University College London

Department of Computer Science

Master's Thesis

Learning to Grasp Under Uncertainty Using Deep Neural Networks

---

Author:
Noorvir Aulakh

Primary Supervisor:
Prof. Lourdes Agapito

Secondary Supervisor:
Dr. Vijay Pawar

September 4, 2017

Abstract

The last ten years have seen a push towards the development of autonomous grasping, an ability that would allow robots to interact with arbitrary objects in unstructured environments, especially to meet the growing demands of the e-commerce industry. In addition to improving robustness, new grasping strategies must account for the loss in robot precision as an inevitable side-effect of the need to lower component costs. We build on top of recent work in learning quality measures on candidate grasps, to account for uncertainty in the pose of a parallel-jaw gripper. We present a clustering based approach to generating quality labels from analytical grasp metrics computed in simulation. These are then used to train a convolutional neural-network on synthetic depth images. We define a quality function over the object to be grasped and smooth it by convolving with the uncertainty in gripper pose. We use the Cross-Entropy Method for iteratively generating new candidate grasps, using quality labels predicted by our neural-network. Our approach demonstrates that using a distribution over grasp qualities rather than binary labels, results in the network learning a grasp heuristic that allows it to identify regions with higher probability of success, when operating under uncertainty.

Word Count: 14,400

Acknowledgements

This project would not have been possible without the support, both academic and emotional of a number of people. I would like to thank:

- My supervisor Prof. Lourdes Agapito for her invaluable guidance throughout this project.
- Dr. Vijay Pawar for providing the initial motivation and resources necessary to complete our work.
- Dr. Edward Johns at the Imperial College London for his insightful counsel at key points in this project that helped steer it in the right direction.
- Jeff Mahler at the University of California, Berkeley for sharing details of their approach to training deep neural-networks for robot grasping.
- Joan Martinez for incisive discussions on statistical measures.
- My family, for their unwavering love and support throughout the duration of this project.

Contents

Chapter 1

Introduction

The dexterity with which humans interact with their day-to-day environment makes it hard for one to appreciate the evolutionary accomplishments that make this possible. Our brains have developed over millions of years to become proficient at wielding tools — a skill that many argue is fundamental to the dominance of our species. Seemingly simple tasks such as operating a hand-saw, prove to be incredibly challenging even for some of our closest relatives in the animal kingdom [1]. However, despite being evolutionarily endowed with these amazing abilities, we are not born with them. In addition to the physical capability, there is a learning component involved. Anyone who has seen a toddler struggle with a spoon or fail to put two Legos together can readily appreciate this. Our brains learn to process enormous amounts of perceptual information, as well as the physics of the surrounding world and our bodies, to run hand-eye-coordination control loops at millisecond timescales.

Our own prowess at grasping makes one wonder why Rosie the robotic maid is not already a permanent fixture in our households[1]. Indeed it is not hard to motivate the usefulness of dexterous robots in our day-to-day environments. Robots that could fold your clothes [2], empty your dish-washer [3] or cook dinner would free up an enormous amount of time and would likely receive wide-spread adoption. Hostile working environments present another potential application where dexterous robots could prove to be indispensable. Search and rescue operations during natural catastrophes or containment operations in disaster areas, such as the Fukushima Nuclear reactors, are standing calls for such technology.

Perhaps the most immediate economic driver for progress in grasping is the rapidly expanding e-commerce sector. Companies such as Amazon and Ocado have transformed

[1] Rosie is a fictional robotic maid popularised by the animated T.V. series, *The Jetsons* dating back to the early 1960s.

the scale and logistics of warehousing. However, in contrast to industrial factory floors, where processes are highly controlled and repetitive, these warehouses feature unstructured environments with high variability in operating conditions. This makes it challenging to automate parts of the process that require the handling and manipulation of objects — an area where robots could yield enormous returns on investment. Like the DARPA robotic challenges for disaster relief [4], the Amazon Picking Challenge [5] is an example of the numerous commercial and government incentives in place to speed up advancement of this field.

Yet, despite such powerful incentives, progress has remained relatively limited until recently. Since some of the first research into robotic teleoperation in the 1950s, the *modus operandi* for grasping systems, similar to the case in computer vision, was based on hand-crafted features and analytical metrics. Although this engendered enormous progress in our understanding of the field's governing principles, these methods proved too inexpressive to represent the complex, high-dimensional functions that constitute these tasks. In recent years, however, *artificial-neural-networks* have emerged as revolutionary tools capable of learning such complex representations, by taking inspiration from the operating principles of the mammalian brain.

Modern neural-networks, also known as *deep*-neural-networks, have achieved remarkable success at pattern-recognition tasks — the most well known benchmark for which is the ImageNet classification challenge [6]. This capability at pattern-recognition has been rapidly adopted in numerous fields to yield state-of-the-art results [7], [8], [9].

In this project, we aim to present a concise overview of the history and challenges in robotic grasping, along with some of the latest attempts at addressing them. In particular, we present a neural-network approach to accounting for uncertainty in grasping. This presents an area of increasing relevance, as low-cost robots attain growing adoption in both industrial as well as domestic settings. The scalability of dexterous robots is likely to trade-off the precision at which most industrial robots operate, thereby introducing higher levels of positional inaccuracies. We expand on previous work in this area [10], and present a novel approach at describing the non-linear relationship between grasp-quality and success.

1.1 Objectives

Our goal is to develop a technique to define the non-linear relationship between theoretical grasp-quality metrics, calculated in simulation, and the eventual outcome of an

attempted grasp. To achieve this, we subdivide the objective into the following list of sub-goals:

1. ***Assimilate the fundamentals of robotic-grasping***: Grasping is a vast field with detailed theoretical constructs that are still relevant to our work, despite the revolutionary new direction afforded by deep-neural networks. Our first aim is to understand this theoretical literature in the hope to gain insights to facilitate new developments.

2. ***Explore existing solutions***: In addition to traditional grasping literature, we also need to understand the theoretical and practical underpinnings of modern techniques. In particular, training and debugging deep-neural networks on simulated data.

3. ***Classify uncertainty in grasp-quality***: We aim to create a new data-set of candidate grasps labelled with normalised grasp-quality scores. To do this, we need to define the relationship between standard grasp-quality metrics and real-world outcomes in a way that is trainable on a deep-neural-network. Neural-networks have been shown to perform better at classification rather than regression tasks [10]. We therefore discretise our grasp quality scores.

4. ***Implement a deep-learning framework***: An important stepping-stone for us is implementing an efficient deep-learning pipeline that allows for training our classifier on a large data-set in a reasonable amount of time.

5. ***Learning a grasp function***: We take a step-by-step approach to learning a representation on our data-set using deep-learning models pre-trained on similar tasks (*transfer learning*).

6. ***Evaluate the approach***: Lastly, we aim to provide an evaluation of our technique and discuss its failure modes and potential pitfalls.

1.2 Challenges

The task we aim to solve in this project is challenging on many levels. Not only do we deal with learning the semantics of a complex task in simulation, but we also face tough implementation challenges in the short time-frame afforded by the project deadline. A list of key challenges faced in this project is given below:

- ***Complexity***: As alluded to earlier, robotic grasping is a complex task that lies at an intersection of multiple fields. It requires one to understand the interplay

between these fields along with their practical motivations. We came into this project with no experience in the field and only a theoretical familiarity with the associated sub-fields of computer-vision and machine-learning. Unravelling the task in all its complexity to make worthwhile contributions was therefore a time-consuming process.

- ***Non-linear scoring function***: As far as we know, there is no well-defined distribution for the popular grasp-quality metrics in literature. These functions often vary non-linearly with the location of the grasp on a candidate object. Furthermore, since there are a lot more ways of incorrectly grasping an object than the vice versa, data-sets of these labelled grasp metrics are often skewed (as in our case). Training a classifier in light of these factors is close to impossible. We therefore had to pre-process the data-set and develop a novel approach to aligning the quality metrics and candidate grasps to attain practical outcomes.
- ***Learning a representation***: Training and debugging deep-neural-networks is often an opaque process, requiring the manual tuning of numerous hyper-parameters. Additionally, given the large size of our data-set and highly constrained resources, we had to develop an efficient data-input pipeline that would be the difference between a week in training time versus a few hours. These two factors were the most time-consuming parts on an already constrained project deadline.

1.3 Contributions

Our contributions to the field are the following:

1. ***Novel approach to classifying uncertainty in grasp-quality***: We present a clustering based approach to binning grasp-quality scores to account for uncertainty, along with results on a number of other methods. We discuss both the data-generation and pre-processing pipe-line.
2. ***Data-input pipeline and code repository***: Since a considerable portion of time in this project was spent on optimising the data-input pipeline for skewed data-sets, we open-source our software for future use. The repository is written in a modular fashion that allows one to train multiple different architectures by only changing a configuration file. It can be found at:

   `https://github.com/noorvir-a/grasp_ucl`

In addition to these, we hope that this thesis serves as a helpful text in summarising forty years of development in the field, as well as its integration with modern tools.

1.4 Outline

In the next chapter, we cover the history of grasping, from the first attempts at autonomy to the theoretical developments that lie at the heart of the field. We then briefly introduce artificial-neural-networks and discuss their use in the current state-of the-art. In Chapter 3, we present an overview of deep-learning frameworks and the data-sets used in this project. We present a novel scoring metric to classify the uncertainty in grasp-quality from pre-existing metrics. In Chapter 4 we describe the data-input pipeline and our step-by-step transfer learning approach to training multiple convolutional neural-networks. Finally, we evaluate our approach, identify failure modes and suggest topics for future work.

Chapter 2

Background

Teaching robots to grasp has been an active field of research for over four decades. To make progress in a field, it is essential to understand the fundamental principles that underlie it and form its history, even when some of the established techniques are abandoned in favour of a new, superior method.

In this chapter we discuss the history and prior work in the field of grasping, the renaissance of neural-networks, their working principles, and recent surge as the foremost tool in grasping research — replacing the more hands-on analytical techniques prevalent a mere ten years ago.

2.1 The Prehistory of Grasping

Early research efforts into grasping sub-divided the field into roughly three parts [11]:

1. ***Haptics***: this relates to the facility of touch as a mode of perception that forms a critical component in dexterous manipulation. Haptic communication is so important to humans that it branched off as a field of its own, with the rise of computer technology and the pervasiveness of human-computer interaction.

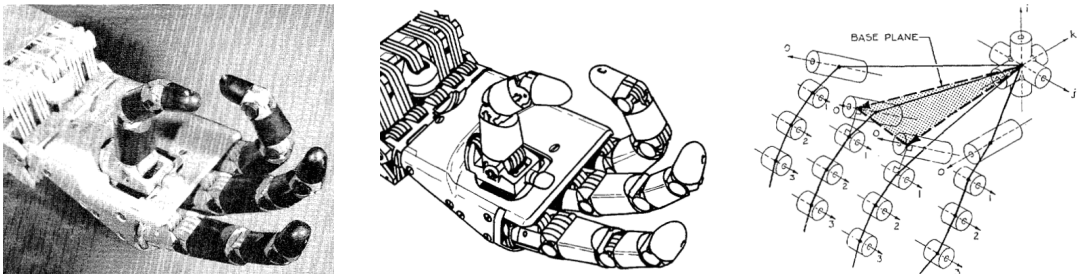
2. ***Fixturing***: the task of restraining objects in a given pose. This is important in a wide-range of applications and is a pre-requisite for dexterous manipulation.

3. ***Dexterous Manipulation***: interacting with the physical world in all its complexity, often with under-actuated systems.

One of the earliest contributions to the field of grasping, one that is still widely employed, was made by the German engineer Franz Reuleaux in his 1875 paper entitled *Theoretische Kinematic* [11]. He introduced the idea of *form-closure*, wherein a combination of forces resolved on the geometric shape of an object can counteract any disturbing forces. Form-closure is one of the two primary restraint properties of an object; the other being *force-closure*. These two properties differ in that, form-closure assumes frictionless surfaces, and therefore requires a greater number of contact-points to ensure equilibrium. Force-closure on the other hand, needs fewer contact point but requires the magnitude of the disturbing forces to be less than the frictional force between the contact surfaces (determined by the coefficient of static friction) [12].

Reuleaux showed that for planar shapes, you need four contact points for complete form-closure. Mishra *et. al.* [13] extended this result to spatial objects with piece-wise smooth boundaries by setting an upper-bound of twelve frictionless contact-points; Somoff had already defined a lower bound of seven in 1900 [14].

These restraint constructs allow us to define fixturing as a fundamental building block towards more complex dynamic *hands* capable of human-like dexterous manipulation. This is in-fact the route some of the first researchers in grasping took in a multidisciplinary effort that resulted the 25 degrees-of-freedom UTAH/M.I.T hand [15]. Constructing such a high degree-of-freedom anthropomorphic hand was partly motivated by the results of Reuleaux, Somoff and Mishra, which indicated the need for versatile end-effectors to ensure form-closure. Additionally, the aim was to enable research into control-systems design and manipulation theory, with the eventual goal of achieving human-level dexterity. Yet, over thirty years after its initial publication, this goal remains unfulfilled.



(A) (B) (C)

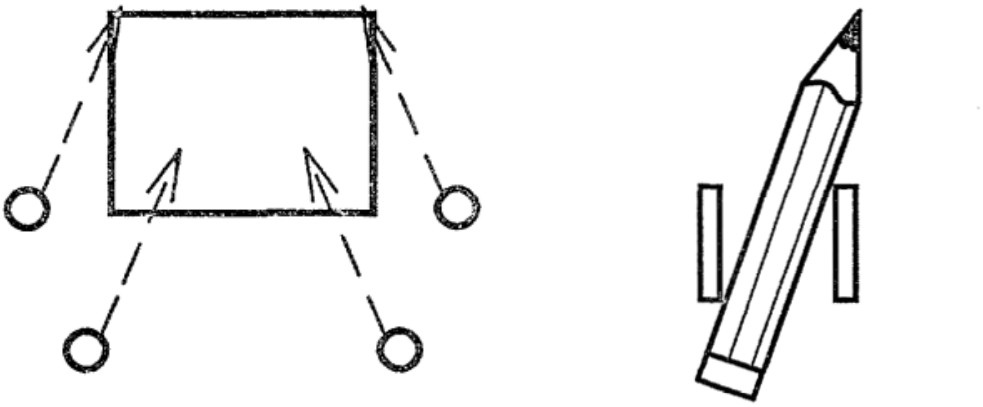
FIGURE 2.1: The UTAH/M.I.T hand [15].

Researchers had underestimated the complexity of the task at hand. Grasping in the anthropomorphic sense spans an enormous state-space, since the high degrees-of-freedom result in an over-defined problem; there are multiple grasp configurations with equivalent

end-results. This is easily motivated by trying to pick-up an object lying close to you; there are, almost always, multiple ways of doing so. In addition to solving for a high-dimensional state-space, the problem was compounded upon by the complex physical dynamics of contact surfaces between the hand and the target object, which could not be sufficiently modelled without appropriate tactile feedback. The combination of these factors resulted in a problem that required new mathematical insight, sensor technology and computational capability of the order that remains unattainable even today, much less in the 1980s [16].

2.1.1 A Simplified Approach

A number of research groups moved towards tackling a less complex problem by focusing on under-actuated, multi-finger manipulators while the versatile, high degrees-of-freedom ones found utility in teleoperation and fundamental research, particularly for space applications [17]. Mason and Sailsbury [18], did seminal work on operating with low degrees-of-freedom arms using only the mechanics of the end-effector to account for uncertainty in the dynamics of grasping. They argued that rather than relying on sensory input to reduce uncertainty, one could use the mechanical properties of the gripper as a simple alternative to cover a large number of cases. Figure 2.2 illustrates this.



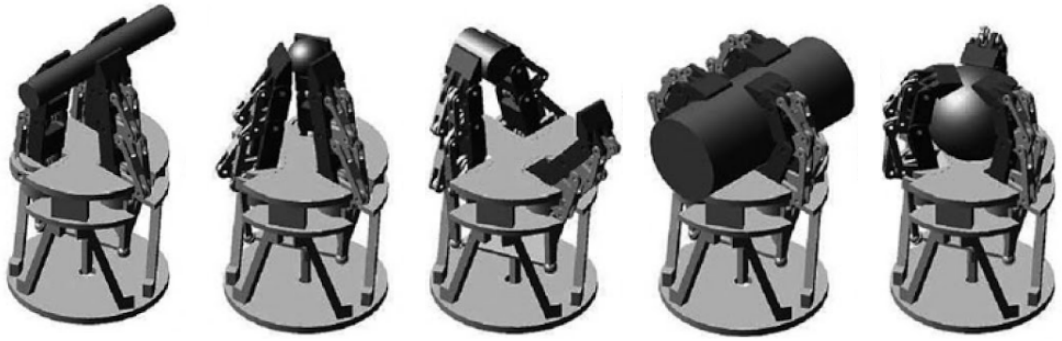
(A) (B)

FIGURE 2.2: Accounting for uncertainty using purely mechanical means [19].

This simple but powerful insight led to extensive work in this direction over the next thirty years, resulting in a number of popular *parallel-jaw* and *three-finger* grippers. New hardware engendered research into the development of new grasping semantics to maximise the utility of these simplified grippers and formalise the various possible grasp configurations. The two most important and widely adopted configurations are the *pinch*

and the *enveloping* grasps [20], [21]. The motivation for these is derived from human grasping, which they aim to replicate in its fundamental form.

Humans tend to use pinch grasps when executing precise operations on small multi-faceted objects, while using enveloping grasps to handle larger, heavier objects where grasp stability is important. In other words, pinch grasps are relevant in dexterous manipulation whereas enveloping grasps serve better in fixturing. These characteristics are evident in the taxonomy of grasping, wherein the pinch and enveloping grasps fall under the *precision* and *power* classes respectively. Figure 2.3 illustrates these configurations using the Self-Adapting Robotic Auxiliary Hand (SARAH), which was designed as an easy to control, yet high degree-of-freedom robot hand for special purpose dexterous manipulations aboard the International Space Station [17].



(A) (B) (C) (D) (E)

FIGURE 2.3: (A-C) Pinch and (D-E) enveloping grasps executed on the SARAH [22].

