2019

Learning to grasp in unstructured environments with deep convolutional neural networks using a Baxter Research Robot

Shehan Caldera
Edith Cowan University

Edith Cowan University

Copyright Warning

Learning to Grasp in Unstructured Environments with Deep Convolutional Neural Networks using a Baxter Research Robot

A thesis submitted for the degree of

Master of Engineering Science

Shehan Caldera

A/ Prof. Alex Rassau, Principal Supervisor

Dr. Douglas Chai, Associate Supervisor

School of Engineering

Edith Cowan University

2019

This thesis is dedicated
to

My Parents
Sunil and **Suba**

For making me who I am and
teaching me to believe in hard work

My Fiancée
Anne
For providing unparalleled support
and encouragement

Acknowledgements

I would like to acknowledge my supervisory panel, Associate Professor Alexander Rassau and Dr Douglas Chai, for their continuous mentoring, support, and guidance throughout my research journey. I would like to acknowledge the many opportunities bestowed upon me during my research candidature.

Furthermore, I would like to acknowledge the School of Engineering at Edith Cowan University for providing me with a scholarship to pursue my research interests in a Masters by Research program. This gratitude is extended to all of the staff of the school including the Executive Dean, Professor Daryoush Habibi; Associate Dean (Research), A/ Prof. Mehdi Khiadani; and all of the academic and professional staff who were involved with my research work with guidance, assistance, and encouragement. Then, I would like to acknowledge my colleagues who shared their research interests with the topic.

Then, I would like to acknowledge the researchers such as Ian Lenz, Joseph Redmon, and Sulabh Kumra whose work has inspired me to explore the applications of deep learning in robotic grasping.

I would also like to express my most profound appreciation to my parents, who have taught me to believe in hard work, to my brother who has always shared similar interests into emerging technologies. Most importantly, to my fiancée who has always known me and who still encourages me to pursue further when I'm on the verge of giving up.

Abstract

Recent advancements in Deep Learning have accelerated the capabilities of robotic systems in terms of visual perception, object manipulation, automated navigation, and human-robot collaboration. The capability of a robotic system to manipulate objects in unstructured environments is becoming an increasingly necessary skill. Due to the dynamic nature of these environments, traditional methods, that require expert human knowledge, fail to adapt automatically. After reviewing the relevant literature a method was proposed to utilise deep transfer learning techniques to detect object grasps from coloured depth images. A grasp describes how a robotic end-effector can be arranged to securely grasp an object and successfully lift it without slippage. In this study, a ResNet-50 convolutional neural network (CNN) model is trained on the Cornell grasp dataset. The training was completed within 30 hours using a workstation PC with accelerated GPU support via an NVIDIA Titan X. The trained grasp detection model was further evaluated with a Baxter research robot and a Microsoft Kinect-v2 and a successful grasp detection accuracy of 93.91% was achieved on a diverse set of novel objects. Physical grasping trials were conducted on a set of 8 different objects. The overall system achieves an average grasp success rate of 65.0% while performing the grasp detection in under 25 milliseconds. The results analysis concluded that the objects with reasonably straight edges and moderately pronounced heights above the table are easily detected and grasped by the system.

Statement of Originality

I hereby declare that the content of this thesis is the original work of the author, and all of the material is presented in accordance with academic rules and ethical conduct.

I also certify that, to the best of my knowledge, this thesis does not infringe upon anyones copyright nor violate any proprietary rights, and that any ideas, techniques, quotations, and material from other peoples works included in this thesis are fully cited as defined by the university policy on standard referencing practices.

Shehan Caldera

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advancements in robotics and automated systems have led to the expansion of autonomous capabilities and more intelligent machines being utilised in ever more varied applications [1, 2]. The capability of adapting to changing environments is a necessary skill for task generalised robots [3, 4]. Machine learning plays a key role in creating such general-purpose robotic solutions. However, most robots are still developed analytically, based on expert knowledge of the application background. Even though this is considered an effective method, it is an arduous and a time-consuming approach, and has significant limitations for generalised applicability. Due to the recent successful results of deep learning methods in computer vision and robotics applications, many robotics researchers have started exploring the application of deep learning methods in their research.

The type of learning that is applied varies according to the feedback mechanism, the process used for training data generation, and the data formulation. The learning problem can vary from perception to state abstraction, through to decision making [5]. Deep Learning, a branch of machine learning, describes a set of modified machine learning techniques that, when applied to robotic systems, aims to enable robots to autonomously perform tasks that come naturally to humans. Inspired by the biological nervous system, a network of parallel and simultaneous convolutions and other mathematical operations are performed directly on the available data to obtain a set of representational heuristics between the input and output data. These heuristics are then used in decision making. Deep Convolutional Neural Network (DCNN) models have proven effective in diverse classification and detection problems [6, 7, 8] and there is a great deal of interest in expanding their utilisation into other domains.

Recently, deep learning has achieved state-of-the-art results in many visual detection applications such as image classification, autonomous driving, and natural language processing. Therefore many robotics researchers have started exploring the

application of deep learning methods for visual detection. Robotic grasping involves detection of the graspable regions on an image. This is identified as the grasp or grasping pose. Current research suggests that the use of a trained deep learning method would generate better results for visual detectionof graspable regions. Deep learning methods are also known for their ability to generalise. In robotic grasping, the detection of the graspable region or a grasp pose from a given image is a critical step. The grasp or grasping pose describes how a robotic end-effector can be arranged to successfully pick up an object. The grasping pose for any given object is determined through a grasp detection system. Any suitable perception sensors including cameras or depth sensors can be used to visually identify grasping poses in a given scene. Grasp planning relates to the path planning process that is required to securely grab the object and maintain the closed gripper contacts to hold and lift the object from its resting surface [9]. Planning usually involves the mapping of image plane coordinates to the robot world coordinates for the detected grasp candidate. The control system describes certain closed-loop control algorithms that are used to control the robotic joints or degrees of freedom (DOF) to reach the grasping pose while maintaining a smooth reach [10].

This thesis comprises the literature review, methodology, results and discussions associated with an investigation into the use of deep convolutional neural networks to enable a Baxter research robot to learn to grasp in unstructured environments using DCNNs and the robotic grasping with a Baxter research robot. The thesis is organised as follows; Chapter 1 provides the introduction and the background information for the project topic, Chapter 2 provides a thorough review into previous literature of deep learning and robotic grasping, Chapter 3 describes the complete method detailing specific steps, Chapter 4 presents all the results from the various experiments, Chapter 5 provides a discussion and analysis of the results analysis, and Chapter 6 provides conclusions and recommendations for future research directions.

This chapter introduces the project topic and aims to provide some background information on general-purpose industrial robotics as well as identifying a sample application environment with variable constraints. It is followed by a description of the problem of formulating a learning method for robots to generalise tasks. The subsequent sections introduce the objectives of this study, and their significance to the related research community, finally the specific research questions aimed to be addressed through this study are provide.

1.1 Background

In order to control the operations of a robotic system using the analytical approaches or "hard coding" requires manually programming a robot with the necessary instructions to carry out a given task. These control algorithms are modelled based on expert human knowledge of the robot and its environment for that specific task [4]. The outcome of this approach explains a kinematic relationship between the parameters of the robot and its world coordinates. Ju et al. [11] suggest that the kinematic model helps in further optimising the control strategies. However, direct mapping of results from a kinematic model to the robot joint controller is inherently open-loop and is identified to cause task space drifts. Therefore, they [2] have, in addition, recommend the use of closed loop control algorithms to address these drifts.

Even though such hard coded manual teaching is known to achieve efficient task performance, such an approach has limitations; in particular the program is restricted to the situations predicted by the programmer, but in cases where frequent changes of robot programming is required, due to changes in the environment or other factors, this approach becomes impractical [4]. According to Ju et al. [2], unstructured environments remain a large challenge for intelligent robots that would require a complex analytical approach to form the solution. While deriving of models requires a great deal of data and knowledge of the physical parameters relating to the robotic task, use of more dynamic robotic actuators make it nearly impossible to model the physics, thus they conclude that manual teaching is an efficient but exhaustive approach [2]. In such cases, empirical methods will provide an increased cognitive and adaptive capability to the robots, while reducing or completely removing the need to manually model a robotic solution [3]. Early work in empirical methods takes a classical form that explores the adaptive and cognitive capability of robots to learn tasks from demonstration. Non-linear Regression techniques, Gaussian process, Gaussian mixture models, and Support Vector Machines are some of the popular techniques related to this context [12]. Although these techniques have provided some level of cognition for the robots, the task replication is limited to the demonstrated tasks [12].

Deep learning has recently made significant advancements in the application backgrounds of computer vision, scene understanding, robotic arts, and natural language processing [10, 13]. Due to the convincing results that have been achieved in the scope of computer vision, there is an increasing trend towards implementation of deep learning methods in robotics applications. Many recent studies show that the unstructured

nature of a generalised robotics task makes it significantly more challenging. However, in order to advance the state-of-the-art of robotic applications it is necessary to create a generalised robotic solution for various industries such as offshore oil rigs, remote mine sites, manufacturing assembly plants, and packaging systems where the work environments and scenarios can be highly dynamic. A desired primary ability for these general purpose robots is the capability to grasp and manipulate objects in order to interact with their work environment. Even though the visual identification and manipulation of objects is a simple task for humans, it is a challenging task for robots that involves perception, planning, and control [14, 10]. Grasping can enable the robots to manipulate obstacles in the environment or to change the state of the environment if necessary. Early work such as [15, 16] show how far researchers have advanced the research methods in robotic grasping. These studies discuss the early attempts of grasping novel objects using empirical methods.

Object grasping is challenging due to the wide range of factors such as different object shapes and unlimited object poses. Successful robotic grasping systems should be able to overcome this challenge to produce useful results. Unlike robots, humans can almost immediately determine how to grasp a given object. Robotic grasping currently performs well below human object grasping benchmarks, but is being continually improved given the high demand. A robotic grasping implementation has the following sub-systems [10]:

- **Grasp detection system**: To detect grasp poses from images of the objects in their image plane coordinates
- **Grasp planning system**: To map the detected image plane coordinates to the world coordinates
- **Control system**: To control a robotic hardware platform to reach the target grasping pose and perform the object grasping

This is further discussed in the literature review in Chapter 2.

The first part of the project is to develop a detection system for graspable regions from object images. Therefore, a deep learning based grasp pose detection method needs to be developed. The objective for this detection system is to identify graspable regions or object grasp poses from images. A popular deep learning method that has been applied in most grasp detection related research is the Deep Convolutional Neural Network or sometimes referred to as the Convolutional Neural Network (CNN) due to the heavy involvement of convolutional layers in their architectures.

There are two principle approaches to apply a CNN to a problem:

1. Create an application specific CNN model
2. Utilise a complete or part of a pre-existing CNN model through transfer learning

Creating a proprietary CNN model requires a deep understanding of the concepts and a reasonable level of experience with CNNs. Therefore, most researchers that implement CNNs in their grasp detection work have opted for transfer learning given the reduced number of parameters to be dealt with. Training of such a CNN requires a large volume of data [17]. The data can be labelled or unlabelled depending on whether a supervised or unsupervised training method is used. Training is the process of tuning the network parameters according to the training data. The studies [18, 19, 10, 20] focus on simplifying the problem of grasp detection and build on the transfer learning model in order to improve the results. While there are several platforms to implement deep learning algorithms, most studies have used Tensorflow [21], Theano [22], or Matlab [6]. With the recent advancements of software applications and programming languages there are now more streamlined tools such as Keras [23], Caffe [24] or DarkNet [25] to implement the same functionality of former deep learning frameworks but in an easier and more efficient way. Even though most recent deep learning approaches for robotic grasping follow purely supervised learning, software platforms such as NVIDIA ISAAC [26], encourage unsupervised learning methods with the support of virtual simulation capabilities.

Robots are widely employed in industries to improve the efficiency and productivity of emerging markets. Robots are also utilised to create work environments with reduced hazards to humans. As a result the term *Collaborative Robots* has been introduced to describe complex robots that are designed to work alongside humans. They provide support to reduce human labour in a shared work environment and assistance to reduce fatigue from repetitive exhausting work [27]. In this project, the robotic grasping is performed on a Baxter Research robot. The Baxter robot is an advanced humanoid robot with dual 7 degree of freedom (DOF) arms. It is a fully compliant robot that is specially made for human-robot collaborative work environments. It was designed and developed by Rethink Robotics in 2012 [28]. Rethink Robotics [28] has created Baxter with a focus on industrial work safety, easy programming, and general-purpose usage as key points for an industrial robotic solution. The robot has multiple sensors on its joints, arms, and head as shown in Figure 2.11 that can be utilised in various applications. The robot joints are actuated by Series Elastic Actuators (SEA) providing human like interactions between its work environment. The display on its

head is used to output various states and alerts from its operating system. Chapter 2 provides a review of the literature relating to the collaborative Baxter robot and its capabilities. This thesis explores deep learning methods for robotic grasp detection that can be used to enable a Baxter robot to work in an unstructured environment.

The application background for this project is utilisation of robotic systems in environments that can change unpredictably. An example of this type of an environment is a Materials Recovery Facility (MRF). In an MRF slow moving conveyors, optical sorting machines and lifting conveyors work hand in hand with human workers to sort waste materials so that they can be processed and used in new products. The major types of materials recycled are paper, plastic, metal and glass [29]. In current MRFs there are human workers required at several points throughout the process to recognise unwanted items in the conveyors. These workers need to remove these materials manually to prevent any accidental harm they may cause in the sorting machines of the MRF [30]. Automating this part of the process is challenging as the ever-changing environment of an MRF requires task-generalised robots. Therefore modelling an analytically expert robotic solution is nearly impossible due to highly variable constraints. These kinds of challenging unstructured environments exist in many industrial contexts outside of standard production line environments. Even though it may be theoretically possible to overcome some of these challenges with a suitably complex standard programming approach, the cost of such a solution will remain unaffordable to most of the smaller to medium scale industries. In order to overcome such problems data-based model learning algorithms are required [31].

Recently there have been numerous depth cameras available for academic research for a reasonable cost. The Microsoft Kinect sensor is one of the more popular depth cameras that has been used with object grasping research. Most of this research work has used coloured depth images (RGB-D: Red, Green, Blue, and Depth channels) arguing that the added depth information allows the system to more effectively isolate the region belonging to the graspable object. The Microsoft Kinect sensor generates point clouds with RGB images that can be combined to create RGB-D images. Over the last few years, the field of robotic grasp detection with deep learning only just begun to receive more attention, and therefore, the availability of collected data is, unfortunately, very limited. One dataset that is available and that has been widely used for research in this area is the Cornell Grasp dataset, a small-scale grasp dataset with 885 RGB images and point clouds of 240 different objects. Each object image has annotations for valid and invalid graspable regions denoted by oriented grasp rectangles. The dimensions of these rectangles provide the area of the graspable

region on the object image. Overall, there are about 5110 valid ground truth grasps and 2909 invalid ground truth labels.

1.2 Problem Statement

Compared to robots, humans show significantly higher success rates in manipulating objects in a dynamic environment by observing the environment first. Inspired by this approach to give the robots in an industrial environment the ability to learn their tasks autonomously, deep learning methods are identified as a promising way of achieving this goal with the proven results of [32, 33, 31, 34, 18, 19]. A deep learning architecture is identified by its capability to model a desired mapping of inputs to outputs using a large set of non-linear transformations of input data [31].

The training data and the testing data for many machine learning algorithms are obtained from the same data distribution. When the distribution or the feature space changes, most machine learnt statistical models need to be remodelled by collecting data anew. It can be expensive and nearly impossible to collect a fresh data set that would generate similar results. Therefore it would be advantageous to use transfer learning when data collection is not an option [35].

A Deep Neural Network (DNN) or a Deep Convolutional Neural Network (DCNN) is built with multiple layers of information representations [36]. The representations are found by convolutional layers thus identifying a deep learning network as a Convolutional Neural Network (CNN) [17]. Deep learning networks solve problems with high-dimensional observation spaces, but can only handle discrete and low-dimensional action spaces. Many tasks of interest, most notably physical control tasks, have continuous (real valued) and high dimensional action spaces.

On occassion, the deep learning methods are adapted to continuous domains by simply discretising the action space. However, this has many limitations, most notably the curse of dimensionality: the number of actions increases exponentially with the number of degrees of freedom. In such higher dimensional applications it would require new deep learning architectures customised to the problem specification [37].

The input representation is important for a DNN. Given that DNNs identify multiple representations from input raw data [36], the raw data must be formulated in a way such that it is clearly differentiated from one input to another [17]. A grasp rectangle is represented as shown in Figure 1.1 with the grasp parameters: $G = \{x, y, \theta, h, w\}$. The force grasping is removed considering this research only

considers rigid objects that are not easily deformed within the grasp force range of the Baxter robot.

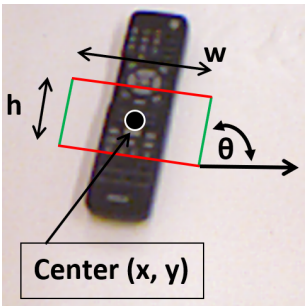


Figure 1.1: A rectangle grasp representation showing the center point (x, y), the height (h), the width(w), and the orientation (θ) with the horizontal axis.

Taking all of this in-to consideration, in this work a method is proposed for implementation of deep learning methods applied to the operation of a Baxter robot for object handling in an unstructured environment. A deep convolutional neural network is trained on the Cornell grasp data [38] to successfully identify grasp rectangles from coloured depth images to use with an inverse kinematic solver to determine the joint angle solution to control the Baxter robotic arm to grasp the objects in order to move them within the workspace.

1.3 Objective

The purpose of this research study is to investigate the use of deep learning to enable a Baxter robot to learn to identify objects and their grasping poses in order to successfully grab the objects between the robotic end-effector and lift them up without slippage. This project explores task generalisation for robotic systems through the utilisation of deep learning.

Most recent changes to the industrial workplace have introduced dynamically varying work environments for robotic systems where structured programming approaches require nearly exhaustive effort to produce effective results. Highly dynamic environments such as off-shore oil rigs or small to medium scale manufacturing plants that have minimal batch sizes of products require continuously adaptable robotic solutions. Deep learning has recently shown the potential to provide this utility.

This study will investigate the application of deep learning methods for detection of grasp poses for objects and will explore robotic grasping with the Baxter robot. Robotic grasping involves the following steps:

1. Identifying and localising the object in its workspace
2. Identifying the object orientation in the workspace
3. Localising the grasping point for the robotic end-effector
4. Controlling the robotic arm to pick-up and move the object to a pre-defined drop-off zone in the workspace.

The first three steps require training of a DCNN from previously collected and annotated grasp data from the Cornell grasp dataset. A suitable DCNN architecture is proposed based on a review of the recent robotic grasp detection work.

1.4 Significance

The Deep Learning community will benefit from the contribution of a deep learning application for robotic motion control. From the literature relating to applications of Baxter robots, it can be assumed that application of deep learning in motion control of a Baxter robot is an area of research that is still under development. There are several methods of applying deep learning in such a task, principally:

- Developing a new deep learning algorithm
- Using a pre-trained deep learning algorithm (Transfer learning [6])

Each method has its own challenges and advantages. To date, most machine learning algorithms have been applied in a variety of computer vision problems. It is only recently that robotic motion control is being explored using the methods of machine learning. As deep learning has the power to allow task generalisation for robots it is a valuable topic to explore in the hopes of finding a lower cost solution to facilitate motion planning for robots in dynamic industrial environments. Once the neural network model is completely trained, the detection model is used to identify grasp poses for the objects that are visible from the images. The located grasp regions are re-mapped to world coordinates. Afterwards, a joint angle solution for the specific Baxter robotic arm is found by solving the inverse kinematic equations. Then, the Baxter robotic arm is actuated to manipulate the detected object.

1.5 Research Questions

This research study has aimed to answer the following questions:

1. What are the suitable Learning methods for object recognition and grasping with a Baxter robot?
2. How to effectively implement Deep Learning methods for grasp detection from images?
   (a) How to best represent the data for training?
   (b) How to select the training algorithm?
   (c) How can the performance of the deep learning model be measured and improved?
3. How to best relate the learnt grasp detection model to the physical robotic grasping?
4. How can the selected robotic grasping method be compared with the current state-of-the-art?

Chapter 2

Literature Review

This chapter provides a thorough review of the relevant literature and focuses on analysing the previous approaches in order to inform the methodology. Section 2.1 discusses different steps associated with a robotic grasping application. Sections 2.2 and 2.3 provide a discussion on various robot learning methods and the importance of data for training such methods. Section 2.4 discusses various neural network architectures and their associated properties relating to grasp pose detection DCNNs. Section 2.5 provides some discussion into physical grasping with a real robot. Finally, Section 2.6 compares various robotic systems with their specifications and collaborative capabilities.

2.1 Robot Learning

This section examines the recent literature relating to robot learning methods. Referring to the literature, three different types of learning approaches have been identified. Traditional methods of robotic control have focused on manually teaching tasks to robots. These robot control methods have been developed based on the expert knowledge of human operators regarding completing such tasks. The next part of the literature describes the incorporation of humans as demonstrators to the robots to learn the tasks. This has taken a task imitation approach in the context of learning. While demonstration based learning has been shown to require a large amount of varying types of data in training, modern approaches of deep learning explore methods of teaching a robot only using perceptual data, which in turn reduces the required amount of data. Another direction for deep learning explores the use of reinforcement to allow the robots to explore and learn the tasks by themselves based on a reward value that is increased or decreased based on whether the action taken is more cor-

rect or more incorrect. The following subsections explore each of these types in more detail.

2.1.1 Manual Programming

The first type of control systems were developed using a model developed by human expertise. This model identified the relationships between the robotic features and the real environment [4]. In motion control of industrial robots, this developed model explained a kinematic relationship between the robotic coordinates and the world coordinates. This model was then used in simulation of the control strategies[2]. Most of the industrial environments were cluttered with obstacles within the workspace of robots being employed. The importance of a model development was recognised in the sense of operating the robots in a manner where there is no harm or damage to the robots or to the work environment. The objective of control design for these industrial robots was accomplished by referring to the previously mentioned model. This process of manual modelling is identified as Manual Teaching.

The models used in programming robots in the position control mode were identified as kinematic models [11]. The Denavit-Hartenberg (DH) notations were widely used to describe the kinematic model of a robot [39]. The robotic programmer's task was to find the method of mapping DH parameters of the robot links used in the mechanism to its environment. However, for the case of the Baxter Research Robot, this parametric information was not provided by the Baxter software wiki information, instead it was provided in a file known as a Unified Robot Description Format (URDF). This is an XML representation of the robot model (kinematics, dynamics, sensors) describing the Baxter robot [40]. Therefore Ju et al. [11] devised a method to find the DH parameters from the URDF and they were able to summarise the key steps to extract the DH parameters.

Ju et al. [2] suggested that unstructured environments remain a big challenge for machine intelligence. Thus, it was generally believed not feasible for a robot to deal with variable surroundings autonomously. Therefore, their view was that semi-autonomous and human-in-loop methods were to be preferred for the majority of applications. The process where an operator controlled a robot remotely, was identified as *Teleoperation* [2]. Ju et al. also presented that the sector of telemedicine was going to be highly interested in teleoperated robots, since they were able to perform various elaborate tasks including remote surgery and teleoperated rehabilitation. Historically, input devices like keyboards and joysticks had been widely used in teleoperating systems. Through the advance of technology, haptic devices had moved

into the focus of research projects during the last decades. The major advantage of these types of devices was to provide the operator with a tactile feeling of the remote environment through force feedback. The forward kinematics model had described the relationship between the joint angles of the serial manipulator and the position and orientation of its end-effector.

As Ju et al. [11] suggested, having the kinematic model would help in design and optimisation of the control strategies. However mapping the joint angles of the master to the slave directly is inherently open-loop and unavoidably causes numerical drifts in the task space. In order to overcome this drawback, the CLIK (Closed Loop Inverse Kinematics) algorithm is applied [2].

Even though the manual teaching method is known to make the learning efficient, such an approach has its own drawbacks; the teaching is limited to the situations predicted by the programmer, but manufacturing plants having smaller batch numbers of products require frequent changes of robot programming [4]. While deriving of models requires a lot of data and knowledge of the physical model relating to the robotic task, use of more dynamic robotic actuators make it nearly impossible to model the physics, thus meaning manual teaching is an efficient but a tedious task requiring lots of data and a lot of physical analysis techniques. Providing increased cognitive and adaptive capabilities to the robots would help in reducing or completely avoiding the need to analyse the data allowing the robots to learn autonomously [3].

2.1.2 Learning from Demonstrations

Another direction identified from the literature has focused more on incorporating humans as demonstrators for robot task learning. This section reviews the literature that explores the adaptive and cognitive capability of robots in autonomously learning demonstrated tasks. This method will reduce or remove the need to find the relationship between provided data for teaching, thus avoiding the heavy calculations and modelling of physical states. The general idea is for robots to learn new tasks autonomously by observing, thus allowing the end-users to teach robots without programming. It is found that statistical analysis for observation data in imitation learning has been done with various non-linear regression techniques extended from machine learning with some of the popular methods being Gaussian Process, Gaussian Mixture Models, and Support Vector Machines [12].

In [41], Calinon et al. identifies two approaches for achieving imitation learning: One approach aims at extracting and encoding low-level features, Low-level features

of a task can be identified as the states of joints on a robotic actuator throughout the demonstration. The second approach aims at higher level features such as complete actions. The paper's authors performed different demonstrations of the same tasks and estimated the relevance using a probabilistic method. This method provides a continuous representation of the constraints, given by a time-dependent co-variance matrix, which can be used to decompose, generalise, and reconstruct gestures. Furthermore, they proved how such a statistical representation of motion can be combined with classical solutions to the inverse kinematics problem, in order to find a controller which optimally satisfied the constraints of the tasks with the use of several experimental setups. They used a kinesthetic approach in demonstrating for the learning. Concluding the discussion they jointly stated how to extract the most important features with spatio-temporal correlations, how to determine a generic metric to evaluate the imitative performance, and how to optimise the robot's reproduction of the demonstrated task after modelling the signals through a probability density function and analysed with non-linear regression techniques of Gaussian Mixture Models (GMM) [41].

