Deep Convolutional Neural Networks for Robotic Grasp Detection

CS39440 Major Project Report

Author: Oliver Thomas (olt13@aber.ac.uk)

Supervisor: Dr Patricia Shaw (phs@aber.ac.uk)

30th March 2021

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a MEng degree in  
Robotics and Embedded Systems Engineering (132C)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name Oliver Thomas

Date ……………………………………………

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Oliver Thomas

Date ……………………………………………

Acknowledgements

I am grateful to my supervisor, Patricia, for guiding and advising me while completing this interesting project.

I would also like to thank my family and my girlfriend Katie for their incredible ongoing support.

Abstract

Include an abstract for your project. This should be approximately 300 words.

The abstract is an overview of the work you have done. Highlight the purpose of the work and the key outcomes of the work.

This paper explores the complicated topic of robotic grasping, following a deep learning approach to ‘learn’ a generalised view of grasping to be applied to novel objects.

Contents

[1 Project Background, Analysis & Process i](#_Toc71137506)

[1.1 Project Description i](#_Toc71137507)

[1.2 Background & Motivation i](#_Toc71137508)

[1.3 Analysis i](#_Toc71137509)

[1.3.1 General Analysis i](#_Toc71137510)

[1.3.2 Grasp Dataset ii](#_Toc71137511)

[1.3.3 Simulation Environment ii](#_Toc71137512)

[1.3.4 Convolutional Neural Network iii](#_Toc71137513)

[1.4 Process iii](#_Toc71137514)

[1.4.1 Feature-Driven Development iii](#_Toc71137515)

[1.4.2 Feature Table and Initial Timeline v](#_Toc71137516)

[2 Convolutional Neural Networks 6](#_Toc71137517)

[2.1 Background 6](#_Toc71137518)

[2.2 Technical Workings 7](#_Toc71137519)

[2.2.1 Convolutional Layers 7](#_Toc71137520)

[2.2.2 Pooling Layers 8](#_Toc71137521)

[2.2.3 Overall Configuration 8](#_Toc71137522)

[2.3 SCW 8](#_Toc71137523)

[3 Experiment Methods 9](#_Toc71137524)

[3.1 Experiment Hypothesis 9](#_Toc71137525)

[3.2 Experiment Design 9](#_Toc71137526)

[3.2.1 Experiment Outline 9](#_Toc71137527)

[3.2.2 Ground Truth Evaluation 10](#_Toc71137528)

[3.2.3 Trained Neural Network Evaluation 10](#_Toc71137529)

[3.2.4 Experiment Results 11](#_Toc71137530)

[4 Software Design and Implementation 12](#_Toc71137531)

[4.1 Iteration 0 12](#_Toc71137532)

[4.1.1 Feature List 12](#_Toc71137533)

[4.1.2 Overall Model 12](#_Toc71137534)

[4.1.3 Workspace Initialisation 13](#_Toc71137535)

[4.2 Iteration 1 14](#_Toc71137536)

[4.2.1 Feature 1 – Visualise Dataset 14](#_Toc71137537)

[4.2.2 Feature 2 – Generate Dataset 14](#_Toc71137538)

[4.2.3 Feature 4 – Simulation Environment 15](#_Toc71137539)

[4.2.4 Reflection 16](#_Toc71137540)

[4.3 Iteration 2 17](#_Toc71137541)

[4.3.1 Feature 3 - Gazebo Models 17](#_Toc71137542)

[4.3.2 Feature 5 – Arm Manipulation Setup 17](#_Toc71137543)

[4.3.3 Reflection 18](#_Toc71137544)

[4.4 Iteration 3 19](#_Toc71137545)

[4.4.1 Feature 6 – CNN Configuration 19](#_Toc71137546)

[4.4.2 Feature 7 – Train Network 20](#_Toc71137547)

[4.4.3 Feature 8 – Network Integration 20](#_Toc71137548)

[4.4.4 Reflection 21](#_Toc71137549)

[4.5 Iteration 4 22](#_Toc71137550)

[4.5.1 Feature 9 – Develop Experiments 22](#_Toc71137551)

[4.5.2 Feature 10 – Save Results 22](#_Toc71137552)

[4.5.3 Reflection 22](#_Toc71137553)

[4.6 Iteration 5 23](#_Toc71137554)

[4.6.1 Feature 11 – Results Visualisation 23](#_Toc71137555)

[4.6.2 Reflection 23](#_Toc71137556)

[5 Testing 24](#_Toc71137557)

[5.1 Overall Approach 24](#_Toc71137558)

[5.2 Automated Testing 24](#_Toc71137559)

[5.3 Visual Testing 24](#_Toc71137560)

[6 Results and Conclusions 25](#_Toc71137561)

[6.1 Ground Truth Success 25](#_Toc71137562)

[6.2 Model Loss and Accuracy 25](#_Toc71137563)

[6.3 Model Success in Simulation 25](#_Toc71137564)

[7 Critical Evaluation 26](#_Toc71137565)

[8 Annotated Bibliography 27](#_Toc71137566)

[9 Appendices 29](#_Toc71137567)

Table of Figures

[Figure 1.1: The feature table for the project, as seen in week 11. v](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137430)

[Figure 1.2: Initial Gantt chart to plan a rough timeline. v](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137431)

[Figure 2.1: Shows a receptive field (Gabor filter-type) typical for a simple cell. Source: [] 6](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137432)

[Figure 2.2: Shows the process of convoluting and input to create a feature map. Source: [] 7](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137433)

[Figure 2.3: Max Pooling Diagram. Source: [] 8](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137434)

[Figure 3.1: The pipeline for evaluating the trained neural network in simulation. 11](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137435)

[Figure 4.1: UML Case diagram showing initial design. 12](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137436)

[Figure 4.2: Initial ROS Graph. 13](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137437)

[Figure 4.3: The visualisation of the data using ACRONYM scripts. Left to right: greyscale, depth, segmentation, grasp visualization. 14](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137438)

[Figure 4.4: Two diagrams - first a simple break down of the feature - second a UML Sequence diagram showing the interaction between the scripts. 15](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137439)

[Figure 4.5: Design of the Gazebo model generation script. 17](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137440)

[Figure 4.6: The deigned configuration for the CNN. 19](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137441)

[Figure 4.7: A plot of the loss metric decreasing after each epoch. 20](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137442)

[Figure 4.8: Design for collating results into one dataset. 23](https://prifysgolaber-my.sharepoint.com/personal/olt13_aber_ac_uk/Documents/Work/Major%20Project/mmp_grasp_detect/Reports/Final%20Report.docx#_Toc71137443)

# Project Background, Analysis & Process

## Project Description

The aim of this project is to research and apply (in simulation) a deep convolutional neural network for the purpose of detecting successful grasp poses of objects. The project utilises many interesting technologies, from machine learning libraries to robotic middleware. This report’s main job is to explore these technologies in more depth, but it will also cover both the experimental and software engineering processes involved in completing the project.

## Background & Motivation

Over the last five to ten years, much work and research has gone into the field of robotics. Perhaps most significant is the work in the field of robotic manipulation and grasping. Despite this, robotic grasping is still a challenging topic due to the many environmental inputs that effect how a grasp can be accomplished. One company at the forefront of robot research is Boston Dynamics [1], who have produced many new and exciting robots; including a dog-like robot that can open doors. However, though this may appear like the robot has successfully tackled the grasping problem, it is still very complex. Robots that are designed to perform grasp still typically use a database that contains information on how to grip specific objects that the robot will come across. However, this method does not work well in a constantly changing environment, in which a robot may find many novel objects.

This project was inspired by reading the Review of Deep Learning Methods in Robotics Grasp Detection [2]. This paper discusses the “current state-of-the-art in regard to the application of deep learning methods to generalised robotic grasping”, which fascinated me. Further reading of the review led to the decision to research more into the topic of deep learning in connection with robotic grasp. This was followed by reading many other papers discussing the topic, such as Lenz et al. ‘Deep Learning for Detecting Robotic Grasps’ [3] (others shown in the Annotated Bibliography [4]–[8]), to ascertain the more details about what the project would entail. They also gave more detailed information about certain design issues that will be discussed in *Chapter 4.1*. For instance, Schmidt et al. paper on ‘Grasping unknown objects’ [9] provided details on a basic Convolutional Neural Network (CNN) structure to build off of for this project.

