2018-2019
COMP 4801 Final Year Project

Intelligent 3D Printed Robotic Arm
LegoARM

Interim Report

Student
Hung Lok Shun
3035184192

Student
Yiu Lok Yan Serena
3035179721

Supervisor
Dr. Y. K. Choi

Submission Date
20th January, 2018
Abstract

Sorting Lego pieces has been an issue to Lego builders and enthusiasts. The current solution adopted by many is to do it manually, as automation solutions that can be easily adopted are yet to be developed. This project is proposed to explore the potential solutions with an intelligent robotic arm that implements object recognition using supervised deep learning. This project is divided into three main stages: Planning and Experimentation, Algorithm Study and Implementation and Algorithm Integration. At present, the project is in the second stage. The focus of this stage is to study various machine learning techniques and fine-tune different parameters to increase recognition accuracy. The objectives of the project are finalised and the methodology design is refined based on preliminary study and experiments done on related subjects.
Acknowledgement

We would like to express our gratitude to our supervisor Dr. Y. K. Choi from the Department of Computer Science for her guidance and advice in the project. She has always provided timely and constructive feedback.

We would also like to thank Mr. David Lee from the HKU MakerLab to provide support on robotics. His technical expertise has been very helpful during implementation and troubleshooting.

We would also like to express our appreciation to Ms. Joanna Lee from the Center of Applied English for providing valuable suggestions regarding the preparation of reports and presentations.

Finally, we would like to thank the University of Hong Kong (HKU) for providing an opportunity to experiment and apply knowledge on practical applications through the project.
# Table of Contents

Abstract 2  
Acknowledgement 3  
Table of Contents 4  
List of Figures 6  
List of Tables 6  
Abbreviations 7  
1 Introduction 8  
  1.1 Background 8  
  1.2 Motivation 8  
  1.3 Objectives 9  
  1.4 Scope 10  
  1.5 Outline of the Report 11  
2 Related Works 12  
  2.1 Automatic Sorters 12  
    2.1.1 Automatic Lego Liftarm Sorter LS-40 [8] 12  
  2.2 Object Detection 13  
    2.2.1 Faster R-CNN 13  
    2.2.2 You Only Look Once (YOLOv3) 13  
  2.3 Object Classification on synthesized images 14  
3 Methodology 15  
  3.1 Prerequisites 15  
    3.1.1 Hardware 15  
    3.1.2 Software 15  
  3.2 Robotic Arm 16  
    3.2.1 Robotic Arm Configurability 17  
    3.2.2 Controlling Articulated Joints 17  
    3.2.3 Encapsulation of Robotic Arm Control in a Python Library 17  
    3.2.4 The Robotic Arm Software Interface 18  
  3.3 Machine Learning Model 18  
    3.3.1 Data Collection Method 18  
    3.3.2 Synthesizing Data 19  
    3.3.3 Convolutional Neural Network 19  
    3.3.4 Data Augmentation 20  
    3.3.5 Evaluation Metrics 20  
    3.3.6 Customized Neural Network 20
3.3.7 Applying Feature Extractor
3.3.8 Object Recognition
3.4 Camera Calibration

4 Results and Findings

4.1 Robotic Arm Head Module Designs
4.2 Rendering Lego 3D Model Files
  4.2.1 Selecting 3D Model Files
  4.2.2 Choice of Rendering Engines
  4.2.3 Extracting 3D Models
  4.2.4 Automation with Modified Rendering Engines
  4.2.5 Data Generation with Scripted Automatic Rendering
4.3 Machine Learning Model Training
  4.3.1 Comparison of Output Model Trained using Real Photographs and Synthesized Images
  4.3.2 Comparison between Before and After Applying Regularization
  4.3.3 Comparison between Fine-tuned Pre-trained Models
  4.3.4 Comparison between Before and After Applying Feature Extractor
  4.3.5 Comparison between gray-scale and coloured images in object recognition
  4.3.6 Summary of the experiment results

5 Limitations and Discussion

5.1 Synthesizing Data
  5.1.1 Rendering Automation Bottleneck
  5.1.2 Rendering Scene Configurability
5.2 Machine Learning Model Training
  5.2.1 Choice of data samples
  5.2.2 Number of training iterations
  5.2.3 Limited evaluation method

6 Future Work

7 Conclusion

8 Reference
List of Figures

Figure 1 Detail drawing of the robotic arm design by ftobler ................................................. 16
Figure 2 Dependency relationships of software components of the robotic arm .................. 16
Figure 3 A conceptual illustration of how feature extractor works [31]................................. 21
Figure 4 Head modules tested for picking and placing Lego pieces: 3-jaw angled gripper (left), suction cup(middle), parallel gripper (right)...................................................... 23
Figure 5 Rendering pipeline using different Lego CAD software........................................ 25
Figure 6 Rendered images of Lego element 4121715 produced using POV-Ray (left), BlueRender (middle), and Blender (right)................................................................. 26
Figure 7 Extracts of the file types .ldr and .lxf of a Lego model created in LDD ............... 27
Figure 8 Inheritance tree of the original BlueRender program (left), and the modified version using an injection point on life-cycle initialization.................................................. 29
Figure 9 Batched image rendering of the data generator ....................................................... 30
Figure 10 Comparison of training metrics between the real photograph dataset and the synthetic image dataset ......................................................................................................... 34
Figure 11 Comparison of training metrics before and after applying regularization to avoid overfitting................................................................................................................ 35
Figure 12 Comparison of training metrics before and after applying regularization to avoid overfitting, using an augmented dataset with size of 2835.............................................. 36
Figure 13 Comparison of training metrics for untrainable VGG-16 and ResNet-50 .......... 37
Figure 14 Comparison of training metrics for training last 4 layers of VGG-16 and ResNet-50 ................................................................. 38
Figure 15 Comparison of training metrics for using of VGG-16 as feature extractor ........ 39
Figure 16 Comparison of training metrics for using of ResNet-50 as feature extractor .... 39
Figure 17 Comparison of training metrics on using gray-scale images................................ 40
Figure 18 Samples of wrong predictions ............................................................................. 41