Ratliff et al. in [42] presented imitation learning as a large-scale classification problem considering actions as labels for a given state. They further investigated a multi-class function that scored each action, thus finding the optimal action considering the one having the highest score. They have stated that the learning was to be finding of this particular scoring function. During the time period of their study the functional gradient techniques were being used for such predication functions. Therefore it was pointed out that they were interested in adopting binary classification or regression techniques that were developed by their peers in order to find this scoring function. They applied these new-found approaches in robotic grasping and quadruped locomotion experiments. In improving the number of labels, simple brute-force enumeration was used. In furthering their investigation they had presented a set of functions and optimisation. The Structured-Margin Loss function would provide a notion of loss when a wrong action was selected. The Functional Gradient Optimisation provided a cumulative improvement on increasing the score achieved. When it was required to keep the score function always positive or when scores were added together in multiple states the Exponentiated Gradient Variant provided a generalisation. This generalisation increased the number of actions that could be trained for a given state.

Gribovskaya et al. [43] proposed that by encoding demonstrated data statistically with Gaussian Mixtures, non-linear multi-dimensional dynamics of motion could be

learnt. Furthermore they had presented methods to prove the stability of the resultant control algorithm in two steps. First, they had formulated conditions for parameters of GMMs to be satisfied to guarantee local asymptotic stability. Secondly, to verify boundaries of the region of applicability of the control algorithm, they had proposed a numerical method. They had tested these hypothesis using two experiments. The first one learnt theoretical dynamics with known mathematical relationships to check the accuracy of learning. The second one focused on applicability of data, which required some optimisation prior to learning. Interestingly they had used the kinesthetic teaching for demonstrations. They identified this method as being advantageous as it allows the human to gain awareness of the physical limitations of the robot. They had further investigated the extensive requirement of data acquiring and data categorising required for imitation learning. Also optimising of data prior to learning was identified as a major challenge. And robotic control is considered kinematical since the concerned learning is focused on mimicking the trajectory provided in the demonstration [43].

Billard et al. in [44] identified that there were three ways of interfacing a demonstration for imitation learning;

1. Directly recording human motions:
   This is used when the the kinematics of the motion was solely to be imitated. Several types of motion tracking systems or sensor were used. This approach was ideal when explicitly mapping human joints with robot joints.

2. Kinesthetic Learning:
   The robot is physically guided throughout the task. This method allowed the human to gain awareness of the physical limitations of the robot. But this approach was not suitable for situations where synchronisation between several robotic links was required.

3. Immersive teleoperation scenarios:
   This approach was identified as quite useful for teaching complex tasks. Teleoperation allowed complex manoeuvres to be performed that needed teaching, tasks that required higher precision and accuracy. Usually these scenarios involve manipulation of redundant robotic arms with multiple links.

The literature concluded that imitation learning had provided the cognitive capabilities to the robots to learn the tasks in a purely autonomous way requiring only demonstrations from humans [3]. In [1], Lopes et al. stated that in addition to visual

demonstrations the motor and sensor information also had assisted in learning the tasks. Thus it proved that imitation learning requires a lot of data and data arrangement prior to the learning. Another drawback of imitation was that the robot was able to learn only the tasks that humans were capable of demonstrating. Additional information was lacking to improve the learning [44].

2.1.3 Deep Learning

An investigation of recent literature found that deep learning methods provided a promising solution for capabilities such as recognising a facial expression or recognising a spoken word, which are easily realised by humans, but have been difficult to achieve for a machine. The scope of these kinds of problems has an overflow with imitation learning, solutions for which require computers to have a capability to learn from experience. The relevant experience is built up during training prior to operation. In a similar manner to how the human brain is thought to work, through deep learning approaches the computers would learn a hierarchy of concepts that arranges complex concepts as a collection of simpler ones [17]. Since the knowledge is to be gathered by experience the need to specify all of the knowledge required, as identified by Lenz et al. [18] was not necessary. It was found that the concepts are built on top of each other deepening with many layers thus identifying as Deep Learning [17].

Most of the literature in deep learning has focused primarily on visual data, with the theory being that it is processed in deep learnign systems in a similar manner to how the human brain processes information for experience based knowledge. Goodfellow et al. [17] identifies the basic concept of neurons of such a system as similar to the ones in a human brain, and proceeds to explain how the same concept has been formalised in literature. In an attempt to standardise the theoretical approach of creating a deep learning network they explain that the neurons of a deep learning network are referred to as the features that have been extracted from the input data. The next part of the deep learning network analyses these features using a probabilistic calculation and ranks the probable candidate neurons. The last part of the network selects the highest ranking neurons as the outputs required for the application.

Brownlee [45] identified there were three different learning approaches in machine learning.

1. Supervised Learning: The input data was manually corresponded with output labels and the system then tries to predict the mapping between input and

output.

2. Unsupervised Learning: The system only has an input data set and tries to find the coherence between probable labels thus making the learning without supervision.

3. Semi-supervised Learning: There is a partially arranged correspondence between input and output data.

Krizhevsky et al. [46] showed that using purely supervised learning, they were able to achieve remarkable results in identifying objects in the images from a highly challenging image database known as *Imagenet* [47]. They had created a deep convolutional neural network and improved the learning avoiding unsupervised approaches in training. They further had studied how computational power helped them in increasing the network size. Their conclusions were that longer duration training helped in improving the results and they also stated that to match the human infero-temporal pathway in image recognising there would be several orders of magnitude more experiments to be conducted. Krizhevsky et al. [46] hypothesised that to obtain information that was missing or less obvious in static images, using video sequences with deeper neural networks would help.

Lenz et al. [18] presented a deep learning system to identify robotic grasps using RGB-D data. They further stated that instead of hand-engineering the features for a grasp detection problem, deep learning had been used in learning the features. They had conducted several experiments for detecting optimal grasping points for a number of different objects. A multimodal feature learning algorithm was formed based on group regularisation. They identified multimodal data as audio with video, images with text, and RGB-D data. In order to improve the efficient but near-exhaustive method of searching for the optimal grasping points on an object, they used a two-stage cascaded system. In the first iteration they identified the possible candidates for the grasping points with a lower feature set for the learning. Feeding the output to the next iteration along with complex feature extractions they selected the optimal grasping modes from the selected candidates. However, their approach took 13.5 seconds to deliver results with a maximum accuracy of 75%.

In an approach to interact with human hands, attempting to predict their future locations Lee and Ryoo [48] had proposed a convolutional neural network for future representation regression. They suggested that the hand locations at each time-step along with the robot positions could be considered as the inputs for the learning

system. After sufficient training, this would allow the system to predict the next hand location with respect to time [48]. They performed several experiments to prove their approach achieving a higher precision of 36.58 ± 16.91 with the Intersection of Union (IOU) between the predicted and ground truth hand locations. Compared to previously achieved 29.97 ± 15.37 IOU this was considered as a performance boost.

Polydoros et al. [31] identified the difficulties in manually identifying a dynamical model for a robot and proposed a neural network approach for learning inverse dynamics of manipulators using sensory data. They tested an initial hypothesis with several robotic platforms including a Baxter research robot [28]. Their algorithm was found to be better exploited when the model was continuously updated, which was partly due to the recurrent structure of their proposed network model. They further identified that their method was more adaptable than traditional approaches in object manipulation tasks, i.e. picking and releasing of objects.

The inverse kinematics problem in robotics was classified by Xia and Wang [49] to be a time-varying quadratic optimisation problem. They further identified that neural networks had conveniently reduced the computational complexity of motion planning referring to various literature [50, 51]. They jointly stated that most cases relating to kinematic control used feed-forward networks, where they had identified that the use of recurrent networks could avoid the need for off-line supervised learning and would be more suitable for unstructured environments. Therefore they had proposed the use of a single layer recurrent neural network reducing the network complexity and focusing on real-time learning. Through this work, they also proved the stability of their network.

In an attempt to win the Amazon Picking Challenge 2016 [52], Schwarz et al. [53] designed a complete robotic solution incorporating deep learning approaches. In it, they extended the DenseCap deep network [54] for its capability in captioning interesting areas of the scene using bounding boxes. The CNN in DenseCap was pre-trained using image data from ImageNet [47]. Throughout the project, they identified the importance of optimal performance of each individual network towards the success of a multiple deep network system. They concluded that the amount of training had acted towards the impressive end result that enabled their system to pickup 10 objects out of the test set of 12, with the grasp solutions obtained in under 340 milliseconds [53].

Mahler et al. [34] developed a deep learning architecture for robotic grasping known as Grasp Quality Convolutional Neural Network (GQ-CNN). It predicted an evaluation metric identified as the grasp robustness. The grasp robustness metric

provided valuable information required in grasping such as probability of grasp success, resistance to arbitrary forces and torques, etc. Mahler et al. trained their GQ-CNN on the Dex-Net 2.0 dataset, which comprised 6.7 million point clouds, parallel-jaw grasps, and robust grasp metrics. They had developed Dex-Net 2.0 for the purpose of reducing the training time through the use of cloud computing. When releasing the Dex-Net 2.0 cloud dataset Mahler [55] explained that a probabilistic model is used by Dex-Net 2.0 to generate synthetic point clouds (from physics based mesh models), grasps, and grasp robustness labels from 3D object mesh models. The main insight behind the method had been that robust parallel-jaw grasps of an object were strongly correlated with the shape of the object. Further referring to their attempt in [34] Mahler [55] described that deep CNNs were able to learn the correlations among the data in Dex-Net 2.0 using a hierarchical set of filters that recognise geometric primitives.

Considering the development and the proven results of Deep Convolutional Neural Networks (Deep CNNs) this research study has further explored the capabilities of Deep CNNs to detect grasping poses for objects in images and the use of inverse kinematics to control a Baxter robot to grasp and manipulate these objects within an unstructured environment.

2.2 Robotic Grasp Detection

Grasp detection is identified as the ability to recognise the grasping points or the grasping poses for an object in any given image [34]. As shown in Figures 2.1 and 2.2, a successful grasp describes how a robotic end-effector can be orientated on top of an object to securely hold the object between its gripper and pick the object up. As humans we use eyesight to visually identify objects in our vicinity and to find out how to approach them in order to pick them up. In a similar manner, visual perception sensors on a robotic system can be used to produce information on the environment that can be interpreted into a useful format [19]. A mapping technique is necessary to classify each pixel of the scene on the basis of belonging or not belonging to a successful grasp. Recent robotic grasping work has used several different definitions for successful grasp configurations [10, 19, 18, 20]. In this regard, a representation or a definition of a good grasp is necessary. This section reviews some of the promising grasp representations and their method of detection. Section 2.2.1 discusses several grasp configurations and how they are represented in images. Section 2.2.2 discusses how these grasp representations are detected from images. While this

section provides an overview of how grasps can be represented and detected in deep learning applications, the reader is directed towards the articles [9, 56] for further information.

2.2.1 Grasp representation

In most of the earlier works, grasps were represented as points on images of actual scenes or from 3D mesh models based on simulations. Using a supervised learning approach, Saxena et al. [15] investigated a regression learning method to infer the 3D location of a grasping point in a Cartesian coordinate system. They used a probabilistic model over possible grasping points while considering the uncertainty of the camera position. Extending their investigation, they had discretised the 3D workspace in order to find the grasping point g, given by $g = (x, y, z)$. They reported that taking object images from different viewpoints greatly helped to increase the effectiveness of the grasping. Smaller graspable regions were considered as grasp points. An example of this grasp representation is shown in Figure 2.1 [16]. In their reinforcement learning approach for grasp point detection, Zhang et al. [32] simply defined a grasp as a point in a 2D image plane. A major drawback of point defined grasps such as these, however, was that it only determined where to grasp an object and it did not determine how wide the gripper had to be opened nor the required orientation for the gripper to successfully grasp the object.

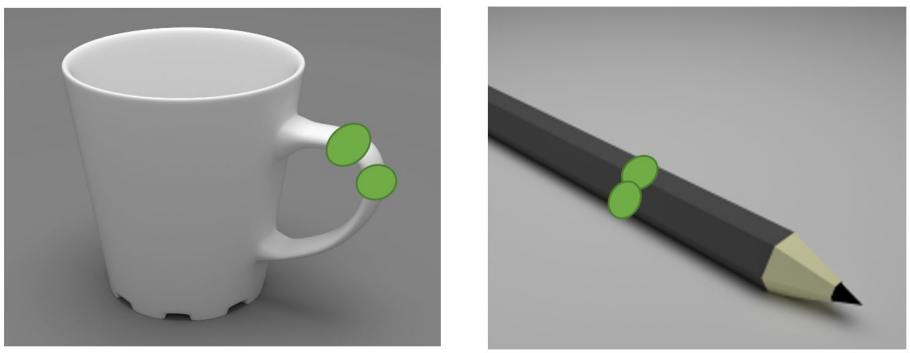


Figure 2.1: Example of the point grasp representation by Saxena et al. [16].

As a way to overcome this limitation, another popular grasp representation that has been proposed is the oriented rectangle representation that was used in [57, 18, 19, 10, 20, 58]. According to Jiang et al. [57] their grasping configuration has a seven dimensional representation containing the information of a **Grasping point**, **Grasping orientation**, and **Gripper opening width**. In world coordinates, their

grasp representation, G is stated as $G = (x, y, z, \alpha, \beta, \gamma, l)$. Their grasp representation is shown in Figure 2.2 (a). The red lines represent the opening or closing width of the gripper along with the direction of the motion. The blue lines represent the parallel plates of the end-effector.

Simplifying the previously introduced seven dimensional grasp rectangle representation from [57], Lenz et al. [18] proposed a five dimensional representation. This was based on the assumption of a good 2D grasp being able to be projected back to 3D space. While they failed to evaluate their approach, Redmon et al. [19] confirmed the validity of the method with their own results. They further supported the statements by Jiang et al. [57] and Lenz et al. [18], that detection of grasping points in this manner was analogous to object detection methods in computer vision but with an added term for the gripper orientation. Adapting the method of [57, 18], they also presented a slightly updated representation of a grasp rectangle as shown in Figure 2.2 (b). This modified rectangle grasp representation has been used in a number of later publications demonstrating its usefulness [10, 20, 58]. In their work to use deep learning algorithms for robotic grasping detection, Kumra et al. [10] have used this grasp rectangle originally proposed by Redmon et al. [19]. A very recent online project page [20] has cited the same Redmon grasp rectangle in their work.

Figure 2.2: Grasping rectangle representations. (a) The representation by Jiang et al. [57]: The upper-left corner (r_G, c_G), length m_G, width n_G and its angle from the x-axis, θ_G for a kitchen utensil. It has multiple grasps defined as shown. (b) The simplified representation by Redmon et al. [19] for a hammer, showing its grasp centre at (x, y) oriented by an angle of θ from its horizontal axis. The rectangle has a width and height of w and h respectively.

Despite not employing a CNN in their dictionary learning method for grasp detection, Trottier et al. [59] used the same grasp rectangle that was used in previous methods, most likely due to its similarity to the object detection representations that

were widely used at that time. In another study conducted by Park et al. [60], the same grasp rectangle representation was again used. They argued that the grasp rectangle was analogous to the standard object detection bounding box with the added feature being the orientation. In their novel classification method for grasp detection Zhou et al. [61] used the similar 5-element grasp rectangle representation following the previous work in [18, 19, 10, 20, 58, 59, 60, 62]. Wang et al. [63] proposed a minor variation to this approach that differed simply by excluding the parameter for gripper plate height (h). They argued that this parameter can be controlled in the robotic set-up configurations thus the authors used a four-element grasp representation of $G = (x, y, \theta, w)$.

Another grasp representation introduced in more recent research is the combined location and orientation representation. In [64], the authors have used the simple $G = (x, y, \theta)$ representation that dropped the dimensional parameters (h, w). The dimensional parameters provided a sense of the physical limitations for certain end-effectors. Similar representations were used in [65, 66]. This representation described a grasp in a 2D image plane. This representation was improved by Calandra et al. to include the 3D depth information just by adding the z coordinates to the representation, resulting in a grasp representation, $G_z = (x, y, z, \theta)$ [67, 68]. In their approach to detect robotic grasps using tactile feedback in addition to visual sensing, Murali et al. [69] used the same representation, (G_z).

From the three grasp representations described in this section, the Rectangle representation can be identified as the most commonly used for any grasp detection applications. While a detailed analysis of the relative suitability of the approaches has not been conducted, the literature survey suggests that any preference is application specific. Table 2.1 summarises the characteristics of each grasp representation type with regards to depth, pose, and physical limitations of the end-effector. Lenz et al. [18] argued that in most cases, the depth information can be manually controlled specific to the application. Therefore the rectangle representation can be selected as the most suitable grasp representation in most cases.

Table 2.1: Comparison between different grasp representations [70]

Type	**Parameters**	**Depth**	**Pose**	**Limitations**
Point	(x, y)	No	No	No
	(x, y, z)	Yes	No	No
Location + Orientation	(x, y, θ)	No	Yes	No
	(x, y, z, θ)	Yes	Yes	No
Rectangle	(x, y, θ, h, w)	No	Yes	Yes

Pixel-wise grasp representations were used when structured grasps were not useful. Ku et al. [71] used convolutional layer activations to find the anthropomorphic grasping points in images. They created a mask that represented the grasping points for the robotic index finger and the thumb. The mask contained all the pixels that were part of the grasp [71]. Their method only worked for cuboid and cylindrical shaped objects. They reported that only one trial failed from the complete set of 50 trials achieving an average success rate of 96%.

In applications where simple object localisation translates back to the simple pick-up points in the 2D image plane, dense captioning was used to localise objects in order to pick them up. In their work with Amazon Picking Challenge [52], Schwarz et al. [53] used the popular dense captioning [54] to localise objects in images. Dense captioning provides a textual description of each region of the image and it can be used to identify or localise objects in an image. During the testing they had successfully picked up 10 objects out of the 12 test object set, and their fine-tuned system responded within 340 milliseconds during the testing.

These graspable region representations are widely used in picking or sorting objects in clutter when there are no particular requirements on the order in which objects are picked up. More structured grasp representations are generally employed in conjunction with object recognition in order to grasp the identified objects [19]. The works discussed in this section demonstrate that a consistent grasp representation method must be adopted in order to start working with learning algorithms for the detection of robotic grasps. The ground truth labels should have the optimal number of parameters to represent a grasp while ensuring that it is not over-defined. The five dimensional grasp representation originally presented by Lenz et al. [18] for a 2D image should, thus, be further explored.

2.2.2 Grasp detection

The conventional analytical method of robotic grasp detection is performed on the premise that certain criteria such as object geometry, physics models, and force analytics are known [9]. The grasp detection applications are built based on a model developed with this information. The modelling of such information is often challenging due to the current rapidly changing industry requirements. An alternative approach is to use empirical methods, also known as data-driven approaches, that rely on previously known successful results. These methods are developed using existing knowledge of object grasping or by using simulations on real robotic systems [56]. A major drawback of analytical methods is that they rely on the assumption that the

object parameters are known, therefore they cannot be used for a generalised solution [56]. There are two types of empirical approaches in robotic grasp detection:

1. Methods that use learning to detect grasps and use a separate planning system for grasp planning

2. Methods that learn a visuomotor control policy in a direct *image-to-action* manner

In the literature, direct grasp detection has been carried out using two different techniques. The most popular method is to detect structured grasp representations from images. Another portion of the research has approached the grasp detection problem by learning a grasp robustness function. Both of these method require a separate grasp planning system to execute the grasp. During the last few years, there has been a growing interest into learning a visuomotor control policy using deep learning. The introduction of tools such as NVIDIA Isaac [26] has enabled the extensive use of reinforcement learning in simulated environments with domain adaptation. These visuomotor control policy learning methods do not require a separate grasp planning system.

The most popular method for structured grasp detection was the sliding window approach proposed by Lenz et al. [18]. In their approach, a classifier is used to predict if a small patch of the image contains a potential grasp. The image is divided into a number of small patches and each patch is run through the classifier in an iterative process. The patches that contain higher ranking grasps are considered as candidates and pushed as outputs. This method yielded a detection accuracy of 75% and a processing time of 13.5 seconds per image. Similar results were reported from the studies by Wang et al. [72] and Wei et al. [51] who followed a similar approach. Guo et al. [62] used the reference rectangle method to identify graspable regions of an image. This method was adapted from region proposal neural networks [73]. The locations of the reference rectangle were identified using the sliding window approach. Due to the repetitive scanning method for identifying graspable regions of images, this method was largely considered unsuitable where a real-time detection speed is necessary. As an alternative, Redmon et al. [19] proposed the one-shot detection method.

In most one-shot detection methods, a direct regression approach for predicting a structured grasp output is used. In these approaches, the structured output represents the oriented grasp rectangle parameters in the image plane coordinates. In the first one-shot detection approach, the authors argued that a faster and more accurate

method was necessary and proposed to use transfer learning techniques to predict grasp representation from images [19]. They reported a detection accuracy of 84.4% in 76 milliseconds per image. This result produced a large performance boost compared to the then state-of-the-art method, the sliding window. The one-shot detection method assumed that each image contained one graspable object and predicted one grasp candidate as opposed to the iterative scanning process of the sliding window approach [19]. Following a similar strategy, Kumra et al. [10] reported an improvement whereby a detection accuracy of 89.21% for their multi-modal grasp detector with a processing speed of 100 milliseconds per image was achieved. They explained that it was due to the deeper network architecture that they had used in their experimentations. Therefore, it was evident that most work in one-shot detection followed deep transfer learning techniques to use pre-trained neural network architectures [20, 58].

Although the preferred method for one-shot detection is the direct regression of the grasp representation, there were numerous occasions where the combined classification and regression techniques were employed for one-shot detection. While arguing that the orientation predictions of a structured grasp representation lay in a non-Euclidean space, where standard regression loss (L_2) had not performed well, Chu et al. [74] proposed to classify the orientation among 19 different classes in the range of (0° : 360°). They used a direct regression method to predict the bounding box of the grasp. The authors reported a detection accuracy of 94.4% with RGB images. Building on the concept by Guo et al. [62], Zhou et al. [61] proposed to use the anchor boxes for predefined regions of the images. Each image was divided into $N \times N$ regions. The orientation of the anchor box was classified between k classes. The authors argued that the k can be a variable integer. By default it was set to $k = 6$. The angles ranged between ($-75^\circ : 75^\circ$) [61]. Zhou et al. [61] achieved a detection accuracy of 97.74% for their work. The literature reasoned that these improvements were achieved as it was easier to converge to a classification during the training and the associated errors were minimum, but a classification would limit the output to a predefined set of classes [64].

Learning a grasp robustness function also had been the central idea of many studies in deep grasp detection. The researchers used this function to identify the grasp pose candidate with the highest score as the output. Grasp robustness described the grasp probability of a certain location or an area of an image [34]. Binary classification was a well researched technique for this approach that classified the grasp points as valid or invalid (1, 0). Park et al. [60] used a multi stage spatial transformer network to predict the success of a grasp candidate. They reported a grasp detection accuracy of

89.60% in 23 milliseconds per image [60]. Using end-to-end learning, Ten Pas et al. [75] performed a binary classification to identify graspable regions in a dense clutter of object, the authors presented 77% detection accuracy with passive point cloud data. In their work, Lu et al. [76] performed a CNN based grasp probability study to achieve a detection accuracy of 75.6% for previously unseen novel objects and 76.6% for previously seen objects during the training. In addition to the applicability of this method to cluttered objects, researchers concluded its usefulness when partial information was observed [60].

A method to learn an optimal grasp robustness function was proposed by Mahler et al. [34]. They considered the robustness as a scalar probability in the range of $(0-1)$. The authors compiled a dataset known as *Dex-Net 2.0* with 6.7 million point clouds and analytic grasp quality metrics with parallel-plate grippers planned using robust quasi-static grasp wrench space analysis on a dataset of 1,500 3D object mesh models. They have further trained a grasp quality convolutional network (GQ-CNN) that was used to learn a robustness metric for a grasp candidate. They have tested their CNN with their dataset achieving an accuracy of 98.1% for grasp detection. Robust grasp detection was explored in [77]. Johns et al. reported that they achieved a grasp success rate of 75.2% with minor gripper pose uncertainties and 64.8% with major gripper pose uncertainties. They described the gripper pose uncertainties as resulting from the varying shapes and contours associated with the objects.

The literature survey suggested that a direct mapping of images to robot actions could be predicted by learning a visuomotor control policy. This method would not require a separate grasp planning system and, thus was also considered a pixel-to-action method. Using their previous findings in [34], Mahler et al. [66] proposed a method to find deep learnt policies to pick objects from amongst clutter. The authors reported that by using a transfer learning technique with their previous findings in [34], they achieved a grasp detection accuracy of 92.4%. When they tested their learnt policies on robotic grasping, they achieved a success rate of 70% with 5 trials for each of 20 objects of the test dataset [66]. The winning team from the Amazon Picking Challenge 2017 [52], Zeng et al. [78] proposed a visuomotor control policy prediction method for images of objects in clutter. The authors proposed an action space with four individual actions: (a) suction down, (b) suction side, (c) grasp down, (d) flush grasp. They reported a maximum accuracy of 96.7% for grasping and 92.4% for suction with the *Top-1* confidence percentile [78]. Zhang et al. [32] proposed a method to use reinforcement learning [79] to determine the action to extend the robot end-effector to a point in a 2D image plane. Closely following the proposed deep Q

network by Mnih et al. [79], the authors managed to adapt it to a robotic system using synthetic images. They tested their system to achieve a 51% success rate in reaching the target point [32]. They further concluded that these results were largely affected by not having an optimal domain adaptation from synthetic to realistic scenes.

In summary, it is identified that most of the work in grasp pose detection has focused on the detection of a structured grasp representation. For structured grasp representation detection, the one-shot detection method has achieved state-of-the-art results according to recent research studies. Most of one-shot detectors use deep transfer learning techniques to use pre-trained deeper convolutional networks to predict the grasp candidates from images. This sections has introduced the popular grasp detection methods and the reader is directed to Section 5 for an in-depth discussion of the convolutional network approach for deep detection.

