeout: TimeInterval = 0.03

    // MARK: - Methods
    init?(_ image: CIImage, referenceImage: ARReferenceImage,
className: String) {
        self.className = className
        self.referenceImage = referenceImage
        anchor?.setValue(className, forKey: "className")
        visualizationNode = VisualizationNode(referenceImage.
physicalSize, className: className)

        // Start the failed tracking timer right away. This ensures
that the app starts
        //  looking for a different image to track if this one isn't
 trackable.
        resetImageTrackingTimeout()
    }

    deinit {
        visualizationNode.removeAllAnimations()
        visualizationNode.removeFromParentNode()
    }

    /// Prevents the image tracking timeout from expiring.
    private func resetImageTrackingTimeout() {
        failedTrackingTimeout?.invalidate()
        failedTrackingTimeout = Timer.scheduledTimer(
    withTimeInterval: timeout, repeats: true) {
            [weak self] _ in
            if let strongSelf = self {
                self?.delegate?.alteredImageLostTracking(strongSelf)
            }
        }
    }

    func add(_ anchor: ARAnchor, node: SCNNode) {
        if let imageAnchor = anchor as? ARImageAnchor, imageAnchor.
    referenceImage == referenceImage {
            self.anchor = imageAnchor

            // Add the node that displays the altered image to the
    node graph.
            node.addChildNode(visualizationNode)
        }
    }

    func update(_ anchor: ARAnchor) {

        if let imageAnchor = anchor as? ARImageAnchor, self.anchor
    == anchor {
            self.anchor = imageAnchor
            if !imageAnchor.isTracked {
                resetImageTrackingTimeout()
            }

        }
    }
}
```

```

// MARK: - Protocol
/**
 Tells a delegate when image tracking failed.
  In this case, the delegate is the view controller.
 */
protocol AlteredImageDelegate: class {
    func alteredImageLostTracking(_ alteredImage: AlteredImage)
}
```

A.4 Visualisation Node Class

```
import Foundation
import SceneKit
import ARKit

class VisualizationNode: SCNNode {

    // MARK: - Initialisation
    public let currentImage: SCNNode

    public let className: String?
    weak var delegate: VisualizationNodeDelegate?

    // MARK: - Methods
    public init(_ size: CGSize, className: String) {

        var path = ""
        var songPath = ""
        var scale : Float = 0
        if className == "tangela"{
            path = "art.scnassets/BreakDance_Mouse.dae"
            scale = 0.0005
            songPath = "art.scnassets/Numb-LinkinPark.mp3"
        }
        else if className == "chocolate" || className == "bread" || className == "milkshake"{
            path = "art.scnassets/HipHopDancing.dae"
            scale = 0.0004
            songPath = "art.scnassets/How You Like That.mp3"
        }
        else{
            path = "art.scnassets/SalsaDance_Mouse.dae"
            scale = 0.0005
            songPath = "art.scnassets/Numb-LinkinPark.mp3"
        }

        currentImage = createPlaneNode(size: size, rotation: -.pi / 2, contents: UIColor.clear,
                                          objectPath: path, scale: scale, songPath: songPath)
        self.className = className

        super.init()
        addChildNode(currentImage)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func display(){

    }

}

// MARK: - Protocol
/// Tells a delegate when the fade animation is done.
/// In this case, the delegate is an AlteredImage object.
protocol VisualizationNodeDelegate: class {
    //func visualizationNodeDidFinishFade(_ visualizationNode:
    VisualizationNode)
}
```