List of Tables

Table 1 Sample images of the 3 Lego used in the experiment ................................................. 31
Table 2 The main setup of the customized neural network used in the experiment ............... 32
Table 3 Project Schedule ....................................................................................................... 45
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKU</td>
<td>The University of Hong Kong</td>
</tr>
<tr>
<td>STEM</td>
<td>Science, Technology, Engineering, and Mathematics</td>
</tr>
<tr>
<td>MOC</td>
<td>My Own Creations</td>
</tr>
<tr>
<td>CNN</td>
<td>Convoluted Neural Network</td>
</tr>
<tr>
<td>YOLO</td>
<td>You Only Look Once</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller Unit</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer-Aided Design</td>
</tr>
<tr>
<td>LDD</td>
<td>Lego Digital Designer</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

Lego has been one of the most popular toys since 1958 [1]. The Lego Group has expanded the Lego series from bricks to robotics, which has been used in ‘Science, Technology, Engineering, and Mathematics’ (STEM) education and competitions in the recent decades [2] [3]. With the huge variety of elements that Lego offers, storing Lego parts becomes a complex problem, especially for enthusiasts who would create their own models which are known as ‘My Own Creations’ (MOCs) and share them with others or even sell building instructions online [4] [5].

Lego sorting is to classify Lego elements and to store them in groups. Many Lego enthusiasts store their Lego by element type due to their sizeable collection. It is crucial to keep their storage tidy, such that parts can be easily located [6] [7]. However, it is a tedious and time-consuming task to sort Lego manually.

1.2 Motivation

Lego builders are looking for ways to organize Lego efficiently. Many efforts were made to automate Lego sorting. However, most solutions are impractical due to their size and inefficiency, and requirement of preliminary manual sorting [8] [9]. The existing solutions do not solve the problem efficiently as they are not fully automated. They will be discussed in further details in Section 2.

Many research studies regarding object recognition have been done over the last fifty years [10]. Major approaches to realise object recognition include the use of image segmentation and
artificial intelligence. Currently, there are researches done on vision-assisted robots for sorting small objects such as bolts and nuts [11]. However, limitations arise from the dependency on image processing and segmentation, such as recognizing similar-looking objects like Lego pieces.

Effective organization of pieces stimulates creativity and encourages open-ended designs, as indicated by survey results [12]. Lego builders could further utilize the enhanced productivity if the sorting procedure is automated, as time and human labour saved could be spent on Lego building.

1.3 Objectives

LegoARM is proposed to explore the applicability on sorting Lego pieces using an approach of integrating modern technologies, specifically applying object recognition and machine learning. The goal can be subdivided into three components, implementation of a robotic arm, building a classification model to identify Lego pieces, and performing camera calibration on the robotic arm. The project will be divided into three stages, where the components will be realised in order sequentially.
The project aims to deliver 2 advancements. First, it explores the applicability to sort Lego pieces automatically using modern technologies and robotics. Second, the project will be open-source and made available to the public upon completion. Components in LegoARM are therefore modularized, where they can be substituted with their counterparts should future extensions be made after the project is concluded, or reused in completely new projects.

1.4 Scope

Lego pieces come in a great variety of categories. Broadly speaking, pieces fall into two major systems, the Brick System and Technic System. There are over 200 categories on BrickLink [13], and each category contains up to thousands of pieces. Thus, it is not feasible to cover all pieces produced by Lego in this project.

This project will focus on sorting pins, axles and connector pieces from the Technic System available in the Lego Mindstorms Education EV3 Core Set [14]. The Technic System is widely used in STEM education and competition, such as Lego Education After School Programs and FIRST Lego League competitions [15] [16]. It is used extensively and favoured over the Brick System in STEM education and robotics, and a very common Technic set owned by many is the Lego Mindstorms Education EV3 Core Set. Since a sorting automation solution will be likely to be adopted by institutes and centers offering STEM education due to their scale, the Technic System is chosen over Brick System for this project. The selected Lego categories are comprised of smaller pieces due to the tediousness of sorting tiny pieces by hand and the limitation of the size of the robotic arm.
1.5 Outline of the Report

The remainder of this paper is structured as follows. First, works related to this project will be reviewed. After that, descriptions of the methodology will be presented, and followed by the experiments performed and the current findings and results. Then, limitations will be brought forward to assess the capabilities of LegoARM and potential risks in this project. Finally, this paper will be concluded with a discussion on future plans of the project.
2 Related Works

In this section, related works are evaluated regarding their applicability and relevance to automate Lego sorting. Automatic sorters will be discussed in prior to various object recognition and classification methods.