2.3 Learning from Data

Research has consistently shown that deep learning requires a large volume of labelled data to effectively learn the features during the training process [17]. This requirement is also apparent in supervised learning methods in robotic grasp detection [18, 19, 10, 20]. In recently published work, researchers either use training data from a third party or introduce their own application specific proprietary data sources or methods to automate the data generation [34, 64]. Johns et al. [77] highlighted that the major challenge with deep learning is the need for a very large volume of training data, thus they opted to generate and use simulated data for the training process. Another challenge of training deep neural networks is the lack of domain specific data as mentioned by Tobin et al. [14]. They proposed a method to generate generalised object simulations in order to address this challenge, although it is yet to be proven how effective the results can be. For real-time applications, use of simulated data and the availability of 3D object models is not practically achievable [19, 10]. As a way to overcome this, there are reports of network pre-training as a solution when there is limited domain specific data [19, 10, 20].

2.3.1 Quality and Quantity of Training Data

Brownlee [80] specified that annotations of the available data will be more important if the learning was purely supervised and less important for unsupervised learning. The author further described the importance of the three subsets of the data for training, validation, and testing. In [19, 10, 20, 58], the authors had followed the same

argument. A comprehensive explanation can also be found in [81]. The literature survey suggests, in addition to the larger training datasets, domain specific data is necessary for effective results.

Multi-modal Data

Use of multi-modal data has become popular in many research studies into robotic grasp pose detection. Early work from Saxena et al. [15] stated that most grasping work assumed prior knowledge of the 2D or 3D model of the object to be grasped, but such approaches encounter difficulties when attempting to grasp novel objects. The authors experimented with depth images for five different objects in their training. They reported the grasp success rates for basic objects such as mugs, pens, wine glasses, books, erasers, and cellphones. An overall success rate of 90% with a mean absolute error of 1.8 cm was reported.

Following this work, Jiang et al. [57] scaled the problem space to 194 images of 9 classes. They stated that the availability of multi-modal data could be useful in identifying edges and contours in the images to clearly differentiate graspable regions. Lenz et al. [18] supported the same claim in their work that used multi-modal RGB-D data. In a few recent transfer learning applications, the authors have used the multi-modality in a way that overcame the 3-channel data limitation with existing pre-trained CNNs. The authors in [19, 58, 20] replaced the Blue channel in RGB images with depth disparity images and created 3-channel RG-D images.

Kumra et al. [10] proposed a novel method to use pre-trained DCNN architectures with 3-channel input limitations. Instead of replacing the Blue channel, the authors trained two convolutional networks for RGB and depth features individually. They used a similar encoding to [82] in order to create 3-channel depth disparity images. They further reported that proper pre-training for the depth CNN was not available since all of the pre-trained networks were pre-trained on RGB images. More streamlined methods such as [83] would help in this endeavour. While most of these works used visual information, there were some reported studies that had used tactile sensing with deep learning approaches in grasp detection.

In [69], Murali et al. explored using tactile sensing to complement the use of visual sensors. This method involved a re-grasping step to accurately grasp the object. The authors reported a success rate of 85.92% with a deep network and 84.5% with an SVM. A similar approach was followed by Calandra et al. [67] in their work to use tactile sensing in robotic grasp detection.

Some researchers had also experimented with uni-modal data as well. Kumra et al. [10] trained their neural network with uni-modal RGB images and achieved an accuracy of 88.84%.

The literature survey indicates there are several types of multi-modalities involved in grasp pose detection research with the most popular one being the RGB-D data. Evidence suggests that the added benefit of edge and contour information in RGB-D images has an advantage over uni-modal RGB images. There is not yet enough evidence to suggest whether tactile sensing has an added advantage over depth imaging for grasp detection work using RGB images. A major challenge with respect to using RGB-D data, however, is the access to suitable training data sets.

2.3.2 Datasets

Goodfellow et al. [17] stated that the performance of a simple machine learning algorithm relied on the amount of training data as well as the availability of domain specific data. The recent publications suggested that the availability of training data was a prevailing challenge for this learning method. Some researchers combined datasets to create a larger dataset while others collected and annotated their own data.

Pre-compiled datasets

The Cornell Grasp Dataset (CGD) from [38] was identified as a popular grasp dataset that was compiled for most transfer learning approaches in robotic grasping [19, 10, 20, 58]. The CGD was created with grasp rectangle information for 240 different object types and it contained about 885 images, 885 point clouds and about 8019 labelled grasps including valid and invalid grasp rectangles. A sample of a set of images is shown in Figure 2.3. The grasps were specifically defined for the parallel plate gripper found on many robotic end-effectors. The CGD appeared in a number of research studies during the recent past, which might suggest that it has a reasonable diversity of examples for generalised grasps [19]. The recent trend of using RGB-D for learning to predict grasps was covered with the CGD dataset through the inclusion of point cloud data by its creators. Lenz et al. argued that having the depth information would result in a better depth perception for an inference system that was trained on depth data [18]. A sense of good and bad grasps was also necessary to differentiate a better grasp from the alternatives [18, 19]. Therefore, the CGD could be selected as a suitable dataset for its quality and adaptability. The CGD was extensively used in [18, 19, 10, 20, 58, 74, 62, 60].

Figure 2.3: Sample of Cornell Grasp dataset [38].

In the grasp detection work by Wang et al. [63], the authors used the Washington RGB-D dataset [82] for its rich variety of RGB-D images. The authors self-annotated as they preferred to combine the resulting dataset with the CGD. The authors further stated that the combined Washington data instances of 25,000 with the 885 instances from the CGD would help in pre-training a deep network [63].

Collected datasets

When application specific data was necessary, researchers have provided intuitive methods for data collection. Murali et al. [69] used a previously learnt grasping policy to collect valid grasp data and performed random grasps to collect invalid grasp data. They have collected data for 52 different objects. Calandra et al. [67] collected data from 9,269 grasp trials for 106 unique objects. Pinto et al. [64] stated how time consuming it was to collect data for robotic grasping and proposed a novel approach inspired from reinforcement learning. Their approach would predict centre points for grasps from a policy learnt using reinforcement learning and the orientation was classified for 18 different classes using grasp probability. The authors scaled the data collection to 50,000 grasp trials using 700 robotic hours. They used a Mixture of Gaussians (MOG) background subtraction that identified graspable regions in images in order to avoid random object-less spaces in images. Levine et al. [84] further

improved this approach through the collection of grasping data from nearly 900,000 grasp trials using 8 robots.

Domain adaptation and simulated data

As pointed out by Tobin et al. [14], most applications lacked domain specific data. While arguing the importance of a large volume of domain specific data, the authors proposed a method to use physics simulations to generate domain specific data using 3D mesh models for a set of primitive shapes. Most of the work that has used simulation and 3D model data relies on domain adaptation to its real world equivalent set of objects. Bousmalis et al. [85] conducted several experiments to verify the domain adaptation capability of a deep grasp detection application that was trained on 3D mesh models randomly created by the authors. The authors randomly mixed simulated data with realistic data to compile a dataset of 9.4 million data instances, using this a grasp success rate of 78% was achieved. In a similar approach, Viereck et al. [65] proposed a method for learning a closed-loop visuomotor controller from simulated depth images. The authors generated about 12,500 image-action pairs for the training. They reported a grasp pose detection success rate of 97.5% for objects in isolation, and 94.8% for objects in clutter [65]. Mahler et al. [34] suggested populating a dataset containing physics based analyses such as caging, grasp wrench space (GWS) analyses and grasp simulation data for different types of object shapes and poses. They further suggested that cloud computing could be leveraged to train a convolutional neural network with this dataset that would in turn, predict a robustness metric for a given grasp instead of directly predicting a grasp. The proposed dataset was called *Dex-Net 2.0* [55] and contained about 6.7 million point clouds and analytic grasp quality metrics with parallel-jaw grasps planned using robust quasi-static GWS analysis on a dataset of 1,500 3D object models [34].

Summary

Mahler et al. [34] concluded that human annotation is a tedious process that requires months of work and the simulations would have to be run for a large number of iterations on a robotic system. With the limited availability of domain specific data, Redmon et al. [19] proposed to use pre-training. Pre-training assumes that by using the weights of the convolutional overhead of a CNN model that was trained on a large dataset such as ImageNet [47] would transfer the universal filtration capabilities to a smaller dataset, providing better results compared to the usual training approach. Even though most of the prior work used 3D simulations to find suitable

grasp poses for objects, Kumra et al. [10] stated that despite those previous works having performed well they required a known 3D model of the object to calculate a grasp. This 3D model would not be known a priori and the complex modelling techniques of forming the 3D model was beyond the capacity for most of the general purpose robots as their desired primary function was faster adaptation to dynamic scenarios [19]. In such cases, a learning algorithm would produce the necessary results provided that there were enough, domain-specific data instances for the training.

In conclusion, the amount of training data plays a key role in the outcome of the trained algorithm. Approaches such as [64, 84, 86] try to reduce or completely avoid the challenges of compiling such huge datasets. In cases where an extended information set is necessary to produce a grasp prediction, a dataset such as the Dex-Net 2.0 [34] could be used. For most generalised grasp prediction networks, however, the CGD [38] would be an optimal starting point considering its adaptability for more generalised object shapes and poses with the added benefit of the inclusion of depth information.

2.4 DCNNs for Grasp Detection

Most recent work in robotic grasp detection apply different variations of convolutional neural networks to learn the optimal end-effector configuration for different object shapes and poses [20, 10, 18, 34, 64, 19]. They do so by ranking multiple grasp configurations predicted for each object image instance. Ranking is done based on the learnt parameters from the representation learning capability of deep learning. As opposed to the manual feature design and extraction steps of classical learning approaches, deep learning can automatically learn how to identify and extract different application specific feature sets [17]. The authors of [17, 79] explained the importance of the CNN architecture towards learning. It was further reported that networks with greater depth would be able to learn more complex hierarchical representations from images [46].

In analytical approaches, various grasping application specific parameters such as closure properties and force analysis are combined to successfully model the grasps [9]. Closure properties describe the force and momentum exerted at the point of contact, also known as a *Grasp Wrench*. Depending on the level of friction at each of these points, the point of contact could be further elaborated. According to Bicchi et al. [9] force analysis describes the required grasping force that should be applied by the robotic gripper on the object so as to grasp it securely without slipping or

causing damage. Kinematic modelling between the contact points is a function that describes the relative motion between two different contact points. Reviews such as [9] suggested how practically impossible it would be to prepare a generalised grasping model using just analytical data. Given how well certain learning algorithms [10, 19, 18, 20] performed in the past, however, it could be concluded that using visual representation of successful grasps as training data with these learning algorithms would result in usable generalised solutions.

2.4.1 Architecture

A deep CNN is built with multiple layers to extract information representations [36]. Goodfellow et al. [17] has stated that the representation of the learning process of a deep neural network has similar attributes to the method through which information is processed by the human brain. During the last five years there have been many active improvements for DCNN architectures. Most of these approaches use ImageNet [47] tests for benchmarking. From inspecting many CNN methods that are originally evaluated on ImageNet data, it is evident that all of them have followed the general structure shown in Figure 2.4. Literature suggests that more lower level features are identified using the convolutional layer while application specific features are extracted by the fully connected portion of the network where pooling and activations are widely employed [87]. This suggests that the results on the ImageNet data provide a reasonably useful evaluation of the architecture even though it is not specific to robotic grasping.

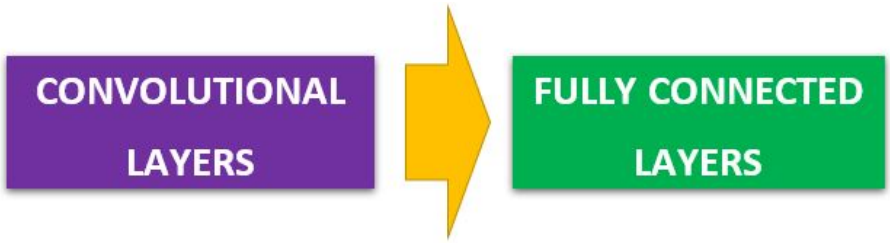


Figure 2.4: General structure of a CNN

The literature survey suggested two types of convolutional layer placements in DCNNs. Early approaches used a stacked architecture where each layer was placed one after the other. More recent DCNNs have used convolutional layers in parallel. Szegedy et al. [87] reported that this trend was accelerated due to the availability of increased computational capacities.

Both of the popular AlexNet and VGG-Net were stacked deep neural network architectures. With AlexNet, Krizhevsky et al. [46] produced a reduced error rate of 16.6%. Simonyan et al. further reduced the error rates to 7.0% with their introduction of VGG-Net [88]. Redmon et al. [19] were the first authors to implement AlexNet with their work in robotic grasp detection. They fine-tuned the DCNN architecture to accommodate their hardware. Their model is shown in Figure 2.5. Their direct regression model that was trained on RG-D images achieved an accuracy of 84.4% and the *MultiGrasp* model that divided an original image to $N \times N$ sub-images achieved an accuracy of 88%. Their work was later followed by Watson et al. [58] who achieved an accuracy of 78% with one fold cross validation. Ebert et al. [20] achieved an accuracy of 71% closely following the same work in [19].

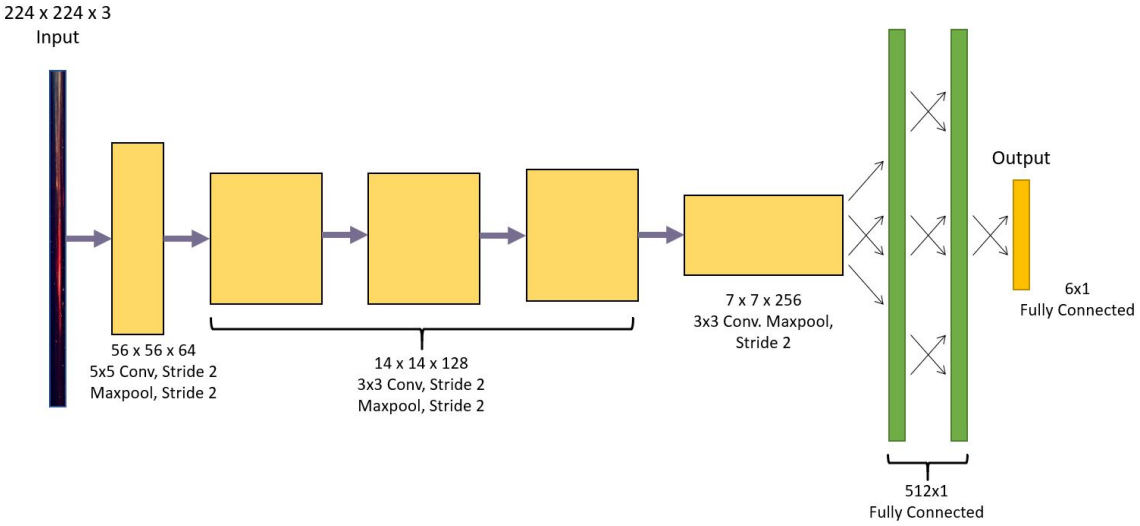


Figure 2.5: Neural network model proposed by Redmon et al. [19].

Modern dense DCNN architectures are developed under the premise that deeper networks are capable of extracting more advanced features from data. Szegedy et al. [87] reported that the drawbacks of increasing the depth of a DCNN are two-fold. In order to train such deep network models there should be a distinguishable variation between the training data and this was challenging even with human labelling. When the depth of a DCNN is increased, the number of trainable parameters automatically increases, which requires higher computational power for training [87]. Therefore they suggested sparsely connected deep network architectures. They proposed their DCNN architecture known as GoogleNet, with a reduced error rate of 6.8% in ImageNet testing [47]. Following that He et al. [89] proposed a DCNN architecture with skip connections that further reduced the error rates to 3.57% using their ResNet

architecture. A sample residual block is shown in Figure 2.6. By combining [89] with their original approach in [90], Szegedy et al. proposed the Inception-ResNet architecture [91].

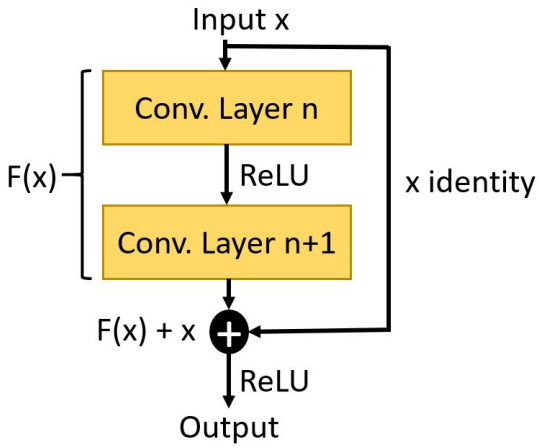


Figure 2.6: Example of a residual block [10].

Denser deep networks have appeared in recent robotic grasp detection work. Kumra et al. [10] used the 50 layer version of the popular ResNet [89] architecture to extract features from RGB-D images in order to detect grasp configurations for the objects in the images from the Cornell grasp dataset [38]. The authors presented two different network models that were aimed at grasp detection using uni-modal and multi-modal images named *Uni-modal* and *Multi-modal* architectures respectively. They achieved the highest grasp detection accuracy of 89.21% using their multi-modal DCNN architecture with image-wise splitting. This architecture is shown in Figure 2.7. Zhou et al. [61] used the ResNet-50 and ResNet-101 networks as feature extractors in their grasp detection work and achieved accuracies over 98% for both versions. Chu et al. [74] have used the same ResNet-50 architecture [89] with their grasp detection work. As opposed to the previous approach in [10], they used grasp labels from the grasp data [38] to propose regions of interest in the images to ultimately propose multiple grasps at once. They achieved a detection accuracy of 96% using image-wise splitting.

Another research direction in robotic grasp detection is the use of custom neural network models inspired from the popular region proposal convolutional networks known as R-CNN [73]. Lenz et al. [18] developed their two cascaded CNN models for grasp detection using a sliding window approach. The first neural network model extracted higher level features such grasp locations whereas the larger second network verified the valid grasps from those detected. In the first stage the authors used a

variant of the *Sparse Auto Encoder* [92] to initialise the weights of the hidden layers. Pre-training in this way was a necessary step to avoid overfitting. The authors reported a grasp detection accuracy of 75% that was tested with a Baxter robot by carrying out object grasping. The creation of application specific DCNNs has received greater attention recently. When there has been enough higher quality training data available, researchers have used their own custom neural network models. Most of these works were motivated from recent DCNN success stories in ImageNet classification tests [47]. Lu et al. [76] used a custom architecture for their work in multi-fingered grasp prediction. The max-pooling and rectified linear units were used as activation functions in their work. They further concluded the adaptability of their work into the realm of two-fingered grasp detection [76]. The detection accuracies of 75.6% for novel objects and 76.6% for previously seen objects were claimed.

The literature survey has further suggested that researchers have used DCNN or CNN methods to find the inverse kinematic or dynamic solutions. The kinematic and dynamic modelling of robotic manipulation were focused in [49] and [31] respectively. The proposed methods were further experimented with by various authors in motion planning in unstructured environments [50, 51]. Even though these works do not directly align with robotic grasp detection, they can be extended to grasp planning.

Szegedy et al. in [87] stated that advances in the quality of image recognition had relied on newer ideas, algorithms, and improved network architectures as well as more powerful hardware, larger training data sets, and bigger learning models. They further commented that neither the deep networks nor the bigger models alone would result in such improvements but combining them in order to create a deeper architecture would suggested these improvements over the classical theories. They experimentally presented that by increasing the depth (the number of deep levels) and the width (the number of units at each level) of a deep network would improve the overall network performance. However, it would also require a commensurately larger set of training data that would result in the following drawbacks:

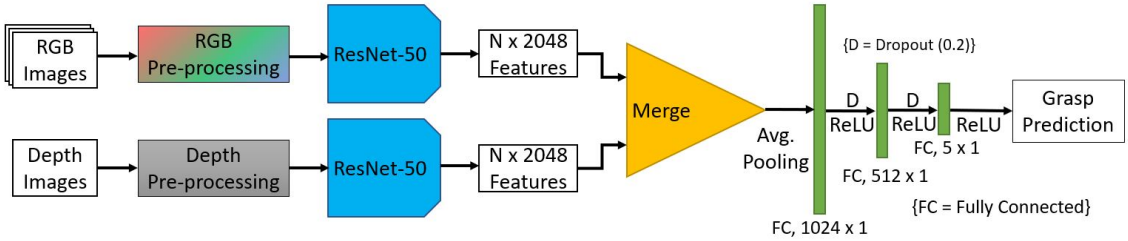


Figure 2.7: Multi-modal grasp predictor [10].

1. Larger data sets would result in increased features to be extracted while limited data sets would result in overfitting.

2. Deeper networks would require increased computational resources during the training.

The authors evaluated their original DCNN performance in [87] that had an error rate of 6.67% with their own improved version in [90] with an error rate of 3.5%. They concluded in [90] that this performance boost was a result of going deeper with convolutions.

2.4.2 Transfer Learning Techniques

The most successful robotic grasp detection work has used transfer learning methods to achieve accuracies close to 90%. Any transfer learning approach includes the following steps:

1. Data pre-processing

2. Pre-trained CNN model

Compared to image classification, robotic grasp detection requires the capability of a DCNN to identify grasp configurations for novel objects. This requires training on generalised object scene images. Therefore, most researchers limit their pre-processing techniques to just accommodate CNN input dimensions such as image width and the number of channels. Unlike in image classification, the ground truth data for grasping was less augmented. Redmon et al. [19] reported the minimum amount of necessary pre-processing for RGB-D datasets as centre cropping of images and replacing the blue channel of RGB images with depth data while normalising the depth data to (0, 255) range, which is the default RGB colour space range. While following the exact same procedure in [19], Watson et al. [58] normalised all RGB values and grasp labels to (0, 1) arguing that training targets should be in the same range as the training data. In their method, Pinto et al. [64] resized images to 227×227 which was the input size for their model. Their network had a similar architecture to AlexNet [46] as shown in Figure 2.8.

Due to the problem of overfitting with the limited available datasets, the deep learning robotic grasp detection literature indicates that many authors have used pre-training. Pre-training was identified as a transfer learning technique where the deep network was pre-trained on larger datasets prior to the training on domain specific

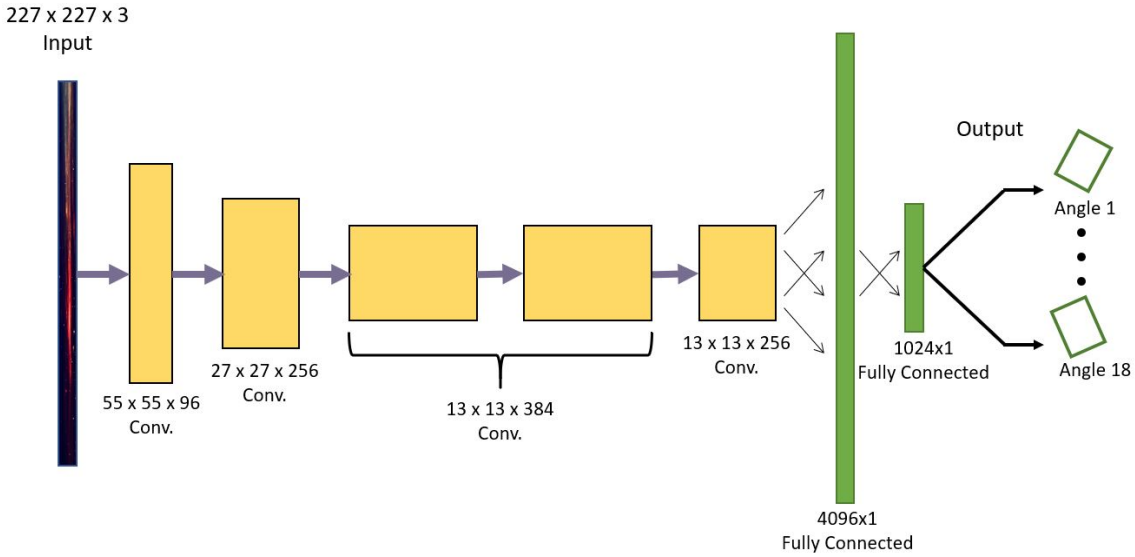


Figure 2.8: 18-way binary classifier by Pinto et al. [64].

data. The weights of the convolutional layers that were originally learnt during the pre-training were kept frozen during the training on domain specific data.

In their one-shot detection method, Redmon et al. [19] used the AlexNet [46] convolutional network architecture in compiling their network model [19] which achieved a grasp detection accuracy of 84.4%. The same approach was used in [20, 58] with similar results reported. They modified the orientation parameter from the grasp representation while arguing that the angle predictions are two-fold (positive or negative) [19]. Therefore the authors replaced θ with $(\sin 2\theta, \cos 2\theta)$ following the trigonometric definitions. The argument was further supported in [20, 58].

A similar pre-training approach was employed in [10] using the deeper ResNet-50 architecture as opposed to the variant of the AlexNet from [19]. They reported that the deeper network model afforded them an increased accuracy of 89.21%. They further reported a second model as shown in Figure 2.9. It was aimed at uni-modal data such as RGB or RG-D images. This model achieved an accuracy of 88.84%.

Even though the transfer learning made it less challenging to use a pre-trained model for the convolutional part of a DCNN, researchers still had to design the fully connected part of the DCNN. While there is not enough evidence to determine the optimal number of units required, Redmon et al. [19] used two fully connected layer that had 512 units in each individual layer. In [10], Kumra et al. used one fully connected layer with 512 units for their *Uni-modal* architecture and two layers with 512 units for their *Multi-modal* architecture. A model similar to the deep network in [19] was later followed in [20] and [58]. In DCNN training applications, the popular

learning optimiser is the *Stochastic Gradient Decent (SGD)*. In deep grasp detection most authors used the SGD but they argued that it was not an optimal optimiser and reported more advanced optimisers were necessary. Ruder provided a comprehensive overview of different learning optimisers in [93].