## Analysis

### General Analysis

The nature of this project is research based; therefore, research questions that the project aims to answer need to be defined before the problem can be properly analysed. The main question to be researched is – is it possible for a CNN to learn grasp patterns for specific objects? It is clear from the extensive research that has been done into this area that this should be possible; however, there may be technical limitations in place that prevent this project to produce the most accurate model. The limitations to the implementation of this project will be discussed further in *Chapters 4.2 and* *6*. This question will be evaluated by using metrics during the model training pipeline, with results coming from the model loss function.

The next question to be answered is – will the trained model produced be able to be applied to novel objects? In other words, has the model been able to get a generalised view of grasping? The answer to this question will be found through simulation – the results of which will be used to evaluate the efficacy of the model and the success of the research. The design of the experiments will be shown in *Chapter 3* along with a statement of the project hypothesis.

Upon analysis of this project, it became clear from the beginning that there would be three major component elements to conduct the experiments. These components for the basis for the technical objectives and are as follows:

1. Generating an image – grasp dataset.
2. Training a CNN to predict grasps from images.
3. Building a simulation environment that will be used to run experiments.

The next few sections will analyse each of these objectives, breaking down the possible technologies that could be used, plus a brief description of possible challenges that may arise (though this will be discussed in more detail in the *Design* and *Implementation* sections).

### Grasp Dataset

By reading Caldera’s review into the subject of deep learning for robotic grasping and the other papers discussed earlier, it became readily apparent that there was a main grasp dataset that was standard. This was known as the Cornell Grasp Dataset [] and seemed to include everything that was required for the purposes of this projects research – a dataset of images with labelled grasps. However, unfortunately when trying to obtain the data from the referenced links, the data seemed to have been taken down from the public domain. Hence, more research was required to find data suitable for the project. During this research, two other datasets were found that seemed to meet the requirements – the Jacquard dataset [10], and the ACRONYM dataset [11], [12]. After more analysis of what was on offer from the two options, the ACRONYM dataset was decided upon, mainly due to the fact it is linked to the ShapeNetSem [13] object database, which could easily be used within the simulation environment. In addition to this, the ACORNYM dataset is provided with access to Python scripts to manipulate the data. This task may be the most important part of the project, as the data used in machine learning is fundamental to the usefulness of the output.

### Simulation Environment

Due to previous experience, the simulation side of the project is based in ROS [14]; however, research was required for which simulator was going to be used. The obvious choice was Gazebo [15], but other options included the OpenRave [16] and the GraspIt! [17] simulators. After analysing the strengths and weaknesses of each option, Gazebo was agreed on. This is mainly due to its ease of use, experience with the software, and compatibility with the ShapeNetSem models. Software for ROS Melodic can be developed in either C++ [18] or Python 2 [19] and it is likely that a combination of both will be used for this project.

The main decision to make regarding the simulation other than the simulator is which manipulator will be used. Upon initial analysis there are two options, the Franka Emika Panda arm [20] or the Fetch robot [21]. The Panda arm was chosen over the Fetch robot for two reasons, the main being that the ACRONYM dataset was generated using the Panda arm and the continuity would be beneficial. The Panda arm would also be simpler to simulate due to it only being an arm, rather than an arm connected to a bigger robot.

To plan and move the arm to specific points, a library is needed to calculate the inverse kinematics and the relevant joint positions. The plan for this project is to use the MoveIt! framework [22] for this purpose. Due to experience with working with this framework, no other library was considered for this project.

Another consideration to make when designing the environment, is to make the scene within the camera as similar to that produced by the grasp dataset. This will help when integrating the trained network into the simulation.

### Convolutional Neural Network

Convolutional Neural Networks (CNNs) are tricky to implement efficiently; however, libraries and platforms such as, Tensorflow [23] and PyTorch [24], enable the implementation of machine learning algorithms such as CNNs easy. They also support customisation (such as with loss metrics or training loops) and optimisation. Tensorflow is the library of choice for this project alongside Python 3 [19], due to some previous experience and the interest in developing these skills further. Initial analysis of the problem shows that the project may need to implement a custom loss function to improve the network for this purpose; however, the standard libraries will likely be used to begin with, in order to create a working prototype. The CNN will likely be the biggest technological limitation of this project, as they require a lot of time to fully train.

## Process

### Feature-Driven Development

During the initial analysis of the project, several different software development methodologies were researched. It was decided that this project will use an agile Feature-Driven Development (FDD) methodology for planning and time management. The FDD process followed over the course of the project has been adapted to make it suitable for an individual developer. The main change is the removal of team roles and feature teams, as it is a solo project. The FDD process can be considered a plan-driven approach that incorporates agile elements. It was designed to follow five fundamental development steps that build around the features of the intended program over several one-to-two-week iterations. The FDD workflow steps are:

1. Develop an overall model of the software and write the project outline. This model is the basis for the initial planning of the project onto which each iteration’s designs are built upon. This can be done through updating existing diagrams or adding new diagrams that give more detail about specific features.
2. Build a feature list of all the requirements by breaking up the target functionality into smaller features. Due to the agile nature of FDD, this list is not necessarily final; however, the small nature of the project makes it likely that the list shown in *Figure 1.1* will be.
3. Plan the development of the features – determine the order in which the features will be developed and who will oversee developing them. A rough timeline for developing the features is shown in *Figure 1.2*.
4. Design by feature – features are selected for the next iteration and assigned to different teams (though not in this case). The features are designed and added to the overall model of the system.
5. Build by feature – once the designs are inspected, the feature is then implemented to that design. Usually, unit and acceptance testing are applied at this stage in the project process. However, due to the time constraints placed on the project, most of the testing was left to the end of the process.

Steps 4 and 5 are repeated until all the features have been designed, implemented, and tested. Once that is completed, the software produced should comprehensively fulfil the functional requirements set out as the features at the beginning.

### Feature Table and Initial Timeline

Figure .: The feature table for the project, as seen in week 11.