2.1 Automatic Sorters

2.1.1 Automatic Lego Liftarm Sorter LS-40 [8]

This is a Technic Liftarm sorting machine made entirely out of Lego, a variation of its previous generation which sorts Technic Axles [9]. Most Lego machines fall short of the ability to sort pieces across categories since pieces are usually identified according to a single property, usually length or colour, and this in particular sorts Technic Liftarm pieces by filtering their lengths. Lego machines are mostly created by enthusiasts for interest rather than practicality, with purposes to demonstrate a proof of idea and various building techniques, so they are often massive and often lack integration of appropriate modern technologies which would greatly enhance their capabilities. LegoARM will focus more on applying modern technologies and creating a practical solution for sorting automation.

2.1.2 Vision-assisted Bolts and Nuts Sorting Robotic Arm [11]

This research focuses on applying computer vision to detect, identify and physically locate objects in conjunction with a robotic arm. Modern technologies are proven to be applicable on sorting automation, as demonstrated by the research. While this research shares a similar backbone with LegoARM, the recognition model which is the core of the project is entirely
different. LegoARM focuses on identifying smaller objects, specifically Lego pieces using machine learning.

2.2 Object Detection

Recent years, there are many research studies and new algorithms on object detection. The aim of object detection is to classify the objects in an image and find the corresponding location of the objects. The performances of different algorithms of convoluted neural networks (CNN) are evaluated in terms of accuracy and speed.

2.2.1 Faster R-CNN

Faster R-CNN is an object detection algorithm based on region proposal method [17]. The network only focuses on the extracted region proposal instead of the whole image. Faster R-CNN is faster than Fast R-CNN and R-CNN due to its Region Proposal Network (RPN) which allows region proposals to be extracted in a very low cost and hence able to achieve a faster training rate in a shorter testing time. The accuracy of Faster R-CNN depends on the backbone classification model, such as ResNet, Inception and so on. As a result, a customized self-build model can also apply this algorithm to collect the location information of the detected object.

2.2.2 You Only Look Once (YOLOv3)

YOLO is different from the region based algorithm used by R-CNN, Fast R-CNN and Faster R-CNN. It uses Bounding Box Prediction [18], which first divides an image into bounding boxes and computes a class probability and offset value for each bounding box. A bounding
box with class probability higher than a predefined threshold will be selected and the location of the object in an image can be found. YOLOv3 uses Darknet-53 as its backbone for feature extraction and it has a better performance than the previous version, Darknet-19. Detection using YOLO can be done in a higher frame per second (fps) than in Faster R-CNN and other region-based detectors. However, it has less accuracy advantage on detecting small objects due to algorithm limitation [19].

2.3 Object Classification on synthesized images

One of the most crucial parts of training a machine learning model is data collection, but it is inflexible to obtain a large amount of data. There are many research studies to explore the possibility of using synthesized data for training, especially for image classification problem.

Researches done by Peng et al [20] and Sarkar et al [21] have shown the effect of training a model with different synthesized images generated from 3D models. They tested the effect of applying backgrounds, textures, number of cameras and colour to the synthesized images in training a model. Their work gave inspiration to perform data collection by rendering images with 3D models.
3 Methodology

This section describes how LegoARM will be realized. It begins with the prerequisites of hardware and software (section 3.1), followed by elaboration of the three components in the following order, the robotic arm (section 3.2), the recognition model (section 3.3), and the camera calibration methods (section 3.4).

3.1 Prerequisites

3.1.1 Hardware

A HuaDuino single-board microcontroller unit (MCU) will be used to control the Robotic Arm. Computation of inverse kinematics of articulated joints will be done on board, where movement commands will be accepted through a communication interface.

A Logitech webcam will be paired with a MacBook. The built-in webcam available on MacBooks is not used since a separate webcam enables better positioning of the camera to a better viewing angle.

3.1.2 Software

Python3 is chosen as the main programming language. It is a suitable language considering extensibility of the project due to the libraries available in Python and scalability of the project as it is relatively easy to Python-enabled machines to perform computation.
3.2 Robotic Arm

The first stage is implementation of the robotic arm. The robotic arm by ftobler is selected from Thingiverse for this project [22]. It is a suitable candidate since all necessary parts can be 3D printed, and that 3 degrees of freedom is sufficient for sorting Lego pieces. The hardware configurability is discussed in section 3.2.1.

![Figure 1 Detail drawing of the robotic arm design by ftobler](image)

Figure 2 describes the dependency relationships of software components of the robotic arm. The software is divided into 3 parts regarding the control of motors on the HuaDuino, the communication between a Python library and the HuaDuino MCU, and the Python interface exposed to control the robotic arm. The details are elaborated in section 3.2.2 through 3.2.4.

![Figure 2 Dependency relationships of software components of the robotic arm](image)
3.2.1 Robotic Arm Configurability

The robotic arm is adapted to fit the requirements of this project. The open-source design files allows the head module to be customized specifically to pick and place Lego pieces. Multiple head modules are 3D printed by the HKU MakerLab and tested to select the optimal hardware configuration. The details of the head module designs are discussed in section 4.1.