While most transfer learning approaches employed pre-trained network models in an end-to-end learning process some researchers used them as feature extractors for shallow network models. Chollet [94] stated that due to the 2-step method of feature extraction it was impossible to employ data augmentation techniques if required as the learnt features would not be the same as the training images. Also running end-to-end learning was costlier as it required the convolutional base of the network to be run on data repetitively.

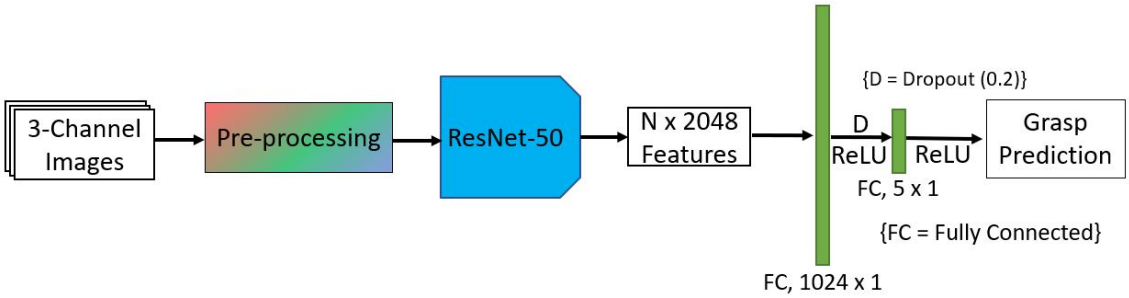


Figure 2.9: Uni-modal grasp predictor [10].

Table 2.2: Comparison between different transfer learning techniques in one-shot grasp detection. Results were reported from tests performed on the Cornell Grasp Dataset [70]

Method	**Architecture**	**Accuracy (%)** (image-wise)	**Accuracy (%)** (object-wise)
Direct regression by Redmon et al. [19]	AlexNet [46]	84.4%	84.9%
Regression+Classification by Redmon et al. [19]	AlexNet [46]	85.5%	84.9%
MultiGrasp Detection by Redmon et al. [19]	AlexNet [46]	88.0%	87.1%
Uni-modal, SVM, RGB by Kumra et al. [10]	ResNet-50 [89]	84.76%	84.47%
Uni-modal RGB by Kumra et al. [10]	ResNet-50 [89]	88.84%	87.72%
Uni-modal RGD by Kumra et al. [10]	ResNet-50 [89]	88.53%	88.40%
Multi-modal SVM, RGB-D by Kumra et al. [10]	ResNet-50 [89]	86.44%	84.47%
Multi-modal RGB-D by Kumra et al. [10]	ResNet-50 [89]	89.21%	88.96%
Direct Regression (RG-D) by Basalla et al. [20]	AlexNet [46]	71%	NA
Direct Regression (RG-D) by Watson et al. [58]	AlexNet [46]	78%	NA

Detecting object grasping configurations from images is still accurately solved using analytical methods but the use of empirical methods is exponentially increasing due to successful results in recent publications. One commonly used method is to train a visuomotor controller using deep learning that iteratively corrects the grasping point until the object is successfully grasped between the gripper jaws. The next best method is to learn a function that scores the possible grasps on an image and use it to select the highest scored grasp as the candidate. There are other methods that learn a certain heuristic and exhaustively search for possible grasps on the images. Training CNNs to detect grasps requires a high volume of manually labelled data. As a solution, most researchers opt to use simulated training data as in [77, 65, 14]. Alternatively, data collection can be automated as in [64]. Recent approaches have suggested network pre-training can help to avoid overfitting due to the limited availability of training data [19, 10, 20]. As shown in Table 2.2, AlexNet [46] and ResNet-50 [89] were widely used in these studies. The reasoning for the reduced results in [20] and [58] compared to [19] was not clarified by any author. However,

the literature survey suggested that these methods utilised a lower number of training data instances and a varied set of data augmentation techniques as compared to [19]. This, further supported the arguments made by many different researchers regarding the limited availability of high quality annotated training data for such robotic grasp detection work.

2.4.3 Evaluation of Results

Each grasp configuration that was predicted by the learning algorithms should go through an evaluation process to identify if it is in fact a valid grasp. There are two evaluation metrics for grasps: **rectangle metric** and **point metric** [19, 10, 20]. The point metric evaluates the distance between the predicted grasp centre and the actual grasp centre relative to a threshold value, but the literature does not provide further insight as to how best to determine this threshold value. Furthermore, this metric does not consider the grasp angle, which neglects the object orientation in the 2D image plane.

The rectangle metric defined a successful grasp under the following two conditions.

1. Difference between the grasp angles to be less than 30°
2. Jacquard index between the two grasps to be less than 25%

The Jacquard index is given by Equation 2.1:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

This method evaluates grasp poses in their image plane. Watson et al. [58] argued that further evaluation metrics would be necessary in order to evaluate grasps on the fly. They proposed an on-line evaluation method known as "Marker based evaluation method". The process evaluates grasps detected on images which had objects with the labels marked by the authors. In Figure 2.10 a marked banana is shown with red markers. The authors evaluated the distance to the predicted grasp centres from these markers and reported the grasp success based on this [58].

Further to this the authors validated their results with one-fold cross validation and implemented an inverse kinematic grasp planner with a Baxter robot [58]. An implementation of grasp planning supported their conclusions in [58] as opposed to previous attempts in [19, 10] that failed to further validate grasp detection results on a real robot.

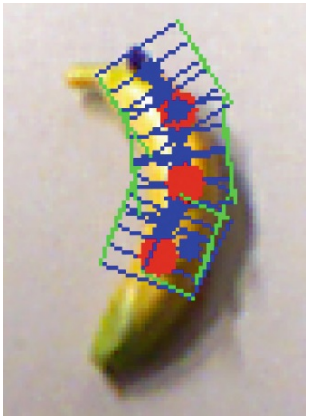


Figure 2.10: Marker based evaluation method [58]

In most of the studies in robotic grasp detection, researchers have proposed the grasp detection as a computer vision problem and presented their results as detection accuracy, mean error of orientation, and the mean jacquard index for predictions. The only conclusion that can be drawn from these results is an evaluation metric for the grasp detection. There are very few approaches that have carried out grasping with real robotic hardware. Although the evaluation metrics draw an acceptable conclusion for the grasp detection, the additional results from actual robotic grasping will further confirm the results from grasp detection.

2.5 Baxter Research Robot Platform

The Baxter Robot shown in Figure 2.11 is a collaborative robot designed and developed by Rethink Robotics in 2012 [28]. Rethink Robotics [28] has designed Baxter with a focus on Industrial Work Safety, Easy Programming, and General-purpose usage as key points for an industrial robotic solution. Baxter is a redundant robot with two 7 degrees of freedom (DOF) humanoid arms. A redundant robot is identified as having more DOF than that required for reaching a given position and orientation in its workspace [49]. There is also a separate platform of the same robot for research and education known as the Baxter Research Robot (BRR). The support documentation provides additional details required in setting up the BRR, setting up a workstation, and the first programming tutorial for the BRR in its wiki [95].

It is desirable to control a robot in an environment in which it will be alongside human colleagues with impact prevention control strategies. This practice is mutually beneficial as damage on impact for humans or the robot are both highly undesirable. Conventional actuators become more stiff over time because of the precise control

Figure 2.11: The Baxter robot in the ECU Robotics Lab

of force. This results in poor joint compliance and results an unsafe work place collaboration with robots. In order to accomplish safety compliance, the Baxter robot has Series-Elastic Actuators (SEA) in each of its joints. Instead of directly coupling the motors to robot joints via gear boxes, the coupling is made through springs which provides the torque by the twists of these springs [11]. This allows a degree of compliance in the joints, meaning that human-robot impacts are significantly less dangerous, and when coupled with accurate force sensing means that the Baxter robot is safe to operate alongside humans.

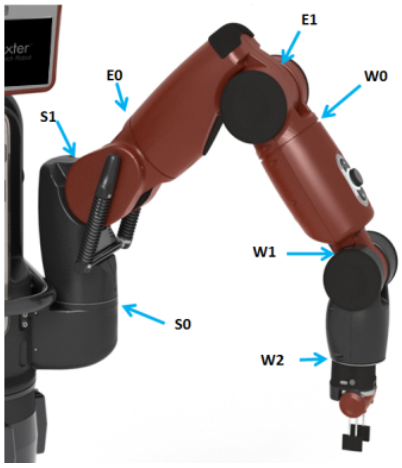


Figure 2.12: Baxter robot joints [96]

The Baxter robot is identified as a humanoid robot having two arms with each arm having 7 DOF. The 7 robotic joints are identified as S0, S1, E0, E1, W0, W1, and W2 respectively as shown in Figure 2.12 [96].

- S0 - Shoulder Roll
- S1 - Shoulder Pitch
- E0 - Elbow Roll
- E1 - Elbow Pitch
- W0 - Wrist Roll
- W1 - Wrist Pitch
- W2 - Wrist Roll

2.5.1 Hardware Specification

The physical specifications [97] of the Baxter robot state that it has a maximum reach of 1210 mm, a maximum payload of 2.2 kg including the weight of the end-effector, and a gripping torque of 4.4 kg. The additional physical specifications are listed in Table 2.3.

Table 2.3: Physical specifications of the Baxter robot [97]

Physical Dimension	**Value**
Robot height	1778 mm - 1905 mm (adjustable)
Maximum Reach	1210 mm
Torso mounting plate diameter	337.82 mm
Weight (Including the pedestal)	138.8 kg
Pedestal foot print	914.4 mm x 812.8 mm
Payload (Including the end-effector)	2.2 kg
Gripping torque	4.4 kg

According to the specification [97], the BRR comes with an on-board computer including an array of sensors: Three cameras, one on its head and one on each of the two arms, sonar sensors on its head, infrared range sensors on its two arms, accelerometer, and navigational cuff buttons. The head also incorporates a display unit with various uses in applications. The components are shown in Figure 2.11. The cameras are shown by red colour rectangles, the infrared range sensor is shown by a green colour rectangle and the display unit is shown by a blue colour rectangle. The computer and sensor specifications are listed in Table 2.4.

Table 2.4: Computer and sensor specifications of the Baxter robot [97]

Specification	**Type or Value**
Computer Processor	Intel Core i7-3770 Processor (8MB, 3.4GHz) with HD4000 Graphics
Memory	4GB, NON-ECC, 1600MHZ DDR3
Storage	128GB Solid State Drive
Camera Max Resolution	1280 x 800 pixels
Camera Effective Resolution	640 x 400 pixels
Camera Frame Rate	30 frames per second
Camera Focal Length	1.2 mm
Screen Resolution	1024 x 600 pixels
Infrared Sensor Range	4 cm - 40 cm

2.5.2 BRR in Research

The BRR has been used in several previous experiments including verification of kinematic models, teleoperation applications, human- robot collaboration and deep reinforcement learning. Ju et al. [11] had verified the kinematic model of the BRR, hoping it would help in further applications requiring a kinematics model for motion control. In a different attempt Ju et al. [2] used this kinematics model in a tele-operation control application for the BRR with leading attempts at tele-controlled industrial robotic solutions.

In [32], Zhang et al. used deep reinforcement learning for training the BRR in a virtual simulation to reach a target location. Having promising results in simulations they further investigated the applicability into physical implementations. The noise levels on the live images, however, had resulted in failure on the reinforcement learning network trained with simulation images [32].

In an attempt to learn effective robotic grasps Lenz et al. [18] used the BRR in their experiments to verify their results. They had used a two cascaded system in optimising the training time and improving the overall outcome [18]. Instead of manually labelling all of the training inputs for possible grasps, the first network would identify possible grasps on the objects using point-cloud data and output the selected grasp candidates. Making this as the input for the second network, it would identify the effective grasp that would have the higher probability of success. In an attempt to learn to interact with humans focusing on their hand locations on the world, Lee et al. [48] had used the BRR with their experiments. Overall, Table 2.5 lists these studies that were conducted based on the research platform of the Baxter robot.

Table 2.5: Studies conducted on the BRR

Authors	**Type of Study**
Z. Ju, C. Yang, and H. Ma	Kinematics modeling [11]
A. Smith, C. Yang, C. Li, H. Ma, and L. Zhao,	Dynamics model [98]
Z. Ju, C. Yang, Z. Li, L. Cheng, and H. Ma	Teleoperation using haptics [2]
H. Reddivari, C. Yang, Z. Ju, P. Liang, Z. Li, and B. Xu	Teleoperation using motion tracking [99]
L. Pinto and A. Gupta	Self-supervised/ unsupervised object grasping [64]
I. Lenz, H. Lee, and A. Saxena	Supervised object grasping [18]
J. Lee and M. S. Ryoo	Human collaboration [48]
F. Zhang, J. Leitner, M Milford, B. Upcroft, and P. Corke	Reinforcement learning for target reaching [32]

There are numerous examples in the recent literature that explore using the BRR as a robotic manipulator in their experiments covering various types of learning problems. Given the preliminary stages of much of this research, it would suggest that there is considerable scope to explore possible applications of deep learning methods with a Baxter Research Robot.

Chapter 3

Methodology

This chapter describes the methodology adopted for the project. Firstly, information regarding the various deep learning software frameworks that were utilised for the project is provided. The hardware requirements and materials involved with the project are then listed, followed by a discussion relating to image pre-processing and the steps involved with the convolutional neural network training. The section is concluded with a detailed discussion of the overall experimental setup.

This project has investigated the use of deep learning methods to train robot systems to be able to manipulate objects in unstructured environments. The use of convolutional neural networks to detect grasps from RGB-Depth (RGB-D) images in order to plan successful grasp manoeuvres that can be carried out by a Baxter robot using inverse kinematics has been explored. Several deep learning software frameworks have been tested for training of the CNN models. Another requirement for training CNNs is access to suitable hardware for accelerated computing. A high-end graphical processor (GPU) is compulsory. An external depth camera is also utilised with the project for acquiring depth images. Image pre-processing is a key step during the network training. Various image processing software libraries have been used for this purpose. Once the training data is ready for the training, neural network training involves several key steps and carefully selection of the optimiser and the cost function to produce optimal results. Once the neural network is trained for a sufficient number of steps, it can then be used to detect grasp rectangles from images. Using inverse kinematics, a joint angle solution can be generated that can be used to move the Baxter robotic arm to grab the object.

3.1 Deep Learning Frameworks

Various software frameworks are used for training of deep learning neural network models in recent literature. According to [100], deep learning software frameworks are developed to provide the capability to design, build, train, and validate neural network models. Some of the popular frameworks are:

- Caffe2
- Cognitive toolkit
- MXNet
- PyTorch
- TensorFlow

Most of these programming interfaces rely on GPU-accelerated libraries such as cuDNN or NCCL to deliver multi-GPU accelerated training.

Initially, the experiments were done with Matlab's Neural Network Toolbox, but after discovering various limitations and licensing issues with Matlab a decision was made to transfer into open-source software. All of the experiments have thus been performed using the Keras [23] Python deep learning libraries with TensorFlow back-end [101] support.

3.1.1 Matlab Deep Learning

Matlab R2017b provides a set of tools specifically for deep learning applications under its *Neural Networks toolbox* [6]. The toolbox introduces deep learning as a machine learning technique referring to the recent state of the art applications made possible with the use of deep learning. The toolbox provides the methods and functions to create and configure a CNN for classification and regression applications. For this purpose, the layers listed in Table 3.1 are provided within the Matlab deep learning toolbox. Table 3.1 also describes what each of the available layers is specialised for. The documentation also provides support for transfer learning by allowing multiple configuration to fine tune parameters with availability of a good selection of the most well known deep learning networks. The documentation further confirms that using the pre-trained networks would allow immediate solutions for small scale deep learning applications [6].

Table 3.1: Supported CNN layers by the Matlab Neural Networks Toolbox [6].

Layer	**Description**
Image input Layer	Defines the size of the input images of a CNN.
Convolutional Layer	Extracts the features from input images. A CNN can have multiple convolutional layers depending on the input data.
ReLU Layer	A nonlinear activation function. It performs the threshold function for all of the elements.
Cross Channel Normalisation (Local Response Normalisation) Layer	This layer performs a channel-wise local response normalisation. It usually follows the ReLU activation layer.
Max and Average Pooling Layers	These layers follow the convolutional layers for down-sampling, hence, reducing the number of connections to the following layers in order to reduce the parameters for learning
Dropout Layer	A dropout layer randomly sets the layer's input elements to zero with a given probability
Fully Connected Layer	It connects neurons of the layer with the neurons of the previous layer in order to combine the features learnt in each layer. It helps in pattern recognition
Softmax and Classification Layers	For classification applications a Softmax and a Classification layer must follow the last Fully Connected layer
Regression Layer	For continuous variable outputs the last Fully Connected layer must be followed by a Regression layer

Transfer Learning with Matlab

The Matlab documentation [102] identifies transfer learning as re-configuring a previously trained network to tailor a specific set of application requirements. It is useful to use pre-trained neural network models for similar learning problems. Figure 3.1 provides an illustration of how input images are processed through neural networks. Matlab also provides the capability to extract features from images. The success of

neural networks relies on large amounts of training data. It is evident that neural networks learn a huge number of features from these images and these learnt features can be leveraged in different applications.

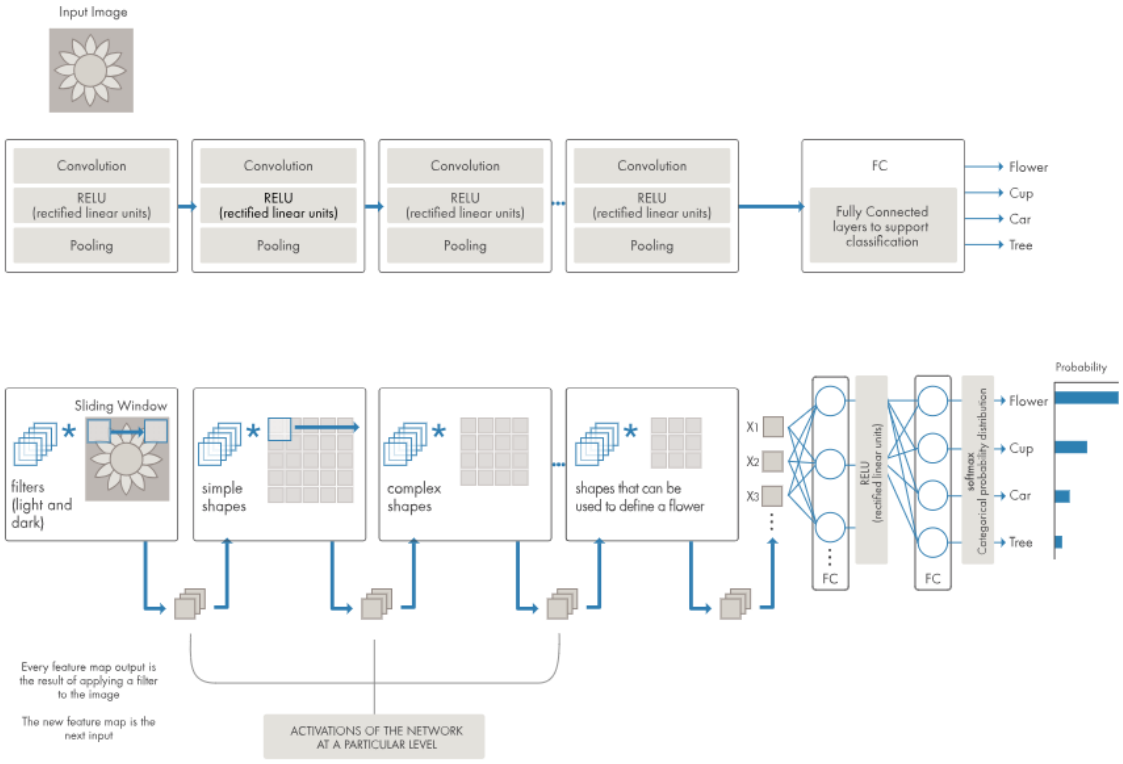


Figure 3.1: An example illustration of how input images are processed through neural networks [102]

The Matlab R2017b provides a limited set of cost functions and learning optimisers. The capability to create custom functions to create other learning optimisers is neither available or the documentation is lacking to describe the process. Therefore, this research project experienced challenges with carrying out the experimentation in Matlab. It was later revealed this capability is available in the newer versions, however, it was challenging to acquire another set of software licenses for a license that was just purchased. Therefore, all the experiments were transferred to open-source as the necessary capability was available as well as the support from the open-source software community. The experiments were then performed with Keras under the TensorFlow backend support.

3.1.2 TensorFlow Deep Learning Backend

Table 3.2: Important TensorFlow backend functions

Function	**Description**
arange()	Creates a 1D tensor containing a sequence of integers
argmax()	Returns the index of the maximum value along an axis
argmin()	Returns the index of the minimum value along an axis
backend()	Publicly accessible method for determining the current backend
batch_flatten()	Turn a nD tensor into a 2D tensor with same 0th dimension
batch_normalization()	Applies batch normalisation on x given mean, var, beta and gamma
batch_dot()	Batchwise dot product
bias_add()	Adds a bias vector to a tensor
categorical_crossentropy()	Categorical crossentropy between an output tensor and a target tensor
concatenate()	Concatenates a list of tensors alongside the specified axis
constant()	Creates a constant tensor
conv1d()	1D convolution
conv2d()	2D convolution
conv3d()	3D comvolution
expand$_d ims$()	Adds a 1-sized dimension at index "axis"
flatten()	Flatten a tensor
floatx()	Returns the default float type, as a string
l2$_n ormalize$()	Normalizes a tensor wrt the L2 norm alongside the specified axis
mean()	Mean of a tensor, alongside the specified axis
pool2d()	2D pooling
pool3d()	3D pooling
pow()	Element-wise exponentiation
relu()	Rectified linear unit
reshape()	Reshapes a tensor to the specified shape
sin()	Computes sin of x element-wise
softmax()	Softmax of a tensor
sqrt()	Element-wise square root
square()	Element-wise square
std()	Standard deviation of a tensor, alongside the specified axis
sum()	Sum of the values in a tensor, alongside the specified axis
tanh()	Element-wise tanh
var()	Variance of a tensor, alongside the specified axis

TensorFlow is an open-source software library for machine learning, specifically for deep learning purposes. TensorFlow performs numerical computations using data flow graphs. The Nodes and Edges of a graph represent mathematical operations and multidimensional data arrays (tensors) respectively. The Google Brain Team's researchers and engineers are responsible for creating TensorFlow, a generalised tool for machine learning applications. It has a flexible architecture which provides cross-platform deployment of numerical computations. The TensorFlow Application Programming Interface (API) enables deployment of computations across one or more CPUs or GPUs in a desktop, server, or a mobile device. Although its preliminary application scope is machine learning or deep learning, the library has a wider application range in Linear Algebra, Image Processing, and Computer Vision [101].

Keras is popular as a model-level Python deep learning library. It provides high-level building blocks for developing neural network models. It does not contain any function to handle low-level operations such as tensor products and convolutions. In this regard it requires a well-optimised tensor manipulation library which is known as the *Backend*. TensorFlow is such an open-source symbolic tensor manipulation backend developed by Google. Some important TensorFlow backend functions [103] are listed under Table 3.2.

3.1.3 Keras: Deep Learning with Python

Keras is a software toolset that enables fast prototyping with neural networks in a modular approach. It is a high-level neural networks API written in Python to use several low-level tensor manipulation backends. Also it has the capability to run convolutional, recurrent, or a combination of neural network models on CPU or GPU accelerated programming environments.

The development team of Keras follows four key guiding principles [94]:

- **User friendliness:** Keras consists of consistent simple APIs and requires a minimum number of user actions to get the required output. Also it provides direct and actionable feedback upon user error.
- **Modularity:** . Keras provides fully-configurable modules that can be plugged together with fewer user actions. All of the neural network layers, cost functions, optimisers, initialisation methods, activation functions, and regularisation methods are all standalone modules that can be combined to create new models.

- **Easy extensibility:** Users can create new program blocks following the existing modules. This enables advanced research into neural networks.
- **Work with Python:** No declaration is required in separate module configuration files. Python is used to describe neural network models. This makes it compact and easier to debug and the source code is easily extensible to advanced programs when necessary.

Currently, there is support for several backends seamlessly integrated into the Keras libraries. The backend support currently available are for: the TensorFlow backend, the Theano backend, and the CNTK backend. In this project, the TensorFlow backend is used as it is the default backend for Keras.

The core data structure of Keras is a model. The simplest type of model is the sequential model, a linear stack of neural network layers. For more complex architectures, it is advised to use the Keras Functional API. A sequential model is created as below:

```
from keras.models import Sequential

model = Sequential()
```

To add neural layers, the add() function is used. The Dense layer is a core layer in Keras. It implements the operation: $output = activation(dot(input, kernel) + bias)$. The *activation* is the element-wise activation function passed as the activation argument. The *kernel* is the weight matrix created by the layer, and the *bias* is the bias vector.

```
from keras.layers import Dense

model.add(Dense(units=32, activation='relu', input_dim=150))
model.add(Dense(units=5, activation='softmax'))
```

Once all the layers are stacked properly, the neural network model is compiled for training as shown below:

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

The network training is initiated by using the $fit()$ function with training data and targets. The following will train the model for 30 epochs with a batch size of 32.

```
model.fit(x_train, y_train, epochs=30, batch_size=32)
```

3.2 Materials

This project involves several hardware components that will be described in this section. Any significant deep machine learning application requires a workstation PC with a high computational power, or access to online cloud computing for neural network training. The workstation PC that was used with this project will be discussed along with some discussion on the trade-offs involved in using local machines compared to cloud services. Some information relating to the utilised Baxter robotic platform and its ROS infrastructure will also be discussed, with attention given to the inverse kinematic solver available with the Baxter software. Another key component of this research is the parallel plate gripper of the robot, and the characteristics of this will be described. For 3D image acquisition, a Microsoft Kinect v2 is modified to interface with the workstation PC. That configuration process is also described in this section.