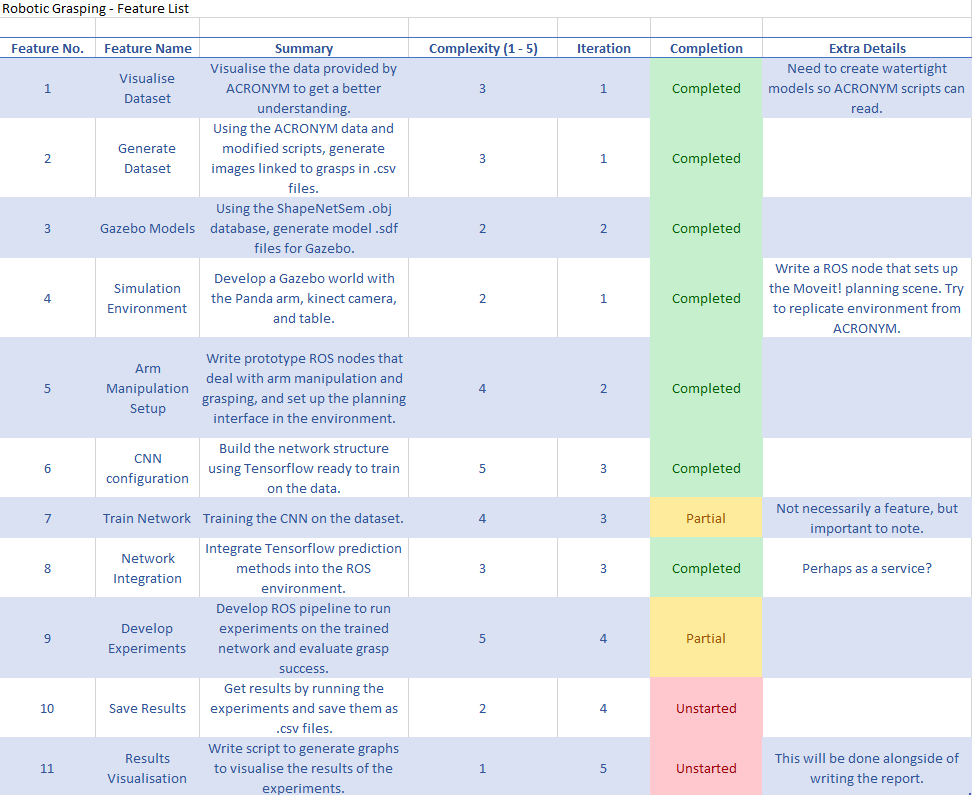
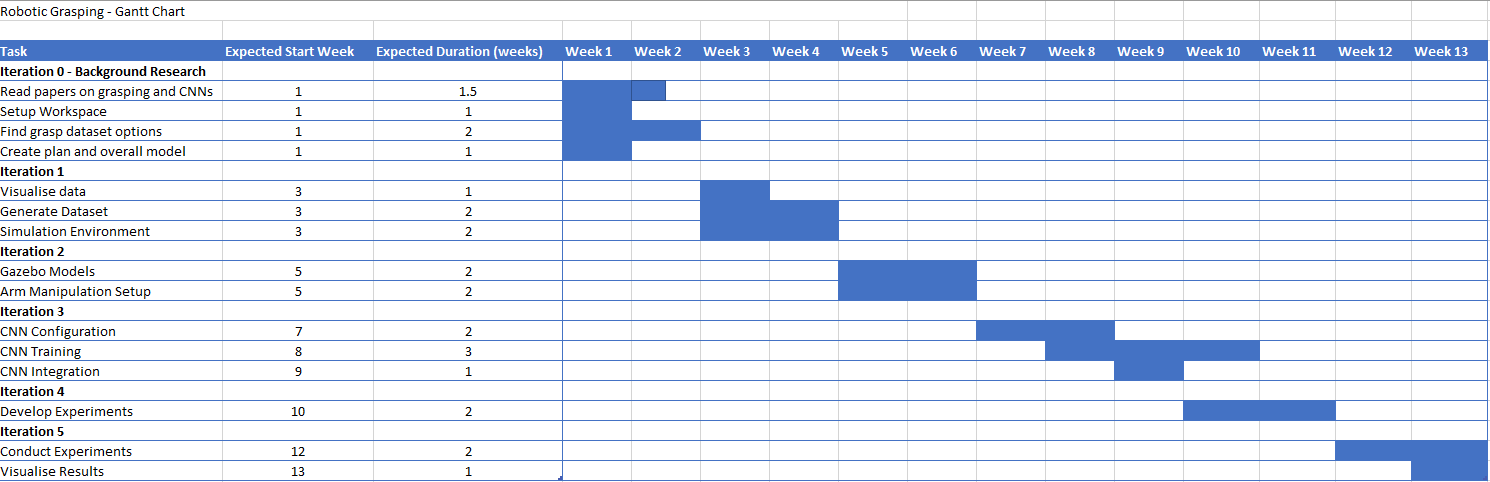


Figure .: Initial Gantt chart to plan a rough timeline.



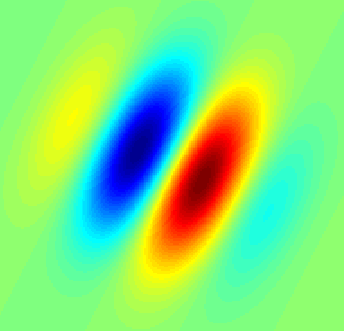
# Convolutional Neural Networks

When implementing technologies, it is important to know where they come from. This helps you to understand how and why they work. This chapter’s focus is on the origins and the inner workings of Convolutional Neural Networks. The aim of this research is to aid the design and implementation of a working CNN.

## Background

In the 1950s, more research was done into the human visual cortex, attempting to answer two questions. First, what does the visual system do? Second, how does it do it? “In 1959, David Hubel and Torsten Wiesel described simple cells and complex cells in the human visual cortex” writes Rachael Draelos [25]. Hubel and Wiesel proposed that a combination of these two types of cell is used in pattern recognition. Each simple cell responds to bars and edges of a particular orientation in the scene as seen in *Figure 2.1*. Complex cells respond similarly to the bars and edges; however, they differ as they have a property known as ‘spatial invariance’. This means the complex cells still respond when the bars and edges are shifted around the scene.

Figure .: Shows a receptive field (Gabor filter-type) typical for a simple cell. Source: []



By 1962 Hubel and Wiesel’s theory had developed. They proposed that spatial invariance of complex cells is achieved by ‘summing’ the output of several simple cells with similar orientations but different locations in the scene (or receptive field in the eye). This summing effect enables the complex cells to respond to similar stimuli that occur anywhere in the field.

Throughout the history of robotics, there are many examples where biology has inspired technical advancement. This is true also in the field of machine learning and convolutional neural networks. Draelos talks about how, in the 1980s, Dr Kunihiko Fukushima was inspired by the simple and complex cells and proposed a neural network model that applied this simple-to-complex concept through mathematical operations. This ‘Neocognitron’ [26], as it was named, was an attempt to create a computational model for visual pattern recognition and it successfully displayed the spatial variance properties desired. The neocognitron was the inspiration for more work in the 1990s that led to the modern CNNs.

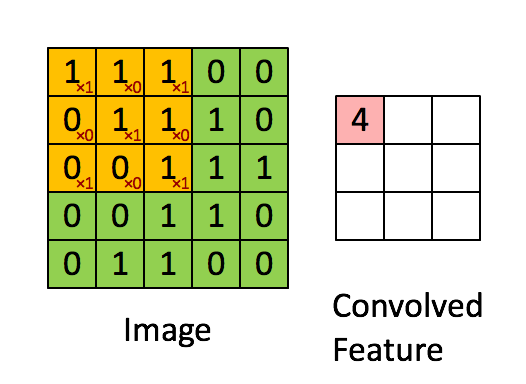
More recently, CNNs have successfully been used for classifying images, such as AlexNet [], which achieved state-of-the-art performance when trained on the ImageNet dataset in 2012. Furthermore, CNNs are now being used for a variety of purposes, from facial recognition to the analysis of medical images. This successfulness in regard to detecting patterns within images is what led to this project attempting to utilise this functionality in the context of robotic grasp detection.

## Technical Workings

### Convolutional Layers

Unlike a standard artificial neural network, a convolutional neural network takes an image as its input. Therefore, the network needs a way to deal with two, three, or even four-dimensional data. The way this is done is through a convolutional layer, which is the core building block for these networks, hence the name. This layer convolves the input by using filters (or kernels) to create a feature map (the matrix on the right of Figure 2.2). This is done by calculating the dot product of the receptive field (highlighted in yellow in Figure 2.2) and the filter (the red numbers in Figure 2.2). The dot product is defined as the sum of the product of each corresponding element in the two matrices and it produces a scalar output. The filter’s size is usually an odd number (3, 5, and 7 are common sizes) and determines the size of the area for features to be identified in the image. It is applied to the whole image by shifting the receptive field by a set number of pixels known as the stride, the default being one pixel. Increasing the stride leads to a decrease in the size of the feature map, due to less overlapping of the receptive fields.

Figure .: Shows the process of convoluting and input to create a feature map. Source: []

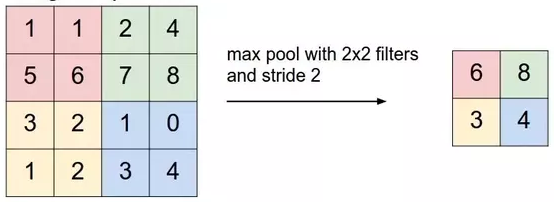


Once the feature map is calculated, a non-linear activation function is applied. The standard choice currently is the Rectified Linear Unit (ReLU) activation, as it is more computationally efficient than the other options. ReLU is a piecewise linear function that will output the input if positive, otherwise, it will output zero. The purpose of using an activation function such as this, is to introduce non-linearity into a network that has so far just been applying linear operations. The rectified feature map can either be the input for another convolutional layer, or it can be ‘flattened’ into a vector to be an input for a fully connected layer – leading to the output layer.

### Pooling Layers

After every convolutional layer in a CNN, it is usual to have a pooling layer applied to the rectified feature map. The main purpose of this is to downsample the features; making the network more efficient and puts less strain on the available resources. There are different types of pooling, including max, sum, and average with max pooling being the most used. Pooling layers work on a similar principle to convolutional layers. A filter, usually an even number this time, is applied to the feature pool with no overlap. So, in the common case of a filter size of 2x2, usually the stride would be set to two also, which leads to a feature map that is half the size. In the case of max pooling, the largest element in the receptive field is taken to represent that area (as shown in *Figure 2.3*). Sum and average pooling are calculated similarly but using the sum and mean average of the receptive field, respectively.