3.2.2 Controlling Articulated Joints

The robotic arm joints are powered by stepper motors connected to a HuaDuino MCU. Since HuaDuino is compatible with Arduino family MCUs, the open-source C++ library encapsulating the reverse kinematics of articulated joint positions and stepper motor controller calls from ftobler can be used [22]. The exposed serial port interface is used by the host machine to control the robotic arm.

3.2.3 Encapsulation of Robotic Arm Control in a Python Library

The communication between the host machine and the HuaDuino MCU are encapsulated in a Python library. Low-level serial port data exchanging are implemented with PySerial where G-Code command strings are packed in little-endian byte order in utf-8 encoding. This exposes the communication interface as Python functions, allowing the robotic arm to be controlled by Python code.
3.2.4 The Robotic Arm Software Interface

An Application Programming Interface (API) is developed to provide an abstraction layer for the robotics components, where all method calls from the other components will be channelled through. By defining an abstract interface, all dependencies of the robotic arm can be encapsulated in one single component, exposing only the interface to move the head module and pick/release Lego pieces.

The use of an abstraction interface enables the component to be interchangeable and adds extensibility to the project. Alterations to the robotic component will not affect any other part of the project. Robotic components to pick and place Lego pieces can be interchanged with the robotic arm model used by providing an implementation of this interface.

3.3 Machine Learning Model

This next section elaborates the data collection method and the supervised learning algorithm.

3.3.1 Data Collection Method

The amount of data required is substantial due to the use of machine learning and the size of the collection of Lego pieces. An intuitive approach to collect data is to take photographs of a few Lego pieces. However, this limits the extensibility of including any new Lego pieces. It is infeasible to manually collect data whenever the classification model is expanded to identify new pieces. To construct an extensible pipeline on training the classification model, this project borrows the method of using synthetic dataset in training [21]. This greatly reduced the amount of data required to be collected manually.
3.3.2 Synthesizing Data

3D models were taken to render images of Lego pieces. Lego 3D model files are available from two major sources, the database of the Lego computer-aided design (CAD) software from The Lego Company, Lego Digital Designer, and an open-source alternative, the LDraw library [23]. Both software are free to use. The same Lego piece from the two sources has different geometry data and file types. Images of Lego pieces can be produced by importing the model files in appropriate rendering engines, where there are several to choose from. The experiments of rendering will be discussed in section 4.2.

3.3.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a deep-learning architecture and it is used in the project for object recognition. It is a feedforward neural network with convolutional layers, and it has been commonly applied in object recognition [24] [25].

The library that is used to build the CNN is TensorFlow, which is a Google library for deep learning. It has been used in more than 6000 online open source repositories since it was open sourced in 2015 [26] [27]. It is popular mainly due to its support on multiple deep learning frameworks, flexible interface, and community resources [28]. Keras library is used on top of TensorFlow to simplify the programming. It also supports CNN and is able to provide numerous common pretrained network as frameworks. Therefore, Keras library is used together with TensorFlow for the model training process.
3.3.4 Data Augmentation

In general, a better accuracy of the model can be achieved with larger amount of data. To expand the dataset for training and testing, data augmentation is applied. The `ImageDataGenerator` class in Keras Library is used to augment data by applying random transformation to an image when the images are inputted to the neural network. As a result, the parameters determining the quantity of data input, specifically the data batch size and step number, would affect the resulting quantity of augmented training samples used. For instance, when the batch size is 20 and steps number is 50, the number of image fit into the model will be 1000.

3.3.5 Evaluation Metrics

The focus of the project is to correctly identify Lego pieces. Therefore the final evaluation of the performance is the prediction accuracy. However, the training model cannot be evaluated solely based on the final prediction accuracy. The validation accuracy and validation loss during the training progress must be taken into account to identify the parts of the model which requires improvement, and determine whether underfitting or overfitting has occurred.

3.3.6 Customized Neural Network

A customized neural network allows better flexibility on adding or removing layers. It is used to test the effect of applying different hyperparameter settings, regularization and the effect of different types of layers. The details are reviewed in section 4.3.
3.3.7 Applying Feature Extractor

To improve the performance of the neural network, pre-trained models are used as feature extractors. Figure 3 illustrates the setup of using the feature bottleneck of a pre-trained model to extract a feature map from input data. The pre-trained models used are VGG16 and ResNet-50. Both models were notable in the localization task in ILSVRC in different years [29]. The pre-trained models and weights from ImageNet are obtained and loaded from the Keras library [30]. The classifier layers of the original models are replaced with a fully connected ReLU layer, a dropout layer and a softmax output layer.