Grippers such as the SARAH are so popular and numerous today that a complete review of the field is impossible. However, since the geometry of the gripper is central to the grasps that can be executed with it, we briefly present the ones most relevant to our work, with visualisations in Figure 2.4.

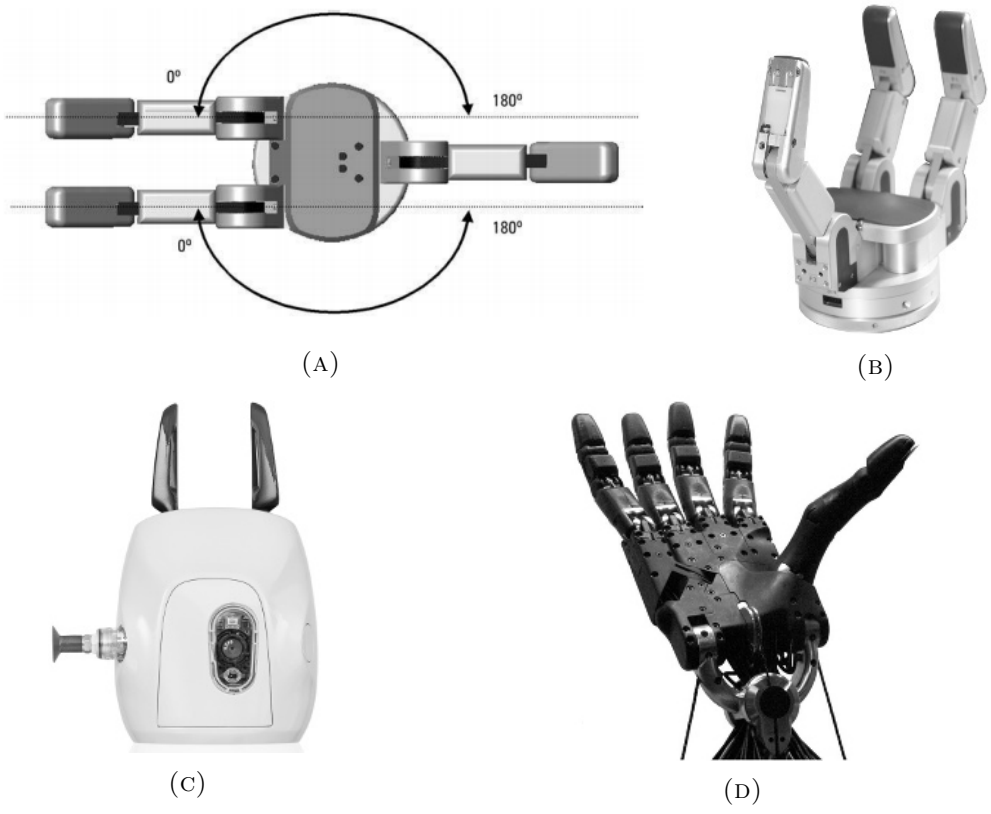
The Barrett Hand Figure 2.4b [23], along with the SARAH, is one of the most popular three-finger grippers in industry and academia today, with a number of associated grasp data-sets (Section 2.4). It is a three-finger, eight-axis gripper that was created with the goal of being flexible and multipurpose; it is capable of handling a wide range of tools without explicit configuration. Owning to its reconfigurable design, it enables both pinch and envelop grasps by adjusting the spread angle of the fingers, as illustrated in Figure 2.4a.

Contrary to what one might expect, the Barrett hand cannot be used to exactly replicate parallel-jaw motion (such as the Yumi Gripper Figure 2.4c). The fingers are under-actuated, with the inner and outer joints closing in the ratio 3-to-4 until one of the

links strikes an obstacle. Although this approach lends to an easier control strategy, it precludes generalisation in a machine-learning setting, as discussed in Section 3.3.

The Yumi parallel-jaw gripper [24] might be considered the two fingered analogue of the Barrett Hand. Although there are no significant distinguishable features for different two-finger grippers, the most notable one is the contact surface on the finger-tips. Guo *et al.* [25] provide an in-depth analysis of the effect of different finger-tip surface materials and print-patterns on gripping performance. We present the Yumi gripper primarily because our data-set (Section 3.3) was generated using its finger-tip geometry.

Lastly, we discuss the ShadowHand [26] as a modern-day parallel to the UTAH/M.I.T. hand. Although it has twenty-four degrees-of-freedom, we include it here since it is the only gripper available to us and because its versatility and individual control of each Distal, Proximal and Metacarpal joints allows us to approximate parallel-jaw motion.



(A) (B)

(C) (D)

FIGURE 2.4: (A-B) The reconfigurability of the Barrett Hand [23] makes it one of the most popular grippers in academia. (C) The Yumi Gripper [24] (D) ShadowHand [26].

Although new gripper geometries are proposed frequently today, the mathematical techniques, formalised in the 1980s—90s, for controlling under-actuated hands, continue to underpin the development of modern approaches prevalent today. The SARAH for

instance, uses these techniques to control its ten degrees-of-freedom with only two actuators; it operates by allowing each finger to conform to the shape of the object, using the self-adaptability of the phalanges; it implicitly couples multiple degrees-of-freedom to achieve the desired grasp [17], [27]. In the case of the enveloping grasp, these self-adapted contact points serve as secondary restraints and ensure the theoretical minimum number of points required to achieve force and form-closure, as described by the work Reuleaux, Somoff and Mishra.

The unanswered portion of the grasping problem remains in identifying the *primary* points of contact, essential in executing both the pinch and enveloping grasps. This relates to the development of various grasp-quality metrics discussed in Section 2.2.

An analysis of these metrics suggests that two and three-finger grippers should be sufficient for simple pick and place tasks. The difficulty therefore must lie in robustly identifying the primary contact points. This has been an open problem for many years; its intractability lies in perceiving and interpreting the robot's environment, and planning a collision-free approach towards retrieving the object.

A standard approach to perception in robot grasping is to construct a 3D point-cloud of the scene using depth cameras, and segment the candidate object for further processing. Early techniques used various analytic and similarity based methods for grasp planning. Miller *et al.* [28], for instance, extract shape primitives, such as cylinders, spheres, etc. from the segmented object and plan multiple candidate grasps on this simplified geometry. They then analyse these grasps using the *GraspIt!* [29] simulator and choose the one with the highest score. Li *et al.* [30] use a shape matching algorithm that compares local patch information to match surface normals and collections of features with similar relative placements, to align grasps with object shapes. Goldfeder *et al.* [31] use a similar approach by defining a shape descriptor for 3D range data, built on top of SIFT descriptors[1], to measure similarity between acquired point-clouds and models in a pre-computed database.

Most of these techniques are predicated on large data-sets of labelled grasps. The first steps towards generating such a data-set were provided by the introduction of the *GraspIt!* simulator. *GraspIt!* allows the user to test candidate grasps or plan new ones by initialising them from a prehensile hand posture, given the object and gripper geometries. It was used to generate The Columbia Grasp Database [33], used in a number of aforementioned papers. OpenRave is a related simulator that automates the grasp planning and testing pipeline and does not rely on a simplifying prehensile posture initialisation [34]; we use it to generate the data-set in our work.

[1] Scale-Invariant Feature Transforms are prominently used in computer vision to describe local features in images [32].

While these techniques provided essential foundations for data-driven grasping, the grasp accuracy achieved was limited to 60—70%. One possible explanation is that the dynamics between the gripper and an object are too complex to approximate using similarity or shape primitive based fitting. What is required is to learn the semantics of grasping in a way that generalises to new objects with complex geometries. Bohg and Kragic [35] presented one such approach using logistic regression and Support Vector Machines trained on synthetic images. Although they achieved 85% grasping accuracy, their method was limited to a few known objects and did not generalise.

The shortcomings of the techniques discussed so far lay in the inability to learn a sufficient function approximation for grasping arbitrary objects. The key breakthrough came with the success of deep neural-networks as the foremost tools for solving the similar but unrelated task of image classification. Although most of the theory behind neural-networks had been formalised since the 1980s, it is only recently that the improved computational capabilities of modern computers allowed them to achieve unparalleled performance as complex high-dimensional function approximators. Following their renaissance with Krizhevsky *et al.'s* 2012 paper [36], deep neural-networks have been so successful in learning representations for spatial information (like images and point-clouds) that within five years, most grasping literature now employs them at some level.

Owing to the relevance of neural-networks to our work and the somewhat nuanced optimisation techniques employed to train them, we briefly discuss their theoretical underpinnings in Section 2.3. First, however, we discuss the theoretical construction of the grasp-quality metrics we wish to learn with them.

2.2 Grasp Quality

Grasp-quality metrics attempt to categorise the stability of candidate grasps. They describe *how well* a grasp can resist disturbances [37]. Central to defining these metrics is the concept of *wrenches* and the *Grasp Wrench Space*:

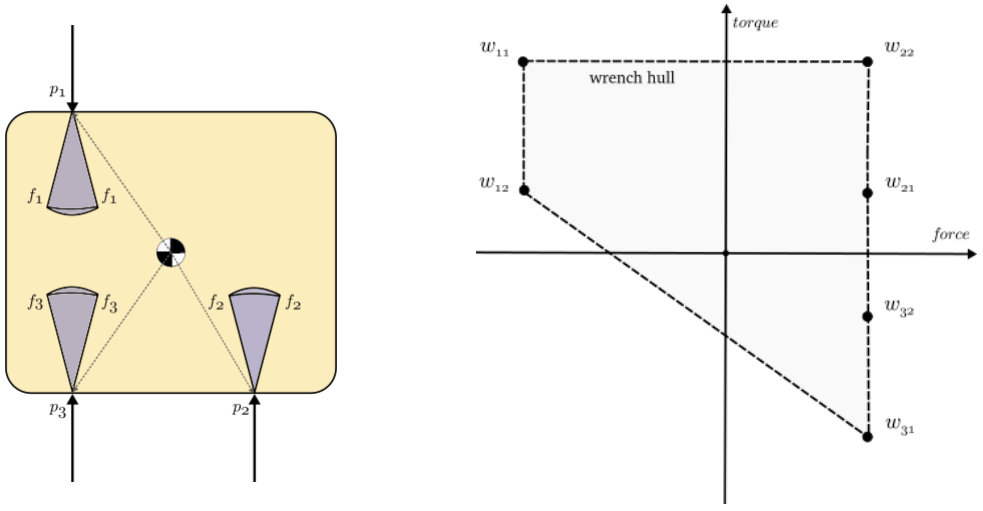
Wrench: A vector $w_i \in \mathbb{R}^6$, composed of the force and torque components experienced by object when a point contact applies a force to it [38].

$$w_i = \begin{pmatrix} f_i \\ \tau_i \end{pmatrix} = \begin{pmatrix} |f_i| \cdot \hat{n}_i \\ |\tau_i| \cdot \frac{p_i - c}{|p_i - c|} \times \hat{n}_i \end{pmatrix} \tag{2.1}$$

where $f_i \in \mathbb{R}^3$ is the force applied to the object at point p_i and $\tau_i \in \mathbb{R}^3$ is the resulting torque at the centre-of-mass of the object located at point c in world coordinates. $\hat{n}_i$ is the direction of the point contact force into the body of the object, normal to the surface tangent. If the object is acted upon by more than one point forces, the net wrench w is given by a linear combination of all wrenches N:

$$w = \sum_{i=1}^{N} w_i \tag{2.2}$$

Grasp Wrench Space: The set of wrenches that can be applied to an object at each point contact. The boundary of the grasp wrench space is defined by a convex-hull. The position of the origin of the space with respect to this convex-hull (wrench-hull) determines form-closure. The wrench space for a 2D object is three-dimensional, where as that for a 3D object is six-dimensional[2]. Figure 2.5 illustrates a toy example demonstrating wrench space analysis for three point forces.



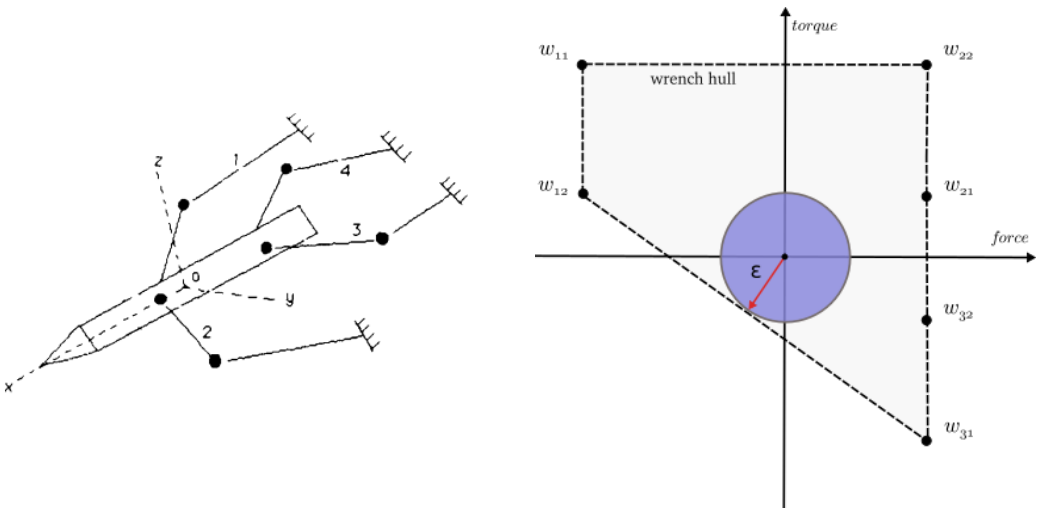
(A) (B)

FIGURE 2.5: Toy example demonstrating wrench space analysis. (A) An object acted upon by three point forces is analysed for force-closure by approximating friction cones (purple) and moments about the centre of mass. (B) The wrench hull of the object exists in a 2D torque-force space.

Li and Sastry [39] presented one of the early analyses of task-specific optimality and stability evaluation by modelling grasps as task-ellipsoids in the wrench space of the object. While their metric was task-specific (for example grasping a pencil), in 1992, Carlo Ferrari and John Canny introduced a new generalised grasp-stability metric called

[2] 2D-wrenches: composed of two-dimensional forces and one-dimensional torques.
3D-wrenches: composed of three-dimensional forces and three-dimensional torques.

the ϵ-metric or the Ferrari-Canny metric — one of the most popular quality measures for grasp stability and the basis for the one used in this project [40].



(A) (B)

FIGURE 2.6: (A) The weakest direction for resisting disturbances in along the x-axis of the pencil [39]. (B) The robustness of force-closure is quantified using the ϵ-metric. It can be interpreted as the radius of the largest circle that can be enclosed in the wrench hull of the object.

The ϵ-metric encodes information about how effectively a candidate grasp can resist disturbing wrenches in its weakest direction. In the case of the pencil in Figure 2.6 for instance, this might be along the x-axis rather than the z-axis since only the frictional forces restrain it in that direction. Geometrically, this can be interpreted as the radius of the largest ball, centred at the origin, that can be enclosed in the wrench hull of the object [37]. Refer to [40] for a derivation of the ϵ-metric.

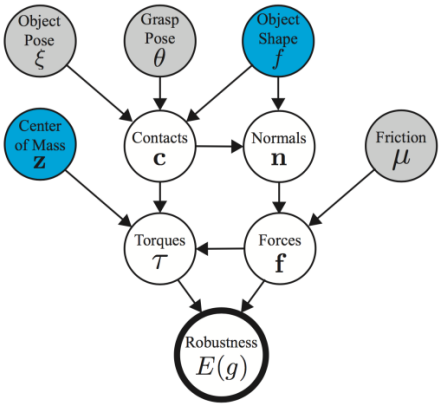


FIGURE 2.7: Graphical model depicting the calculation of the *robust* ϵ-metric. The model incorporates uncertainty in estimation of grasp metrics to generate a robustness score. [41]

The final property of relevance to this project is the robustness measure of a grasp. Robustness in the context of grasping is defined as the probability of grasp-stability in the presence of perturbations to gripper pose, estimated by sampling contact points in the neighbourhood of the nominal grasp points [41]. In effect, it accounts for positional uncertainty of the gripper fingers. Seita *et al.* [41], employ a probabilistic approach to estimate a robust ϵ-metric to generate the DexNet-1.0 grasp data-set — a derivative of which is used in this project. Their approach is described in Figure 2.7, where the model is sampled one-hundred times to compute the robust ϵ-metric, $E(g)$ for each grasp candidate.

2.3 Artificial Neural Networks

Artificial-neural-networks are node based computational constructs that were originally inspired by the structure of the mammalian brain. The individual nodes, also called units or neurons, that make them up were motivated by the eponymous structures in the brain. Biological neurons, also called nerve cells, are particular in their elongated shape, and tentacular signal transmitters and receivers. The anatomy of undifferentiated nerve-cells is visualised in Figure 2.8: they are composed of a cell-body (*soma*) which branches numerous *dendrites* and a single *axon* which might it-self branch multiple times before terminating. Nerve cells communicate through electrical signals transmitted from the axon of one neuron to the dendrites of the other across specialised connections called *synapses*. The highly branched nature of the dendrites and axon-terminals allows multiple neurons to receive signals from a signal axon — a broadcast signal in effect.

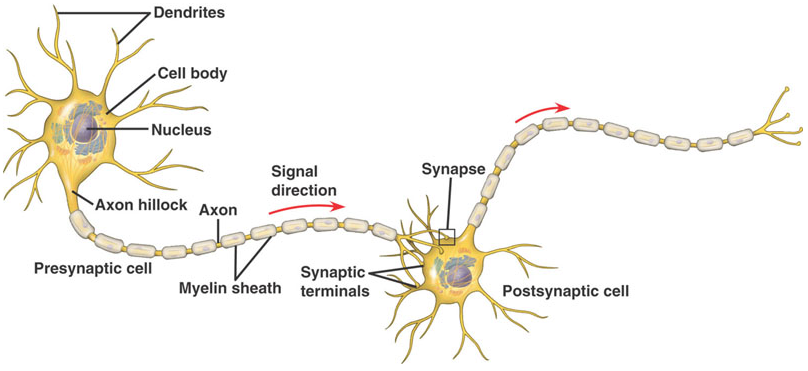


FIGURE 2.8: The structure of a biological neuron [42].