Figure .: Max Pooling Diagram. Source: []



### Overall Configuration

The small details of a CNN can vary greatly for different projects; however, the standard building blocks are usually used universally in a set order. This standard configuration is as follows:

* Input layer
* Convolutional and Pooling Layers alternated
* Flattening for input to Fully Connected Layers
* Output Layer with a softmax or linear activation depending on the task

This chapter has tried to explain the workings of a convolutional neural network in enough detail for the purposes of this project. However, if more information is required, there are many easily accessible papers and articles written on the subject, including [] and [].

## SCW

This project has been given access to the Supercomputing Wales (SCW) [27] cluster, in order to train the network. The reason this has been granted is that CNNs need a lot of training on a lot of data, which means they are computationally intensive. This project therefore required more computing power than was initially available locally. The process to use this service will be discussed further in *Chapter 4.4*.

# Experiment Methods

This project poses the question: can robotic grasps be successfully learnt by using a deep learning algorithm? *Chapter 3* will outline the original hypothesis made at the start of the process, and the experiments that were designed to evaluate the aforementioned hypothesis and the success of the project. The supporting software design will be covered in *Chapter 4*.

## Experiment Hypothesis

* Can a robot learn to grasp objects just from input images?
* This approach has been tried before – as mentioned in intro. Quote papers and success. Mention how this differs slightly as only in simulation and using different from norm data.

The hypothesis being investigated in this project is that if a deep convolutional neural network can learn to predict grasps from images, then the trained network can be used to grasp novel objects in simulation. This is functionality is trying to mimic the human intuition and experience that enables us as a species to pick up objects that we may have never encountered. This hypothesis has been tested in other papers, as mentioned in *Chapter 1*, with some reports quoting success rates of up to … This project is not necessarily searching for the most accurate neural network or the most successful grasps; due to time and technical limitations imposed due to the nature of the project (discussed further in *Chapter 4*). However, the expected outcome is a proof of concept that could be further refined to produce better results.

## Experiment Design

### Experiment Outline

The aim of this project is to test this hypothesis through experimentation in simulation. This will involve writing software and handling different data, further discussed in *Chapter 4*. The software and data will be used to test the hypothesis by taking the following two measurements:

* Evaluation of the ground truth.
* Evaluation of the trained network.

Before describing the experiments to take these two measurements, I must define two things within the scope of this project:

* What is a successful grasp?
* What is a ground truth?

Perhaps most importantly, for the purpose of this project, I will be defining a successful grasp as a combination of two factors. Furthermore, I will also be recording semi-successful grasps that come close to picking up the object (e.g., if the object is thrown to the side). The two factors that are going to be considered for grasp success are: gripper closure, and object movement within the camera frame. This will be further explained in the Design section of *Chapter 4*.

A ground truth in the field of machine learning is the ‘reality’ that the model is attempting to predict, in this case the ground truths are successful grasp patterns for specific objects. The ground truth is dictated by the input data for a model; hence, the capability and efficacy of all machine learning models is directly linked to the quality of the ground truth. This project is making use of the ACRONYM dataset, as discussed in *Chapter 1.3*, which contains grasp and object information in relation to the ShapeNetSem object database. This grasp dataset is then considered to be the ground truth for these experiments.

As stated before, the training data for a neural network directly impacts the trained output; therefore, it is necessary to determine the quality of the dataset that is being used. The ACRONYM grasp dataset does contain information on the success of each grasp from their physics simulator; however, this does not necessarily mean the labelled grasps will work in this project’s simulation. Therefore, it is essential to conduct an experiment to ascertain the accuracy of the grasps stated in the dataset within the project simulation environment, evaluating the ground truth of the scenario, thus allowing a comparison to the accuracy of the neural network.

### Ground Truth Evaluation

The ground truth experiment will be simple. The objects named in the dataset (a description of how this is produced is in *Chapter 4*) will be spawned into the Gazebo environment and each grasp labelled for that object will be attempted three times. Repeating the test enables an average success rate to be taken for both the different objects and the dataset as a whole.

* Extrapolating from a tested subset due to lack of time to test all grasps?

### Trained Neural Network Evaluation

Firstly, while training the model it is possible to assess the accuracy of the network by viewing the loss metrics. This will evaluate how close the predictions of the model are to their ground truth labels. It will also be possible to conduct tests on unseen data to check for overfitting during the training. However, this value will not necessarily indicate whether the network will work well in the simulation. Therefore, this loss metric will not be used to assess the network in this report.

The main experiments to evaluate the efficacy of the trained neural network model will be similar to that of the ground truth data. The pipeline for this evaluation is shown in F*igure 3.1*. Firstly, a random model from the ShapeNetSem database is spawned into the Gazebo simulation. An image of the object will be generated from the camera in the environment and fed into the trained model, outputting the predicted grasp. This prediction will define a transform from the object, which will then be used to attempt the grasp. The grasp success will be monitored and recorded; looking for partial gripper closure and the object to be moving up in the camera frame for a fully successful grasp. This pipeline, as in the ground truth evaluation, will be repeated three times to enable the taking of an average, thus decreasing the likelihood of outliers in the results.

Figure .: The pipeline for evaluating the trained neural network in simulation.

Spawn a random model

Generate grasp pose from image

Attempt the grasp

Evaluate the successfulness of the grasp

### Experiment Results

There will be two ways to evaluate the results of the project’s experiments:

* Looking at the average success rate of all objects.
* Looking at the category of object the model performs best on.

The first of these two options best evaluates the initial hypothesis laid out originally, as it focusses on the generalisation of the model and its ability to predict grasps for all kinds of objects. A high average success rate would show that the model (or a similar one) could be robust enough to be applied in many scenarios. On the other hand, a low success rate could be the result of different factors, such as: a low-quality ground truth, an incorrectly setup simulation environment, or a poorly built convolutional neural network.

The second option could perhaps be more interesting in the situation where the grasp success rate is mid-ranged (i.e., 40% - 60%). This is because in this situation the model appears to be working well to some degree; however, we might assume that some types of objects are holding it back. Looking at the success rate of the different category of objects could enable this to be fixed in future iterations of the project. There could be many reasons why some objects have a higher grasping rate than others, for example, the object is bigger or more complicated; therefore, will naturally be harder and more awkward to pick up. The object could even be too large for the gripper on the robot, though this can be mostly mitigated during this project due to its simulated nature.

By using both of these evaluation methods, we can generate a good assessment of how successful the model is performing, how it can possibly be improved, and how correct the original experiment hypothesis was.

# Software Design and Implementation

This chapter merges the discussion of the design and implementation of the project, due to the decision to follow a feature-driven-development-like methodology in *Chapter 1.4*. A subsection in this chapter has been dedicated to the design and implementation of each iteration in the project

## Iteration 0

As part of the FDD methodology, the first iteration always includes creating an initial model that encompasses the overall design. This model is then updated during each following iteration. Iteration 0 will also include the development of a feature list, the setup of the workspace environment, and the development of a rough plan to help manage time.

So, the work for Iteration 0 is as follows:

* Initial workspace setup
* Feature list
* Create plan (rough Gantt chart to manage time)
* Overall model of the project

### Feature List

As stated in the project analysis (*Chapter 1.3*), the project has been split into three fundamental tasks:

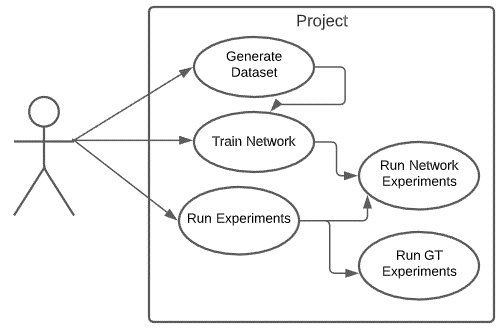
1. Generating an image – grasp dataset.
2. Training a CNN to predict grasps from images.
3. Building a simulation environment that will be used to run experiments.

By analysing these tasks and the experiments outlined in *Chapter 3*, a list of basic requirements or features can be established. These features can be viewed in more detail as a table in *Figure 1.1*.

### Overall Model

The nature of this project does not necessarily lend itself to one overall model for all software – there are many unconnected parts of software to design and implement the designs for which will be left to later iterations. However, it is still possible to create an overview of how the data will be transferred between the sections. In other words, how the output of one task becomes the input to the next.

Figure .: UML Case diagram showing initial design.