3.2.1 Workstation PC

Training a deep learning model can take a long time, and some models may even take several days or weeks to complete the training [10, 19]. Using Graphical Processing Unit (GPU) acceleration can speed up the process significantly. It is reported that training neural network models in Matlab with a GPU reduces the time required to train a network [6]. It and can cut the training time for such an image classification problem from days down to hours. When training deep learning models, MATLAB uses GPUs (when available) without requiring an understanding of how to program GPUs explicitly [104]. TensorFlow, however, requires CUDA 7.5 support for GPU-accelerated network training. And it requires CUDA 8.0 if the GPUs use the nVidia Pascal architecture. The accelerated training also requires cuDNN version 5.1 or version 6 depending on the version of TensorFlow. Due to these complexities with TensorFlow, Matlab was selected as the initial platform for the training of the Deep Learning network in this study [105]. However, the very latest TensorFlow versions come with a clear documentation to properly setup the deep learning environment.

The open-source software license was found to make it significantly easier to experiment with different deep learning libraries, therefore all later work was conducted using Keras and TensorFlow.

Table 3.3: Current specifications of the Workstation PC

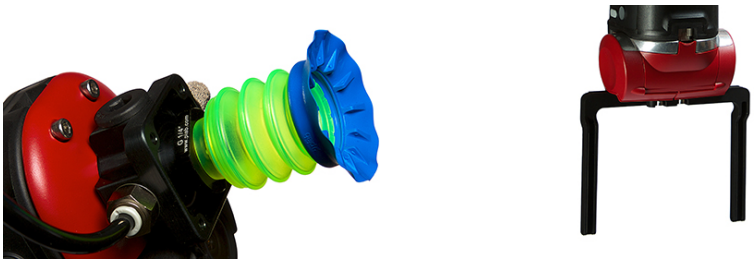
Hardware	**Description**
GPU	NVIDIA GTX GeForce TITAN X (2015) (12GB VRAM)
RAM	DDR4 16GB, 8 extra slots for upgrading
Storage	250GB SSD, 2TB HDD
CPU	Intel Xeon E5, 22 cores/ 44 threads

Recently, the use of cloud computing has appeared in many research works in the field of deep learning. Google and Amazon each have their own cloud GPU computing solutions that can be accessed by paying a premium fee monthly depending on the usage. There have been several reports of the complexities involved with setting up cloud GPU instances and running multiple network training instances per day can multiply the cost involved with the cloud services. As an alternative, researchers suggest the use of a local workstation PC built with a deep learning capable GPU [106]. The most popular GPU manufacturer is NVIDIA and they are the only facilitators of GPU accelerated training with modern deep learning software frameworks. Researchers suggest to use an NVIDIA GTX Titan X series GPU as it is the most capable consumer version available on the market. The current flagship model is the NVIDIA GTX Titan Xp. For research work it is recommended to use at least 2 GTX TItan X GPUs. The next important component is the random access memory (RAM). The minimum required is around 16GB but the researchers recommend at least 32GB of RAM to allow larger batch sizes. For storage of datasets, it is recommended to use a solid sate drive (SSD) as they offer high-speed data reading and writing. Even though the central processing unit (CPU) is not heavily used during the network training, it is recommended have one with a higher thread count as they increase the parallel computing capability of the GPUs. The least acceptable thread count is 8, requiring at least a quad-core CPU. Table 3.3 lists the specifications of the workstation PC used for this work.

3.2.2 Robotic Gripper

A Baxter research robot is used in this project. It is controlled via the Robot Operating System (ROS), an open-source control software for advanced industrial robots. This also provides an inverse kinematics solver that can be used to solve for joint

angles given the end-effector position and orientation vectors. The Baxter robot has 3 cameras but all of them are standard RGB cameras and are not useful with the specific requirement of the project to have the capability to capture RGB-D. The robotic structure is half of a humanoid with dual 7-DOF arms and a head. Each arm can accommodate various end-effectors such as a vaccum suction cup or a parallel plate gripper as shown in Figure 3.2.



(a) Vaccum suction cup (b) Parallel plate gripper

Figure 3.2: Various robotic end-effectors for the Baxter robot: a vaccum suction cup as shown in (a), and a parallel plate gripper as shown in (b).

In this project, the use of the parallel plate gripper is explored. The parallel plate gripper has 4 different gripper plate sets that can be swapped according to the grasping application. The gripper plates shown in the top-right of the image are used as it allows reasonable depth while still allowing the robot to grab the smaller objects involved with the project. The different gripper plates are shown in Figure 3.3.

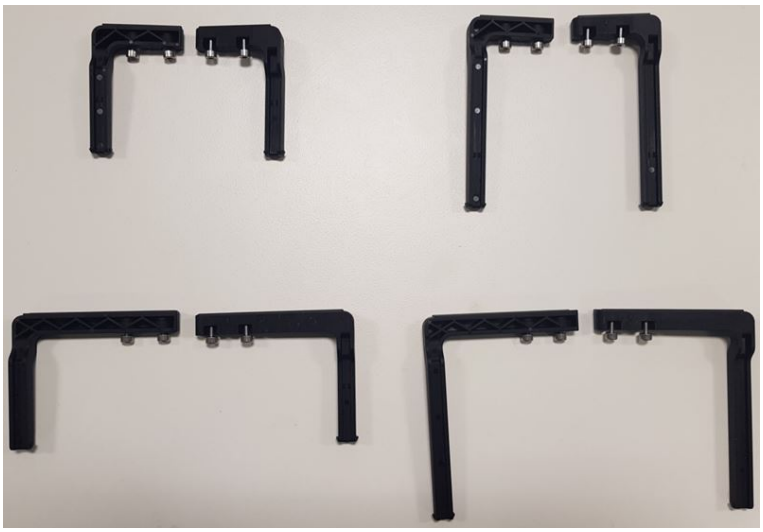


Figure 3.3: Various gripper plates of the Baxter parallel plate gripper. The top-right gripper plate set was used in the project considering the size of the objects involved with the project.

The selected parallel plate gripper has a maximum opening width (w) of 151mm and a gripper plate height (h) of 12mm. The opening width can be configured according to the object dimensions by using different finger positions as shown in Figure 3.4. The stoppers can be used to limit the gripping depth.

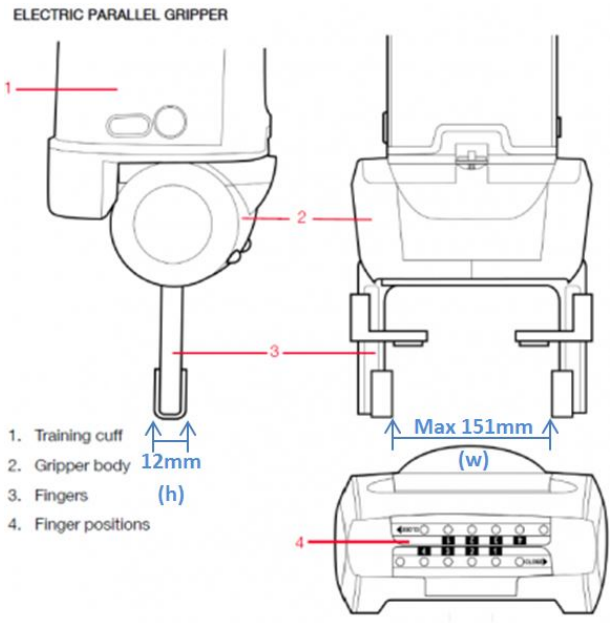


Figure 3.4: Opened gripper dimensions

3.2.3 Microsoft Kinect v2

Microsoft Kinect is a motion sensor that was initially introduced for the Xbox 360 and there-after for the Xbox One as *Kinect v2*. It is a webcam-style input peripheral that allow users to interact with applications in their gaming consoles via their body motion. THe interaction also includes spoken commands and hand gestures. The latest version Kinect for Xbox One (v2) that is shown in 3.5, was released in 2013 alongside the new gaming console. This version has seen much improvements compared to the previous version. The resolution of the camera has been improved to 1920×1080. The infrared vision has been improved to detect changes in the room lighting and adapt accordingly [107].

Figure 3.5: Microsoft Kinect for Xbox One (Kinect v2) [107]

Table 3.4: Microsoft Kinect v2 specifications and requirements.

Property	**Specification**
Colour camera	1920 × 1080 @ 30fps
Depth sensor	512 × 424 @ 30fps
Sensor type	Time of Flight (ToF)
Horizontal/ Vertical angle of view	70°/60°
Range	0.5 - 4.5 (m)
Connection Interface	USB 3.0
Minimum required RAM/ Processor	2GB/ 3.1 Ghz Dual core processor

The Kinect sensor tracks motion and gestures by creating a 3D point cloud of the scene infront of it. Although the colour images can be generated in a higher resolution, the point cloud data has a width of 512 and a height of 424 pixels at a rate of 30 frames per second (fps). Each pixel contains the distance to objects and people in front of the sensor. In addition to the point cloud, the Kinect generates standard coloured images with its RGB camera. By combining coloured images with an additional channel of depth values extracted from the point cloud data, the coloured depth images (RGB-D images) can be generated. Considering the limited availability of computing resources, it was decided to use the RGB-D images with a reduced scale as appropriate to the neural network training. The output of the Kinect sensor images have four channels and they are: {Red, Green, Blue, Depth}. Some specifications of the Kinect v2 is listed in Table 3.4.

3.3 Image Pre-processing

In this project, the Cornell Grasp Dataset is used for the training data. The dataset contains 885 RGB images and point clouds for 240 various objects such as tools, cups, fruit, vegetable, shoes, etc. These images are 640 pixels in width and 480 pixels in height. When the depth and colour images are combined they form 4-channel RGB-D images. However, the neural network model only accepts 224×224 3-channel images as the pre-trained network models are trained on RGB images from large datasets such as the ImageNet [47]. As the point clouds are providing the depth information, a conversion technique is implemented to re-generate the grayscale depth images. In depth images the white value will be increased as the depth increases. The depth values are normalised between [0, 255] from their actual range (0.5, 4.5) m. Some pixels of the depth image have non-numerical values (NaN) due to the occlusion in the stereo image. These pixels are replaced by zeros. Once the grayscale depth images (D

images) are ready, the Blue channel (B channel) of coloured RGB images are replaced by the D images creating the RG-D images. An RG-D image of a whisk is shown in Figure 3.6. Matlab image processing is used to generate these images.

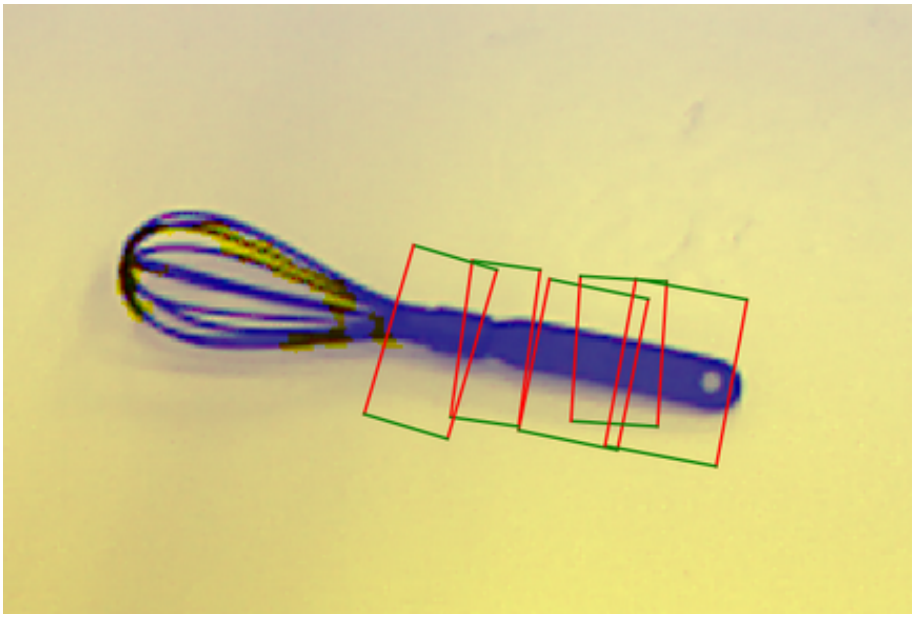


Figure 3.6: A pre-processed RG-D image from the Cornell grasp data showing an image of a whisk; the pixels with higher depth values are in blue, and the grasp rectangles are overlaid.

It is visible from the RG-D image that by incorporating depth information in this way, it is possible to easily visualise the object area or specific regions of interest for grasping. From these RG-D images, a centre crop of 400×400 is extracted around the grasp centre and it is then re-scaled to 224×224. During the neural network training all the pixel values are normalised between [0,1] to reduce the initial loss occurrence. While OpenCV libraries have been used with RG-D image generation, all of the image manipulation tasks such as image reading, cropping, and re-scaling are carried out with the Python Pillow image processing libraries. The image pre-processing process is visualised in Figure 3.7.

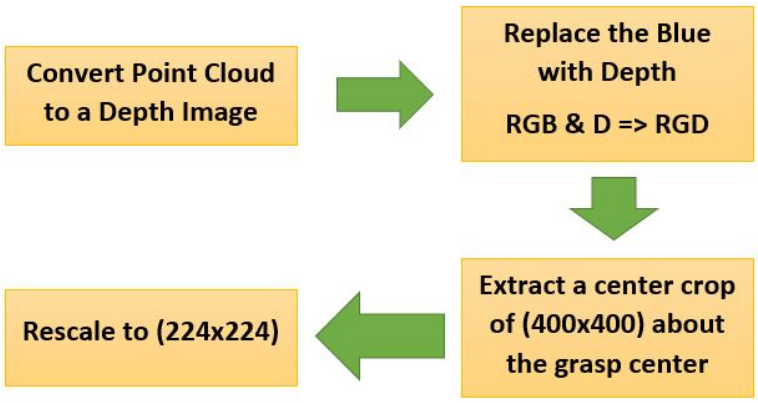


Figure 3.7: Image pre-processing flowchart

Image Compression

The images are stored as numerical Python arrays once they are read from the raw files. When a substantial number of images are read it creates a large array, sometimes making it impossible to proceed within the limited computing resources available. Therefore, it is convenient to use compression techniques in order to read a large amount of training data. The HDF5 algorithm is one such technique to compress data. The HDF5 for Python library is H5PY and it is available at: `https://www.h5py.org`. These arrays are stored in storage drives as opposed to storing them in the RAM. The library allows easier manipulation of HDF databases similar to numerical Python array manipulation. The HDF files can contain different datasets and folders. The folders are referred to as groups in HDF files. All the image and grasp labels were read into Python arrays and after pre-processing with batch processing, the image arrays were stored in a dataset named "training data" within an HDF file. When necessary this HDF file was accessed to import the training data into the neural network training program. This resulted in a reduced delay at the start of the network training by avoiding the need to access the hard drive to read raw image data every time the network training is initiated. Even though this has not resulted in a significant improvement in this study, with extended datasets such as [84], the HDF5 algorithm can be noticeably helpful.

3.4 CNN Training for Grasp Detection

The workstation is installed with Ubuntu Linux 16.04 LTS. TensorFlow was installed with CUDA enabled GPU support under its own Python virtual environment. Then

Keras was installed to use the TensorFlow backend. The scikit learn and pillow Python libraries were used for the image pre-processing requirements. Pre-training is a necessary step when domain specific data is limited to avoid overfitting. Pre-training the network models on larger datasets such as ImageNet [47] allows for broad training of the first few layers of a network model, which are considered to learn high-level features. This intuitively reduces the number of parameters to be trained during the more application specific training stage, while also reducing the susceptibility to overfitting due to the limited availability of training data. The *Keras Applications* module provides pre-trained ResNet50 network models that can be used for this project. Only the convolutional part of the pre-trained model is used. This convolutional base provides a set of universal filters for domain specific data while the application specific fullyconnected layers extract the much lower-level features that are important for grasp prediction. The weights of the convolutional base are fixed with pre-trained weights during the end-to-end training.

3.4.1 Data for Training, Validation, and Testing

Brownlee [45] identified that there were three different learning approaches in machine learning according to the availability of data and further categorised learning algorithms into three categories as explained before: supervised, unsupervised, and semi-supervised.

The effective accuracy of a learning algorithm relies heavily on how extensively it was trained. The outcome accuracy of any learning algorithm depends mostly on the following three factors [17, 79]:

- Data for training
- Architecture of the algorithm
- Self-exploration capacity of the algorithm

In an online article, Brownlee [108] explained the three different categories of training data. They broke down the training data into *training data*, *testing data*, and *validation data* and further explained the following terms as mentioned below [108]:

1. **Training data**: Sample of the data that was used to fit the neural network model.

2. **Validation data**: Sample of data that was used to evaluate the network model that was fit using the training data while tuning model hyper-parameters.

3. **Testing data**: Sample of previously unseen data during the training that was used to evaluate the model.

Goodfellow et al. [17] stated that the performance of a simple machine learning algorithm could rely on the input data representation. Furthering their statement they explained that for an AI system that was intended to diagnose a patient, the system would not have enough information to perform the diagnosis if the only input for the AI system is a MRI (Magnetic Resonance Imaging) scan image. There should also be enough guidelines to identify which type of data and what kind of data labels are represented by this particular input [17]. In similar terms Tobin et al. [14] stated that it is important to have a large volume of domain specific data to fit a model in any learning algorithm and proposed a method to use physics simulations to generate domain specific images of a particular set of generic object shapes to use in a generalised robotic grasping application. Preparing such a large database, however would require a tedious amount of human annotations. In this case, Mahler et al. [34] suggested to populate a dataset containing physics based analyses such as caging, grasp wrench space (GWS) analyses and grasp simulation data for different types of object shapes and poses. They further suggested that cloud computing could be leveraged to train a convolutional neural network with this dataset that would in turn, predict a robustness metric for a given grasp instead of directly predicting a grasp [34]. The proposed dataset was called *Dex-Net 2.0* and contained about 6.7 million point clouds and analytic grasp quality metrics with parallel-jaw grasps planned using robust quasi-static GWS analysis on a dataset of 1,500 3D object models [34].

In this project, there are 885 image instances available from the Cornell grasp dataset. Given that one valid grasp is selected for each image as the ground truth data the number of total available data instances become 885. And the first 100 images are allocated for the validation data, the next 35 images are allocated for the testing data, and the rest of the 750 image instances are allocated to the main training data portion.

3.4.2 Network Model

Even though the proposed ResNet50 network is much deeper compared to the standard feed-forward networks, it requires less computing capabilities for training due to the presence of the residual learning. The skip connections also reduce the number of

learn-able parameters resulting in quicker training times. The pre-trained ResNet50 is complemented with our own Fully-Connected shallow network for the detection purposes. After flattening the output from the ResNet50, a fully-connected layer is attached with 512 units, and it is followed by a 0.5 drop-out factor. A rectifier linear unit is used for activation. The output layer does not have an activation as this is a continuous variable prediction application. The complete network model is shown in Figure 3.8.

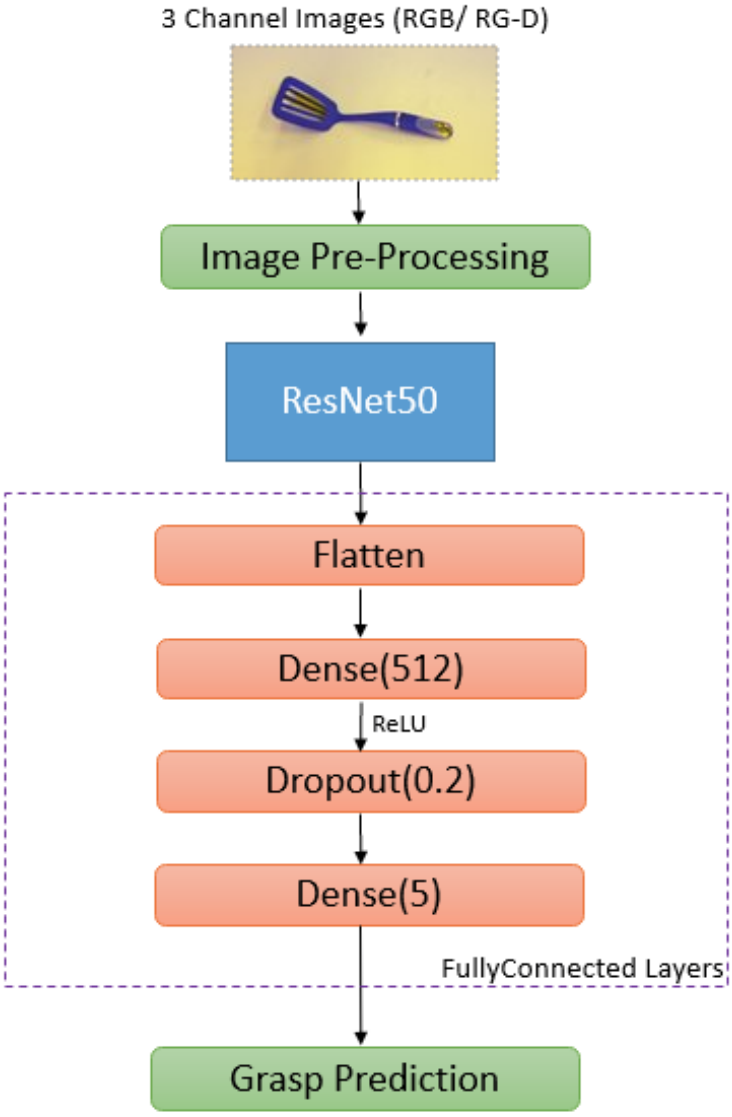


Figure 3.8: Proposed CNN network model

The network model is compiled with the Adam optimiser algorithm. The name Adam is derived from adaptive moment estimation. The optimisation algorithm is

required to find the direction of the weight updates in order to reduce the loss and it decides how fast a network model is completely trained to converge to a lower loss value. In contrast to the classical optimiser algorithm, Stochastic Gradient Decent (SGD), which maintains a constant learning rate throughout the weight updating, Adam maintains different learning rates for updating the weights of the parameters during the training and the learning rate is separately adapted as the training unfolds. The Adam optimiser is known to converge faster and it also has factors to control the learning rate decay whereas the SGD has no learning rate decay option.

The ResNet50 is a 50 layer convolutional neural network that produced state-of-the-art image detection results in 2015 with the ImageNet competition. The complete ReseNet50 neural network model is shown in Figure 3.9.

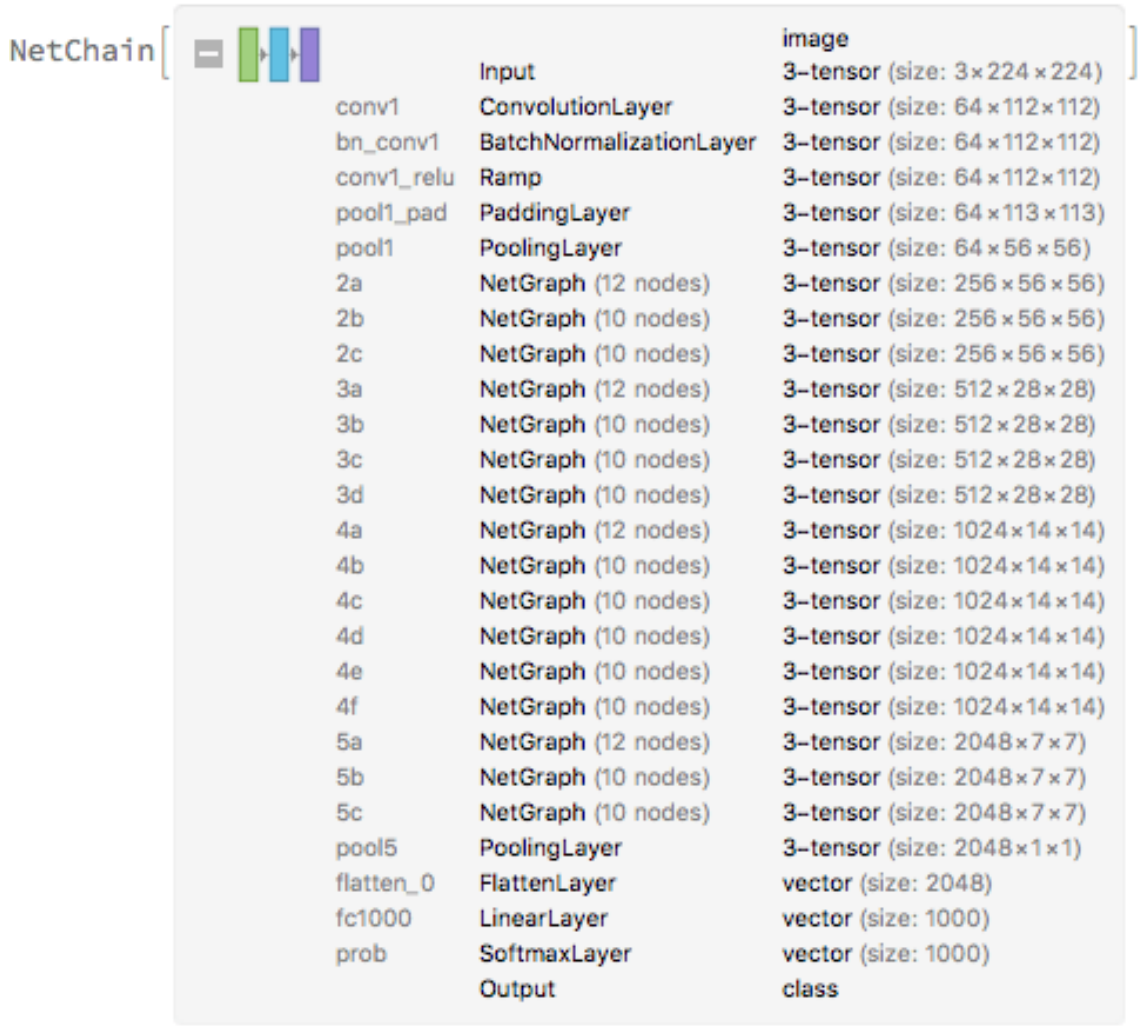


Figure 3.9: Complete ResNet50 CNN Architecture [109]

3.4.3 Vector Regression

We propose a direct regression method to predict the grasp coordinates directly from the raw RGB-D images. Large scale image datasets only contain RGB images, thus the networks are limited to pre-training with 3 channel data. Therefore, the raw RGB-D data is converted to 3 channel RG-D images by replacing the Blue channel with Depth values. As a baseline we have used RGB images during the training removing the depth channel information altogether. These 3 channel images are then fed to the convolutional overhead to extract features. The extracted features are then sent through the fully-connected layers to extract the predictions which produces a five element vector. The five elements are the location and the lengths of the grasp rectangle along with its orientation. Therefore, this method is identified as the five element vector regression. This model is trained with the strong assumption that each image contains only one grasp label. This step is enforced to avoid the need for the sliding window approach. During the training each image is mapped with the first ground truth grasp to avoid the averaging effect during the prediction stage. In the training stage, the weights of the convolutional overhead are kept fixed while the weights of the fully-connected layers are randomly initialised.