Similar to their biological counterparts, artificial-neurons receive inputs from multiple neurons and produce a single output, that might be broadcast to multiple other neurons. They can be thought of as computational units that apply a function (*activation*) to

the input. While biological neurons *learn* by strengthening the synaptic connections between networked neighbours, artificial neurons can be be *trained* by adjusting the weights on the input channels of each neuron. By chaining together these broadcast connections between individual neurons, one can create a large networked graph capable of performing massively parallel computations. A schematic of what such a graph might look like is given in Figure 2.9.

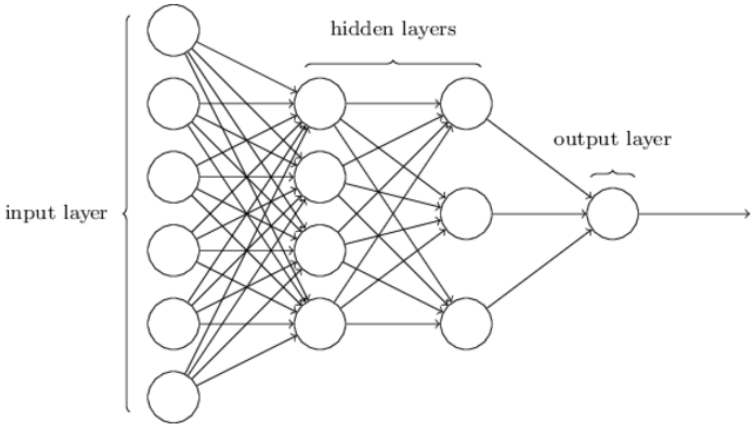


FIGURE 2.9: Schematic of a simple artificial-neural-network [43] with six inputs, two hidden layers and one output.

Each arrow depicts a weighted connection between the output (axon) of a neuron in the previous layer and the input (dendrites) of neurons in the current layer. In the simplest case, the activation of each neuron can be considered to be a unit-step function. For weights w_{ij}^l relating the output x_i of the i^{th} neuron in layer $l-1$ to the j^{th} neuron in layer l, the equations relating the transmittance of information from one layer to the next, left-to-right (forward-propagation) in the network are given by:

$$x_j = \begin{cases} 0 & \text{if } \sum_i w_{ij}^l x_i \leq \text{ threshold} \\ 1 & \text{if } \sum_i w_{ij}^l x_i \geq \text{ threshold} \end{cases} \tag{2.3}$$

Since these connections represent linear transformations, they can be expressed concisely as matrices. The affine transformation from layer $l-1$ to layer l can then be expressed using the following equation, where $\mathbf{b}$ represents the bias term[3].

$$\mathbf{x}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b} \tag{2.4}$$

[3] The bias term translates the entire distribution; for a 2D function this equates to shifting the function along the x-axis.

We can therefore represent large neural networks as simple chained linear transformations. Although Figure 2.9 only depicts 14 neurons, in theory, there is no upper-bound on this number — estimates of the number of neurons in the human brain vary between 85 billion and 105 billion [44]. Artificial-neural-networks are universal function approximators [45] and can exactly compute *any* function in the limit as the number of units goes to infinity. However, training these networks beyond a certain number of units becomes computationally intractable. This was the primary bottleneck that stifled the progress of neural-networks for forty years after the introduction of the backpropagation algorithm [46] — the technique used to train them.

2.3.1 Backpropagation

Although the backpropagation algorithm was independently described by multiple authors in the field of control theory in the 1960-70s [47], [48], [49], its utility in training neural-networks was popularised in the seminal paper by Rumelhart *et al.* in 1986 [50].

The backpropagation algorithm relates the cost-function C, of the output of a layer to its weights $\mathbf{W}$ and $\mathbf{b}$, by the partial derivative $\partial C/\partial \mathbf{W}$. Using the chain-rule, this expression can be generalised to compute the gradients of error propagation of the cost-function with respect to the weights in any layer of the network. In essence, we compute the contribution of the weights in each layer of the network to the error in the output of the final layer. This allows us to construct a gradient field relating the output of the network to its weights, which can then to used to minimise the cost-function using an array of numerical optimisation techniques. For a detailed discussion of the equations of backpropagation, refer to the excellent text by Michael Nielsen [43].

2.3.2 Optimisation

Given a cost-function and its associated gradient field, one of the most popular methods to minimise it is using the Gradient Descent Algorithm. Gradient Descent is a first-order optimisation technique — it relies on the first-order partial derivatives of the cost function. As discussed in the previous section, the partial derivatives $\partial C/\partial \mathbf{W}$, tell us the direction in which the cost-function changes with respect to the network weight. The gradient-descent algorithm allows us to step in the direction that best reduces the cost-function, which might itself lie on a high-dimensional manifold. This means that as long as we can define a cost-function that is Euclidean at all points on the manifold, we can use gradient descent to minimise it. This however does not guarantee that we will find the optimal solution; the solution might settle at a local minimum. Figure 2.10 motivates the intuition behind gradient-descent on a cost-function in three dimensions.

So far we have assumed the gradient-field is defined on the entire cost-function at each step of gradient-descent. In the context of the training data-set, this translates to computing gradients on the entire set simultaneously — also known as batch gradient descent. As we discuss later, for modern neural-network architectures, the size of the data-set immediately renders this approach computationally intractable. A popular alternative is to use *mini-batch* gradient-descent. Instead of computing exact gradients on the entire data-set, it uses a small subset of data to approximate the global gradient field. The errors introduced through this approximation cancel out over multiple iterations. In practice, this approach is several orders of magnitude faster than batch gradient-descent with no significant loss in accuracy [51]. When each mini-batch is composed of a single data-point, the method is referred to as *stochastic* gradient-descent. However, this distinction has become increasingly vague, with most numerical optimisation software using the term stochastic to refer to both cases.

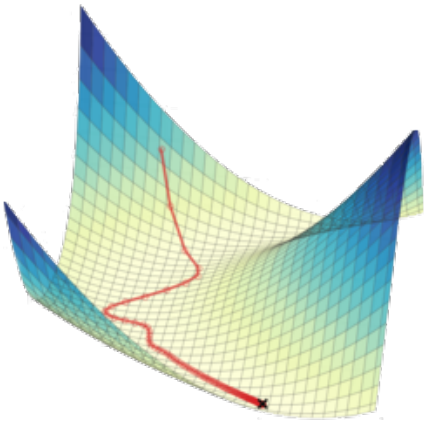
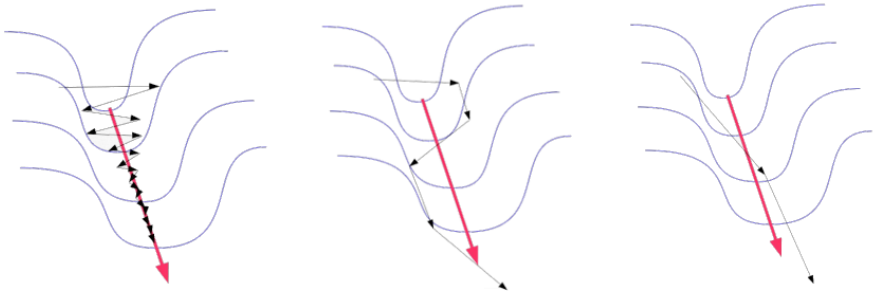


FIGURE 2.10: Intuition behind gradient descent for a three-dimensional cost function [52].

There are a number of variants of the gradient-descent algorithm used today. The two most relevant to our work are:

1. ***Momentum***: A shortcoming of gradient-descent is that it does not take advantage of the curvature information of the cost-function; it treats low-curvature (flat) regions in the same way as high-curvature ones. The Momentum algorithm takes this into account and uses it to accelerate gradient-descent in the direction of consistently downward pointing gradients. It can be thought of as an approximation to second-order optimisers. The idea is motivated by imagining a ball rolling down a hill. Figure 2.11 provides an illustration.

2. ***Adam***: Tuning the hyper-parameters (learning rate, momentum-rate etc.) of a neural-network can often be cumbersome. The Adam optimiser introduced by Kingma and Ba in 204 [53], keeps track of the gradients for each weight in the network and adaptively updates the learning rates from the estimates of the first and second moments of these gradients. It therefore minimises the manual tuning required by the user by automatically adjusting these parameters during training. The trade-off however, is the computational time and storage required to compute these estimates.



(A) (B) (C)

FIGURE 2.11: Accelerating gradient descent with curvature heuristics. (A) Standard gradient descent takes uniform steps along the cost-function. (B) The Momentum optimiser uses a heuristic to speed up optimisation by giving the optimiser inertia in the direction of optimisation. (C) The ideal optimisation step theoretically achievable using higher order curvature information.

As mentioned earlier, gradient descent is a first order optimiser; it ignores local curvature information about the cost-function and provides pessimistic estimates for the gradient direction. The Generalised Gauss-Newton algorithm on the other hand, uses second-derivatives to improve these estimates. However, computing second-derivatives has a time complexity of $\mathcal{O}(\mathcal{N}^2)$ compared to $\mathcal{O}(\mathcal{N})$ for first derivative. In practice, this proves to be too computationally expensive to be viable.

2.3.3 Deep Learning

Deep-learning refers to the branch of machine learning dedicated to training neural-networks with multiple hidden layers. The number of hidden layers published in the literature ranges mostly between 4 and 150 [54]. In recent years, deep-neural-networks have achieved state-of-the-art performance in a wide variety of fields, starting with image classification [36] and progressing to applications in speech recognition, translation [7], robot navigation and grasping [55], [56]. The success of these networks can be attributed to the aforementioned universal approximation theorem and the observation that adding

more layers (depth) to a network gives it more expressiveness than adding more units to a layer (width) [57]. The improvement in computational resources and the advent of *big-data*, propelled neural-networks from their status as a forgotten tool from the 1980s to the most popular method for pattern creation (generative networks) and recognition (discriminative networks) available today.

2.3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are one of the most popular and powerful neural-network architectures available today. Their success lies in the ability to process the spatial structure of data, such as images and audio, both of which form prominent communication modalities in our society. CNNs are based on three key concepts:

Local Receptive Fields:
The neural-network architectures discussed earlier consider layers in which each unit in layer l is connected to all the units in layer $l-1$; the network is *fully-connected*. In CNNs instead, each unit in layer l is connected to a spatial region in layer $l-1$, known as the *local receptive field* of the hidden unit. This spatial region, also known as a kernel or a filter, is then slid across the entire input layer to produce the next hidden layer. This concept is visualised in Figure 2.12.

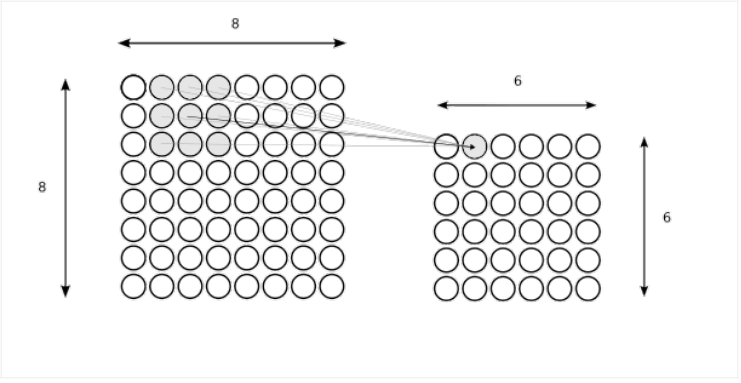


FIGURE 2.12: The local 3x3 receptive field of a hidden neuron.

Each connection in the receptive field learns a weight which is used to generate to value of the hidden neuron. The hidden neuron can be thought of as analysing a particular region of the input space. Given input activations a_{ij}^{l-1}, kernel weights w_{mn} and bias b, the output for a neuron with a *tanh* activation can written as:

$$a^l = tanh(b + \sum_{m=1}^{3}\sum_{n=1}^{3} w_{m,n} a_{i+m,j+n}^{l-1}) \tag{2.5}$$

Shared Weights:
The weights of the connections learned in a receptive field are shared across all neurons in the subsequent hidden layer. This is the fundamental element that allows CNNs to detect features in the input image wherever they might appear, since each filter is slid across the entire image. This feature also means that there are significantly fewer weights to learn during training, which allows construction of deep networks.

Shared weights however mean that all the neurons in the hidden layer detect exactly the same features; they are a *feature map* from the input layer to the hidden layer. To allow the network to learn additional features, CNNs contain multiple feature-maps or channels, that can be imagined as a 3D volume of weights. Figure 2.13 illustrates this for the case of three channels.

Pooling:
The last key component of CNNs are pooling layers. Pooling layers operate on the output channels of the convolutional layers, and apply an kernel that performs some sort of summarising operation on its receptive field. A common approach is to choose the maximum value from the kernel — referred to as *max-pooling*.

A complete step of the pipeline is visualised in Figure 2.13.

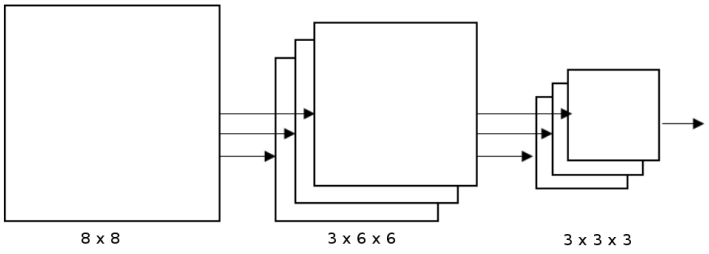


FIGURE 2.13: Pipeline for convolution operations in a CNN. The 8x8 input is convoluted with 3 channel 3x3 kernels using a step-size for 1 to yield a 3x6x6 hidden layer.

2.3.5 Transfer Learning

Since deep-neural-networks usually take multiple days to train for any reasonably complex task, it is common to use pre-trained weights to initialise networks with the same or similar architectures. This technique is known as *transfer-learning*, and allows one to reuse representations learned on a similar task. These pre-trained weights can then be used to fine-tune the final layers of the network or backpropagate through the entire network, depending upon the size of the data-set available and the level of similarity of the new and original tasks.

Deep learning is an active field with a number of nuances and tricks for training that are impossible to cover in this brief overview. Refer to [58] for a detailed discussion.

2.4 Related Work

Deep-neural-networks have been used extensively in grasping research over the last five years. The ability of CNNs to parse complex spatial patterns lends well to the perception task often central to a robot grasping environment. A key challenge in the field however, is the lack of large, labelled data-sets required to train deep-networks. Neural-networks require to be trained on data in the same format as that used at test time. The training input therefore must be in a form that can be acquired using a depth camera in a real-world setting. Pre-existing mesh data-sets such as the ones in the Columbia Grasp Database, do not provide this. A number of research groups have adopted varying strategies and techniques to overcome this hurdle.

Pinto and Gupta [59], present a *self-supervised* approach to grasping using trial-and-error on a parallel jaw gripper. They use a Baxter robot [60] equipped with a depth camera, and allow it to automatically attempt grasps on objects placed in a bin, without any human supervision or labels. The network receives a binary success label based on whether the gripper jaws closed completely on an attempted grasp. The network starts off by randomly sampling locations on a 2D grid, upon which the grasp is centred, as well as one of eighteen bins specifying the angle of the gripper jaws with respect to the x-axis of the depth-image frame[4]. They generate a data-set of 50,000 attempted grasps acquired over 700 robot hours, with success-rate of 79.5% on a test set of seen objects.

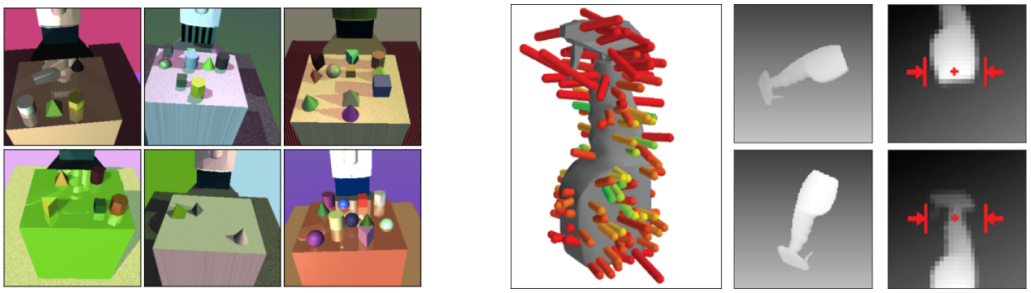
Levine *et al.* [61] take this a step further by using an end-to-end approach to learning hand-eye-coordination. They employ between 6 and 14 robots to run simultaneously over the course of several months, thereby implicitly allowing the wear and tear of physical components to regularise the learnt grasp strategy. They use a similar self-supervised approach, but instead increase the size of the collected data-set by an order of magnitude, to 800,000 attempts. Although they achieve an 8% improvement in accuracy over Pinto and Gupta, the law of diminishing returns would suggest that such an approach based on physical data collection is not viable for up-scaling.

Deep-reinforcement learning of end-to-end visuomotor policies [56] is another promising approach that requires a significantly smaller data-set. It aims to train CNNs to directly control motor torques on a robot to accomplish grasping and object manipulation tasks. This allows the robot to adjust its grasping policy on-the-fly to account for uncertainty

[4] Each bin represents 10 degrees of rotation for antipodal grasps, leading to equivalent grasps every 180 degrees.

and perturbations in perception. This is fundamentally different to methods discussed so far, which rely on an offline computation of the grasp trajectory.

A related strategy, referred to as One-shot Imitation Learning, is considered by Duan *et al.* [62], where they train a neural-network to complete tasks at test-time following a single demonstration. The aim is to create a framework for the network to learn the semantics of performing a task separately from the perception and planning. Tobin *et al.* [63], address the latter task through training in a simulated environment and introducing domain randomisation as a technique to transfer the learned representation to a real-world environment.



(A) (B)

FIGURE 2.14: Using simulated training data to learn real-world representations. (A) Domain randomisation for transferring function representations learnt in simulation to the real-world [63]. (B) Synthetic grasping data [64].

While the goal of such policy driven methods is different to the grasp function we aim to learn, they introduce a new transfer-learning paradigm for object manipulation which could greatly aid learning of an otherwise hard task. Such models allow one to specify tasks that might be more easily represented in a supervised-learning setting [65], [66], [67].