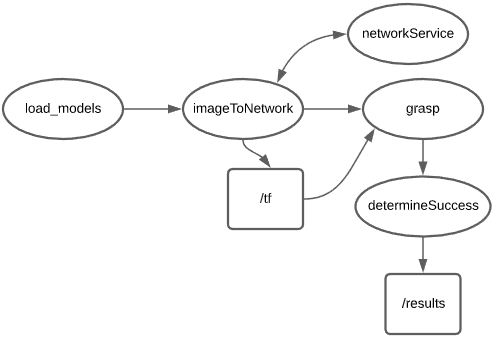


This initial design can be seen in *Figure 4.1* as a UML case diagram. The tasks listed above form the cases in the diagram, which also shows how the tasks link together.

#### Initial ROS Graph

As part of the initial design, a basic plan for the structure of the ROS nodes was created. This simple overview of the ROS structure is shown in *Figure 4.2*.

Figure .: Initial ROS Graph.



It demonstrates the possible connections between the nodes to create the functionality for running the experiments defined in *Chapter 3*. The details of this structure will be further designed in future iterations.

### Workspace Initialisation

At the beginning of the project, time was spent to setup the three main elements of the workspace. The tools setup during this Iteration were:

* GitHub
* Catkin workspace / ROS
* PyCharm and VSCode

#### GitHub

GitHub is used to create a private online git repository. This repository was used as a project management and version control tool. Two branches were used in the repository, a master and development branch. At the end of each iteration, the development branch will be merged with the master branch so that the next iteration can be developed.

#### Catkin Workspace

A catkin workspace is required for ROS development, as it enables the building of the C++ source code. As part of the catkin setup, the requirements for the project are established, though more libraries can be added later in the process if required. This process also creates a tidy filesystem, making working with the files easy.

#### PyCharm and VSCode

Most features of this project require programming. It was decided during the initial analysis that this would be split between two languages, C++ and Python. Therefore, it is necessary to setup an IDE for each of these languages. PyCharm will be used to develop the Python scripts for generating the dataset and building the Tensorflow CNN. VSCode will be used entirely for the ROS development, predominantly in C++ with some Python code written as well.

## Iteration 1

Iteration 1 focusses mainly on the understanding of the ACRONYM dataset in order to generate usable data to train the network on. The simulation environment is also designed in this iteration.

### Feature 1 – Visualise Dataset

*Visualise the data provided by ACRONYM to get a better understanding.*

Feature 1 did not require any major design, due to the fact that the ACRONYM dataset comes with tools [28] that enable the visualisation of the data. However, to enable the reading of the object files by these tools, I was required to use the Manifold [29] library to create watertight models. By integrating the Manifold code into a Python script (‘ACRONYM OBJ WATERTIGHT FILES.PY’), I was able to format each of the ShapeNetSem object files.

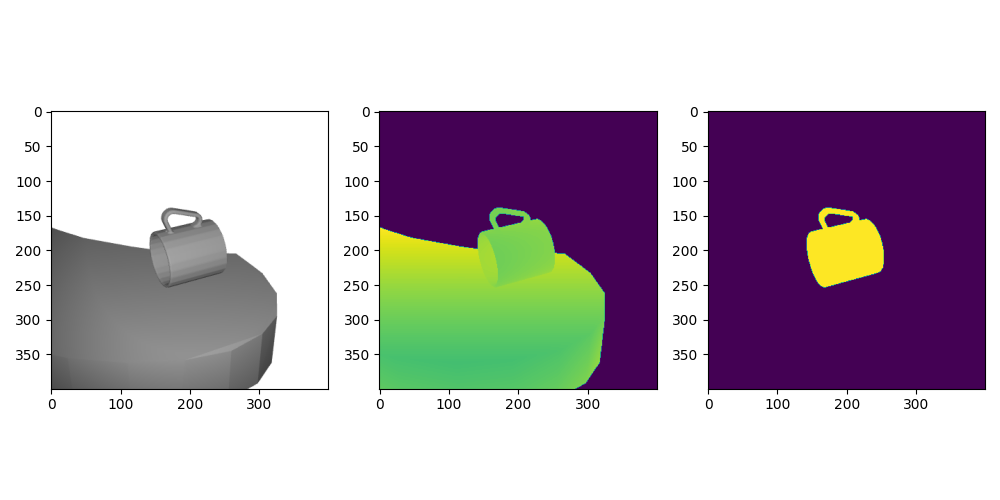
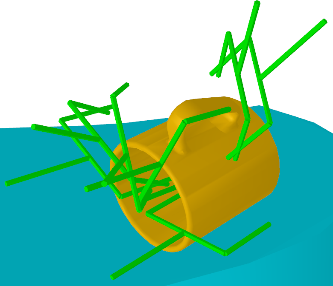


Figure .: The visualisation of the data using ACRONYM scripts. Left to right:  
greyscale, depth, segmentation, grasp visualization.

The ACRONYM tools provide three command-line interfaces to interact with the models: visualising the model, visualising the grasps, and generating a random scene with grasps that are not in collision. The image rendering and grasp visualisation tools are demonstrated in *Figure 4.3*. The implementation of these visualisations enables an analysis of the information provided in the dataset. It will also be of use when designing Feature 2.

### Feature 2 – Generate Dataset

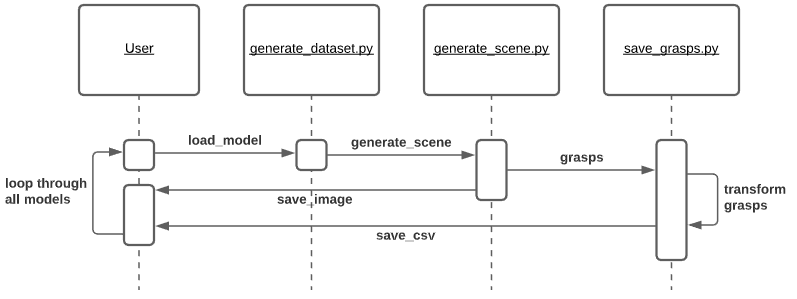
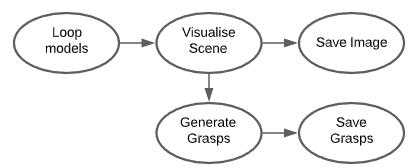
*Using the ACRONYM data and modified scripts, generate images linked to grasps in .csv files.*

Feature 2 required some thought for the initial design, as it is vital in any deep learning task that there is enough data to train on. The ACRONYM dataset certainly provides enough data (17.7M grasps on 8,872 objects), but how it is presented to the machine learning model is important. For this project, it was decided that each object would be randomly placed into a scene fifty times with an image of the scene generated and ten of the available successful grasps would be stored. The dataset provides each grasp as a 4x4 transformation matrix from what is estimated to be the centre of the object. Using this as a label for the object images would be computationally inefficient, requiring the data to be simplified into a six-dimensional format (x, y, z, roll, pitch, yaw). This was accomplished by using the transformations library provided by tf. It was decided that the dataset would have the following structure:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X | Y | Z | ROLL | PITCH | YAW | IMAGE |

Once that was determined, the scripts to generate the dataset could be designed. This design is shown in *Figure 4.4* and includes a combination of modified ACRONYM tool scripts and new scripts.

Figure .: Two diagrams - first a simple break down of the feature - second a UML Sequence diagram showing the interaction between the scripts.



This series of scripts produces two kinds of file: a .png image and a .csv containing the selected grasps. The image and corresponding grasp files share a name, meaning another Python script (‘load\_dataset.py’) can match the two files to create one large .csv file with the structure mentioned above. The image column in the resulting table is a string that corresponds to the image file name.

The implementation of this caused some issues due to different Python versions being required by the transformations library and the ACRONYM tools. This problem was solved by using the python subprocess module that enabled a Python 2 to be called from Python 3. While this is probably not the most computationally efficient way to do this, it was the quickest solution enabling the feature to be produced as quickly as possible. It also meant that no existing libraries had to be re-written for the purpose of this project.