3.4.4 Training Evaluation

In this research work, the root mean squared error function that is given by

$$rmse = \sqrt{mean((y_{pred} - y_{true})^2)} \tag{3.1}$$

is used to calculate the prediction loss in each training iteration.

There are two different types of evaluation metrics that can be used in this research context. In previous work authors have argued that the **Rectangle metric** evaluates the regions and the orientations of the predictions whereas the **Point metric** only evaluates the regions of the predictions. Since it is important to evaluate the orientation of the grasp predictions in our work, we use the rectangle metric in validating the predicted grasps from our DCNN. The rectangle metric validates a grasp if both of the following conditions are met:

1. Grasp angle difference between the orientation of the predicted and the ground truth grasps is less than 30°

2. Jacquard index given by

$$J(G_{pred}, G_{true}) = \frac{|G_{pred} \cap G_{true}|}{|G_{pred} \cup G_{true}|} \tag{3.2}$$

is greater than 25%.

The rectangle metric follows an effective validation method compared to the point metric for suitable grasps and it is much closer to the metrics used in object detection other than having a reduced threshold at 25% for the Jacquard index.

Initially, we have used five-fold cross validation with **Image-wise splitting** and **Object-wise splitting**. However, this did not show the expected improvement in results. We believe that this is due to the presence of objects with similar shapes but different colours such as different sunglasses or different kitchen utensils in various colours. Considering the limited variations in the data instances, we have opted for one-fold cross validation as in [58].

3.5 Experimental Setup

The Baxter research robot [28] has been used to demonstrate the robotic grasping capability of the trained model. The Baxter is a humanoid robot with dual 7 degrees of freedom arms with the capability to manipulate objects in human-like fashion. Each arm and the head have attached cameras. The complete robotic system is controlled through its ROS infrastructure. The software development kit also provides an inverse kinematic solver for the Baxter robot, which can be easily accessed to control the Baxter arms to pick objects up once their grasp points are recognised. A Microsoft Kinect v2 sensor mounted on top of the work space is used to capture colour and depth (RGB-D) images of the work area. The experimental setup is shown in Figure 3.10.

Figure 3.10: Experimental setup

3.5.1 The Grasping System

The complete grasping mechanism starts with the Microsoft Kinect v2 capturing a coloured RGB image and a depth mask for each scene with an object on the table that is intended for grasping. Then the captured images will be combined together to create coloured depth (RG-D) images. The Blue channel of colour images will be replaced by the depth mask. The depth mask is a grayscale image that contains values between [0, 255] to represent how close or far away the objects that are viewed from the Kinect v2 camera are. The combined RG-D images will then be sent through the trained convolutional neural network (CNN) and the grasp parameters (x, y, θ, h, w) are then predicted. These parameters represent an oriented rectangle on the image plane with a centre point at (x, y), a height (h), and a width (w). The rectangle is oriented to the horizontal axis (x-axis) by an angle of (θ). The grasp parameters are then fed to the inverse kinematic solver to re-map them into the world coordinates in order to find an acceptable robotic joint solution that can be reached by the Baxter robot for object grasping. After the object has been successfully placed at the drop-off zone, the robotic arm will be reset for the next object. This process is shown in Figure 3.11.

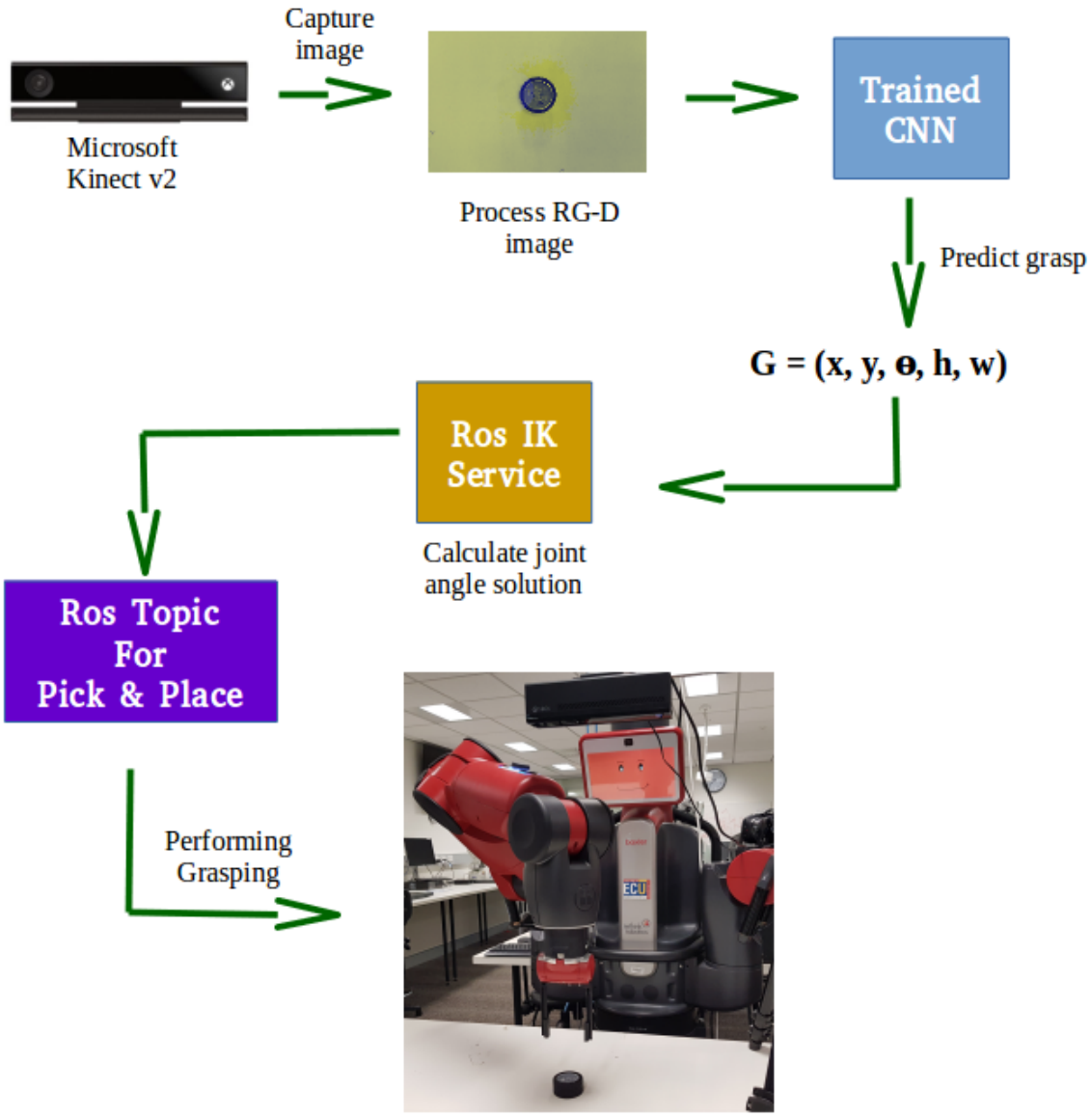


Figure 3.11: The complete system for robotic grasping with deep learning based grasp detection.

3.5.2 Image Acquisition

The Microsoft Kinect v2 requires a proprietary interfacing hardware to properly connect with a computer before retrieving images. It is understood that the standard interface that is accepted for the game consoles is a customised version of the USB 3.0 protocol. Therefore a custom interfacing method was implemented to provide separate 12 VDC power and USB 3.0 communication as shown in Figure 3.12.

(a) Microsoft's proprietary cable (b) Customised connection

Figure 3.12: Manually customised interface for Microsoft Kinect v2

The Kinect requires several software interfacing libraries to read images. As most of the functionality is limited for Windows operating systems, the open-source software community has developed `libfreenect2`, an open-source software interface written in C++ for accessing a Microsoft Kinect v2. Recently, a Python wrapper known as `pylibfreenect2` for this software library was developed and it is available at: `https://github.com/r9y9/pylibfreenect2`. The flowchart in Figure 3.13 shows the image reading process using these software interfaces.

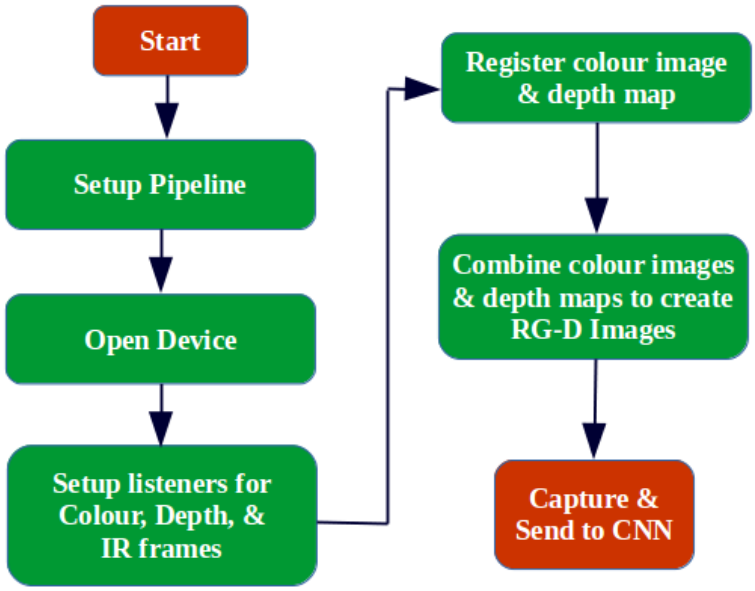


Figure 3.13: Capturing live RG-D images from the Kinect v2

3.5.3 Robotic Pick and Place

Once the Baxter robot control PC was setup with Baxter SDK as described in Appendix A and the Kinect v2 is completely setup, a coordinate mapping is performed to convert the detection results back to the Robot workspace. A $45cm \times 45cm$ workspace area is identified that allows the camera to be at a safe distance without interfering with the Baxter robot arm movements. After a camera calibration, a pixel to workspace distance relationship is found showing 0.2cm per pixel. A ROS topic as described in Appendix B was then developed to enable the right limb of the Baxter robot to be controlled and to use the inverse kinematic solver to determine the joint angle solution for each grasp point detected. Another ROS topic was developed to move the Baxter arm over the object location, orient the gripper, open the gripper, move downward to grab the object, rise 15cm above the table, move to the drop-off point, move downward, and drop the object at the position. After the process the robotic arm will be reset to a standby position for the next pick. The waypoints that are included in the object pick-and-place trajectory are as follows:

1. HOME: Starting position/ Reset position
2. PRE-PICK: 15 cm above the object location with the gripper oriented with the required grasp pose before picking the object
3. PICK: Picking point
4. POST-PICK: 15 cm above the object location after picking
5. PRE-DROP: 15 cm above the drop zone before the object drop
6. DROP: Object drop-off point
7. POST-DROP: 15 cm above the drop zone after the drop

This process is shown in Figure 3.14.

Figure 3.14: The sequence of object pick and place with the Baxter robot

3.6 Limitations

This project has the following limitations.

1. The amount of available application-specific data is lower than the optimal amount.

2. The original RGB-D images or the coloured depth images are 4-channel numerical tensors and ideally require pre-trained CNN architectures with the capability to input 4-channel data. However, at the time of writing there is no known pre-trained network architectures that accomodate such input modalities.

3. The Kinect v2, which is the utilised depth camera has a considerably lower depth resolution than anticipated. Therefore, minor variations in depth information are not visible.

4. The open-source hardware drivers for Kinect v2 has limited support from the manufacturer and it has resulted in slower developments for image optimising techniques. This has affected the automatic white-balancing of the camera.

5. The region in the workspace that the Baxter robotic arms can be arranged to in order to achieve a gripper pose that is normal to the surface is smaller than expected. This ultimately results in a work area of $35cm \times 45cm$. This restriction also limits the overall size of the objects that can be grasped. In

this project, only objects that have a width less than 7cm are considered. The weight of the objects must also be less than 2kg as the Baxter arms have a payload capacity of 4kg and 2kg of this are taken up by the gripper weight.

6. On some occasions, the system fails to detect objects under certain variations of light conditions and background colour. In some research studies, the background colour has been subtracted in order to improve the results. However, with this project there was no improvement observed with this approach.

3.7 Budget

Table 3.5 provides detailed information of the expenses associated with the project. The important materials of the project are the workstation PC for deep learning, data for training, the Baxter robot, a depth camera, and camera mounts. The speed of the neural network training depends on the performance capability of the workstation PC. In this project, it was decided to use the existing workstation PC available within the ECU robotics lab. The training data is shared under an open-source license with the intention of academic knowledge sharing. Currently, the lab is equipped with a Baxter research robot. All of the software used in the project is open-source and involves no licensing fees.

Table 3.5: Budget Proposal

Item	**Value (AU$)**
Microsoft Kinect v2	$150.00
Data cable for the Kinect v2	$20.00
Power adaptor for the Kinect v2	$50.00
Camera tripod	$250.00
Camera mount for the tripod	$50.00
Tools	$150.00
Total required	**$670.00**

3.8 Project Timeline

The time-line that is included in Appendix C describes the task-specific work plan against the time period that is spent on each task. The project included the following seven milestones.

1. Formulation of the research topic

2. Identification of the key research questions
3. Inverse kinematic control of the Baxter robot
4. Acquiring of training data and annotations
5. Running the first successful neural network training
6. Evaluation of the trained CNN with grasping and compilation of the results
7. Thesis preparation and submission including review and feedback

Chapter 4

Results and Discussion

The network model was compiled with the Adam optimiser and trained for more than 30 hours until a trained accuracy of 93.91% was achieved. The trained CNN was tested for its accuracy and it was processed through other evaluation metrics to evaluate the grasp detection effectiveness. A mean jacqurd index of 0.6523 was achieved during the training and 0.4356 during the testing. During the evaluation of the trained neural network model an orientation error of $-5.4°$ was achieved during the testing and the training orientation error was $6.78°$. During the physical grasping trials the Lego EV3 brick and the Lego Large motor both scored the maximum success rate of 65% for successfully grasping 13 times out of the overall 20 trials per each object. The minimum success rate, 10% was achieved by the pen through it only being successful in 1 round out of the 20. From the grasp trials it was observed that linearly formed objects with greater than average height above the table were easier to detect through the depth camera as there was a clear overlay of depth information available compared to shallower objects.

4.1 Grasp Detection

Training Process

The network was compiled with the Adam optimiser with an initial learning rate of 0.001, $\beta_1 = 0.99$, and $\beta_2 = 0.9$. The learning rate was plateaued if the training accuracy had not improved within the preceding 10 training epochs. The fully connected layers of the network model were randomly initialised. We followed an end-to-end training process as opposed to [10] as the 2-stage training did not produce effective results for us. We have trained the DCNN with a batch size of 128 for a maximum

epochs of 30. The maximum accuracy of 93.91% was achieved at the 25^{th} epoch. The complete training has taken about 32 hours to approach the 30 epoch milestone.

Testing the CNN

Table 4.1 shows that the trained CNN has achieved a state of the art training accuracy when compared to the previous approaches at a value of 93.91%. The table only includes the accuracies from the image-wise split training data for the previous methods as the one-fold cross validation is employed. It has been concluded previously, based on the evidence of [10] and [19], that the image-wise splitting and the object-wise splitting have not made any difference for the training results other than reducing the variation of the training dataset.

Table 4.1: Accuracy comparison with previous approaches by training on the Cornell Grasp Dataset

Authors	**Algorithm**	**Acc. (%)**
Jiang et al. [57]	Fast search	60.5
Lenz et al. [18]	Sliding window, two stage	73.9
Redmon et al. [19]	One-shot detection, direct regression	84.4
Kumra et al. [10]	Uni modal, ResNet50	88.84
Kumra et al. [10]	Multi modal, ResNet50	89.21
Watson et al. [58]	Direct regression, AlexNet	78.00
Ours	Vector regression, ResNet50 (RG-D)	**93.91**
Ours	Vector regression, ResNet50 (RGB)	78.71

All of the detected valid grasp rectangles were evaluated using the rectangle metric that has been introduced in the previous section. A predicted grasp is to be validated by the rectangle metric. The rectangle metric provides a binary result as either *PASS* or *FAIL* according to the conditions outlined previously. The trained network model has achieved the results shown in Table 4.2. The mean of Jacquard index (J) and the orientation error (θ) are listed for each stage of the network model. The success rate ($S.R.$) which is calculated as

$$\%S.R. = \frac{(J - 25) + (30 - \theta_{error})}{105} \times 100 \tag{4.1}$$

provides an indication of how likely the predicted grasp will be to accurately grasp an object.

Table 4.2: Rectangle metric evaluation of the trained network model

Stage	**J (mean)**	θ (°)	**S.R. (%)**
Training	0.6523	-5.4	61.74
Testing	0.4356	6.78	39.79

Two types of valid grasps were identified during the prediction stage. **True positive** grasp is a predicted grasp that aligns well with the ground truth grasps. Conversely, **False positive** grasp is an acceptable grasp that does not pass the rectangle metric evaluation. Figure 4.1 illustrates a sample of this phenomenon.

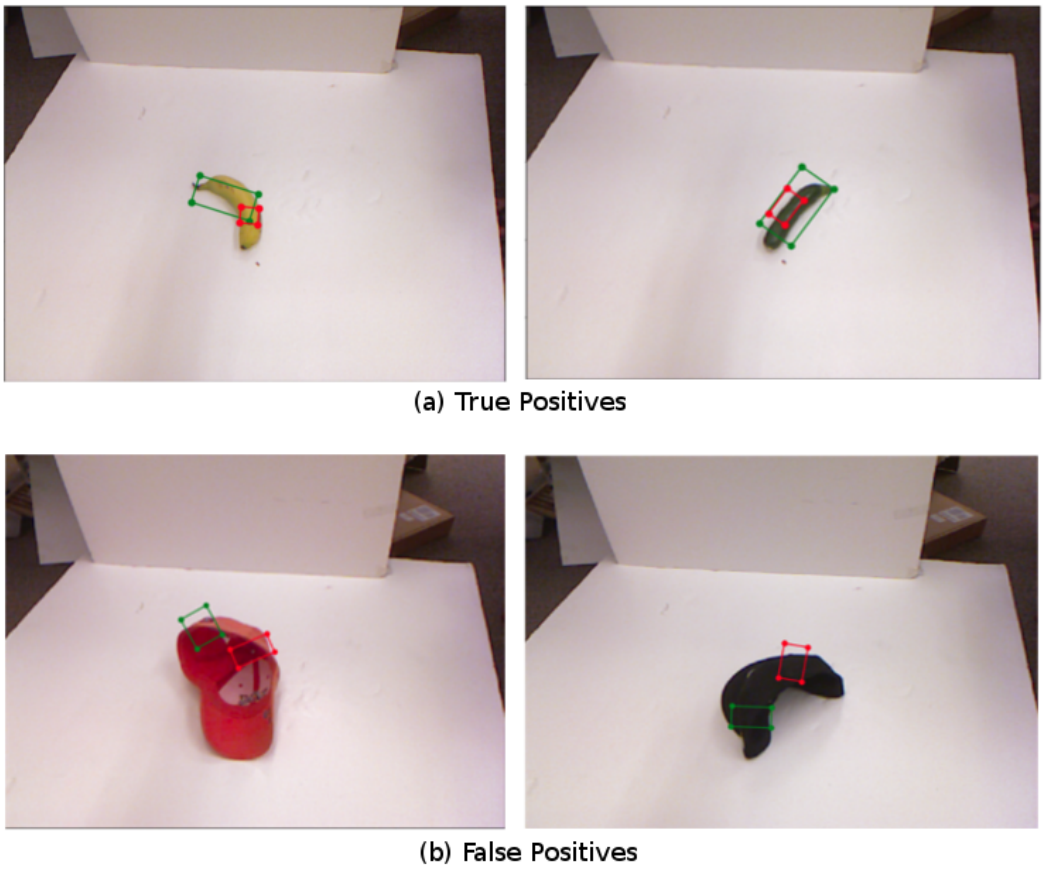


Figure 4.1: Examples of grasps detected from the Cornell Grasp Dataset, showing (a) true positive and (b) false positive grasp predictions. The ground truth grasp rectangles are shown in green while the predicted grasp rectangles are shown in red.

Figure 4.2 illustrates the training progress. From the graph, it is evident that the

training has reached its maximum accuracy of 93.91% towards its final stage while the cost has converged to its minimum value.

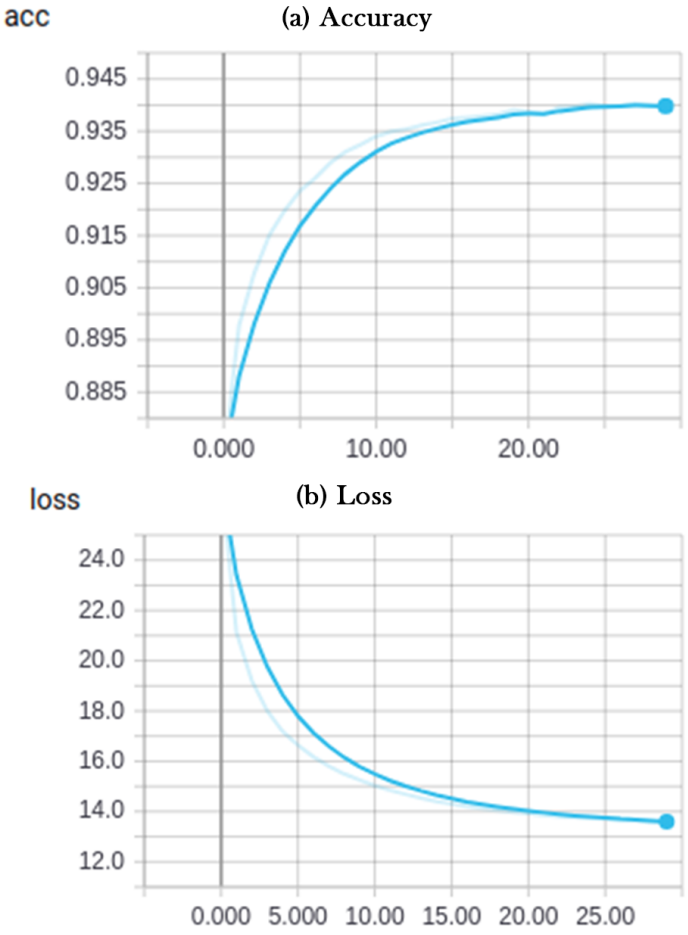


Figure 4.2: Training loss and training accuracy plots

The vector regression network model has processed 448, 000 images per epoch during the training with a mini-batch size of 128 images. Based on the literature, this performance outcome is due to the computing capability of the NVIDIA GeForce GTX Titan X GPU used. This network model is trained using the first grasp from the ground truth grasps to avoid the averaging introduced by [58]. All the ground truth grasps were also tested but the results showed reduced training accuracies supporting the conclusions by Redmon et al. [19].

At the time of this study, pre-trained ResNet50 models are unavailable for 4-channel data. Therefore, we have formed the RG-D training data from RGB-D data to overcome this limitation. Since the original ResNet50 is trained on RGB data,

the features that are learnt from RG-D data would not be the same as those originally learnt from RGB images. The full impacts of this are difficult to conclude without further experiments, but as better training datasets become available further improvements can be expected. Our baseline network model that was trained with RGB images resulted in a final accuracy of 78.71%, which supports the conclusion in [10]. Figure 4.3 shows some valid and invalid grasp predictions from our network.