![Feature Extractor Diagram](image)

*Figure 3 A conceptual illustration of how feature extractor works [31]*

3.3.8 Object Recognition

Faster R-CNN will be selected to implement object recognition. Its ability to detect small objects allows it to be a better option for object recognition over YOLO. Faster R-CNN might not work as well as YOLO in real time detection, but the detection speed of the Lego pieces is less important in this project in comparison to accuracy.
3.4 Camera Calibration

The robotic arm and the camera feed have separate coordinate systems, and calibration is required to perform actions which utilize information from both components. The webcam will be placed in a fixed alleviated position to obtain a better viewing angle. A calibration grid will be placed in a known fixed location relative to the robotic arm, such that the image coordinates can be translated back to world coordinates on the ground plane.
4 Results and Findings

The progress on the robotic arm and machine learning model will be presented in this section. A pipeline is set up to explore the relationship between achieving better prediction accuracy and adjustments tuning on the synthetic dataset and training model. A data generator is built to produce a synthetic dataset, which will be used in machine learning. The data generation and training model will be adjusted according to the output model, with the aim to optimize prediction accuracy.

4.1 Robotic Arm Head Module Designs

Figure 4 displays three head modules that were tested for picking and placing Lego pieces. A 3-jaw angled gripper, a suction cup and a parallel gripper were tested by gripping Lego pieces of different sizes and shapes.

The parallel gripper was able to consistently grab and release Lego pieces, possibly due to the larger contact surface area. The 3-jaw angled gripper fails to grip in the majority of the test
rounds, while Lego pieces tend to be stuck on the suction cup due to their lightweight. The parallel gripper is therefore best suited for the use case of picking and placing Lego pieces.

4.2 Rendering Lego 3D Model Files

The experiment on automating data generation is described in this subsection to find out the optimal workflow to synthesize data in large batches. The selection of 3D Lego model files, choosing the appropriate rendering engine, and automating the rendering process are elaborated in order. The results are discussed in section 5.1.

4.2.1 Selecting 3D Model Files

3D model files describe the Lego pieces’ geometry, where the resemblance to its real-life counterpart depends on the accuracy of geometric data and the polygon count. The two primary sources for 3D model files of Lego pieces are the Lego Digital Designer (LDD) database, and the LDraw library. The former is provided by The Lego Company in 2004, and the latter is an open-source project created back in 1995 [23]. Files from the two sources are rendered in different rendering engines, which will be discussed in section 4.2.2.

While the two sources both provide all common Lego parts, the pipeline for generating images are different. Figure 5 lists the pipeline for generating images using POV-Ray, a common engine which accepts file types from both sources.
Rendered images of Lego pieces from two sources are produced to find out which source is better suited to generate the test dataset for machine learning. Models are taken from both sources and rendered in POV-Ray for comparison. Upon close inspection, the geometry of models from LDD has a higher resemblance to its real-life counterpart, and a higher polygon count which means it describes the 3D model in finer details. Therefore, LDD database is chosen as a viable candidate of model files.

### 4.4.2 Choice of Rendering Engines

The quality of rendered images is significantly dependent on the rendering engine used. Different rendering engines are available for the two Lego file sources.

The major photorealistic rendering engines available for LDD file types are POV-Ray in conjunction with LDD to POV-Ray converter, BlueRender, and Blender. The engines available for LDraw file types are POV-Ray, and Blender. Figure 6 shows rendered images produced by importing geometry data from model files from LDD to test out the rendering qualities. The images in left to right order are produced by POV-Ray, BlueRender, and Blender respectively.
The primary selection criteria comes down to the adjustability of the rendering engine as all engines are able to produce images in similar quality. The scene setup in Blender has the highest degree of freedom to fine-tune rendering parameters such as shaders, reflectance and lighting; However, there is a steep learning curve before these can be tuned to produce desirable results, which significantly increases the complexity to automate rendering. POV-Ray and BlueRender are feasible candidates despite the lesser options for tuning.

4.2.3 Extracting 3D Models

LDD and LDraw comes with little documentation. To use the 3D model files from the libraries, two approaches can be taken. The first one is to reverse engineer the LDD database or the LDraw library primitives and directly extract geometry data, which would involve a notable amount of effort. The second approach is to reverse engineer the CAD file types which are normally imported into the rendering engines and use the geometry data extraction methods in existing rendering programs.

The second approach is more feasible since the CAD file types are text files or zipped text files referencing the Lego parts with corresponding rotations and transformations [32], as shown in Figure 7. LDraw accepts .ldr file types while LDD accepts both .ldr and .lxf file types. By
experimenting and examining the correlation of changing different values in the text files, the reference to the rigid body transformation of Lego pieces can be found.

Results from extensive testing has shown that for a rigid body motion transformation of a Lego piece where $R$ is the rotation matrix and $T$ is the translation vector:

$$
\begin{bmatrix}
R_{00} & R_{01} & R_{02} & t_x \\
R_{10} & R_{11} & R_{12} & t_y \\
R_{20} & R_{21} & R_{22} & t_z
\end{bmatrix}
$$

is flattened to the text string $r_{00}, r_{01}, r_{02}, r_{10}, r_{11}, r_{12}, r_{20}, r_{21}, r_{22}, t_x, t_y, t_z$ in .lxf files, the primary file type used by LDD and LDD rendering engines. A script is created to generate .lxf files containing different Lego pieces, and integrated into the data generator described in section 4.2.5.
4.2.4 Automation with Modified Rendering Engines

The most difficult part is automating the rendering process. Most rendering tools available are intended for the general public to use and requires interaction with a graphical user interface (GUI), which is unfavourable in automating rendering. Experiments on POV-Ray and BlueRender are conducted to test if rendering jobs can be automated without interaction with the GUI, by loading 3D model files into the rendering engines.