### Feature 4 – Simulation Environment

*Develop a Gazebo world with the Panda arm, kinect camera, and table.*

When designing the ROS simulation environment, it was important to mimic the setup from the ACRONYM scenes. This is because the images that the neural network will be trained on will expect a similar scene when predicting in the simulation. The initial design for the environment utilises the Franka Emika Panda arm, a Kinect depth camera, and a table, as seen in the feature description. This would have worked well; however, for an unknown reason the Panda gripper was not working in the Gazebo simulation. Unfortunately, due to this complication the environment feature was delayed while trying to fix the gripper controllers. After spending a lot of time trying to fix this, it was decided to change from the Panda arm to use the Fetch robot instead (due to previous experience of this working). The fetch robot also has an inbuilt head camera which makes the environment simpler to build.

The design of the environment is purely observational; therefore, does not require any design diagrams. The implementation of this feature utilises ROS launch and world files to successfully load the same environment each time.

### Reflection

Upon reflection, the main challenge during this iteration was surprisingly setting up the Gazebo environment. These problems could have been avoided if the fetch robot were used initially. Another issue faced was the time it took to generate the watertight models, in future this could be sped up by multithreading the task; however, this process would still take a long period of time due to the large volume of files.

## Iteration 2

Iteration 2’s focus is on the simulation environment; generating the models that can be spawned; creating the nodes that will control the arm and gripper.

### Feature 3 - Gazebo Models

*Using the ShapeNetSem .obj database, generate model .sdf files for Gazebo.*

Spawning objects into a Gazebo world requires a specific file format – SDF. SDF files are formatted as xml, with tags to represent specific properties of a model. Gazebo needs these properties in order to simulate the object successfully. In SDF there is a tag <uri>, which takes a mesh input provided by the ShapeNetSem database (.obj or .dae) and enables the model to be displayed in Gazebo. The physical information needed for the SDF file is provided by the ACRONYM dataset. Alongside the SDF files, each model needs a .config file in its folder to initialise basic information about the model, such as its name.

To generate all the models for simulation in Gazebo, a Python script was designed as show in the sequence diagram in *Figure 4.5*.

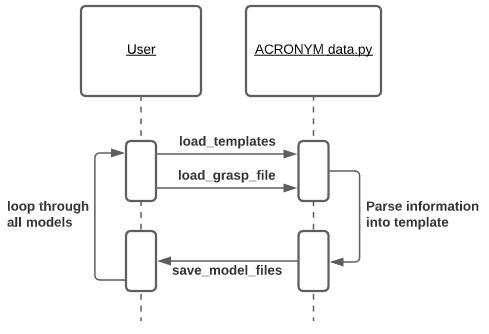


Figure .: Design of the Gazebo model generation script.

This script was implemented using the loading of template text files (for both SDF and config files) and using the Python string format function to parse the object physical information. This makes the script neater, more maintainable, and more understandable. The script also implements a multithreading pool that enables the script to run faster.

### Feature 5 – Arm Manipulation Setup

*Write prototype ROS nodes that deal with arm manipulation and grasping and set up the planning interface in the environment.*

This feature was designed and developed using a mixture of spike work and experience. The initial plan for the grasping functionality was to use the Moveit! library, as discussed in the project analysis section. The system makes use of Moveit! for inverse kinematics and path planning to reach a goal position. The goal position is set by a static transform broadcaster from the base link of the fetch robot to the grasp pose. The transform tree will initially be hard coded within the grasping node; however, in later iterations, the grasp frame will be determined by another node. The grasp poses will either be from the trained model or the grasp dataset. Once the location has been reached, the node uses a gripper controller action to close the gripper around the target object and then retreat. In future iterations, this is when the success of the grasp will be measured.

### Reflection

This iteration made very clear progress in regard to the completion of the project, as two key components were put in place. Furthermore, due to experience with most of the technologies used during this iteration, there were not many issues that arose.

## Iteration 3

Iteration 3 focuses mainly on implementing and training the Convolutional Neural Network (CNN). The following features form a major part of the project but rely heavily upon the data generated in the previous iterations.

### Feature 6 – CNN Configuration

*Build the network structure using Tensorflow ready to train on the data.*

The design for the CNN was inspired by that of a CNN from Schmidt’s paper, as discussed in *Chapter 1*. The configuration follows the standard structure of convolutional layers followed by fully connected layers discussed in *Chapter 2* and can be seen in *Figure 4.6*.

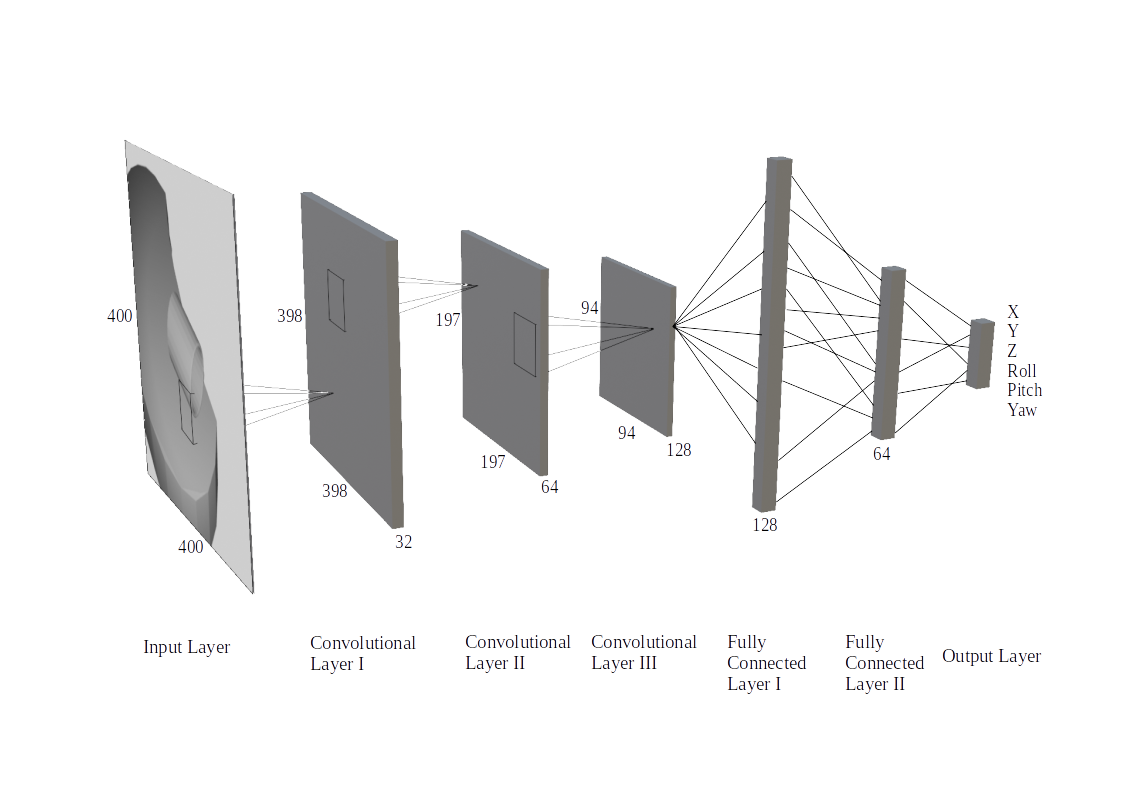


Figure .: The deigned configuration for the CNN.

This network is seven layers in total, with four sub-sections:

1. The input layer – the network takes a 400 x 400 x 1 greyscale image converted to a tensor as input.
2. The three convolutional layers uses a filter of size 3 x 3, a stride of 1, and an increasing kernel size of 32, 64, and 128, respectively. After layer two and three there is a max pooling layer, with a 2 x 2 filter and a stride of 2.
3. There are two fully connected layers following the flattening of the output of the third convolutional layer. The first layer has 128 nodes, decreasing to 64 in the second.
4. The output layer consists of 6 nodes to represent the grasp coordinates.

The implementation of this configuration was made simple by using a Tensorflow Keras Functional model. This also allowed batch normalisation and dropout to be applied to the layers, to help reduce overfitting and to increase the accuracy. An example of a Tensorflow functional model can be seen in *Appendix C*.

Later on in the project, it was decided to implement a new design of CNN, keeping the structure, but creating new training and loss functions. This was done to try and fine tune the network for the specifics of the project. The loss function was updated to minimise the mean absolute error from the closest grasp for that object. Updating the loss function in this way meant the training loop also had to be rewritten to allow access to the current input data. These new functions can be viewed in *Appendix D*.