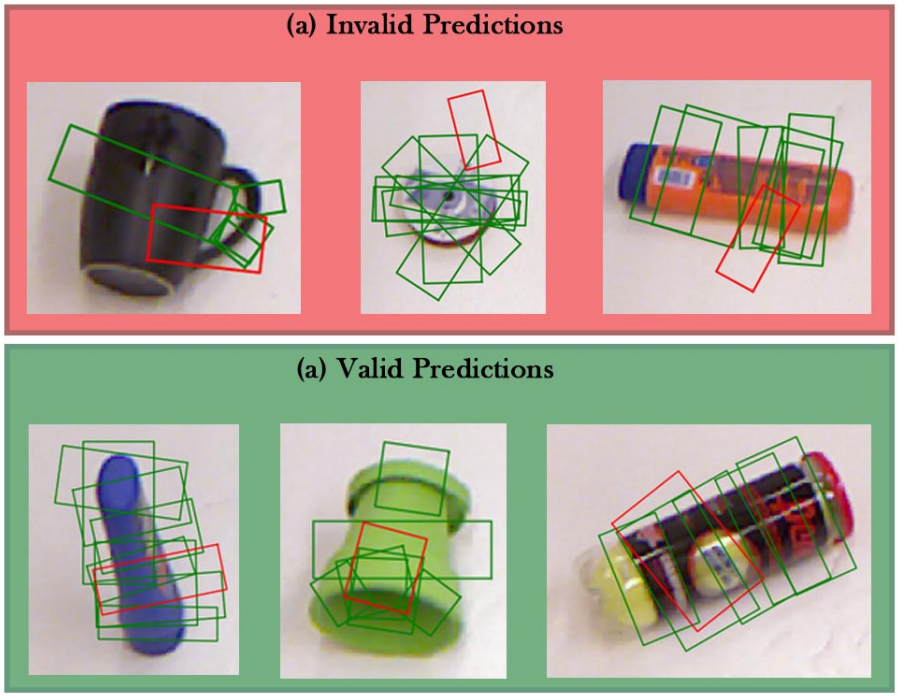
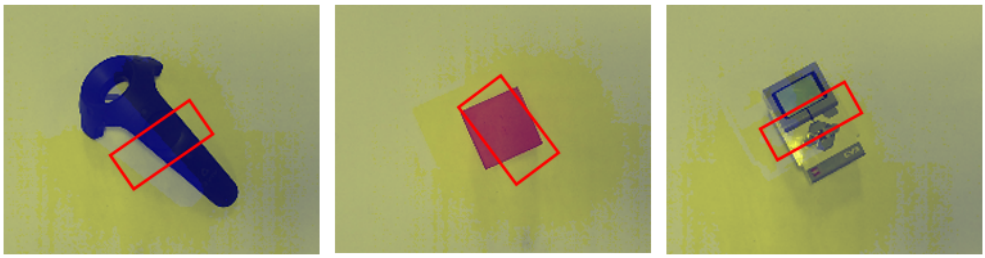
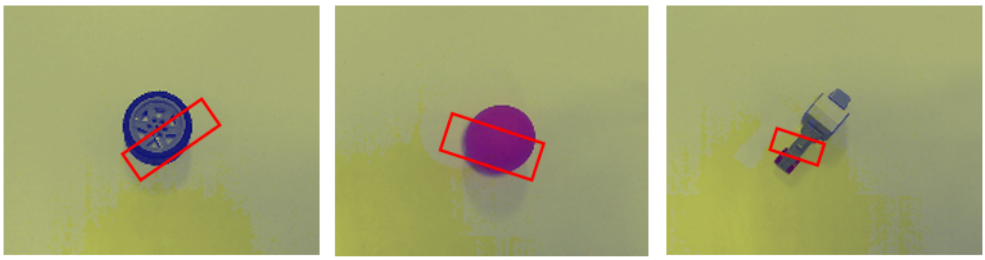


Figure 4.3: Some grasp predictions from our network showing (a) invalid and (b) valid grasp predictions from the images of Cornell Grasp Dataset [38]. Green rectangles show the ground truth grasp rectangles while Red reatcngles show the predictions.

Previous studies have conducted separate ground truth labelling to evaluate the jacquard index and orientation error with live images. A similar step has also been carried out to analyse the results from grasp detection from live images. However, it has been concluded that the ground truth labelling is heavily reliant on semantics and the labels can be biased based on the human understanding of a good grasp. However, there could be numerous other ways to label a valid grasp on an image. Therefore jacquard index would not provide an effective evaluation metric for false positive grasps even though they can be considered as valid grasp predictions. Figure 4.4 shows some instances where grasps were successfully and unsuccessfully achieved.



(a) Successful



(b) Unsuccessful

Figure 4.4: Some grasping instances with live RG-D images showing (a) successful grasps and (b) unsuccessful grasps.

4.2 Physical Grasping Trials

For a physical grasping system, it is difficult to further analyse the results beyond a *PASS* or *FAIL*. In this study, a Baxter robot has been used to grasp objects once their grasp points are identified.

4.2.1 Initial Experiment

In this initial experiment, eight different objects were used for the grasp trials and they are: Lego Mindstorm EV3 Brick, Stress Cube, Robotic Wheel, Lego Large Motor, HTC Vive Joystick, Sunglasses, Rubber Ball. A sample of the objects are shown in Figure 4.5. Each object received 20 trials. Each trial was classified as a *PASS* or *FAIL*.

Figure 4.5: A sample of objects involved with the grasp trials.

The Lego Large Motor and Lego Mindstorm EV3 Brick successfully completed 13 grasps out of the 20 trials. The Pen successfully passed only 1 grasp trial, while failing all the rest of the trials. The Rubber Ball also scored a close to minimum result, passing only 2 trials. It was identified that due to the fixed distance from the camera and the shorter object height of the Pen it was almost undetectable with depth data. A similar issue was observed with the Sunglasses. The spherical shape of the Rubber Ball was challenging to securely grasp between the narrow parallel plate gripper. A similar challenge was observed with the Robotic Wheel due to its cylindrical shape. The grasp trial data are listed in Table 4.3.

Table 4.3: Grasp trials data

Object	**Pass**	**Fail**	**Success Rate (%)**	**Avg. Duration (s)**
Pen	1	19	5	133.2
Rubber Ball	2	18	10	111.4
Sunglass	5	15	25	121.4
HTC Vive Joystick	12	8	60	118.2
Lego Large Motor	13	7	65	122.3
Robotic Wheel	7	13	35	129.5
Cube	11	9	55	119.7
Lego Mindstorm EV3 Brick	13	7	65	120.9

The maximum success rate achieved was 65%, which is close to the theoretical values calculated in the previous section. Figure 4.6 illustrates that the objects that were taller and had straighter edges are more easily grasped with the deep learning based detection system. Almost all trials were completed within 2 minutes with initial grasp detection as low 25 milliseconds.

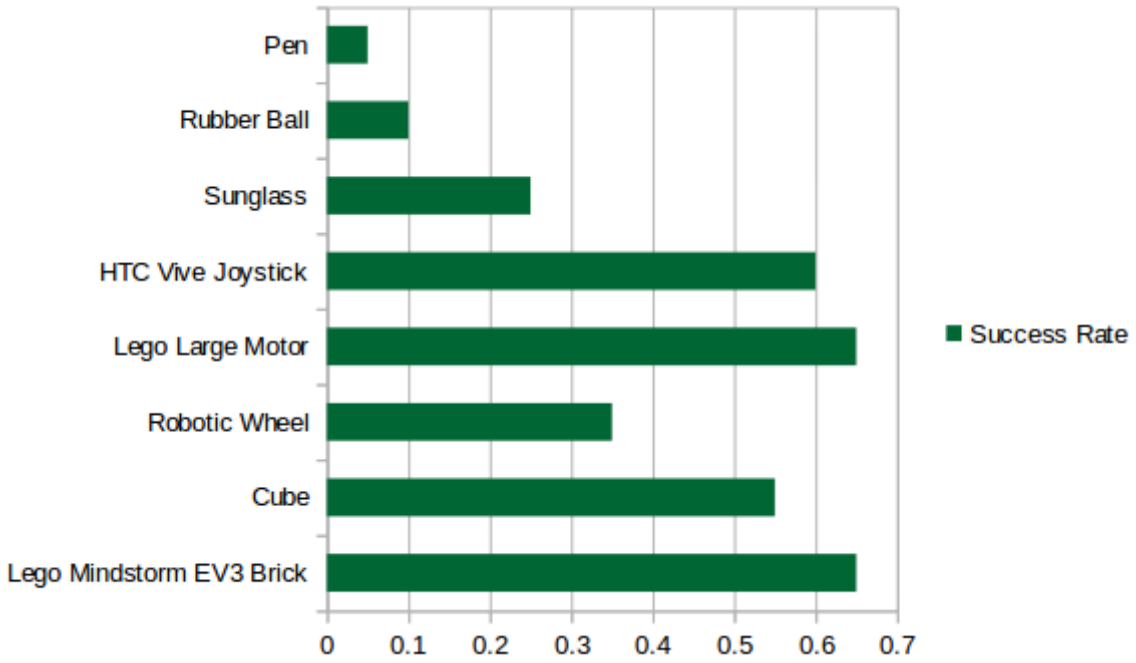


Figure 4.6: Grasp success from 20 trials per object.

4.2.2 Extended Experiment

An additional round of experiments was conducted with an extended set of objects in order to verify the initial results. The object set was included with fifteen different objects such as Aluminium Container, Metal Plate, Combination square, Pliers, Rubber Ball, Whiteboard Eraser, Sunglasses, Wooden wedge, Bottle, Sports Watch, Calculator, Stapler, Highlighter, Pen, Medicine container. Each object received 40 trials. Each trial was classified as a *PASS* or *FAIL*. A sample of the set of objects are shown in Figure 4.7 and they are: aluminium container, combination square, pliers, whiteboard eraser, sunglasses, bottle, calculator, stapler, medicine container, highlighter.

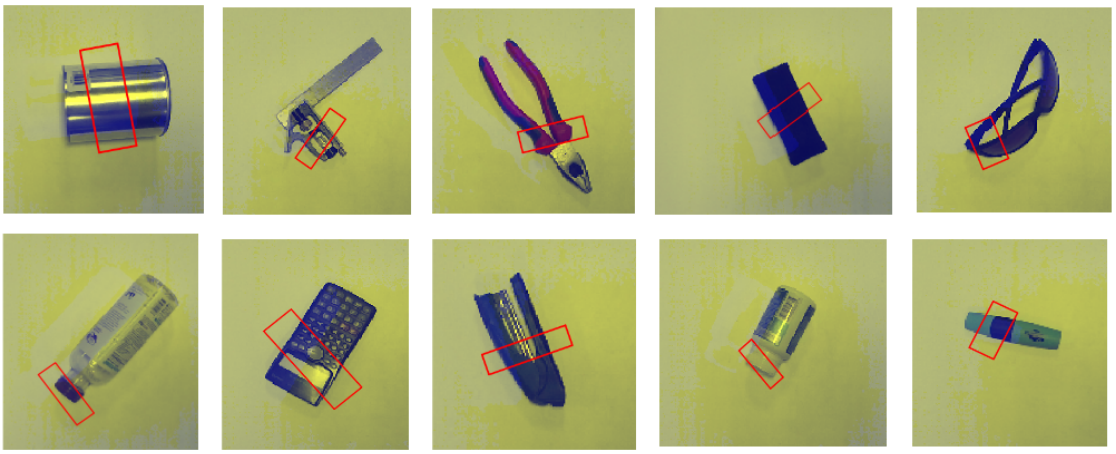
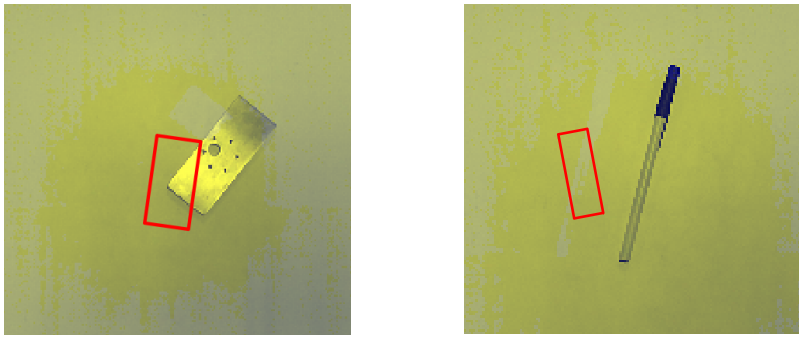


Figure 4.7: A sample of the extended set of objects showing random grasp predictions for an aluminium container, a combination square, a set of pliers, a whiteboard eraser, a pair of sunglasses, a bottle, a calculator, a stapler, a medicine container, and a highlighter

During this extended experiment, it was found that the Pen and the Metallic plate recorded no grasp rectangles at all. Given the fixed distance from the objects recorded a minimum information in the depth images and were undetectable for the neural network model. This is shown in Figure 4.8.



(a) Metallic Plate (b) Pen

Figure 4.8: Visually undetectable objects with minimum depth registration showing the (a) Metallic Plate and the (b) Pen

Although the set of pliers was detected successfully, it was challenging to grasp and hold it securely due to its weight distribution. Only 47.5% of the time it was successfully grasped and held by the Baxter parallel plate gripper. The aluminium container and the sports watch were successfully grasped by increasing the plate width of the Baxter parallel plate gripper. The aluminium container and the sports watch recorded a success rate of 67.5% and 55% respectively. The increment of the plate

width also increased the chances of the rubber ball being grasped successfully. Infact compared to the initial experiment, the rubber ball was successfully grasped 52.5% of the time.

The complete set of grasp success rates are illustrated in Figure 4.9. The maximum grasp rate was 72.5% and it was obtained with the bottle.

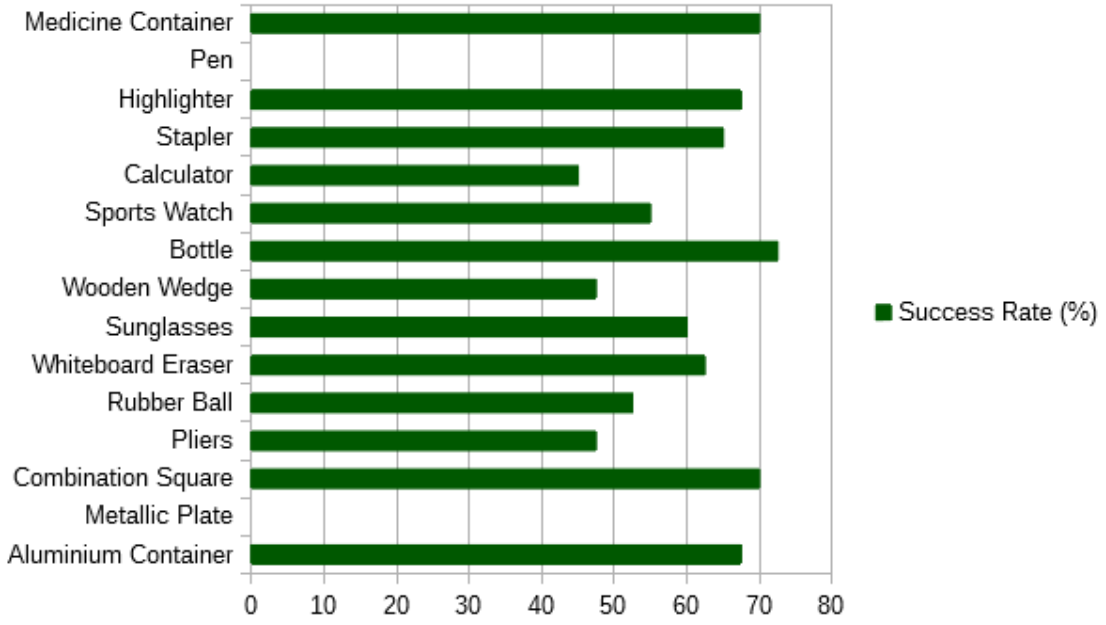


Figure 4.9: Grasp success rates in the Extended Experiment with 40 trials per object.

Chapter 5

Conclusions

This study explores the application of deep learning methods to operate a medium scale general purpose robot, such as the Baxter robot, in an unstructured environment. Recent literature suggests that modelling a robotic solution for the type of problem considered calls for a very tedious process of identifying variables given that the environment is dynamic. The Deep Convolutional Neural Network (DCNN) or the Convolutional Neural Network (CNN) has shown its success in many computer vision problems. In recent years there have been a number of breakthroughs in new CNN applications with promising results. CNNs have been developed that have surpassed the current state-of-the-art in a number of fields including image classification, object recognition, and localisation. The application of variations of these networks have led to recent advancements in robotic grasping applications. This thesis has explored an extension of these recent deep learning based robotic grasping works with an implementation of a layered deep learning network for identifying and localising object grasps from images captured with an external depth camera system to enable a Baxter robot to manipulate objects in unstructured environments.

Grasp pose detection is an important step to enable general purpose robots to manipulate objects that can be found in their dynamic environments. In this project, the use of the one-shot detection methods have been the focus, as these have become the state-of-the-art compared to the sliding window approach that was originally used by Lenz et al. [18]. It is shown that DCNNs can be used to learn features from multimodal data to use with grasp pose detection. This thesis has explored the use of the denser, 50 layer, ResNet CNN architecture to train a grasp pose detection model. The sole challenge for network training was the limited availability of training data. It has been shown previously that residual learning requires less training data as there are a significantly lower quantity of parameters to be trained compared to other architectures. Several other steps were additionally taken to avoid overfitting. These

included pre-training of the ResNet-50 network architecture with ImageNet data and performing heavy image augmentation techniques were performed on the training images. The model generated after training the CNN for more than 30 hours was able to achieve the state-of-the-art detection accuracy of 93.91% with object grasps detected in under 25 milliseconds.

The trained grasp pose detection model was configured to detect the pixel locations of grasp poses in live images and it was mapped to world coordinates. By using an inverse kinematic solver, the world coordinates were converted back into the robot's frame of reference to get the robotic joint space solution. A ROS topic was created to move the Baxter right hand side arm through several waypoints such as HOME, PRE-PICK, PICK, POST-PICK, PRE-DROP, DROP, POST-DROP, and HOME. A set of 8 physical objects were selected as a test set for evaluating object grasping. Each object was allocated with 20 trials. Evaluaiton of object grasping with live images was limited to a **PASS** or **FAIL** metric. More complex evaluation metrics such as the jacquard index and the orientation were omitted as these metrics were heavily reliant on semantics of object grasping and their ground truths. With grasping trials, a maximum success rate of 65% was achieved for physical grasping. It was observed that objects with a greater than average height and a linearly distributed volume are comparatively easier to detect and securely grasp whereas shorter and smaller objects are prone to miss grasp detection. Spherical or cylindrical objects are challenging to grasp with the two-fingered gripper due to their surface curvature when the detected grasp is not accurately centred with the object centres.

Analysis of the results shows that even though there is considerable evidence to prove that the grasp detection is highly accurate, with physical grasping there are several further challenges to real-world grasp success. One of the major reasons is identified to have been a hardware limitation with the depth sensor that was used with the project. The limited availability of the training data was identified as a prevailing challenge of the project. The training dataset only contained about 885 object instances with 240 different objects. With the number of parameters of modern CNN architectures it is necessary to have a reasonably large amount of data to avoid overfitting. Previous research suggested a reasonable number of data instances would be between 10,000 - 1,000,000. But collecting such large-scale datasets is highly challenging as it requires heavy human involvement with annotations and extended hours of access to equipment.

Future Improvements

Going forward, it is worthwhile to extend the experimentation with transfer learning approaches for robotic grasping work as the robotic grasping is one of the primary tasks for a general purpose robotic system. Handling multi-modal data is much more important considering the different sensor types that are available on modern robotic systems. A major bottleneck for this research was the limited availability of training data for robotic grasp pose detection. While pre-training and image augmentation managed to reduce the overfitting, it would be extremely valuable to have a large annotated robotic grasps dataset. Another key challenge with transfer learning is the number of input dimensions. Currently, the ResNet-50 architecture is pre-trained with with 3-channel ImageNet RGB images. Therefore, it can only learn from data with 3 dimensions. In order to use 4-channel RGB-D data with training it is required to pre-train the network with 4-channel data. Once a sufficient dataset is collected and annotated, a ResNet-50 model with 4 input dimensions could be pre-trained to use with 4-channel RGB-D data. Song et al. [83] has effectively used 4-channel RGB-D data with DCNN training by feeding coloured images and depth information in two different CNN architectures and by merging the learnt features in the end. Some attention should also be given to their approach.

Recently, unsupervised learning techniques have emerged in generalised robotic solutions for providing effective results. By combining reinforcement learning as in [64, 84], researchers have automated the large-scale data collection for robotic grasping. Further experiments with such large-scale datasets are expected to perform better compared to the current achievements. In supervised learning techniques, a trained grasp detection model requires a separate path planner to manipulate the robotic arms to successfully grasp the object. Regardless to the collected dataset, the trained neural network trains best to predict grasps that are validated by human annotators. It is commonly identified that the grasp annotations are heavily biased by semantics perceived by the human annotators. In such cases, learning an algorithm for robotic grasping in an unsupervised manner would allow a robotic system to comb through any unpredictable scenario through which an object can be identified and grasped. By combining reinforcement learning, the training can be converged to an optimal model in an effective manner with a sense of right or wrong. However, the trial-and-error step of such methods are heavily limited due to the numerous precautions that are necessary to mitigate the harm and damage to and from the robotic equipment. The release of platforms such as NVIDIA Isaac [26] will accelerate the research with

reinforcement learning by removing the challenges and limitations involved with the trial-and-error approaches by allowing the learning to be carried out in simulated environments with the domain adaptation capability.

References

[1] M. Lopes and J. Santos-Victor, "Visual learning by imitation with motor representations," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 3, pp. 438–449, 2005.

[2] Z. Ju, C. Yang, Z. Li, L. Cheng, and H. Ma, "Teleoperation of humanoid baxter robot using haptic feedback," in *Multisensor Fusion and Information Integration for Intelligent Systems (MFI)*, December 2014.

[3] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *International Journal of Robotics Research (IJRR)*, vol. 31, no. 3, pp. 360–375, 2011.

[4] J. Kober and J. Peters, "Imitation and reinforcement learning," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 55–62, 2010.

[5] J. Peters, D. D. Lee, J. Kober, D. Nguyen-Tuong, A. Bagnell, and S. Schaal, "Robot learning," *Springer Handbook of Robotics* 2^{nd} *Edition*, *pp.*$357 - -394, 2017.$

[6] Mathworks. Deep learning in matlab. Accessed: 2017-04-28. [Online]. Available: https://au.mathworks.com/help/nnet/ug/deep-learning-in-matlab.html

[7] Z. Chen, T. Zhang, and C. Ouyang, "End-to-end airplane detection using transfer learning in remote sensing images," *Remote Sensing*, vol. 10, no. 1, 2018. [Online]. Available: http://www.mdpi.com/2072-4292/10/1/139

[8] I. Rosenberg, G. Sicard, and E. O. David, "End-to-end deep neural networks and transfer learning for automatic analysis of nation-state malware," *Entropy*, vol. 20, no. 5, 2018. [Online]. Available: http://www.mdpi.com/1099-4300/20/5/390

[9] A. Bicchi and V. Kumar, "Robotic grasping and contact: a review," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, 2000, pp. 348–353.

[10] S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 769–776.

[11] Z. Ju, C. Yang, and H. Ma, "Kinematics modeling and experimental verification of baxter robot," in *Chinese Control Conference (CCC)*, July 2014.

[12] A. G. Billard, S. Calinon, and R. Dillmann, "Learning from humans," *Springer Handbook of Robotics* 2^{nd} *Edition, pp.*$1995 - -2014, 2017$.

[13] M. Jeon, "Robotic arts: Current practices, potentials, and implications," *Multimodal Technologies and Interaction*, vol. 1, no. 2, 2017. [Online]. Available: http://www.mdpi.com/2414-4088/1/2/5

[14] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 23–30.

[15] A. Saxena, J. Driemeyer, J. Kearns, and A. Y. Ng, "Robotic grasping of novel objects," in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 1209–1216. [Online]. Available: http://papers.nips.cc/paper/3010-robotic-grasping-of-novel-objects.pdf

[16] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, 2008. [Online]. Available: https://doi.org/10.1177/0278364907087172

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[18] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *Int. J. Rob. Res.*, vol. 34, no. 4-5, pp. 705–724, Apr. 2015. [Online]. Available: http://dx.doi.org/10.1177/0278364914549607

[19] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," *CoRR*, vol. abs/1412.3128, 2014. [Online]. Available: http://arxiv.org/abs/1412.3128

[20] M. Basalla, F. Ebert, R. Tebner, and W. Ke, "Grasping for the real world (Greifen mit Deep Learning)," https://www.frederikebert.de/abgeschlossene-projekte/greifen-mit-deep-learning/, accessed: 2017-02-13.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[22] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 06 2010, oral Presentation.

[23] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[24] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[25] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[26] nVIDIA. Virtual Simulator for Robots. Accessed: 2017-10-30. [Online]. Available: https://www.nvidia.com/en-us/deep-learning-ai/industries/robotics/

[27] Institut für Auslandsbeziehungen (IFA). Collaborative Robots. Accessed: 2017-09-03. [Online]. Available: http://www.dguv.de/ifa/fachinfos/kollaborierende-roboter/index-2.jsp

[28] Rethink Robotics. Baxter collaborative robots for industrial automation. Accessed: 2017-03-02. [Online]. Available: http://www.rethinkrobotics.com/baxter/

[29] CLEANAWAY. Perth MRF. Accessed: 2017-06-20. [Online]. Available: http://www.cleanaway.com.au/contact-us/perth-material-recovery-facility/

[30] Recycleright. Yellow top bins. Accessed: 2017-06-20. [Online]. Available: http://recycleright.wa.gov.au/what-we-do/yellow-top-bins/

[31] A. S. Polydoros, L. Nalpantidis, and V. Krüger, "Real-time deep learning of robotic manipulator inverse dynamics," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, December 2015.

[32] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," in *Australasian Conference on Robotics and Automation*, December 2015.

[33] D. Malowany and H. Guterman, "Visual reinforcement learning for object recognition in robotics," in *IEEE International Conference on the Science of Eletrical Engineering*, Novemebr 2016.

[34] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," *CoRR*, vol. abs/1703.09312, 2017. [Online]. Available: http://arxiv.org/abs/1703.09312

[35] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345 – 1359, 2010.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature: Macmillan Publishers Limited.*

[37] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[38] Cornell University, "Robot Learning Lab: Learning to Grasp," http://pr.cs.cornell.edu/grasping/rect_data/data.php, accessed: 2017-04-12.

[39] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Transaction of the ASME Journal of Applied Mechanics*, vol. 22, no. 2, pp. 215 – 221, 1965.

[40] Rethink Robotics. URDF. Accessed: 2017-05-02. [Online]. Available: http://sdk.rethinkrobotics.com/wiki/URDF

[41] S. Calinon, F. Guenter, and A. Billard, "On learning, representing and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, 2007.

[42] N. Ratliff, J. A. Bagnell, and S. S. Srinivasa, "Imitation learning for locomotion and manipulation," in *IEEE-RAS* $7^{th} International Conference on Humanoid Robots, December 2007$.

[43] E. Gribovskaya, S. M. Khansari-Zadeh, and A. Billard, "Learning non-linear multivariate dynamics of motion in robotic manipulators," *The International Journal of Robotics Research*, vol. 30, no. 1, pp. 80 – 117, 2010.