The use of POV-Ray to render .lxf files requires a bridging software, LDD to POV-Ray converter. POV-Ray rendering can be executed with command line and injected with rendering parameters using .ini files, which eliminates the requirement of interaction with GUI. However, the use of virtual file system in LDD to POV-Ray converter to access the LDD database requires the rendering to be executed via the converter’s GUI [33]. Automation of rendering batch jobs is therefore unmanageable using POV-Ray.

On the other hand, BlueRender, as a JavaFX Application, can be decompiled and injected with custom code to eliminate the use of GUI on rendering. The injection point is at the life-cycle initialization, where the code to parse arguments, execute rendering and auto-closing can be packed into the JavaFX Application start() method, as illustrated in Figure 8. The modified rendering engine enables batched rendering jobs and is integrated into the data generator in section 4.2.5.
4.2.5 Data Generation with Scripted Automatic Rendering

The goal is to create a program to generate 3D model files and scene setups, then render images in batch jobs automatically. Limitations of the data generator are presented in section 5.1.

The data generator utilizes the model file generator in section 4.2.3 and the modified rendering engine in section 4.2.4. Model files describing different camera setups and Lego piece placements are generated by computing the rigid body transformations over a range of valid angles, then loaded into the rendering engine via shell commands repeatedly. By preconfiguring a rendering parameter list, image can be rendered in batch jobs, as illustrated in Figure 9.
4.3 Machine Learning Model Training

The following subsection describes the experiments done on training the model to explore the effect of applying different techniques and how they may be configured to increase the accuracy to classify unseen Lego images. This included applying regularization, feature extraction and using some pretrained models.

The 3 Lego elements selected for experimenting with the machine learning model were Lego elements number 4514553, 4142865 and 4121715. Samples of real photographs, synthesized images, and the samples produced using data augmentation are displayed in Table 1 below.
<table>
<thead>
<tr>
<th></th>
<th>4514553 (connector peg W. Friction 3M)</th>
<th>4142865 (2M cross Axle W. Groove)</th>
<th>4121715 (connector peg W. Friction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real photographs</td>
<td><img src="image1" alt="Real photographs" /></td>
<td><img src="image2" alt="Real photographs" /></td>
<td><img src="image3" alt="Real photographs" /></td>
</tr>
<tr>
<td>Real photographs (augmented)</td>
<td><img src="image4" alt="Real photographs (augmented)" /></td>
<td><img src="image5" alt="Real photographs (augmented)" /></td>
<td><img src="image6" alt="Real photographs (augmented)" /></td>
</tr>
<tr>
<td>Synthesized images</td>
<td><img src="image7" alt="Synthesized images" /></td>
<td><img src="image8" alt="Synthesized images" /></td>
<td><img src="image9" alt="Synthesized images" /></td>
</tr>
<tr>
<td>Synthesized images (augmented)</td>
<td><img src="image10" alt="Synthesized images (augmented)" /></td>
<td><img src="image11" alt="Synthesized images (augmented)" /></td>
<td><img src="image12" alt="Synthesized images (augmented)" /></td>
</tr>
</tbody>
</table>

Table 1 Sample images of the 3 Lego used in the experiment

Two datasets were collected for experimentation, a real photograph dataset and a synthetic image dataset. The former contains 50 samples per class, and the latter contains 325 samples per class. The datasets were randomly augmented by flipping, shifting and rotating during runtime, which significantly increases the dataset size, at least by a factor of 3.
Table 2 illustrates the basic structure of the customized neural network. It consists of 2-4 sets of convolutional layers with ReLU as activation function and a max-pooling layer, a flatten layer, dropout layer, dense layer and a softmax output layer. This network was modified based on existing models for some simple classification problems [34] [35]

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Activation function</th>
<th>No. parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv2d (Conv2D)</td>
<td>(None, 298, 298, 32)</td>
<td>ReLU</td>
<td>896</td>
</tr>
<tr>
<td>max_pooling2d (MaxPooling2D)</td>
<td>(None, 149, 149, 32)</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_1 (Conv2D)</td>
<td>(None, 147, 147, 64)</td>
<td>ReLU</td>
<td>18496</td>
</tr>
<tr>
<td>max_pooling2d_1 (MaxPooling2D)</td>
<td>(None, 73, 73, 64)</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_2 (Conv2D)</td>
<td>(None, 71, 71, 128)</td>
<td>ReLU</td>
<td>73856</td>
</tr>
<tr>
<td>max_pooling2d_2 (MaxPooling2D)</td>
<td>(None, 35, 35, 128)</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>conv2d_3 (Conv2D)</td>
<td>(None, 33, 33, 64)</td>
<td>ReLU</td>
<td>73792</td>
</tr>
<tr>
<td>max_pooling2d_3 (MaxPooling2D)</td>
<td>(None, 16, 16, 64)</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>flatten (Flatten)</td>
<td>(None, 16384)</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>dropout (Dropout)</td>
<td>(None, 16384)</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>dense (Dense)</td>
<td>(None, 64)</td>
<td>ReLU</td>
<td>1048640</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 3)</td>
<td>Softmax</td>
<td>195</td>
</tr>
</tbody>
</table>