### Feature 7 – Train Network

*Training the CNN on the dataset.*

In *Chapter 2*, it was briefly discussed that access to the Supercomputing Wales cluster (SCW) has been granted for this project. This was used to train the CNN on the generated images. While this helps with lifts certain technical limitations with training the CNN, to fully train a large network still takes a lot of time.

In order to use SCW, certain things needed to be initially setup. Setting up the Python 3 environment on SCW was done using Anaconda 3 [30], which gave access to the most updated Python libraries – particularly Tensorflow 2, which was not available by default on SCW. Furthermore, to run jobs on SCW, a batch file containing information about the job and commands to run the job is required (an example is in *Appendix E*). This enables the automatic queueing of all jobs. This does mean that occasionally a job must wait a while before resources are available to run it. Due to the slow speed, large amount of data, and limit resources, the network was trained for ten epochs at a time. At the end of each set of epochs, the weights would be saved so that the training could restart in the same place.

Part of the process of training a neural network is testing different hyperparameters, such as learning rate, batch size, and the number of epochs. It was while testing these different options that it was decided to write the new training functions previously mentioned to see if these would increase the performance of the network. The loss results of training the custom model are shown in *Figure 4.7*.

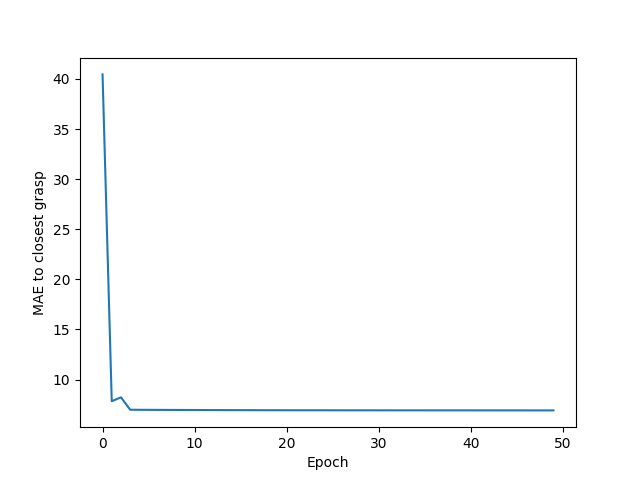


Figure .: A plot of the loss metric decreasing after each epoch.

It is clear that the loss converges very soon after starting the training process. This also occurred when training the standard model, which achieved similar loss performance.

### Feature 8 – Network Integration

*Integrate Tensorflow prediction methods into the ROS environment.*

This feature deals with the utilisation of the trained CNN model within the ROS environment. Since the main structure for ROS was designed in Iteration 0 (see *Figure 4.2*), including how the network would be integrated, there was no need for anymore design. It was decided to integrate the network into the environment using a ROS service, meaning it could be called from any node within the system if required.

A ROS service requires a definition for a pair of messages: one to define the request type, and the other for the response. In this case, the service will be called with a sensor\_msgs Image. The response will be floating point array of size six, which will be the predicted grasp pose. Within the service an OpenCv bridge will be used to format the data from the Image message so that it is readable by the Tensorflow model.

### Reflection

Looking back, this iteration may have produced the first signs that the project would not be as successful as first hoped. This can be seen through *Figure 4.7* as the loss metric converged after very few epochs to a value higher than expected from both network models. However, as stated in *Chapter 2*, this project’s success will not be determined by the network loss metric, but by conducting the experiments in the simulation environment.

## Iteration 4

Iteration 4’s main goal was to develop the software for running the experiments in the simulation environment. The experiments to be implemented have already been designed in *Chapter 3*.

### Feature 9 – Develop Experiments

*Develop ROS pipeline to run experiments on the trained network and evaluate grasp success.*

The initial design for the ROS pipeline to conduct the experiments is shown in *Figure 4.2* and does not need much alteration.

ACTUAL ROS GRAPH

### Feature 10 – Save Results

*Get results by running the experiments and save them as .csv files.*

### Reflection

## Iteration 5

### Feature 11 – Results Visualisation

*Write script to generate graphs to visualise the results of the experiments.*

The output from running the experiments is three separate CSV files. The initial part of this feature is to combine these files into one usable table of results. The data also includes different types of each object, mugs for example, and these will also be combined to make larger categories containing the average success of the individual objects. This ensures the visualisation of the data less crowded and more understandable. This functionality will be achieved through a Python script using Pandas to build the datasets with the following design.

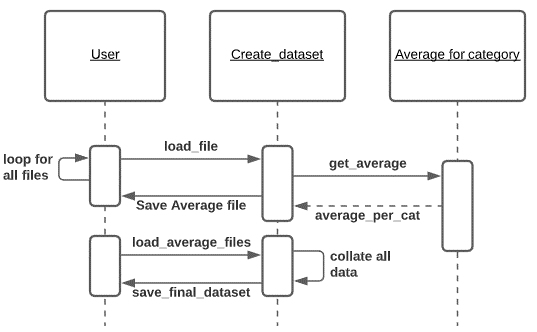


Figure .: Design for collating results into one dataset.

The collated dataset will then be visualised using Python and Matplotlib to produce three plots: a word-cloud based on the highest success rate objects, a ‘top fifteen’ bar chart, and a ‘bottom fifteen’ bar chart. These charts should give a comprehensive view of the success of specific objects in the simulation. The average success of all the grasps will also be available from the dataset.

### Reflection

There was not much to go wrong during this iteration. This meant there was more time to analyse the different results that were received from running the experiments. This analysis is discussed in *Chapter 6*.

# Testing

## Overall Approach

The overall approach to testing for this project is a combination of two methods: automated testing and visual testing. Due to the nature of the project, there is no need for user, interface, or stress testing, as there are no intended users. The following sections will discuss both aspects of the testing plan; considering what testing has been achieved and what would be planned if there were no time constraints.

A short list of main tests has been created to keep track of the testing of fundamental features. This can be seen in *Appendix F*. Alongside these broader tests, while the different elements and programs were being developed, smaller tests were also conducted during each iteration.

## Automated Testing

## Visual Testing

# Results and Conclusions

## Ground Truth Success

## Model Loss and Accuracy

* Show graph and discuss both implementations

## Model Success in Simulation

* Discuss effect of different objects for detecting grasps
  + Which are easier?

# Critical Evaluation

Examiners expect to find a section addressing questions such as:

* Were the requirements correctly identified?
* Were the design decisions correct?
* Could a more suitable set of tools have been chosen?
* How well did the software meet the needs of those who were expecting to use it?
* How well were any other project aims achieved?
* If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

The questions are an indication of issues you should consider. They are not intended as a specification of a list of sections.

The evaluation is regarded as an important part of the project report; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things in the work and aspects of the work that could be improved. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

# Annotated Bibliography

[1] ‘Spot | Boston Dynamics’. https://www.bostondynamics.com/spot (accessed May 05, 2021).

[2] S. Caldera, A. Rassau, and D. Chai, ‘Review of Deep Learning Methods in Robotic Grasp Detection’, *Multimodal Technol. Interact.*, vol. 2, no. 3, Art. no. 3, Sep. 2018, doi: 10.3390/mti2030057.

[3] I. Lenz, H. Lee, and A. Saxena, ‘Deep Learning for Detecting Robotic Grasps’, *ArXiv13013592 Cs*, Aug. 2014, Accessed: Feb. 04, 2021. [Online]. Available: http://arxiv.org/abs/1301.3592.

[4] S. Sapora, E. Johns, and A. Cully, ‘Grasp Quality Deep Neural Networks for Robotic Object Grasping’, p. 116.

[5] U. Asif, J. Tang, and S. Harrer, ‘GraspNet: An Efficient Convolutional Neural Network for Real-time Grasp Detection for Low-powered Devices’.

[6] H. Cao, G. Chen, Z. Li, J. Lin, and A. Knoll, ‘Lightweight Convolutional Neural Network with Gaussian-based Grasping Representation for Robotic Grasping Detection’, *ArXiv210110226 Cs*, Jan. 2021, Accessed: Feb. 04, 2021. [Online]. Available: http://arxiv.org/abs/2101.10226.

[7] J. Watson, J. Hughes, and F. Iida, ‘Real-World, Real-Time Robotic Grasping with Convolutional Neural Networks’, Jul. 2017, pp. 617–626, doi: 10.1007/978-3-319-64107-2\_50.