This suggests that an ensemble-learning methods trained on a variety of related tasks might be the most effective technique of learning to grasp. The prior results of Levine *et al.* [61] on physical robots also suggest that from a deep-learning point-of-view, there is a need for large high-quality grasp data-sets. A promising direction in this case might be to use simulated data for *pre-training* before fine-tuning on physical robots.

Kappler *et al.* [68], present such a large-scale data-set consisting of 500,000 simulated grasps annotated with both an ϵ-metric and a physics-simulator based metric for the Barrett hand. The authors argue that using a combination of these metrics yields improved results. They also identify a score of 0.002 on the ϵ-metric as a threshold for a positive grasp by comparing it to the physics based metric.

Mahler *et al.* [64], use this threshold to train a CNN on their custom data-set, the Dexterity Network 2.0, of 6.3 million data-points consisting of depth-images, hand-poses and grasp-quality labels. Using the robust ϵ-metric described in Figure 2.6, they achieve state-of-the-art results on both seen (93% accuracy and 99% precision) and unseen (80% accuracy and 100% precision) objects. Despite these unparalleled results, one of the failure modes cited by the authors is collisions of the gripper with the object during an attempted grasp. This is likely due to the uncertainty in positioning the gripper at the desired location, resulting from factors such as the miscalibration of rigid-transformations between the robot gripper and depth image or imprecision in joint movements of robot.

Such imprecisions are inherent to most non-industrial robots and only become more relevant with wear and tear. Johns *et al.* [10], suggest an approach to accounting for this uncertainty in position by smoothing the grasp function with the gripper pose uncertainty. Instead of defining grasp success as a binary label, they split the score, calculated by averaging the outcome of multiple trials in a physics based simulation, over six bins ([0.0, 0.2, 0.4, 0.6, 0.8, 1.0]). These bins are used to define a smoothed distribution over all possible grasps on the object. This allows one to pick a candidate grasp that is much less likely to lie close to a region of poor grasp quality.

In the next chapter, we discuss a novel approach to combining the last two methods in order to account for uncertainty while using the largest grasp data-set available.

Chapter 3

Framework and Data Sets

Garbage in, garbage out; a popular machine-learning adage summarising the relevance of having good quality data to learning a meaningful representation. Data-sets are rarely suited to a learning task in their raw form. Cleaning and pre-processing them is therefore central to any machine learning problem. In this chapter, we discuss our approach to selecting the appropriate frameworks and data-sets for our task. We present a brief overview of popular libraries, discuss the benefits and trade-offs of each, and justify our choices. We then describe the construction of a custom data-set, and introduce our approach to dealing with the non-linear relationship between grasp-quality and potential for success.

3.1 Machine Learning Framework

The popularisation of deep-learning in recent years has led to the emergence of a number of deep-learning libraries implemented for different programming languages:

TensorFlow
TensorFlow is a deep-learning framework developed and open sourced by Google, that allows the user a large amount of flexibility in defining a *computation-graph* — a concept shared by most deep-learning frameworks. It involves the offline construction of a data-flow graph that consists of independent units of computation, and *Tensor* objects that represent units of communication between these operations. The motivation behind computation graphs is that multiple computations can be parallelised: not all operations during optimisation need to happen in serial order. They can be thought of as weakly analogous to a compiled language: graph construction allows various optimisations that allow for speed-up at execution time.

Caffe

Developed in late 2014, *Caffe* was one of the first deep-learning frameworks, written in C++ with a Python and MATLAB API. With its popular multi-platform support, Caffe has gained wide-spread adoption and boasts a large *model-zoo* containing pre-trained models uploaded by the community. Aside from the model-zoo, one of its biggest advantages is its excellent documentation, especially since Facebook took up the project in 2017, resulting in Caffe2.

Pytorch

Pytorch is a Python port of the Torch deep-learning library written for the Lua programming language. The most powerful feature of Pytorch is the ability to define dynamic computation graphs. These allow the user to specify varying I/O tensors to the optimiser without keeping track of the exact shapes of the tensor operations — primarily useful when dealing with recurrent neural-networks. However, since the shape of the computation graph is not fixed, the resulting optimisation is slower and requires greater memory usage.

Although the official documentation for TensorFlow is lacking in some regards compared to libraries such as Caffe, we use it as our development framework in this project. The reason of this is two-fold: first, it allows us to execute precise control over the structure and optimisation of our code; second, is contains functionality to wrap Python code into tensor operations that can be added to the graph. This allows us to run native Python I/O and image transformation operations in an efficient manner[1].

3.2 Hardware

We train all our models on an Nvidia Titan X GPU alongside 16 CPUs that are used to fetch data asynchronously. However, despite the available architecture, these resources were shared with other research groups which placed a cap on the maximum time and memory available to us. This restriction made it necessary to create a highly optimised data-input and training pipeline as discussed in Section 4.1.

Our simulated data was generated using the geometry of the Yumi parallel-jaw gripper (see Section 2.1.1). Although we had the option of using other end-effectors, we chose the Yumi as it afforded the opportunity to test our system on the setup provided by [64][2], and was compatible with pre-existing data-sets.

[1] Using native Python was important because it allowed for flexibility and rapid prototyping.

[2] Mahler *et al.* offer to test selected networks on their set-up based on theoretical performance.

3.3 Data-set

One of the early research decisions we faced was the choice between a physics based simulation versus an analytical one. The difference in the two lies in the way in which the dynamics of object interactions are modelled. An analytical approach as used by [64], assumes a simplification of these dynamics and models them as point interactions to compute grasp quality scores, as described in Section 2.2. Although this results in a significant speed-up in labelling candidate grasps, Kappler *et al.* [68], argue that it results in an imprecise approximation of the true interactions. They instead present an approach that augments standard analytical metrics (such as the ϵ-metric) with physics based measures. They model the interactions between the contact surfaces of the object and the gripper, with finite-element models using physics engines —the Open Dynamics Engine (ODE) [69] in this case. The object exists in free-space and is allowed to move upon contact with the gripper. This is in contrast to the analytical case, wherein objects are assumed to be stationary and the metrics are computed offline. The physics-metric is a binary grasp success label which is averaged over multiple trials to attain a normalised score in the range [0, 1]. A similar strategy is adopted by [10], who however do not combine it with an analytical score.

Kappler *et al.* have open sourced their data-set of 500,000 labelled grasps computed for the Barret hand. Figure 3.1 shows an example of a subset of grasps in their data-set. By empirically comparing both the physics and analytical metrics of candidate grasps, they identify an ϵ-metric threshold of 0.002 for successful grasps.

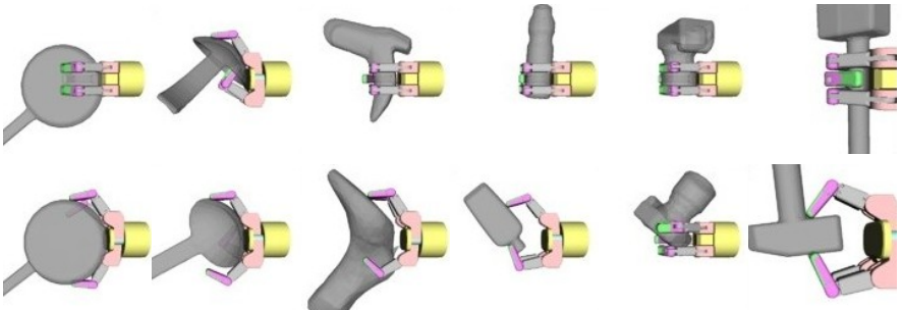


FIGURE 3.1: Example subset of grasps from the Kappler data-set. Top row: successful grasps. Bottom row: unsuccessful grasps [68].

However, due to the relatively small size of the data-set, their use of the Barrett hand instead of the Yumi, and since this data-set does not contain rendered depth images, we chose to use the Dex-Net 2.0 data-set by Mahler *et al.* [64] instead. In hind-sight, the Kappeler data-set could be made amenable to our goal by using the approach described in Section 3.3.3. We leave detailed analysis of our decision to Chapter 6.

Dex-Net 2.0

The Dex-Net 2.0 data-set contains 6.3 million depth images of 1500 objects placed on an empty table in different poses, and labelled with a corresponding grasp-quality score. The gripper position for a candidate grasp is implicitly encoded in the depth-image such that the gripper jaws are always aligned with the y-axis and centred on the image. In addition, the distance between the grasp centre and the depth camera is stated explicitly. Figure 3.2 illustrates these transformations.

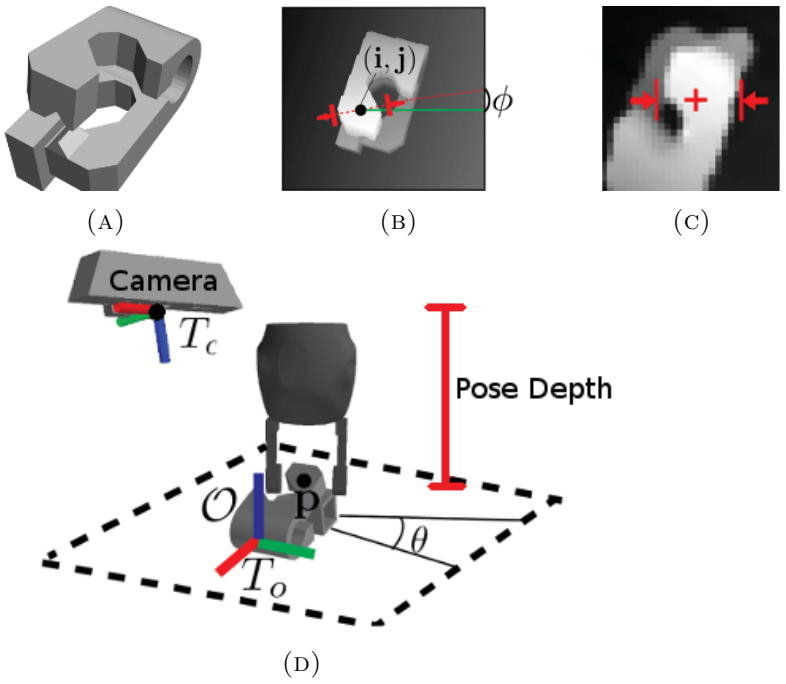


FIGURE 3.2: Transformation from raw depth-images to centred and aligned input [64]. (A) Input object mesh. (B) Sampled raw depth image. (C) The depth image is rotated and translated to coincide with the gripper centre and align parallel to the jaws. (D) Scene arrangement.

The robust ϵ-metric described in Section 2.2 is used as labels with a threshold of 0.002 adopted from [68]. As mentioned in Chapter 2 the number of unsuccessful grasps in the data-set is much higher than the number of successful ones. This results in a highly skewed distribution of the grasp-quality scores, as visualised in Figure 3.3. The histogram roughly resembles a Gamma distribution with the mean lying to the left of the threshold for positive grasps. This approximately translates to a 20-80, positive-negative split.

Furthermore, the relationship between these quality scores and the probability of success varies non-linearly from left to right. A grasp with a quality score of 0.023 at the far right of Figure 3.3 for instance, might be only slightly better than a grasp with a score of 0.005 at the left of the distribution.

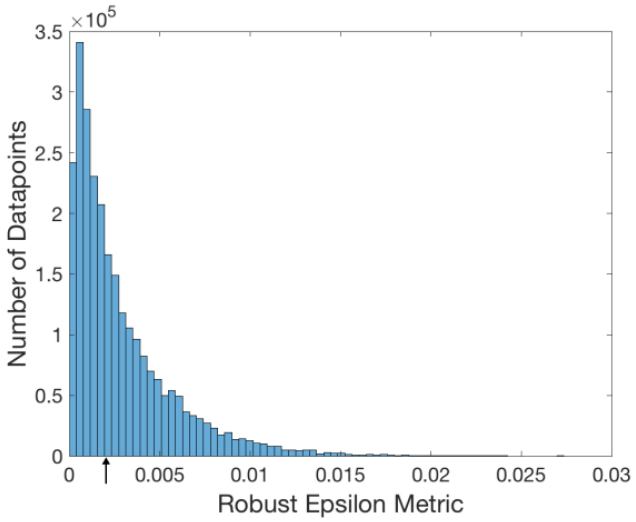


FIGURE 3.3: Distribution of the robust ϵ-metric labels in the data-set. The raw data-set is highly skewed, with a bias towards grasps in collision with the object. The arrow indicates an empirically determined success threshold of 0.002.

The combination of these factors makes it impracticable to train a classifier without first pre-processing the data in some way. In the ideal case, we want the data to be equally distributed in each class, and for the labels to be normalised to reduce non-linear effects. However, this turns out to be a non-trivial task, as discussed in Section 3.3.2.

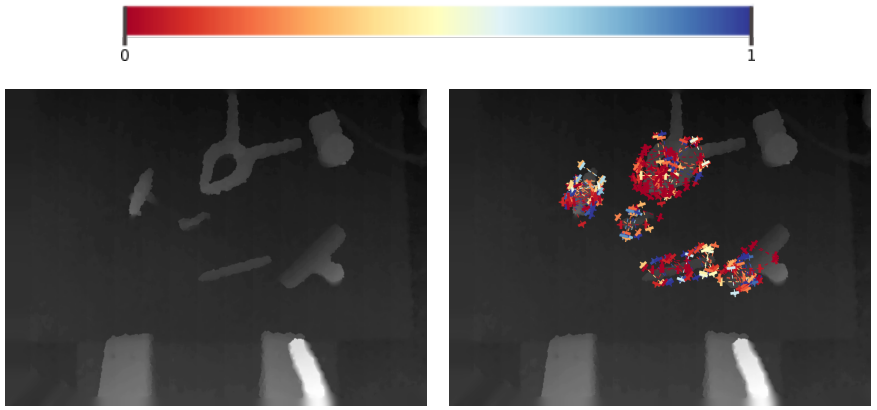
3.3.1 Expressing Uncertainty

Our goal in this project is to learn a representation of the uncertainty in grasping an object as a function of its geometry. Rather than assign binary success labels, we aim to indicate the *degree* of expected success for each potential grasp on the object. This can be thought of as describing a distribution over the input domain of a *grasp-function* that encodes the uncertainty in choosing a particular gripper pose. This function can be parameterised in homogeneous image coordinates (x_i, y_i) using the following transformations:

$$\lambda \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{\Lambda}} \underbrace{\begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} & \tau_x \\ \omega_{21} & \omega_{22} & \omega_{23} & \tau_y \\ \omega_{31} & \omega_{32} & \omega_{33} & \tau_z \end{bmatrix}}_{\mathbf{\Omega}} \begin{bmatrix} u_i \\ v_i \\ w_i \\ 1 \end{bmatrix} \tag{3.1}$$

Where, λ is an arbitrary scale factor, $\mathbf{\Lambda}$ is the intrinsic camera matrix and $\mathbf{\Omega}$ is the transformation matrix relating the camera to the global reference frame.

As a simplification, we only consider grasps that are perpendicular to the table surface (Section 3.3.3). This allows us to reduce the grasp function into a distribution in two-dimensions that can be easily visualised[3] (Figure 3.4).



(A) (B)

FIGURE 3.4: Visualising the grasp function as a 2D distribution over the depth image. The colour of the grasps represents the normalised quality score.

As proposed by Johns *et al.* [10], convolving the uncertainty distribution with the original grasp function results in a representation that is smoothed by its neighbourhood. As a side-effect, this means that we only care about the relative goodness of a grasp with respect to other points in the grasp function. This concept is best explained visually through Figure 3.5.

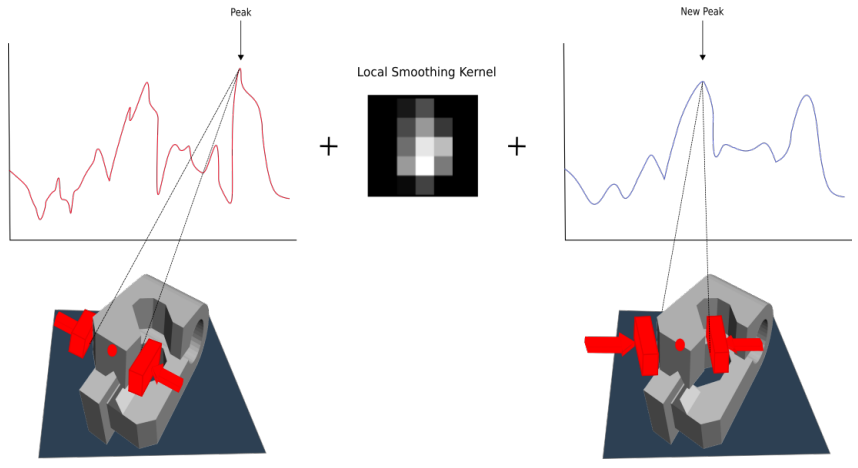


FIGURE 3.5: Smoothing a grasp function by convolving it with the uncertainty distribution. Note that both the 1D grasp function and the smoothing kernel shown are simplifications for motivating the idea. We are only interested in the relative peak of the function and not the absolute quality scores.

[3] In reality, despite this simplification, the grasp function is still three-dimensional owing to the depth component of the gripper pose. This encodes the vertical position on the object where a grasp is executed. However, for the sake of simplicity, we do not visualise this parameter.

We represent the grasp-function as one-dimensional for ease of demonstration. The peak of the original function is smoothed by convolving it with the grasp quality/uncertainty kernel of its neighbourhood. This results in a new peak on the smoothed function, that corresponds to a more robust grasp.

3.3.2 Generating Uncertainty Labels

To learn the grasp function, we split our grasp quality scores into six bins, [0.0, 0.2, 0.4, 0.6, 0.8, 1.0] in ascending order of the ϵ-metric. Bin zero is reserved for candidate grasps that collide with the object on approach. We attempted a number of methods to discretising the grasp-quality scores:

Percentile Binning

Our first experimental splitting strategy was to normalise the data-set (by subtracting the mean and dividing by the standard-deviation) and split the quality score into six bins spanning equal intervals, between the minimum and maximum of the distribution. However, since the normalised data-set is still highly skewed, the classifier learnt a naive representation, wherein it categorised the majority of the input as the dominant class.