Table 2 The main setup of the customized neural network used in the experiment
4.3.1 Comparison of Output Model Trained using Real Photographs and Synthesized Images

Experiments were done to examine the difference in training result between real photographs of the Lego pieces and the synthesized images of those Lego pieces. Figure 10c and 10d plot the training process of using synthesized image of the Lego pieces. Since the metrics did not fluctuate much after the 30th epoch, the training process was terminated at the 50th epoch. On the other hand, the accuracies and losses during the training process of using real photographs was still fluctuating at the 50th epoch, so the training was extended to 100 epochs. Both experiments were conducted under the same setup. The accuracies in training with synthesized images increased quicker than that with real photographs. This could be explained by the exclusion of unnecessary information in synthesized images, in comparison to real photographs.

![Figure 10a](image.png) The training and validation accuracy for real photographs.

**Blue**: Training accuracy  
**Green**: Validation accuracy

![Figure 10b](image.png) The training and validation loss for real photographs.

**Blue**: Training accuracy  
**Green**: Validation accuracy
4.3.2 Comparison between Before and After Applying Regularization

During the training process, overfitting was suspected and hence regularization techniques like applying batch normalization layers and dropout layer were applied to the customized neural network. The graphs in Figure 11 show the effect of applying regularization.

Figure 11c shows the training and validation accuracy before applying regularization. The fluctuations may indicate a possibility of overfitting due to insufficient data samples used. The data size was 315 per class, where 20% of samples were used as validation data. In order to determine whether the model was overfitted and mitigate the issue, regularization methods were applied. A batch normalization layer was applied after each convolution layer, and a dropout rate of 0.2 was applied to the network. The results are shown in Figure 11a and Figure 11b. The validation accuracies and losses during the training fluctuated at a smaller range after regularization was applied.
Further experiments were conducted to examine the effect of applying regularization when the quantity of training data increases. When the dataset size was augmented to 2835 images, the validation accuracies and losses after applying regularization display little fluctuations, as shown in Figure 12c and 12d. The results are similar to those before applying regularization, as shown in Figure 12a and 12b. This evidence suggests that for sufficiently large training
datasets, regularization may be unnecessary. Additionally, this also suggests that the model in Figure 11c and 11d was indeed overfitted.

Figure 12a The training and validation accuracy for the network that applied regularization with 2835 images.

(Blue): Training accuracy  
(Green): Validation accuracy

Figure 12b The training and validation loss for the network that applied regularization with 2835 images.

(Blue): Training accuracy  
(Green): Validation accuracy

Figure 12c The training and validation accuracy for the network without regularization for 2835 images.

(Blue): Training accuracy  
(Green): Validation accuracy

Figure 12d The training and validation loss for the network without regularization for 2835 images.

(Blue): Training accuracy  
(Green): Validation accuracy

Figure 12 Comparison of training metrics before and after applying regularization to avoid overfitting, using an augmented dataset with size of 2835.
4.3.3 Comparison between Fine-tuned Pre-trained Models

In addition to using a customized neural network, the applicability of pre-trained models were also tested, specifically VGG-16 and ResNet-50. The models were imported from Keras library and the weights pre-trained on ImageNet was applied. Experiments were conducted by retraining only partial layers of the model, with all other layers kept constant. If both models were used directly without being fine-tuned, the results are unsurprisingly no better than random guessing, as shown in Figure 13a and 13b, since the training data were not fit into the model. If only the last 4 layers of the model were fine-tuned, the training data was fit into the model, which was reflected by the rise of training accuracy in Figure 14a and 14b. However, validation accuracies of the models show either wild oscillations (Figure 14a) or no fluctuations (Figure 14b). These suggest the models may not be suitable to differentiate the selected Lego pieces and may require more testing to determine which layers to be fine-tuned to fit the data better. Furthermore, the difference between the results in using VGG-16 and ResNet-50 has suggested that VGG-16 may be a model that could fit Lego images as inputs better.

<table>
<thead>
<tr>
<th>Figure 13a Using VGG-16 with all layer untrainable.</th>
<th>Figure 13b Using ResNet-50 with all layer untrainable.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Training and validation accuracy" /></td>
<td><img src="image" alt="Training and validation accuracy" /></td>
</tr>
<tr>
<td>(Blue): Training accuracy</td>
<td>(Blue): Training accuracy</td>
</tr>
<tr>
<td>(Green): Validation accuracy</td>
<td>(Green): Validation accuracy</td>
</tr>
</tbody>
</table>

Figure 13 Comparison of training metrics for untrainable VGG-16 and ResNet-50
4.3.4 Comparison between Before and After Applying Feature Extractor

Further experiments were conducted by using VGG-16 and ResNet-50 as feature extractors. For the same data amount, using ResNet-50 feature extractor (Figure 15a and 15b) yields lower and more fluctuating validation accuracies in comparison to VGG-16 (Figure 16a and 16b). Furthermore, the loss in ResNet-50 feature extractor (Figure 15b) was higher than that in VGG-16 (Figure 16b). This indicates that VGG-16 is potentially more suitable to extract features for Lego images as input data. The validation accuracies of using ResNet-50 showed a rising trend and it could possibly be increased further for a sufficiently large epoch. However, this remains to be verified.
4.3.5 Comparison between gray-scale and coloured images in object recognition

To examine whether colour contributes to the differentiability of the model, experiments were conducted on grayscale images in a duplicate setup. The validation accuracies of the training
process that was using grayscale images (Figure 17a and 17b) are lower than using color images (Figure 12c and 12d). This evidence suggests that the colour information of the images was taken into account by the neural network during training.