[8] S. Kumra and C. Kanan, ‘Robotic Grasp Detection using Deep Convolutional Neural Networks’, *ArXiv161108036 Cs*, Jul. 2017, Accessed: Feb. 04, 2021. [Online]. Available: http://arxiv.org/abs/1611.08036.

[9] P. Schmidt, N. Vahrenkamp, M. Wächter, and T. Asfour, ‘Grasping of Unknown Objects Using Deep Convolutional Neural Networks Based on Depth Images’, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 6831–6838, doi: 10.1109/ICRA.2018.8463204.

[10] A. Depierre, E. Dellandréa, and L. Chen, ‘Jacquard: A Large Scale Dataset for Robotic Grasp Detection’, *ArXiv180311469 Cs*, Sep. 2018, Accessed: Feb. 04, 2021. [Online]. Available: http://arxiv.org/abs/1803.11469.

[11] C. Eppner, A. Mousavian, and D. Fox, ‘ACRONYM: A Large-Scale Grasp Dataset Based on Simulation’, *ArXiv201109584 Cs*, Nov. 2020, Accessed: Feb. 04, 2021. [Online]. Available: http://arxiv.org/abs/2011.09584.

[12] ‘ACRONYM: A Large-Scale Grasp Dataset Based on Simulation’. https://sites.google.com/nvidia.com/graspdataset (accessed Feb. 04, 2021).

[13] ‘ShapeNet’. https://www.shapenet.org/ (accessed Feb. 07, 2021).

[14] ‘ROS.org | Powering the world’s robots’. https://www.ros.org/ (accessed Feb. 06, 2021).

[15] ‘Gazebo’. http://gazebosim.org/ (accessed Feb. 07, 2021).

[16] ‘OpenRAVE | Welcome to Open Robotics Automation Virtual Environment | OpenRAVE Documentation’. http://openrave.org/docs/latest\_stable/ (accessed Feb. 04, 2021).

[17] ‘GraspIt!’ https://graspit-simulator.github.io/ (accessed Feb. 04, 2021).

[18] ‘C++ documentation — DevDocs’. https://devdocs.io/cpp/ (accessed May 05, 2021).

[19] ‘Python’, *Python.org*. https://www.python.org/ (accessed Feb. 07, 2021).

[20] ‘Franka Panda Arm’, *Franka Emika*. https://www.franka.de/technology (accessed Feb. 07, 2021).

[21] ‘Autonomous Mobile Robots That Improve Productivity | Fetch Robotics’. https://fetchrobotics.com/ (accessed May 05, 2021).

[22] ‘MoveIt Motion Planning Framework’. https://moveit.ros.org/ (accessed Feb. 07, 2021).

[23] ‘TensorFlow’, *TensorFlow*. https://www.tensorflow.org/ (accessed Feb. 04, 2021).

[24] ‘PyTorch’. https://www.pytorch.org (accessed May 05, 2021).

[25] R. Draelos, ‘The History of Convolutional Neural Networks’, *Glass Box*, Apr. 13, 2019. https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/ (accessed May 02, 2021).

[26] K. Fukushima, ‘Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position’, *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, Apr. 1980, doi: 10.1007/BF00344251.

[27] ‘SCW’, *Supercomputing Wales*. https://www.supercomputing.wales/ (accessed May 05, 2021).

[28] *NVlabs/acronym*. NVIDIA Research Projects, 2021.

[29] J. Huang, H. Su, and L. Guibas, ‘Robust Watertight Manifold Surface Generation Method for ShapeNet Models’, *ArXiv180201698 Cs*, Feb. 2018, Accessed: May 05, 2021. [Online]. Available: http://arxiv.org/abs/1802.01698.

[30] ‘Anaconda | Individual Edition’, *Anaconda*. https://www.anaconda.com/products/individual (accessed May 05, 2021).

# Appendices

1. Third-Party Code and Libraries

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [5]. The library is released using the Apache License [6]. This library was used without modification.

Include as many declarations as appropriate for your work. The specific wording is less important than the fact that you are declaring the relevant work.

* Tensorflow
* Moveit
* Fetch lib
* Manifold
* ACRONYM
* Numpy
* OpenCv
* Pandas

1. Ethics Submission – Application Number: 18838

**AU Status**

Undergraduate or PG Taught

**Your aber.ac.uk email address**

olt13@aber.ac.uk

**Full Name**

Oliver Thomas

**Please enter the name of the person responsible for reviewing your assessment.**

Neil Taylor

**Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment**

nst@aber.ac.uk

**Supervisor or Institute Director of Research Department**

cs

**Module code (Only enter if you have been asked to do so)**

CS39440

**Proposed Study Title**

Deep learning for robotic grasp detection

**Proposed Start Date**

25 January 2021

**Proposed Completion Date**

1 June 2021

**Are you conducting a quantitative or qualitative research project?**

Mixed Methods

**Does your research require external ethical approval under the Health Research?**

**Authority?**

No

**Does your research involve animals?**

No

**Does your research involve human participants?**

No

**Are you completing this form for your own research?**

Yes

**Does your research involve human participants?**

No

**Institute**

IMPACS

**Please provide a brief summary of your project (150 word max)**

The project aims to apply deep learning to robotic grasp detection; using a deep convolution neural network fed with RGB-D (depth camera) images in order to predict successful grasps for novel objects. This will then be applied in simulation.

**Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?**

Yes

**Will appropriate measures be put in place for the secure and confidential storage of data?**

Yes

**Does the research pose more than minimal and predictable risk to the researcher?**

Not applicable

**Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?**

No

**Please include any further relevant information for this section here:**

**Is your research study related to COVID-19?**

No

**If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.**

Yes

**Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.**

Yes

**Please include any further relevant information for this section here:**

1. Tensorflow Model

def build\_model():

inputs = keras.Input(shape=(400, 400, 3))

x = layers.Conv2D(32, 1)(inputs)

x = layers.BatchNormalization()(x)

x = keras.activations.relu(x)

x = layers.MaxPooling2D()(x)

x = layers.Conv2D(64, 1)(x)

x = layers.BatchNormalization()(x)

x = keras.activations.relu(x)

x = layers.MaxPooling2D()(x)

x = layers.Conv2D(128, 3)(x)

x = layers.BatchNormalization()(x)

x = keras.activations.relu(x)

x = layers.MaxPooling2D()(x)

x = layers.Conv2D(256, 3)(x)

x = layers.BatchNormalization()(x)

x = keras.activations.relu(x)

x = layers.Flatten()(x)

x = layers.Dropout(0.35)(x)

x = layers.Dense(128, activation='relu')(x)

x = layers.Dense(64, activation='relu')(x)

x = layers.Dense(32, activation='relu')(x)

outputs = layers.Dense(6, activation='linear')(x)

model = keras.Model(inputs=inputs, outputs=outputs)

return model

1. Custom Training and Loss Functions

def training\_loop(model, dataset):

optimizer = keras.optimizers.RMSprop(learning\_rate=0.001)

loss\_obj = keras.losses.MeanAbsoluteError()

train\_loss\_results = []

epochs = 10 + 1

widgets = [progressbar.SimpleProgress(), ' ', progressbar.Bar(), ' ', progressbar.ETA()]

b = progressbar.ProgressBar(len(dataset))

for epoch in range(1, epochs):

epoch\_loss\_avg = tf.keras.metrics.Mean()

print("Epoch: {e}/{n}".format(e=epoch, n=epochs-1))

b.start()

for index, (x, y) in enumerate(dataset):

with tf.GradientTape() as tape:

inp = tf.expand\_dims(x[0], axis=0)

loss = grasp\_loss(loss\_obj, y, model(x))

# print(loss)

grads = tape.gradient(loss, model.trainable\_variables)

optimizer.apply\_gradients(zip(grads, model.trainable\_variables))

epoch\_loss\_avg.update\_state(loss)

b.update(index)

print("Epoch loss: {l}".format(l=epoch\_loss\_avg.result()))

train\_loss\_results.append(epoch\_loss\_avg.result())

b.finish()

model.save\_weights("custom\_model\_weights.h5")

plt.plot(train\_loss\_results)

plt.xlabel('Epoch')

plt.ylabel('MAE to closest grasp')

plt.show()

def grasp\_loss(loss\_obj, labels, y\_pred):

losses = []

for y in labels[:1]:

loss = y - y\_pred

losses.append(sum(loss))

closest = min(losses)

return loss\_obj(closest, y\_pred)

1. SCW SBatch File
2. Testing Table