To avoid this class imbalance, we modified our approach to instead use a percentile splitting of data (Figure 3.8a). This ensures an equal number of data-points in each bin. We sort the labels in ascending order of grasp-quality, and split them equally amongst six bins, such that each bin edge represents a difference of 16.67%. We also believed that since only the relative distribution of labels mattered to our end-goal, we could safely ignore the non-linear relationship between the grasp quality scores and the bins they were assigned to. In essence, it did not matter if two successive labels had the correct spatial mapping to the initial distribution, as long as the grasp quality scores in two consecutive bins were strictly increasing. The toy-example in Figure 3.6 demonstrates this case.

Bins	1			2			3			4		
Score (metric)	0.10	0.11	0.12	0.14	0.17	0.45	0.46	0.70	0.71	0.72	0.75	0.98

FIGURE 3.6: Percentile quality labels are generated by sorting the raw ϵ-metric labels in ascending order and splitting equally into each bin. Since we only care about the relative quality of points on the grasp function, a perfect mapping to bins is not essential.

However, as discussed in Chapter 4, this assumption fails due to the very non-linearity is presumes to ignore.

Log-scale Binning

Here, we split the data-set into bins where the domain size lies on a log-scale, with larger bins towards the left of the distribution (Figure 3.8b). Since the bulk of the labels towards the left represent unsuccessful grasps, it is unlikely that these regions will end up lying at the peak of the distribution. We therefore classify them at a coarser scale. Conversely, points towards the right of the distribution are more likely to be relevant in determining the optimal grasp, which in turn warrants classification into finer bins.

Clustered Binning

This approach is motivated by the observation that descritising a non-linear distribution is fundamentally a clustering problem. The Jenks Natural Breaks Optimisation [70] algorithm is one of the techniques specifically designed to determine the optimal arrangement of data into classes[4]. The algorithm aims to minimise intra-class variance while maximising the distance between the means of different classes. While the algorithm was initially conceived for classification of geographical data, it lends itself naturally to the task at hand.

This allows us to split the data-set in a way that maximises the difference between classes, making the learning task easier. The objective function for clustering $\mathbf{x}$ labels into a set of K classes $\mathbf{S}$ is given by:

$$\mathcal{O} = \arg\max_{\mathbf{S}} \sum_{i=1}^{K} \sum_{\mathbf{x} \in S_i} \| \mathbf{x} - \boldsymbol{\mu}_i \|^2 \tag{3.2}$$

Where, $\boldsymbol{\mu}$ is the vector of cluster means. Figure 3.7 illustrates a toy example of clustering in one dimension.

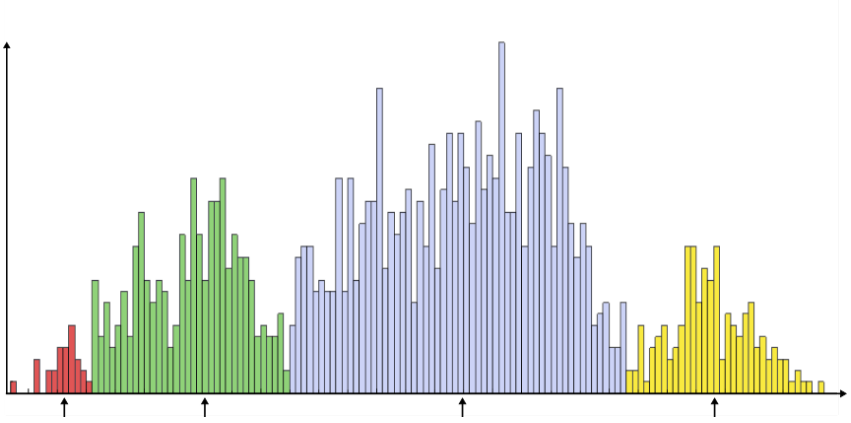


FIGURE 3.7: Toy example demonstrating clustering in 1D. Bins might not necessarily contain an equal number of elements. The focus is on maximising inter-class difference while minimising intra-class variance.

[4] Jenks Natural Breaks is equivalent to K-means clustering applied to univariate data.

Normalised Clustered Binning

This approach is equivalent to the last one except in that we normalise the data-set around a mean of zero and clip it to three standard-deviations. This accounts for outliers that otherwise skew the cluster means. Figure 3.8 visualises the four binning strategies discussed.

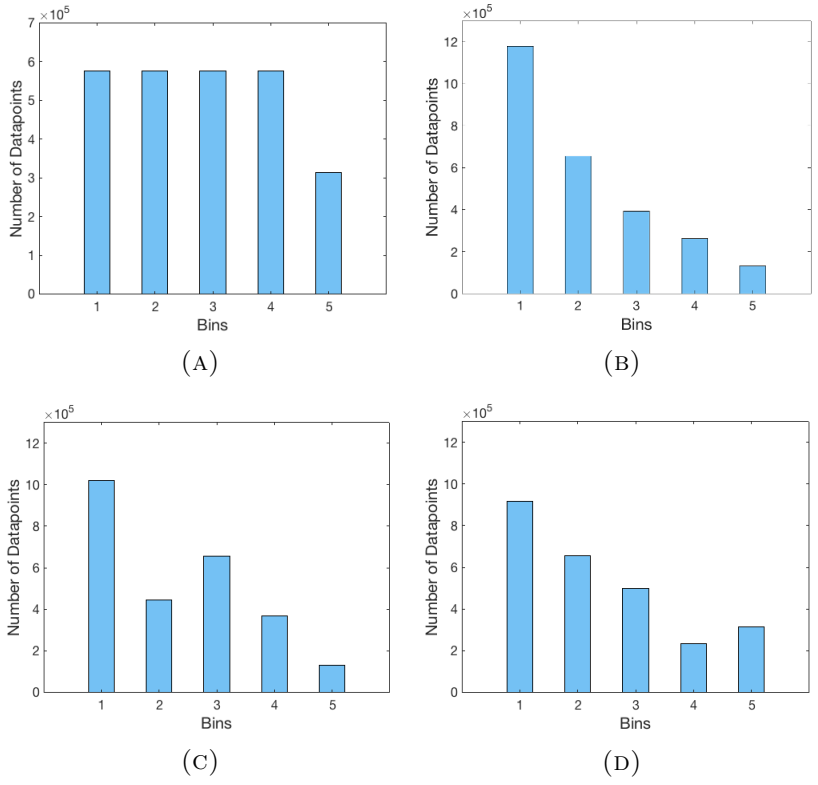


FIGURE 3.8: Four different types of binning, with bin zero reserved for grasps in collision (not visualised here). (A) Percentile binning: labels are split equally into each bin. The last bin has fewer elements because we threshold the normalised scores at the mid-point of neighbouring bins. For example, grasps with scores in the range 0.7 - 0.9 are assigned to bin 4. The last bin therefore only gets values between 0.9 - 1.0. (B) Log scale binning: this captures the intuition that lower bins are not important to identifying successful grasps and can therefore be classified at a coarser scale. (C) Clustered Binning: data is still skewed but the cluster centres shift the distribution. This can result in unwanted effects, especially due to outlier grasps to the far right of the distribution. (D) Normalised binning: the edges of the distribution are clipped to three standard deviations.

3.3.3 Generating Raw Depth Images

The data-set considered so far assumes a discriminative model. To test our method, we define a policy for sampling potential grasps from an input depth image which are then fed to the model for assigning quality scores. In order to express better control over the data-set and enable the potential for training generative models, we develop

a pipeline for sampling new depth images from 3D mesh models with the help of the `dexnet` Python package [64]. This also allows us to augment our training data with images rendered from larger mesh data-sets, such as the ShapeNet [71], in the future.

Before describing the image-rendering pipeline, let us first describe the geometry of the scene, visualised in Figure 3.9:

- ***Grasp-axis***: vector between the gripper jaws.
- ***Grasp-angle*** ***(θ)***: angle between the grasp-axis and the x-axis of the image frame.
- ***Approach-axis***: vector pointing out of the gripper jaws.
- ***Approach-angle*** ***(ϕ)***: angle between the approach-axis and the normal to the table upon which the object is placed.

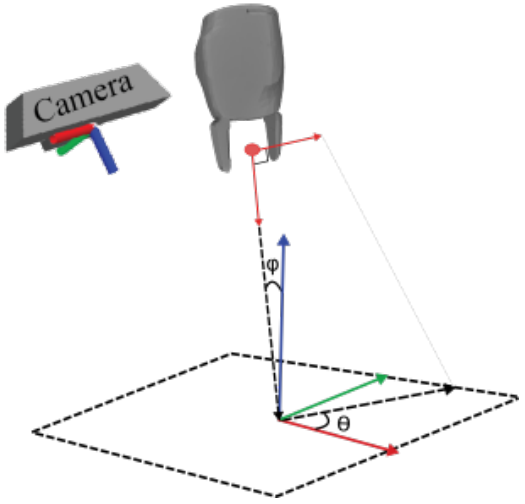


FIGURE 3.9: Geometry of the setup. The figure describes the projection of the Yumi gripper onto a 2D surface parameterised by grasp-angle θ and approach angle ϕ.

The pipeline for image-rendering is summarised in Figure 3.10. We first retrieve the pre-computed candidate grasps associated with a particular object-mesh and gripper combination (the Yumi in our case). Next, we estimate the stable-poses of the object-mesh on the table, and filter the grasps to only those accessible within a specified tolerance of the maximum approach-angle. We repeat this process for each object in the database and cache the resulting grasps to file.

Once all the candidate grasps for each object have been extracted, we iterate through each stable pose for all the 1500 objects and use a random-variable based image sampler to simulate the generation of depth images from a Xbox Kinect, with known coordinates

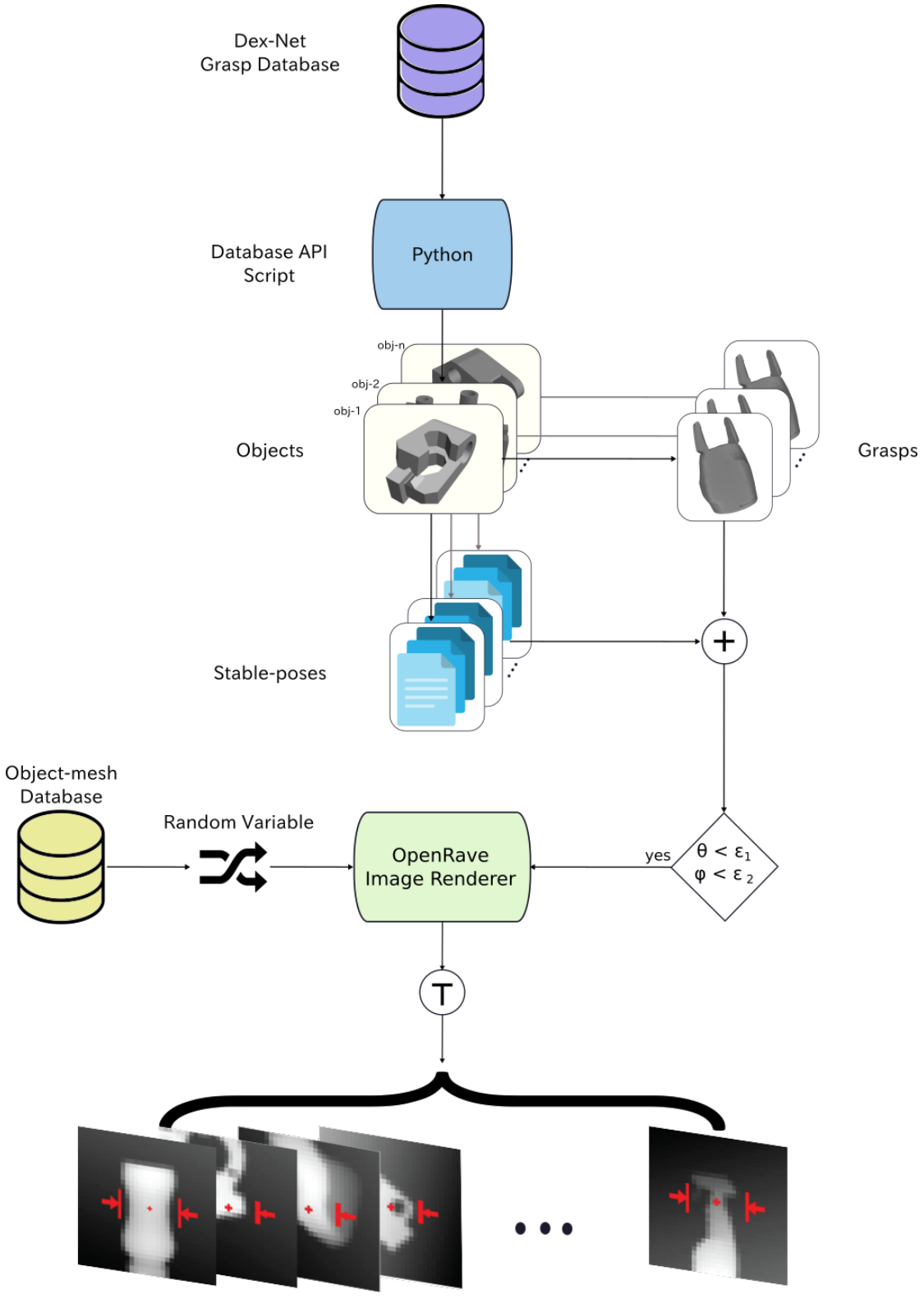


FIGURE 3.10: Pipeline for grasp-image generation from 3D object meshes. The Dex-Net data-set contains object meshes that are sampled using the `dexnet` Python package and the OpenRave API to synthesise grasps. Multiple images with random perturbations are sampled for each object mesh placed in a stable pose on the table. These images and grasps are filtered by approach constraints to form the input to our network.

in the global frame of reference. The random-variable sampler allows us to simulate variation in the scene lighting and camera pose. While the exact number of samples varies for each object, a rough estimate can be approximated as follows: for each of the 1500 object-meshes, there are 10 stable poses; for each stable poses there are 20 candidate grasps; we also sample 20 images from each stable pose. This results in approximately 6 million grasps, which is close to the 6.3 million in the Dex-Net 2.0 data-set.

Once the images have been sampled, we can either rotate and centre them on the grasp to yield the Dex-Net 2.0 data-set, or we can leave them un-transformed for use in generative models. We provide the capability to achieve both these cases but leave the implementation of generative models to future work (Section 6.1).

Chapter 4

Learning to Grasp

Despite the emergence of deep-learning as the foremost tool for classification tasks, as well as the recent advances in machine-learning frameworks, the inner workings of deep neural-networks remain obscure. Although several research groups have attempted to decode the semantics of the representations learned by layers within these networks [72], our ability to interpret them is limited by the capacity of our brains in dealing with high-dimensional data. This "curse of dimensionality" [73] combined with the long training times required to achieve meaningful results, makes it incredibly hard to debug modern neural-networks.

In this chapter, we walk through our approach towards deconstructing these challenges and training multiple deep neural-networks for learning grasp-quality metrics. We discuss the data-input pipeline used in training, along with the related challenges and caveats. Finally, we discuss the different architectures considered, hyperparameter tuning and the training regimes adopted to learn progressively complex tasks. We take a step-by-step approach, with the network trained at each step used to initialise the weights for the next task (*transfer-learning*).

4.1 Data Input Pipeline

Given the large volumes of data required to train deep neural-networks, an efficient optimisation pipeline is of existential importance to the field. While the inability to do so constrained progress for over thirty years, recent advances in computing power, especially in massively parallel operations afforded by powerful Graphics Processing Units (GPUs), has lifted this barrier and accelerated the field at an unprecedented rate. Yet, the speed-up enabled by GPUs is bottle-necked by the rate at which data can be

fed into the optimiser. With a data-set of the size used in this project, optimising the data-input pipeline can result in orders of magnitude difference in training time — one day versus several weeks. Additionally, since we wish to balance our skewed data-set during pre-processing, we further need to under-sample the dominant class(es) while ensuring sufficient data-mixing. We also perform a number of linear operations on the input images for data-augmentation. Efficient pre-processing and data input is therefore a critical requirement for our project.

While the native TensorFlow approach to data-input involves the use of TensorFlow *record* files, we decided to use the NumPy *'.npz'* format in which the data-set was originally provided by the Dex-Net team. The motivation for this was to allow flexibility in manipulating and visualising the data-set during prototyping, as well as minimising data duplication — an important consideration from a storage point of view, given that our data-set is roughly 300GB.

Since we need to under-sample the dominant classes, we end up rejecting between 10-60% of data from each NumPy file, depending on the type of network being trained. This means that in the worst case, for each batch of data used for optimisation, we sample approximately 2.5 batches from the data-set. As the number of samples from each class, randomly distributed in the NumPy files, might be fewer than that required to construct a balanced batch, we buffer surplus data from these files, and allow for sufficiently randomised over-sampling.

To do this, we use a dual *First-In-First-Out* (FIFO) queue architecture to hold the input stream:

1. **Data-Queue**: a FIFO queue containing raw data loaded from disk.
2. **Batch-Queue**: a second FIFO queue to hold shuffled batches of pre-processed input images paired with their corresponding labels drawn from the data queue.

We initially used the TensorFlow `RandomShuffleQueue` and `FIFOQueue` operations for the data and batch-queues respectively, with the `enqueue_many` and `dequeue_many` operations to add and draw elements. Both these choices proved to be highly inefficient due to the blocking behaviour of latter two operations. Inestead, we now manually shuffle the paired input-label data before adding it to the sequential queues. This results in an approximately $\times 12$ speed-up in overall training, visualised in Figure 4.2, as opposed to a naive implementation. Our final architecture is illustrated in Figure 4.1.