[44] A. Billard and D. Grollman, "Robot learning by demonstration," *Scholarpedia*, vol. 8, no. 12, p. 3824, 2013, revision #138061.

[45] J. Brownlee. Supervised and Unsupervised Machine Learning Algorithms. Accessed: 2017-07-03. [Online]. Available: http://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[47] Stanford Vision Lab. Imagenet. Accessed: 2017-05-11. [Online]. Available: http://www.image-net.org/

[48] J. Lee and M. S. Ryoo, "Learning robot activities from first-person human videos using convolutional future regression," *CoRR*, vol. abs/1703.01040, 2017. [Online]. Available: http://arxiv.org/abs/1703.01040

[49] Y. Xia and J. Wang, "A dual neural network for kinematic control of redundant robot manipulators," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 31, no. 1, pp. 147–154, Feb 2001.

[50] H. Ding and J. Wang, "Recurrent neural networks for minimum infinity-norm kinematic control of redundant manipulators," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 29, no. 3, pp. 269–276, May 1999.

[51] J. Wang, Q. Hu, and D. Jiang, "A lagrangian network for kinematic control of redundant robot manipulators," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1123–1132, Sep 1999.

[52] Amazon Robotics. Amazon Picking Challenge 2016. Accessed: 2017-07-06. [Online]. Available: https://www.amazonrobotics.com//pickingchallenge

[53] M. Schwarz, A. Milan, C. Lenz, A. Muńoz, A. S. Periyasamy, M. Schreiber, S. Schüller, and S. Behnke, "Nimbro picking: Versatile part handling for warehouse automation," in *IEEE International Conferrence on Robotics and Automation (ICRA)*, May 2017.

[54] J. Johnson, A. Karpathy, and L. Fei-Fei, "Densecap: Fully convolutional localization networks for dense captioning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[55] J. Mahler. Releasing the Dexterity Network (Dex-Net) 2.0 Dataset for Deep Grasping. Accessed: 2017-07-06. [Online]. Available: http://bair.berkeley.edu/blog/2017/06/27/dexnet-2.0/

[56] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis-a survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, April 2014.

[57] Y. Jiang, S. Moseson, and A. Saxena, "Efficient grasping from rgbd images: Learning using a new rectangle representation," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 3304–3311.

[58] J. Watson, J. Hughes, and F. Iida, "Real-world, real-time robotic grasping with convolutional neural networks," in *Towards Autonomous Robotic Systems*, Y. Gao, S. Fallah, Y. Jin, and C. Lekakou, Eds. Cham: Springer International Publishing, 2017, pp. 617–626.

[59] L. Trottier, P. Gigure, and B. Chaib-draa, "Sparse dictionary learning for identifying grasp locations," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2017, pp. 871–879.

[60] D. Park and S. Y. Chun, "Classification based grasp detection using spatial transformer network," *CoRR*, vol. abs/1803.01356, 2018. [Online]. Available: http://arxiv.org/abs/1803.01356

[61] X. Zhou, X. Lan, H. Zhang, Z. Tian, Y. Zhang, and N. Zheng, "Fully convolutional grasp detection network with oriented anchor box," *CoRR*, vol. abs/1803.02209, 2018.

[62] D. Guo, F. Sun, H. Liu, T. Kong, B. Fang, and N. Xi, "A hybrid deep architecture for robotic grasp detection," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1609–1614.

[63] Z. Wang, Z. Li, B. Wang, and H. Liu, "Robot grasp detection using multimodal deep convolutional neural networks," *Advances in Mechanical Engineering*, vol. 8, no. 9, p. 1687814016668077, 2016. [Online]. Available: https://doi.org/10.1177/1687814016668077

[64] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 3406–3413.

[65] U. Viereck, A. Pas, K. Saenko, and R. Platt, "Learning a visuomotor controller for real world robotic grasping using simulated depth images," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 291–300. [Online]. Available: http://proceedings.mlr.press/v78/viereck17a.html

[66] J. Mahler and K. Y. Goldberg, "Learning deep policies for robot bin picking by simulating robust grasping sequences," in *CoRL*, 2017.

[67] R. Calandra, A. Owens, M. Upadhyaya, W. Yuan, J. Lin, E. H. Adelson, and S. Levine, "The feeling of success: Does touch sensing help predict grasp outcomes?" *CoRR*, vol. abs/1710.05512, 2017. [Online]. Available: http://arxiv.org/abs/1710.05512

[68] R. Calandra, A. Owens, D. Jayaraman, J. Lin, W. Yuan, J. Malik, E. H. Adelson, and S. Levine, "More than a feeling: Learning to grasp and regrasp using vision and touch," *CoRR*, vol. abs/1805.11085, 2018. [Online]. Available: http://arxiv.org/abs/1805.11085

[69] A. Murali, Y. Li, D. Gandhi, and A. Gupta, "Learning to grasp without seeing," *CoRR*, vol. abs/1805.04201, 2018. [Online]. Available: http://arxiv.org/abs/1805.04201

[70] S. Caldera, A. Rassau, and D. Chai, "Review of deep learning methods in robotic grasp detection," *Multimodal Technologies and Interaction*, vol. 2, no. 3, 2018.

[71] L. Y. Ku, E. Learned-Miller, and R. Grupen, "Associating grasp configurations with hierarchical features in convolutional neural networks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 2434–2441.

[72] J. Wei, H. Liu, G. Yan, and F. Sun, *Multi-modal Deep Extreme Learning Machine for Robotic Grasping Recognition*. Cham: Springer International Publishing, 2016, pp. 223–233.

[73] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 580–587. [Online]. Available: https://doi.org/10.1109/CVPR.2014.81

[74] F. J. Chu, R. Xu, and P. Vela, "Real-world multi-object, multi-grasp detection," *IEEE Robotics and Automation Letters*, pp. 1–1, 2018.

[75] A. ten Pas, M. Gualtieri, K. Saenko, and R. Platt, "Grasp pose detection in point clouds," *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1455–1473, 2017. [Online]. Available: https://doi.org/10.1177/0278364917735594

[76] Q. Lu, K. Chenna, B. Sundaralingam, and T. Hermans, "Planning multi-fingered grasps as probabilistic inference in a learned deep network," *CoRR*, vol. abs/1804.03289, 2017.

[77] E. Johns, S. Leutenegger, and A. J. Davison, "Deep learning a grasp function for grasping under gripper pose uncertainty," *CoRR*, vol. abs/1608.02239, 2016. [Online]. Available: http://arxiv.org/abs/1608.02239

[78] A. Zeng, S. Song, K. Yu, E. Donlon, F. R. Hogan, M. Bauzá, D. Ma, O. Taylor, M. Liu, E. Romo, N. Fazeli, F. Alet, N. C. Dafle, R. Holladay, I. Morona, P. Q. Nair, D. Green, I. Taylor, W. Liu, T. A. Funkhouser, and A. Rodriguez, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," in *IEEE International Conference on Robots and Automation (ICRA)*, May 2018.

[79] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[80] J. Brownlee, *Deep Learning with Python: Develop Deep Learning Models on Theano and TensorFlow Using Keras*. Machine Learning Mastery, 2017. [Online]. Available: https://books.google.com.au/books?id=eJw2nQAACAAJ

[81] J. Ruiz-del-Solar, P. Loncomilla, and N. Soto, "A survey on deep learning methods for robot vision," *CoRR*, vol. abs/1803.10862, 2018. [Online]. Available: http://arxiv.org/abs/1803.10862

[82] K. Lai, L. Bo, X. Ren, and D. Fox, "A large-scale hierarchical multi-view rgb-d object dataset," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1817–1824.

[83] L. H. Xinhang Song, Shuqiang Jiang, "Combining models from multiple sources for rgb-d scene recognition," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 4523–4529. [Online]. Available: https://doi.org/10.24963/ijcai.2017/631

[84] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018. [Online]. Available: https://doi.org/10.1177/0278364917710318

[85] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. P. Sampedro, K. Konolige, S. Levine, and V. Vanhoucke, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," 2018. [Online]. Available: https://arxiv.org/abs/1709.07857

[86] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," *CoRR*, vol. abs/1704.05588, 2017. [Online]. Available: http://arxiv.org/abs/1704.05588

[87] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[88] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[89] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[90] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 2818–2826.

[91] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," 02 2016.

[92] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng, "Measuring invariances in deep networks," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 646–654. [Online]. Available: http://papers.nips.cc/paper/3790-measuring-invariances-in-deep-networks.pdf

[93] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. [Online]. Available: http://arxiv.org/abs/1609.04747

[94] F. Chollet, *Deep Learning with Python*. Manning Publications Company, 2017. [Online]. Available: https://books.google.com.au/books?id=Yo3CAQAACAAJ

[95] Rethink Robotics. Baxter Research Robot Wiki. Accessed: 2017-07-06. [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Main_Page

[96] ——. The Baxter Robot Arms. Accessed: 2017-07-24. [Online]. Available: http://sdk.rethinkrobotics.com/wiki/Arms

[97] ——. Technical Specification. Accessed: 2017-07-24. [Online]. Available: http://www.rethinkrobotics.com/baxter/tech-specs/

[98] A. Smith, C. Yang, C. Li, H. Ma, and L. Zhao, "Development of a dynamics model for the baxter robot," in *2016 IEEE International Conference on Mechatronics and Automation*, Aug 2016, pp. 1244–1249.

[99] H. Reddivari, C. Yang, Z. Ju, P. Liang, Z. Li, and B. Xu, "Teleoperation control of baxter robot using body motion tracking," in *2014 International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI)*, Sept 2014, pp. 1–6.

[100] NVIDIA Accelerated Computing. Deep Learning Frameworks. Accessed: 2018-08-19. [Online]. Available: https://developer.nvidia.com/deep-learning-software

[101] TensorFlow. About TensorFlow. Accessed: 2017-07-24. [Online]. Available: https://www.tensorflow.org/

[102] Avi Nehemiah. Deep Learning Tutorial Series. Accessed: 2018-08-19. [Online]. Available: https://blogs.mathworks.com/pick/2017/06/02/deep-learning-tutorial-series/

[103] Tensorflow. Keras TF Backend API. Accessed: 2018-08-19. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/backend

[104] Mathworks. What is Deep Learning? Accessed: 2017-08-30. [Online]. Available: https://au.mathworks.com/discovery/deep-learning.html

[105] nVidia. GPU-ACCELERATED TENSORFLOW. Accessed: 2017-08-30. [Online]. Available: https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/tensorflow/

[106] T. S. Gokkulnath, "Choosing components for personal deep learning machine," Nov. 2017, [Accessed: 2018-08-20]. [Online]. Available: https://medium.com/mlreview/choosing-components-for-personal-deep-learning-machine-56bae813e34a

[107] E. Andy, "Hands on: Xbox one kinect," May 2013, [Accessed: 2018-08-20]. [Online]. Available: https://www.techradar.com/reviews/gaming/gaming-accessories/xbox-one-kinect-1153962/review

[108] J. Brownlee, "What is the Difference Between Test and Validation Datasets?" https://machinelearningmastery.com/difference-test-validation-datasets, accessed: 2017-12-03.

[109] Wolfram, "Resnet50 architecture," Jun. 2018, [Accessed: 2018-08-21]. [Online]. Available: https://resources.wolframcloud.com/NeuralNetRepository/resources/ResNet-50-Trained-on-ImageNet-Competition-Data

Appendix A

Upgrade the Baxter Robot SDK

This chapter walks through the upgrading procedure for updating and installing Baxter robot software development kit (SDK) in Ubuntu Linux 16.04 LTS. Standard SDK documentation only provides instrtuctions to setup Ros Indigo in Ubuntu 14.04. This operating system version is old to handle most of the modern deep learning drivers provided by NVIDIA.

Install ROS Kinetic

The complete desktop version of ROS Kinetic was installed following the official instructions from here: `http://wiki.ros.org/kinetic/Installation/Ubuntu`.

Install Latest Baxter SDK

1. A catkin workspace was created by running the following commands in the terminal:
   ```
   mkdir -p ~/baxter_ws/src
   cd  /baxter_ws/src
   catkin_init_workspace
   ```

2. Build and install the Baxter workspace
   ```
   cd ~/baxter_ws
   catkin_make
   ```

3. Setup environmental variables
   ```
   echo "source ~/baxter_ws/devel/setup.bash" >>  /.bashrc
   ```

4. Install Baxter SDK dependencies
   ```
   sudo apt-get update
   sudo apt-get install git-core
   python-argparse python-wstool
   pythonvcstools
   ```

```
python-rosdep ros-kinetic-control-msgs
ros-kinetic-joystickdrivers
```

5. Install Baxter SDK packages
```
cd ~/baxter_ws/src
wstool init
wstool merge https://raw.githubusercontent.com/RethinkRobotics
/baxter/master/baxter_sdk.rosinstall

wstool update
cd ~/baxter_ws
catkin_make
catkin_make install
```

6. Configure Baxter shell
```
cp ~/baxter_ws/src/baxter/baxter.sh ~/baxter_ws
chmod +x baxter.sh
```

7. Open `baxter.sh` file in a text editor and edit the following:
```
your_ip="192.168.XXX.XXX"
ros_version=kinetic
```

Appendix B

Source Code: ROS Topic for Pick and Place with a Baxter Robot

Prepare the Baxter robot for Object Grasping

```
rosrun deep-grasping enable_baxter_pnp.py -l right
```

```
#!/usr/bin/env python
import argparse
import struct
import sys
import time

import rospy
import baxter_interface
from baxter_interface import RobotEnable
import cv2
import cv_bridge

from geometry_msgs.msg import (
    PoseStamped,
    Pose,
    Point,
    Quaternion,
)
from std_msgs.msg import Header

from baxter_core_msgs.srv import (
    SolvePositionIK,
    SolvePositionIKRequest,
)
```

```
from sensor_msgs.msg import (
    Image,
)


def ik_solution(limb, p_x, p_y, theta, depth):
    """
        Object Pick and Place script
        by Shehan Caldera(shehancaldera@gmail.com)

    Uses IK solver service client and baxter_interface classes.

        The following arguments are required:
        '-l' [left or right],
        '-x' [x-coordinate],
        '-y' [y-coordinate],
        '-th' [orientation around z-axis]

    """
    ns = "ExternalTools/" + limb + "
␣␣␣␣/PositionKinematicsNode/IKService"
    iksvc = rospy.ServiceProxy(ns, SolvePositionIK)
    ikreq = SolvePositionIKRequest()
    hdr = Header(stamp=rospy.Time.now(), frame_id='base')
    poses = {
        'left': PoseStamped(
            header=hdr,
            pose=Pose(
                position=Point(
                    x=0.58,
                    y=0.18,
                    z=0.10,
                ),
                orientation=Quaternion(
                    x=-0.13,
                    y=0.99,
                    z=0.01,
                    w=0.02,
                ),
            ),
        ),
        'right': PoseStamped(
            header=hdr,
            pose=Pose(
```

```
                position=Point(
                    x=p_x,
                    y=p_y,
                    z=0.0 + depth,
                ),
                orientation=Quaternion(
                    x=0.0,
                    y=0.1,
                    z=0.0,
                    w=0.0,
                ),
            ),
        ),
    }

    ikreq.pose_stamp.append(poses[limb])
    try:
        rospy.wait_for_service(ns, 5.0)
        resp = iksvc(ikreq)
    except (rospy.ServiceException, rospy.ROSException), e:
        rospy.logerr("Call␣for␣IK␣service␣failed:␣%s" % (e,))
        return 1

    limb_joints = 0

    # Check if result valid, and type of seed
        ultimately used to get solution
    # convert rospy's string representation of uint8[]'s to int's
    resp_seeds = struct.
    unpack('<%dB' % len(resp.result_type), resp.result_type)
    if (resp_seeds[0] != resp.RESULT_INVALID):
        seed_str = {
            ikreq.SEED_USER: 'User␣Provided␣Seed',
            ikreq.SEED_CURRENT: 'Current␣Joint␣Angles',
            ikreq.SEED_NS_MAP: 'Nullspace␣Setpoints',
        }.get(resp_seeds[0], 'None')

        print("SUCCESS␣-
␣␣␣␣␣␣␣␣␣␣␣␣Valid␣Joint␣Solution␣Found␣from␣Seed␣Type:␣%s" %(seed_str,))

        # Format solution into Limb API-compatible dictionary
        limb_joints =
            dict(zip(resp.joints[0].name, resp.joints[0].position))
```

```
        print "\nIK␣Joint␣Solution:\n", limb_joints
        # print "------------------"
        # print "Response Message:\n", resp
    else:
        print("INVALID␣POSE␣-␣No␣Valid␣Joint␣Solution␣Found.")

    return limb_joints


if __name__ == '__main__':
    arg_fmt = argparse.RawDescriptionHelpFormatter
    parser = argparse.
    ArgumentParser(formatter_class=arg_fmt,
                                   description=ik_solution.__doc__)
    parser.add_argument(
        '-l', '--limb', choices=['left', 'right'], required=True,
        help="the␣limb␣to␣use␣with␣object␣pickup"
    )
    args = parser.parse_args(rospy.myargv()[1:])


    rospy.init_node("Enable_Baxter_PnP")
    RobotEnable().enable()

    # Change the Baxter Face
    baxter_display_face
    = cv2.
    imread("path_to_face_image.jpg")
    baxter_display_face_rosmsg
    = cv_bridge.CvBridge().
    cv2_to_imgmsg(baxter_display_face, encoding="bgr8")
    head_display = rospy.Publisher('/robot/xdisplay', Image, latch=True)
    head_display.publish(baxter_display_face_rosmsg)
    # Sleep to allow for image to be published.
    rospy.sleep(1)

    navigator_arm = baxter_interface.Navigator(args.limb)
    navigator_torso = baxter_interface.Navigator('torso_right')
    navigator_arm.inner_led = True
    navigator_arm.outer_led = True
    navigator_torso.inner_led = True
    navigator_torso.outer_led = True

    robot_arm = baxter_interface.Limb(args.limb)
```

```
pre_start_pos = ik_solution(args.limb, 0.1, -0.9, 0.0, 0.25)
robot_arm.move_to_joint_positions(pre_start_pos)

home_pos = ik_solution(args.limb, 0.45, 0.0, 0.0, 0.20)
robot_arm.move_to_joint_positions(home_pos)

sys.exit()
```

Pick and Place sequence for the Baxter robot

rosrun deep-grasping baxter_pnp.py -l right -x X_COORDINATE -y Y_COORDINATE -th GRASP_POSE_ORIENTATION

```
#!/usr/bin/env python

"""
       Object Pick and Place script by
       Shehan Caldera(shehancaldera@gmail.com)
that uses IK solver service client
and baxter_interface classes.

"""

import argparse
import struct
import sys

import rospy
import baxter_interface

from geometry_msgs.msg import (
    PoseStamped,
    Pose,
    Point,
    Quaternion,
)
from std_msgs.msg import Header

from baxter_core_msgs.srv import (
    SolvePositionIK,
    SolvePositionIKRequest,
)


def ik_solution(limb, p_x, p_y, theta, depth):
    """
        Object Pick and Place script
        by Shehan Caldera(shehancaldera@gmail.com)

    Uses IK solver service client
    and baxter_interface classes.
```

```
    The following arguments are required:
    '-l' [left or right],
    '-x' [x-coordinate],
    '-y' [y-coordinate],
    '-th' [orientation around z-axis]

"""
ns = "ExternalTools/" + limb +
"/PositionKinematicsNode/IKService"
iksvc = rospy.ServiceProxy(ns, SolvePositionIK)
ikreq = SolvePositionIKRequest()
hdr = Header(stamp=rospy.Time.now(), frame_id='base')
poses = {
    'left': PoseStamped(
        header=hdr,
        pose=Pose(
            position=Point(
                x=0.58,
                y=0.18,
                z=0.10,
            ),
            orientation=Quaternion(
                x=-0.13,
                y=0.99,
                z=0.01,
                w=0.02,
            ),
        ),
    ),
    'right': PoseStamped(
        header=hdr,
        pose=Pose(
            position=Point(
                x=p_x,
                y=p_y,
                z=0.0 + depth,
            ),
            orientation=Quaternion(
                x=0.0,
                y=0.1,
                z=theta,
                w=0.0,
            ),
        ),
```

```
        ),
    }

    ikreq.pose_stamp.append(poses[limb])
    try:
        rospy.wait_for_service(ns, 5.0)
        resp = iksvc(ikreq)
    except (rospy.ServiceException, rospy.ROSException), e:
        rospy.logerr("Call␣for␣IK␣service␣failed:␣%s" % (e,))
        return 1

    limb_joints = 0

    # Check if result valid, and type of seed
        # ultimately used to get solution
    # convert rospy's string representation of uint8[]'s to int's
    resp_seeds =
    struct.unpack('<%dB' % len(resp.result_type), resp.result_type)
    if (resp_seeds[0] != resp.RESULT_INVALID):
        seed_str = {
            ikreq.SEED_USER: 'User␣Provided␣Seed',
            ikreq.SEED_CURRENT: 'Current␣Joint␣Angles',
            ikreq.SEED_NS_MAP: 'Nullspace␣Setpoints',
        }.get(resp_seeds[0], 'None')

        print
        ("SUCCESS␣-␣Valid␣Joint␣Solution
␣␣␣␣␣␣␣␣Found␣from␣Seed␣Type:␣%s" %(seed_str,))

        # Format solution into Limb API-compatible dictionary
        limb_joints =
        dict(zip(resp.joints[0].name, resp.joints[0].position))

        print "\nIK␣Joint␣Solution:\n", limb_joints
        # print "------------------"
        # print "Response Message:\n", resp
    else:
        print("INVALID␣POSE␣-␣No␣Valid␣Joint␣Solution␣Found.")

    return limb_joints


if __name__ == '__main__':
    arg_fmt = argparse.RawDescriptionHelpFormatter
```

```
parser = argparse.ArgumentParser(formatter_class=arg_fmt,
                                 description=ik_solution.__doc__)
parser.add_argument(
    '-l', '--limb', choices=['left', 'right'], required=True,
    help="the␣limb␣to␣use␣with␣object␣pickup"
)
parser.add_argument(
    '-x', type=float, required=True,
    help="x-coordinate␣of␣centre␣of␣pickup"
)
parser.add_argument(
    '-y', type=float, required=True,
    help="y-coordinate␣of␣centre␣of␣pickup"
)
parser.add_argument(
    '-th', type=float, required=True,
    help="Gripper␣orientation␣about␣z-axis"
)
args = parser.parse_args(rospy.myargv()[1:])

rospy.init_node("IK_Baxter_PnP")

robot_arm = baxter_interface.Limb(args.limb)
robot_gripper = baxter_interface.Gripper(args.limb)
robot_gripper.calibrate()
robot_gripper.close()

home_pos = ik_solution(args.limb, 0.62, -0.6, 0.0, 0.20)
robot_arm.move_to_joint_positions(home_pos)

before_pick_pos =
ik_solution(args.limb, args.x, args.y, args.th, 0.20)
robot_arm.move_to_joint_positions(before_pick_pos)
robot_gripper.open()
pick_pos = ik_solution(args.limb, args.x, args.y, args.th, 0.05)
robot_arm.move_to_joint_positions(pick_pos)
rospy.sleep(2)
robot_gripper.close()
rospy.sleep(2)
robot_arm.move_to_joint_positions(before_pick_pos)

before_drop_pos = ik_solution(args.limb, 0.35, -0.8, 0.0, 0.20)
robot_arm.move_to_joint_positions(before_drop_pos)
drop_pos = ik_solution(args.limb, 0.35, -0.8, 0.0, 0.05)
```

```
robot_arm.move_to_joint_positions(drop_pos)
rospy.sleep(2)
robot_gripper.open()
rospy.sleep(2)
robot_arm.move_to_joint_positions(before_drop_pos)
robot_gripper.close()

robot_arm.move_to_joint_positions(home_pos)

sys.exit()
```

Appendix C

Project Timeline

The time-line in Figure C.1 describes the task-specific work plan against the time period that is spent on each task. The project included the following seven milestones.

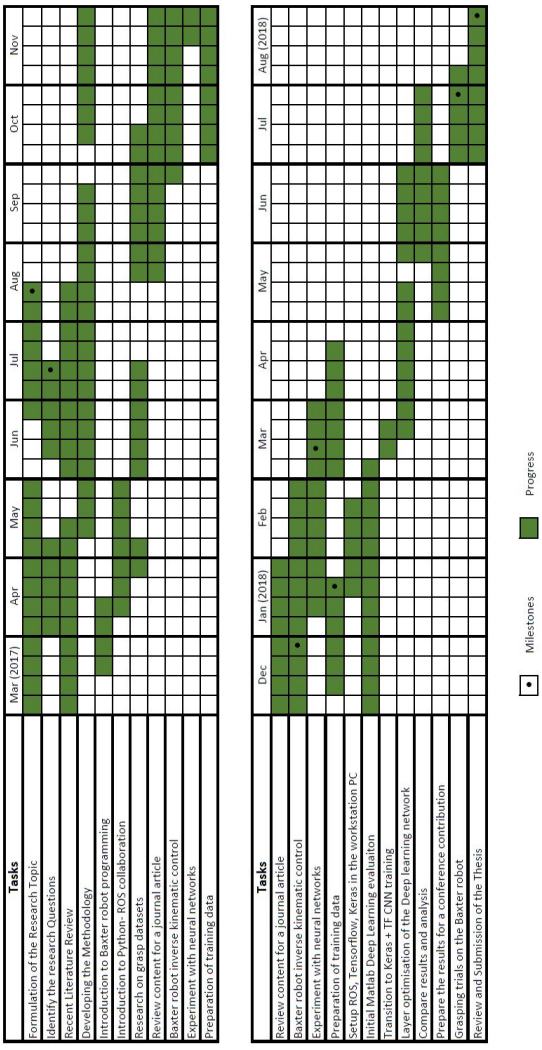


Figure C.1: The project timeline