4.3.6 Summary of the experiment results

In most cases the validation accuracies were irrationally high, ranging from 90% to 100%, while the actual prediction ability, conversely, is rather poor. Among all the trials, the testing accuracies ranges from 33% to 34% and in the majority of the experiments, and the confidence of wrong predictions were fairly high. Figures 18a, 18b and 18c depict examples of confident wrong predictions during the trials on customized neural network, which are common situations throughout all experiments.
Various approaches were taken with the aim to increase the testing accuracy. However, the behaviour could not be completely explained with the current experiment results, which would require further investigation through other directions. A possible factor contributing to the low accuracy, for instance, could be the choice of Lego pieces being too difficult to be differentiated by the neural network. This will be further discussed in section 5.
5 Limitations and Discussion

5.1 Synthesizing Data

The method for synthesizing data despite able to produce a usable dataset for machine learning, has a number of limitations. The data generator will be updated to mitigate the issues and to include more functionalities required by the training model.

5.1.1 Rendering Automation Bottleneck

Multiple instances of BlueRender concurrently executing may produce a system deadlock. The BlueRender setting file may lock up with multiple file read requests opened by multiple instances of the application. This limits the number of image rendering to be one at a time.

The current solution is to divide the rendering parameter list into multiple batches and run the data generator on multiple computers. Owing to recent progress on machine learning model training, the amount of data required may be increased. A proposed solution to increase rendering efficiency is to reduce the overhead on the application startup and closing, which opens and closes the connection to the LDD database for every rendering job, by utilizing the camera rotation function built into BlueRender. An alternative solution is to adapt the data generator to render images using POV-Ray and LDraw library in parallel with BlueRender and LDD database. Depending on the requirement updates of the dataset, different solutions will be adopted.
5.1.2 Rendering Scene Configurability

The current dataset was rendered with one preconfigured scene in BlueRender, due to insufficient time on development on the data generator and lack of documentation. The next batch of data will be rendered in different scene setups, which requires an update to the data generator. Since the pipeline for machine learning and dataset tuning was properly configured, more time can be put into tuning the data generator. A rough documentation on configuring the BlueRender scene file is produced in recent progress. The data generator will be updated after the submission of this report.

5.2 Machine Learning Model Training

5.2.1 Choice of data samples

The choice of classifiable targets potentially impact the result. As mentioned in section 4, the actual prediction on unseen data was about 33%. This may be related to the choice of Lego pieces used in the dataset and an overestimation on the capabilities of machine learning. To unbiasedly determine the effect of the model, another set of Lego pieces with more differentiable characteristics will be used for the upcoming experiments.

5.2.2 Number of training iterations

Due to extensive amount of experiments and time constraints, the training process was terminated when little changes over the validation accuracy and loss was observed. The possibility of substantial changes over a longer training process is yet to be confirmed. However, it would be infeasible to run at infinite iterations to seek for sudden changes.
5.2.3 Limited evaluation method

Evaluating the model by relying on training accuracy and loss, validation accuracy and loss, in addition to the final prediction rate may be insufficient to provide adequate insights to improve the model. Other evaluation methods, such as k-fold cross evaluation, could be required.
6 Future Work

Building a better neural network would be the primary focus in the remainder of this project. A new set of Lego pieces with more differentiable features will be selected for experimentation to correctly pinpoint the reasons for the unexpected results in section 4.2.5.

In the case where a better accuracy can be achieved by making improvements over the previous model, object detection will be integrated to obtain location information of identified objects. However, if the result of the trained model could not be improved, the focus of this project will be adjusted to experimenting the effects of training model with synthesized images with different settings. Table 3 presents an overview of the project schedule.

<table>
<thead>
<tr>
<th>Date</th>
<th>Stage</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 March, 2019</td>
<td>3</td>
<td>Algorithm Integration and Implementation</td>
</tr>
<tr>
<td>14 April, 2019</td>
<td></td>
<td>Deliverables:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Finalised tested implementation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Final report</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Project Poster</td>
</tr>
<tr>
<td>15-10 April, 2019</td>
<td></td>
<td>Final presentation</td>
</tr>
<tr>
<td>29 April, 2019</td>
<td></td>
<td>Project exhibition</td>
</tr>
</tbody>
</table>

Table 3 Project Schedule
7 Conclusion

LegoARM aims to explore the applicability of machine learning and object detection on Lego identification. The current results indicate that achieving the initial objectives with machine learning may require more work. However, the directions for making improvements are clear for future exploration.

Throughout the first semester, effort was placed primarily on acquiring knowledge from scratch regarding machine learning and data synthesis. The experience gained during the process will in turn be practical during further exploration and experimentation on the subject. In the ideal situation, the unresolved issues will be settled in the second semester, and insights will be provided on achieving a better prediction accuracy on trained models.
8 Reference