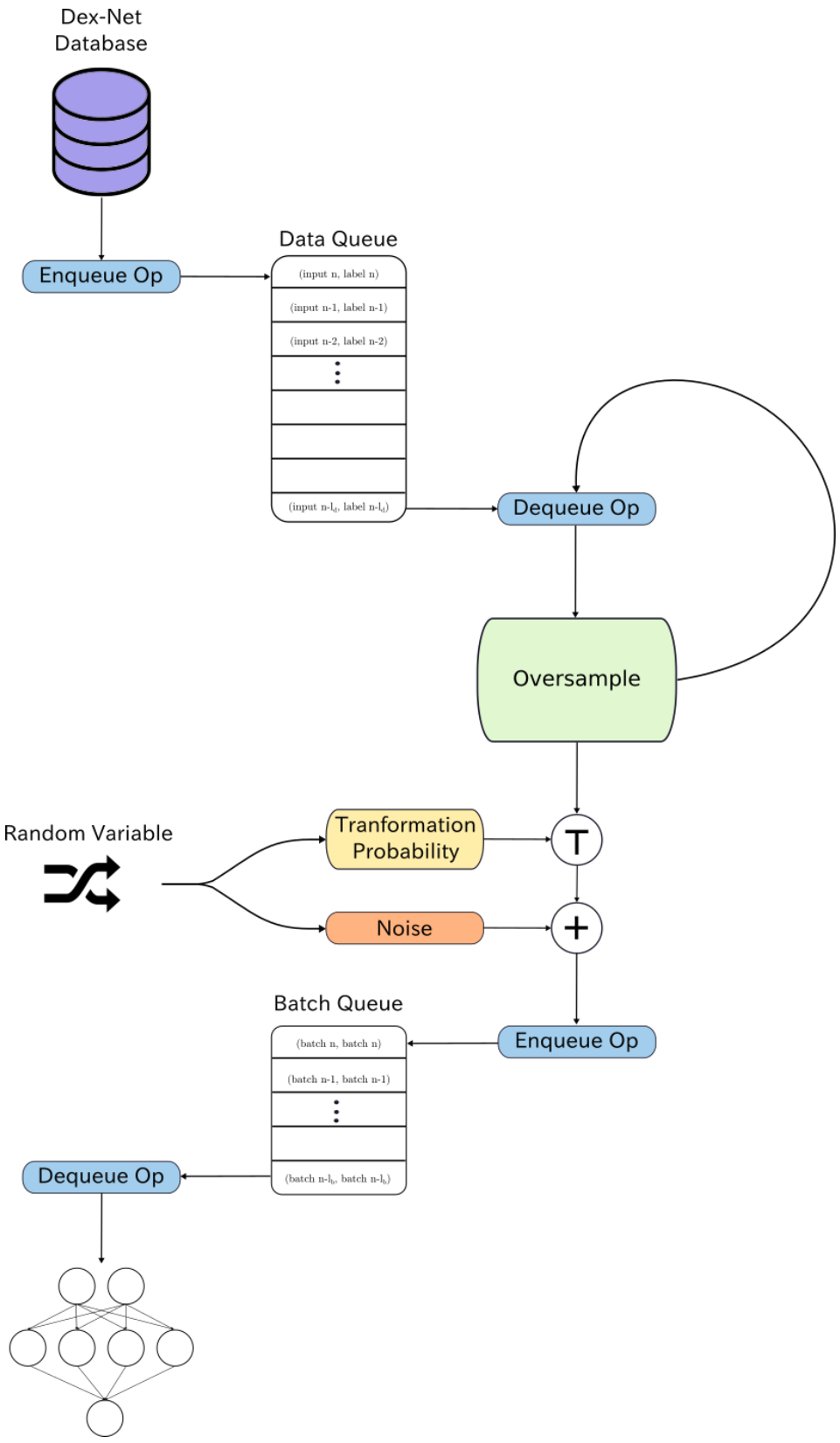


FIGURE 4.1: The architecture of the data-input pipeline adopted for efficiently feeding data to a GPU for optimisation. We employ a dual-queue approach, where the data-queue stores raw shuffled input loaded from disk, which is then oversampled and pre-processed and organised into batches in the second queue.

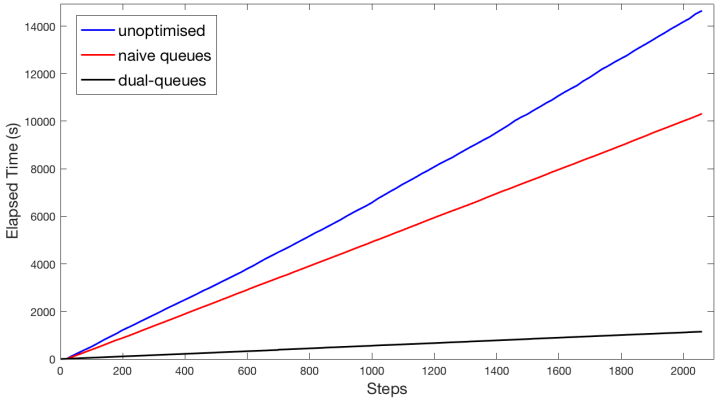


FIGURE 4.2: A comparison of the optimisation speeds of different input pipelines. Our dual-queue approach results in a twelve fold speed-up over a naive single-queue approach.

4.2 Pre-processing

Our pre-processing pipeline consists of a number of stages. The foremost amongst these is to sufficiently shuffle the data-set such that each training batch consists of samples drawn randomly from the entire distribution of object poses. As discussed in Section 3.3.3, the Dex-Net 2.0 data-set is constructed in a hierarchical manner: each object has multiple stable-poses, for each one of which there are multiple sampled images and candidate grasps. Considering the serial nature of these operations, data-points are generated such that similar images are batched together in the 6729 NumPy files from the original data-set[1]. A sub-sample of the distribution of images in these files is depicted in Figure 4.3.

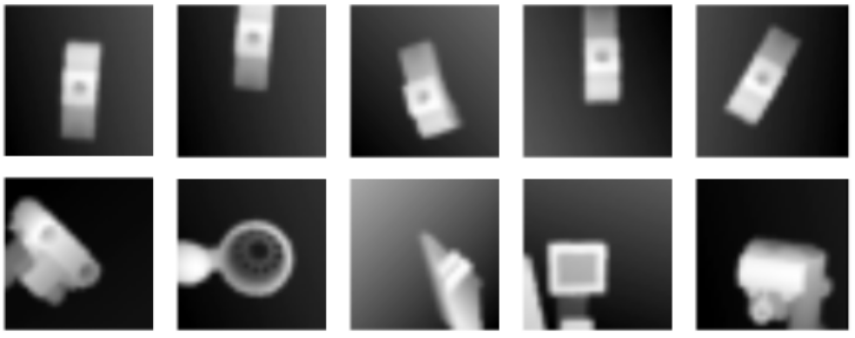


FIGURE 4.3: Five consecutive images drawn from a random file. Top Row: unshuffled data from the Dex-Net data-set. Due to the hierarchical construction of the data-set, consecutive data-points represent similar object poses. Bottom Row: shuffled data used for training in this project.

[1] Each NumPy file contains 1000 data-points, except for the last one, which contains 380.

We randomly shuffle this data-set while keeping track of the original indices, to allow debugging. Figure 4.4 presents a comparison of overfitting a binary classifier on 0.02% of shuffled and un-shuffled data. In the unshuffled case, the training accuracy oscillates widely and fails to converge, most likely because images within each batch are similar but completely different from the next randomly selected batch.

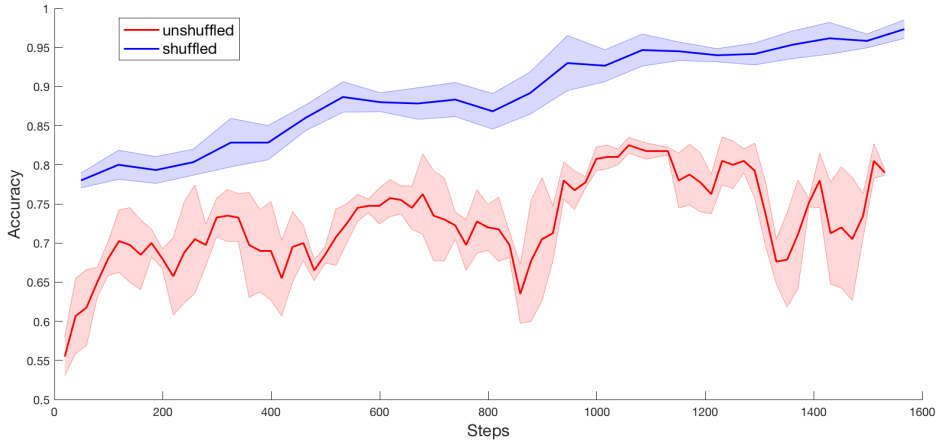


FIGURE 4.4: Results of attempting to overfit 0.02% of the data-set with shuffled and unshuffled input. The latter data-set presents a high batch-to-batch variance which leads to a poor training signal, and causes the training accuracy to oscillate.

Next, to balance out the data-set, we downsample dominant classes by randomly discarding data-points, so that each class has the same number of elements. Due to the high degree of skew of our data, this extensive resampling would have been too slow without the data-input pipeline discussed in Section 4.1.

Since the images used in our data-set are rotated and translated to align with the jaws of the gripper, we can augment our data-set as long as we do not break the symmetry of these transformations. We use two augmentation operations carried out back-to-back with 50% probability: we flip the images on the vertical axis, with the output mirroring the original, and also flip them on the horizontal axis, which is equivalent to rotating the object by 180 degrees on the table. This strategy can also be used in place of the random-shuffling we discussed earlier. However, for the sake of simplicity we decided to extricate this process into two separate steps.

Lastly, we distort the images with random Gaussian noise to allow for better transfer from simulation to the real world. Maheler *et al.* [64] argue that this is the most critical step in improving the performance of their robot on real objects. This claim is further propounded on by Tobin *et al.* [63] who suggest *Domain Randomisation* as a technique for transferring from simulation to the real world.

To ensure that our input pipeline remains efficient, we use Python wrappers in TensorFlow to encapsulate these operations. An additional pre-processing step for making images compatible with the Alex-Net is described in Section 4.5

4.3 Hyperparameters

We trained our models with a combination of Stochastic Gradient Descent (SGD), Momentum, and Adam optimisers. Both SGD and Adam were too slow to converge. We achieved best results with the Momentum optimiser which we used to train our final models. The hyperparameters used are listed in Table 4.1.

Table 4.1: Hyperparameters used to train the final models.

Parameter	Network
Batch Size	150
Base Learning Rate	0.01
Momentum Rate	0.7
Exponential Decay	True
Decay Rate	0.9
Decay Step	100,000

4.4 Grasp Uncertainty Dex-Net

Debugging a neural-network can be a challenging process. To make it tractable in the limited time available to use, we decided to use a transfer-learning approach in this project. We start off with the simplest possible network, using pre-trained weights where available, to progressively learn more complex functions.

The Grasp Uncertainty Dex-Net or GUDNT uses weights from the model trained by [64]; the architecture is depicted in Figure 4.5. We remove the last fully connected layer and train it on the six uncertainty classes while experimenting with a number of different hyper parameters.

The training batch-size plays an important role in the ability and the speed at which a model learns. Although mini-batch gradient descent works on an approximation of the local gradients, and should therefore result in an approximate solution, in practice, large batch-sizes often reduce the quality of the model [74]. In our case, we found the most optimal batch size to be 150, even though larger batches would increase the

optimisation speed, since it reduces the communication overhead on the GPU. We keep this parameter fixed for all training regimes henceforth, although we acknowledge the room for fine-tuning.

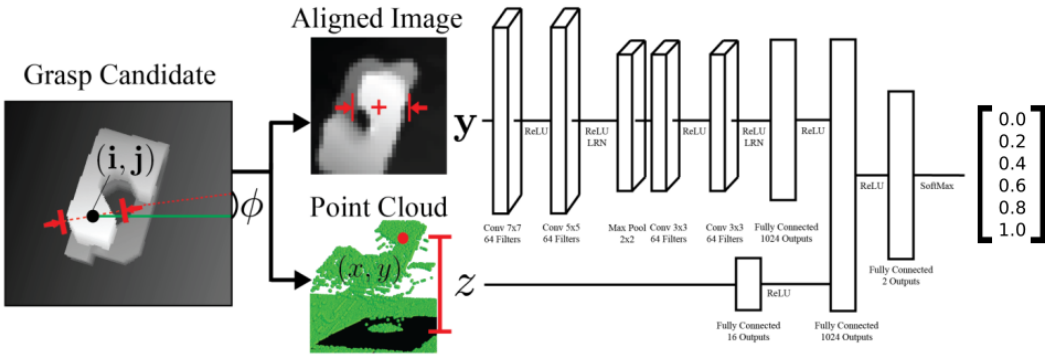
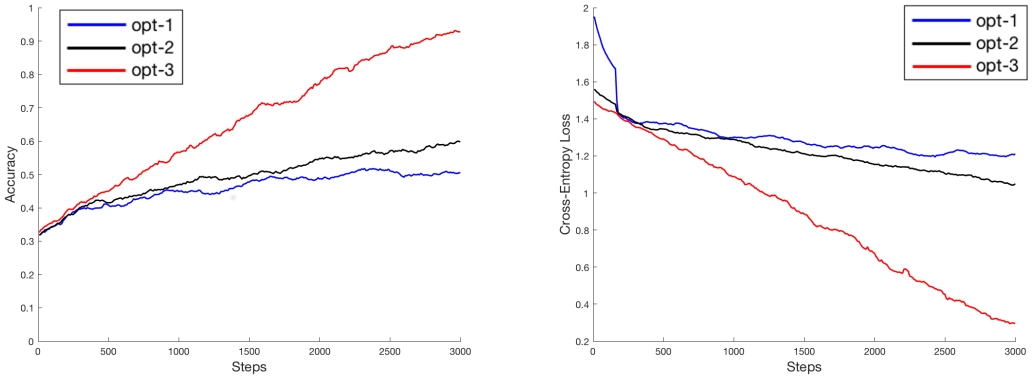


FIGURE 4.5: The GUDNT network architecture [64]. The network takes a transformed depth image as input along with the distance of the grasp centre to the camera. The two inputs are combined together in a fully connected layer before outputting a probability distribution over six quality labels.

We also experimented with the ability of the network to learn as a function of the number of layers trained during optimisation, on 0.02% of the data-set. As illustrated by Figure 4.6, even though the pre-trained weights were learnt on a very similar (binary) task, our network learns fastest when the backpropagation gradients are allowed to flow through the entire network. This is most likely due to differences in pre-processing and the type and magnitude of simulated noise added on training.



(A) (B)

FIGURE 4.6: (A) Accuracy and (B) cross-entropy loss curves for optimising on 0.02% of the data-set as a function of the layers trained. The curves depict the overfitted performance when training (opt-1) only fully connected layers, (opt-2) fully connected and 2 convolution layers, and (opt-3) all layers layers of the GUDNT architecture. Allowing the backpropagation gradients to flow all the way through the network increases the training speed and performance.

We tune the hyperparameters of the GUDNT by training it on the percentile binned labels (Section 3.3.2) with different optimisers and learning rates. The training curves are summarised in Figure 4.8.

These results indicate sub-optimal training. One reason for this might be the resolution at which the ϵ-metric labels are classified into bins. We test this hypothesis by training on log-scale bins. However, this only marginally improved accuracy. Another more likely reason lies in the linearity and normality assumptions of the percentile binning method. The resulting effects of this suboptimal binning are best described through the toy example visualised in Figure 4.7.

Bins	1			2			3			4		
Score (metric)	0.10	0.11	0.12	0.14	0.17	0.45	0.46	0.70	0.71	0.72	0.75	0.98

FIGURE 4.7: Toy example illustrating the failure of the percentile binning method to distinguish between related classes. Equally splitting labels into each class does not take into account the structure of the underlying function. This could lead to similar grasps being classified into different bins or the vice versa.

Since the percentile-binning methods involves sorting all the grasps in the data-set and equally dividing them into the six classes, it takes neither the skewness nor the non-linear distribution of the data into account. This can result in cases where two images with very different qualities end up in the same bin (yellow and red elements). On the other hand two data-points with very similar qualities might end up lying on a bin edge and therefore classified into different classes (edge between bin 2-3 and bin 3-4). Either case results in a difficult to learn function for the classifier. In-fact one can only learn this distribution by overfitting the data, which is why Figure 4.4 achieved high training accuracy.

To overcome this problem, we use clustered bins instead[2]. This allows us to minimise the edge effects at bin boundaries, while ensuring similar classes contain similar data. However, this means that our data-set is once again skewed and we must employ the over-sampling strategies mentioned earlier, at training time. This results in a 10% increase in test accuracy as illustrated by the training curves in Figure 4.8.

[2] In practice, we use the normalised clustered bins since they help us achieve better results by effectively accounting for outliers.

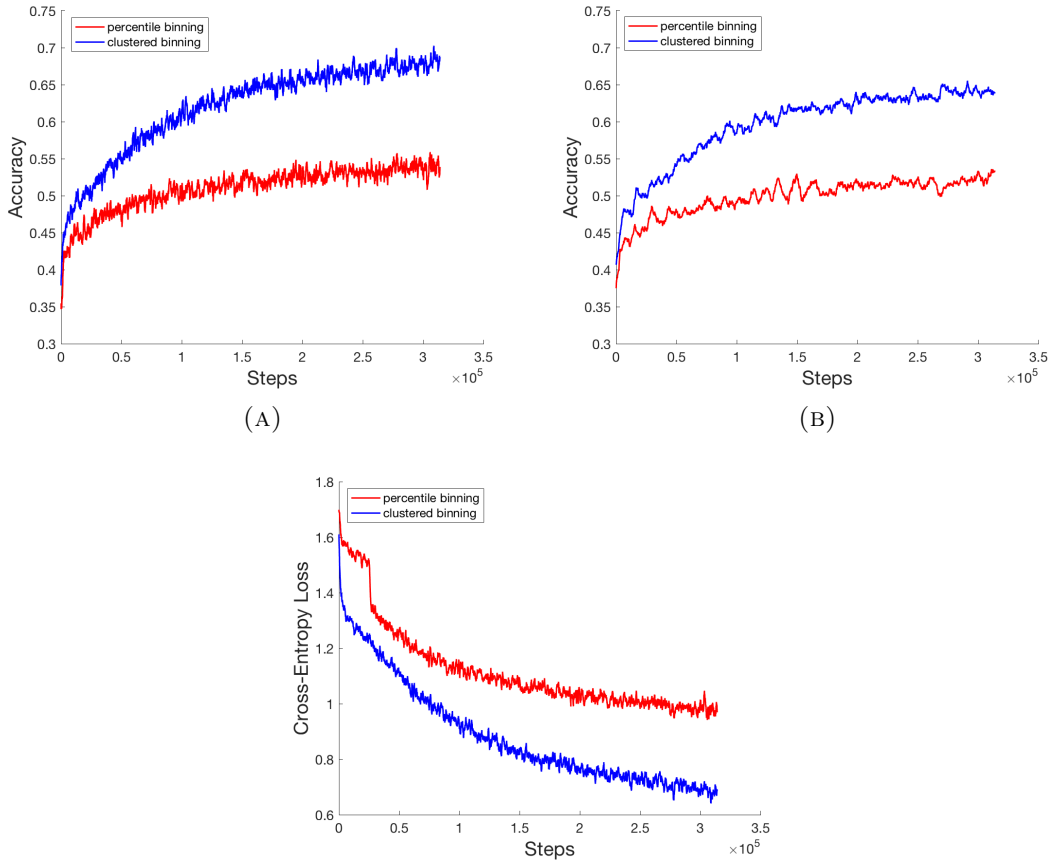


FIGURE 4.8: (A - B) Training and validation accuracies and (C) the cross-entropy loss curves for the GUDNT with percentile and clustered labels.

The GUDNT with its base architecture fails to achieve better training accuracy despite trying multiple hyper-parameter combinations. We believe that this is due to the lack of expressiveness of the network. To overcome this, we consider the deeper and wider Alex-Net architecture in the next section.

4.5 Grasp Uncertainty Alex-Net

The Alex-Net, proposed by Krizhevsky *et al.* [36] in 2012 resulted in a 11% improvement in top-5 test-error over other methods for the ImageNet classification challenge, and initiated the deep-learning revolution. Although deeper and wider networks, with higher accuracies have been proposed since, the Alex-Net is perhaps the most well understood and easy to implement network with pre-trained weights available today. The architecture is illustrated in Figure 4.9. Note that the network is split into two halves to allow for training on two GPUs. While GPUs in 2012 lacked sufficient memory to hold the entire network, modern ones do not face this stipulation. However, to re-use

the pre-trained weights, we must use the same split architecture as the one described in the original paper.

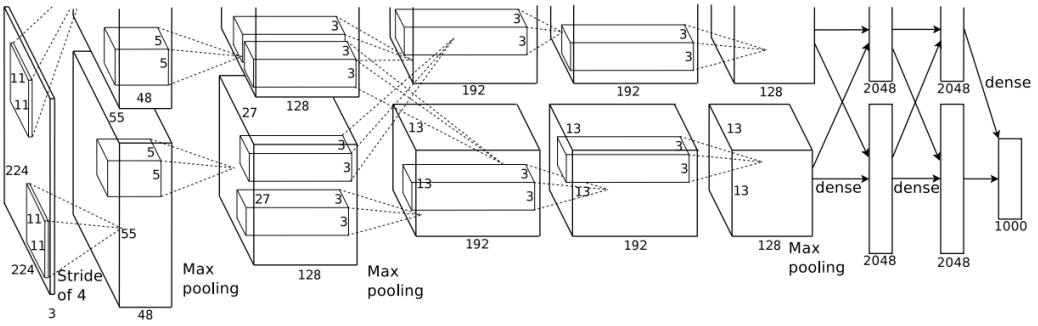


FIGURE 4.9: Architecture of the Alex-Net [36]. In our case, we change the last fully-connected layer to output six classes and add an additional pose input.

Although the Alex-Net was pre-trained on ImageNet, we can still benefit from re-using the learnt lower level visual features shared between the two tasks. In-fact Pinto and Gupta [59] suggest that training on ImageNet results in an increase in test accuracy from 64.6% to 76.9% on their grasping network. However, since the input the the Alex-Net is different to our data-set, we need to further pre-process our images to conform to the correct shape.

The Alex-Net takes a 227x227 input image with three channels, with pixel values in the range [0, 255]. Our data-set, on the other hand consists of 32x32 depth images. Since CNNs share weights across *kernels*, the input dimensions do not strictly matter. However, in our case, since the convolution operation in the Alex-Net does not preserve shape, the 32x32 images would be reduced to degeneracy (shape zero). To reconcile these difference, we re-normalise the images to the range [0, 255] and remove the pre-trained weights from the second and third input channel; we expect to be able to overcome any differences arising from the weights' initialisation during training. Finally, we use cubic interpolation to upscale the images to size: 227x227.

As in the case of the GUDNT, we start by training on a binary classification task using robust ϵ-metric labels thresholded at 0.002 using a momentum optimiser. We achieve a validation accuracy of 89% without fine-tuning the hyperparameters. We use this network to then train on uncertainty labels generated using percentile, log-scale and clustered binning. The training curves for the percentile and clustering methods are summarised in Figure 4.10.

The deeper architecture of the Alex-Net results in a roughly 10% improvement in accuracy for both percentile and clustered labels. This suggests that there might be further improvement to be gained through fine-tuning the architecture and hyperparameters.

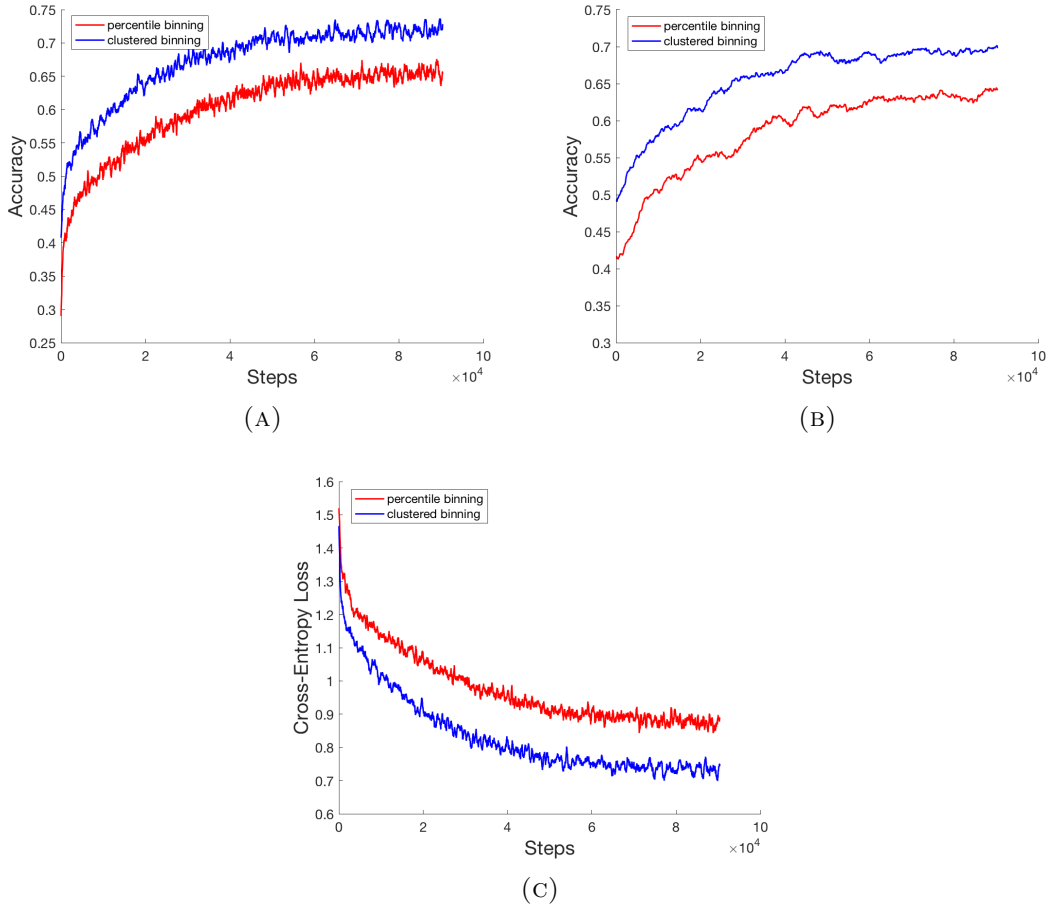
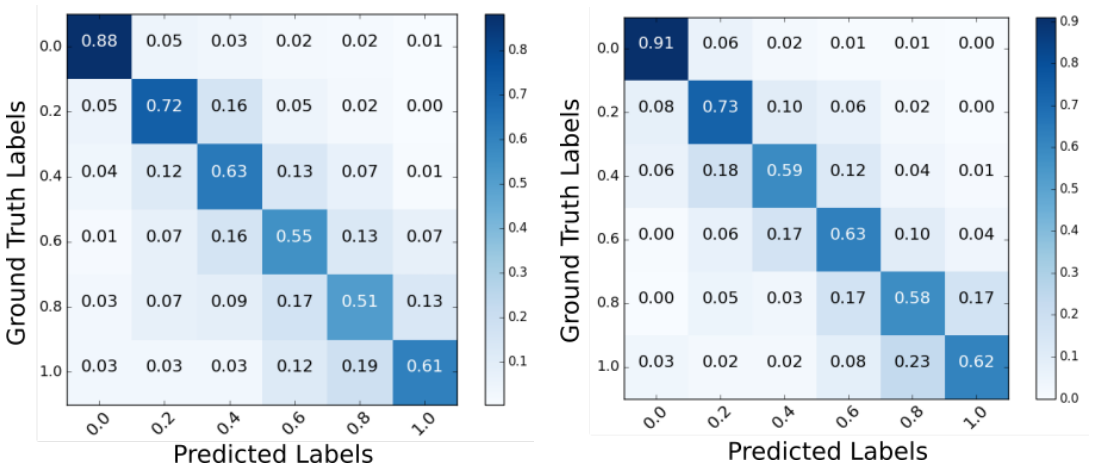


FIGURE 4.10: (A - B) Training and validation accuracies and (C) the cross-entropy loss curves for the GUANT with percentile and clustered labels. As in the case of the GUDNT, the network performs better when trained on clustered labels, achieving a text accuracy of 76.67%.

4.6 Evaluating Architectures

In both the GUDNT and GUANT architectures discussed, the test accuracy with the clustered bins plateaus at 67.67% and 76.67% respectively. While this might appear suboptimal, we must consider that our network is now learning a much harder task than binary classification; it must differentiate between subtle differences in the input images that might be imperceptible to the human eye. Eventually, what matters more than the raw accuracy, is the spread of the predictions around the ground-truth class. A bin 2 grasp classified as bin 6 is a lot worse than it being classified at bin 3. To quantify this statistic, we plot confusion matrices for both architectures in Figure 4.11.



(A) (B)

FIGURE 4.11: Confusion matrices for the best performing GUDNT (A) and GUANT (B) models, tested on 1500 unseen data-points. The concentration of predictions around the diagonal indicates that the networks successfully learn to predict the correct neighbourhood of the ground truth class. Since our underlying grasp metric is a continuous 1D function, this suggests that the networks can discriminate the relative quality of grasps. The deeper architecture of the GUANT helps increase accuracy while reducing the spread around the diagonal, which might indicate that higher expressiveness of the architecture allows the network to learn a more nuanced representation.

We can see that for both architectures, the network predictions are distributed close to the diagonal of the confusion matrices. This suggests that even when the network does not predict the correct class, it is often in the correct region of the distribution. This is of particular importance since our underlying metric is a continuous distribution. Both architectures perform best at identifying grasps in collision (bin 0), while struggling to distinguish between grasps at the higher end of the quality spectrum. This might partly be due to the skewness of our data; there are significantly fewer high-quality grasps to learn from than grasps in collision. A second reason might be that there are not enough visual features present in a 2D depth image to distinguish between grasps calculated in 3D. In other words, we only see the projection of the object onto the camera plane and therefore loose both local and global geometry information.

Chapter 5

Evaluation

So far, we have discussed elements related to the learning component of grasping. We trained a CNN to *discriminate* between different grasps and classify them into one of six *quality* bins, depending upon their likelihood of producing robust force-closure. While a discriminative model such as this allows for a more learnable grasp function, it leaves the question of finding candidate grasps unanswered.

In this chapter, we discuss our approach towards implementing a grasp generator for depth images. We pre-process depth images, sample candidate grasps, classify them based on grasp-quality, and iteratively refine the results to produce an a smoothed grasp function. In particular, we use the Cross-Entropy Method [75], [61] to sample new grasps from a Gaussian Mixture Model, fitted on the best grasps from the previous iteration. Lastly, we present a qualitative analysis of our results and compare it to related work.

5.1 Optimal Grasp Planning

Our goal in this project has been to define a quality-function over an object that allows for robust grasping, while accommodating uncertainty in gripper pose. Since our neural-network is discriminative, we must first generate candidate grasps from input RGB-D images, that can be fed into the classifier, and reassemble the output to construct the grasp function.

We start off by sampling antipodal grasps on the input image using the `gqcnn` Python package [64]. This module takes RGB-D images as input along with the camera calibration and transformation matrices, and outputs sampled grasp coordinates in the image frame. The work-flow for this process is as follows: the depth channel of the input is used to extract edges, that are turned into a 2D point cloud by discarding the gradient

information; next, the RGB channels are used to extract edges at these 2D points along with their associated surface normals. Finally, the 2D point-normal pairs are used to sample antipodal grasps, which are filtered by testing for force-closure using simulated frictional properties of the object-gripper interface. Section 5.1 visualises the stages of this process.

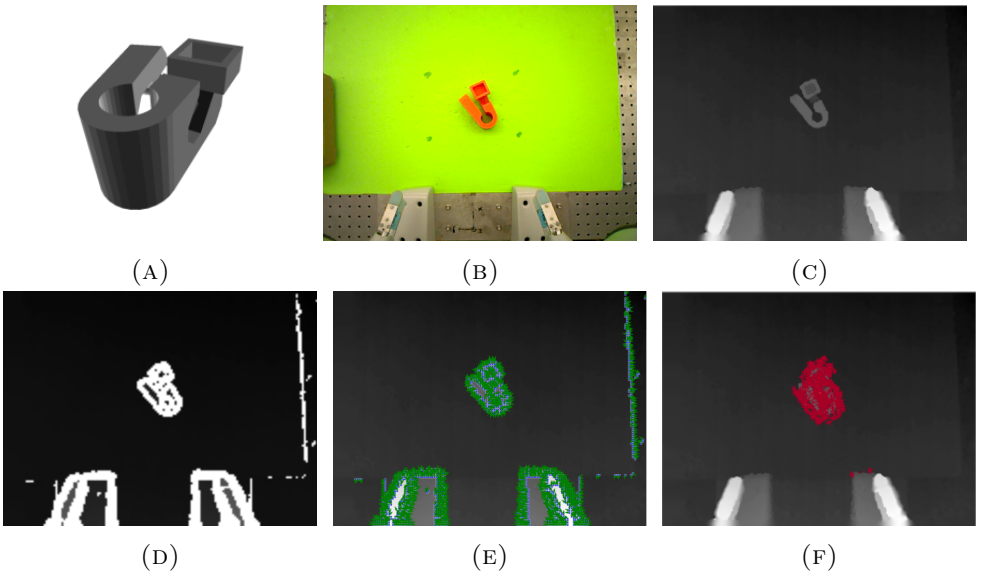


FIGURE 5.1: The stages of grasp sampling. (A) The model mesh used in training. (B-C) RGB and depth images of the real-world object. (D) Edge map computed on the depth image. (E) 2D points (blue) sampled on the depth edges along with surface normals (green) computed on the RGB image. (F) Initial antipodal grasp samples.

This gives us the grasp locations as pixel coordinates in image space. To convert this into input compatible with our network, we generate new depth-images by centring and aligning each sampled grasp on the image and cropping it to the appropriate dimensions. At the same time, we also keep track of the inverse transformation to reassemble the raw depth image. This is best motivated visually through Figure 5.2.

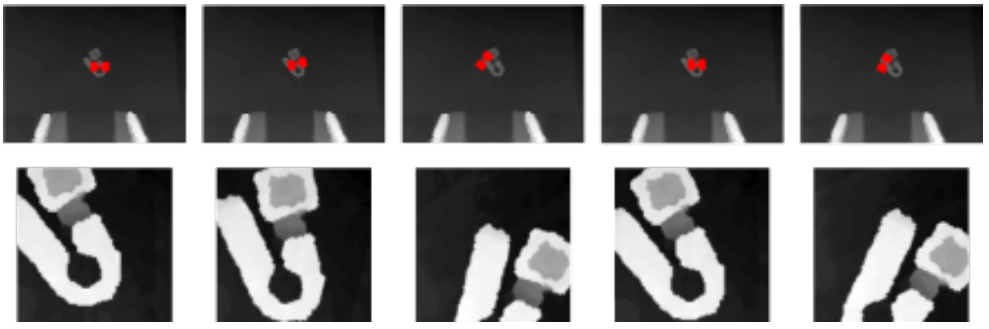


FIGURE 5.2: Top row: grasps sampled on the input depth image. Bottom row: cropped depth images transformed to align with the gripper jaws.

We feed the pre-processed images and grasps to our network to predict the grasp quality bins for each candidate. However, owing to noise and sampling imperfections on the input image, in particular the first-order approximation of surface-normals, the initial candidate grasps are mostly of low quality. They are either in collision with the object or oriented at a large angle to the ground-truth surface normals. This results in a majority of the initial grasps being classified as bin 0 or 1. Figure 5.3 illustrates a sub-sample of the classes predicted by our network on these initial grasps; the model correctly identifies grasps in collision and distinguishes them from those that are viable but of extremely poor quality. In order to implement successful grasps on real-world objects, we need a better strategy to sample the initial candidates.

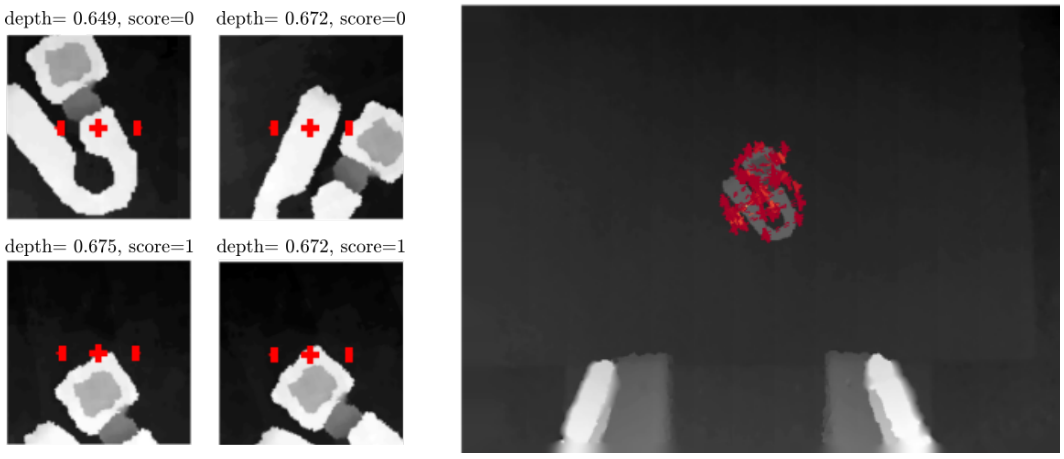


FIGURE 5.3: (A) Initial grasp candidates generated using an antipodal sampling policy. Most of the grasps are either in collision or represent extremely poor quality scores. (B) Grasps visualised on the raw depth image.

5.1.1 Smoothing the Grasp Function

To improve our grasp sampling, we use the Cross-Entropy Method (CEM), described by Deng [75] and adapted to a grasping setting by Levine *et al.* [61]. It allows us to use information about the most robust candidate grasps and iteratively generate similar ones by drawing samples from a Gaussian-Mixture Model (GMM). Since our network predicts a distribution over quality labels, we simultaneously smooth the grasp function when sampling, as motivated in Section 3.3.1. In effect, the sampling and smoothing policies are coupled. We discuss the side-effects and potential alternatives to this approach in Chapter 6.

We start by feeding the samples, generated using the antipodal sampling policy, through our network and applying a *soft-max* function, $\sigma(x)$ to the output, instead of classifying

it into bins. The soft-max has the property that the output is in the range [0, 1] and integrates to one. This allows us to interpret the network output as probabilities over a six class categorical distribution, z. Using this representation, we calculate the expected value of each grasp candidate, which can now be interpreted as a quality score in the range [0, 1].

$$\sigma(x) = \frac{exp(x)}{\sum_{k=1}^{K} exp(x)} \tag{5.1}$$

$$\mathbb{E}(x) = \sigma(x_1)z_1 + \sigma(x_2)z_2 + \ldots + \sigma(x_K)z_K \tag{5.2}$$

Where, $\mathbb{E}$ is the expected value of the categorical grasp quality distribution with class labels $\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ and x is the vector of raw network outputs.

We sort the samples based on their expected quality and use the top N grasps to fit a GMM with P components. The model is fitted on 5D vectors containing the pixel coordinates of each gripper jaw as well as the depth of the gripper centre. This allows us to iteratively refine our grasp candidates by sampling the GMM to generate data-points that inherit the best properties of the previous samples. At the end of the CEM optimisation, we run the final grasp candidates through our network and classify each one into its respective quality label. The algorithm is summarised in Algorithm 1.

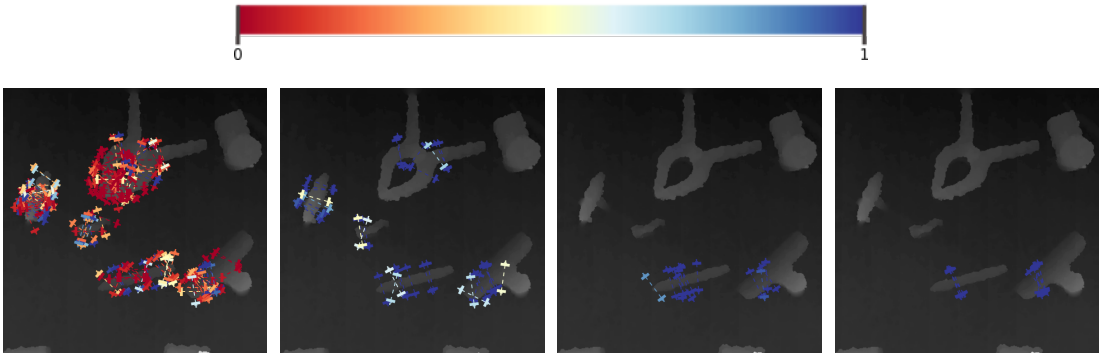
Data: Set $\mathcal{U}$, of M uniformly sampled grasps; number of grasps N to fit GMM $\mathcal{G}$; number of components P of $\mathcal{G}$. Number of iterations I of CEM optimisation;
Result: Set $\mathcal{V}$, of the highest ranking grasps.
for $i = 1, \ldots, I$ **do**
  $\mathbb{E} \leftarrow$ top N grasps from $\mathcal{U}$ % Expectation Step
  $\mathbb{M} \leftarrow$ argmax over model parameters % Maximisation Step
  $\mathcal{G} \leftarrow (\mathbb{E}, \mathbb{M})$ % Construct GMM
  $\mathcal{U} \leftarrow M$ samples drawn from $\mathcal{G}$
end
$\mathcal{V} \leftarrow \arg\max_{\mathcal{U}} \mathcal{G}$

Algorithm 1: Cross-Entropy Method for generating new candidate grasps [75].

A key point to note is that we use the expected value of the categorical distribution rather than the discrete class labels $\{0 \rightarrow 1\}$. This is necessary since the class labels in themselves do not contain enough information to seed better grasps. For the example illustrated in Figure 5.3 for instance, out of the 200 sampled grasps, 198 are bin 0, indicating grasps in collision and 2 are bin 1, implying a poor quality score of 0.2. In

effect, we fail to capture information regarding grasps that might have been good (bin 0.8 or 1.0) but were marginally in collision. By using the expected quality instead, we incorporate this information by computing the weighted average of all possible classes.

Since the grasp function is smoothed locally, planning and execution can be carried out on multiple objects simultaneously. Figure 5.4 visualises the best grasps used for fitting GMMs during the CEM optimisation and smoothing, in a scene containing 6 objects. The peaks of the resulting grasp function are relatively higher at each iteration, with the final one containing grasps qualities in the top bin (0.9 - 1.0). Note that the object at the top right of the image fails to yield any candidate grasps because it is too wide for the gripper to fit around.

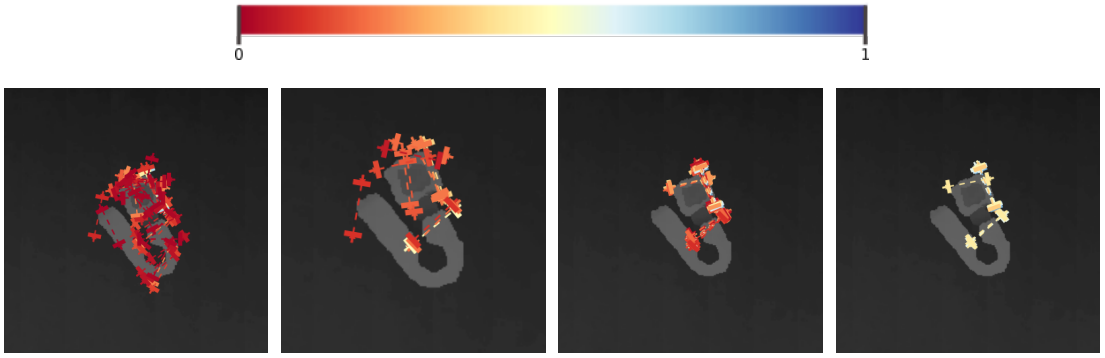


(A) (B) (C) (D)

FIGURE 5.4: Candidate grasps at different stages in the CEM optimisation, where the colour of the grasp indicates its quality score. In this case, we resample fewer grasps M at each successive iteration since they clusters into regions of high quality.

Figure 5.5 visualises the CEM optimisation for an object with an adversarial geometry for grasping. There are very few regions on the object that can accommodate the gripper dimensions; most of the potential grasp locations are too narrow for the gripper jaws (0.5 cm) to fit. This yields a suboptimal grasp function, with the final scores from the CEM optimisation lying midway on the quality spectrum, as evident in Figure 5.5

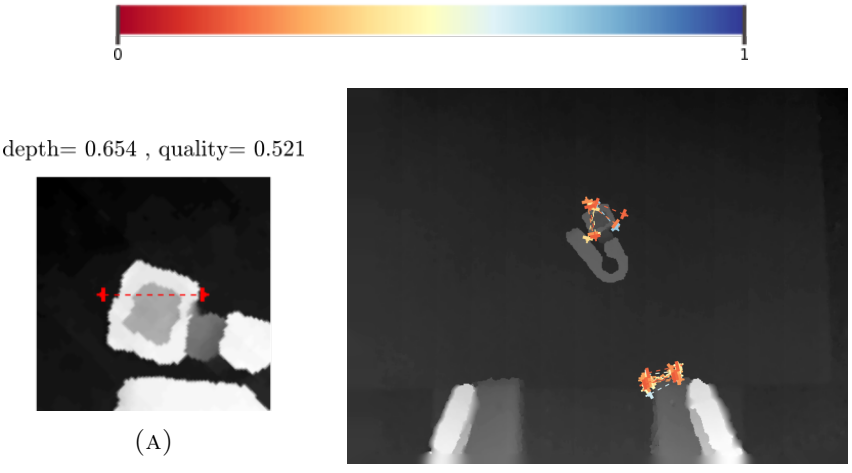
Despite the quality of the overall grasp function, we can see that the model chooses final grasps in the vicinity of the widest opening in the object geometry, where the gripper is most likely to fit without collision. In other words, it takes the uncertainty in gripper position into account when assigning quality scores.



(A) (B) (C) (D)

FIGURE 5.5: Best sample grasps at different stages in the CEM optimisation for an object with an adversarial grasp geometry.

A potential shortcoming of our approach is that we do not include the region surrounding the object in our grasp smoothing kernel. This results in the best candidate grasps lying close to the edges of the object (top right in this case) — an outcome we wish to avoid. In effect, poor grasps *on* the object push the peak of the grasp function towards the edge, since there is no counter push from the region surrounding the object. In hindsight, incorporating off-object grasps is a critical requirement for improving the grasp function close of the object edges. Implementing this however would require an entirely different approach to grasp sampling, and we therefore leave it to future work.



(A) (B)

FIGURE 5.6: (A) Final grasp selected following the CEM optimisation with an expected quality of 0.521. (B) Failure mode of our approach resulting from outliers and misclassifications in grasp sampling.

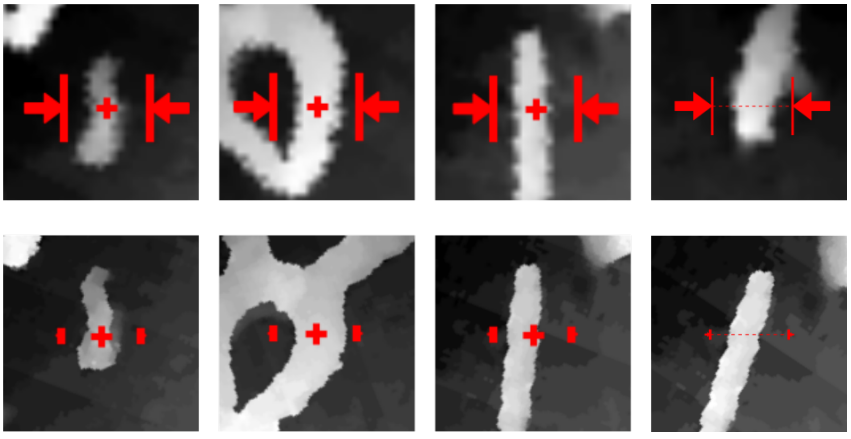
Another failure mode is caused by outlier grasps depicted in Figure 5.6b. If the original randomly sampled grasp function contains an outlier with a high relative quality score, it gets chosen to fit the GMM and continues to make its way into the next iteration if

the remaining grasps are in collision. In this case in particular, our model misclassifies the edge of the robot gripper, visible in the image, as a potential grasp.

5.2 Comparison to Related Work

Since our approach is based on generating candidate grasps on the fly, the theoretical state-space of potential grasps is infinite[1]. As the dynamics of the real-world play such an important part in grasping, traditional test methods rely on analysing performance on a physical robot. However, due to the limited time and resources available to us in this project, such testing is not possible. Instead, we analyse our approach by proxy, using a qualitative comparison of our results to those of Mahler *et al.* [64], and leave real-world testing to future work.

We use the Grasp Quality Convolutional Neural-Network (GQCNN) described by [64] along with the CEM sampling methods discussed earlier to generate grasps for comparison. As the GQCNN is a binary classification network, instead of taking the expected value of the six classes we use the probability of the positive bin to score candidate grasps as described by [64]. A comparison of the generated grasps is presented in Figure 5.7.



(A)

FIGURE 5.7: Sub sample of grasps generated by the GQCNN (top row) and the GUANT (bottom row) with the last column on the right representing the final selected grasps. In this case, both networks identify similar regions of the grasp function as peaks.

Both networks return similar regions for the best grasp location. Unlike the GQCNN, which only learns binary success labels, our network learns a *distribution* over the multiple quality bins. This allows for a smoother training signal when fitting the GMM

[1] This is not strictly true, since we are dealing with discretised representation of the state (pixels).

for grasp sampling. The difference in the two architectures is more evident for grasp planning on the adversarial object geometry visualised in Figure 5.8.

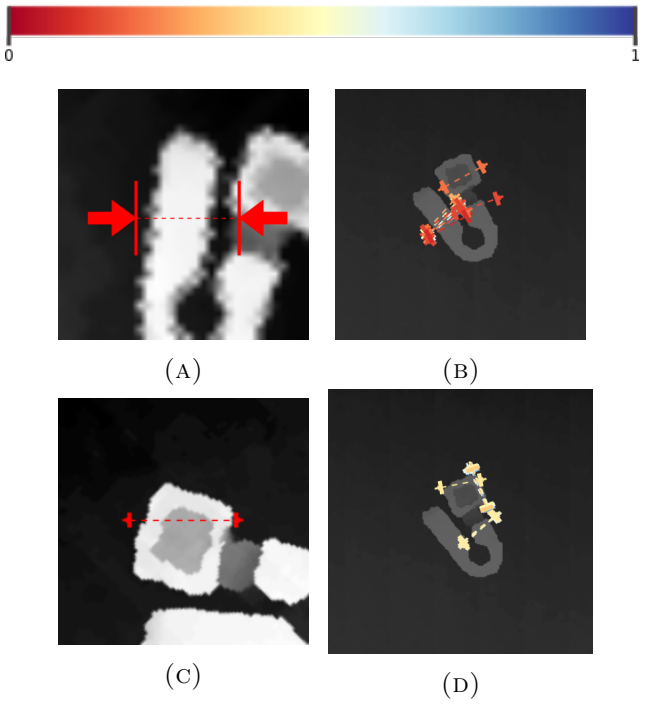


FIGURE 5.8: Grasps planning an an adversarial grasping geometry using (A-B) the GQCNN and (C-D) the GUANT. Our approach favours the region of the object that present a lower probability of collision. It however drifts towards the object edge due to smoothing by negative grasps lying on the interior.

While a qualitative analysis demonstrates the intuition behind our network, a thorough perusal of our approach can only be carried out on a physical robot. We discuss the steps necessary to achieve this along with other potential modifications to the grasping pipeline presented so far, in the next chapter.

Chapter 6

Conclusion

Incorporating measurement or belief uncertainty in estimating a random variable is a popular and well studied approach, used in a variety of applications. From predicting stock prices, to reconstructing 3D scenes and autonomous navigation, it allows us to account for measurement inaccuracies that are inevitable in real-world scenarios. Our goal in this project was to transfer this motivation to robotic grasping. In a complex task such as this, errors can accumulate from a range of sources: visual perception, robot joints, etc., which if unaccounted for, could lead to debilitating effects on performance. In our approach to this problem, we used a deep neural-network to learn a grasp quality function over an input RGB-D image of the object. The quality function encodes the uncertainty in successful execution of grasps predicted by our network. It is learnt in simulation in order to provide the large amount of data necessary for deep-learning. To this end, we reconcile the analytical grasp metrics from the early days of the field, with powerful machine-learning tools for learning complex functions.

We used a novel approach to generating quality bins by clustering non-linear grasp-metrics to form our data-set. We also tested different CNN architectures to study the effect of layer depth on the ability of the network to learn meaningful task representations. Our best architecture: the Grasp Uncertainty Alex-Net (GUANT) achieved a test accuracy of 76.67%, but more importantly learnt to minimise the deviation from the ground truth labels (Section 4.6). In addition, we also presented a highly optimised pipeline for loading and pre-processing data from file, which is useful when the data-set cannot be converted into a TensorFlow `.record` file.

Lastly, we presented our approach to using the expected value of the grasp function with the Cross Entropy Method [75], for iteratively refining the generated grasp candidates. Our network manages to identify graspable regions of the object, that appear intuitively accurate to a human labelling the same task. Indeed, our aim in this project can be

interpreted as teaching the network a grasp-heuristic, wherein grasps lying closer to regions of low quality (high uncertainty) are avoided.

In hindsight, we would modify our grasping strategy to include points not lying on the object when smoothing the grasp function. This would prevent the case where potential grasps are pushed towards the object's edges. We could also use a local RANSAC [76][1] approach to prevent outliers and misclassifications from adversely affecting the initialisation to the CEM optimisation.

6.1 Future Work

In our approach, we focused on clustering the ϵ-metric labels into classes as distinct from one another as possible. Nonetheless, the underlying structure of the task remains one dimensional. In other words, if one were to sort the quality labels, grasps in bin 1 would be more similar to bin 2 than bin 3, 4 or 5; neighbouring bins are more closely related than distant ones. We could use this information to weight our loss function such that misclassification on similar bins is penalised less heavily. This should help guide the optimisation towards the function minima more smoothly.

In addition to the analytical ϵ-quality, using a physics metric, as described by Kappler *et al.*, [68] could help improve results by more effectively modelling the dynamics of the real-world. Another approach is to use domain randomisation [63] to aid transfering from simulation to the real-world. This considers different types of noise and illumination to help the model learn higher-level details from the input.

We can also minimise the noise in the input, from a grasp sampling point of view, by using second order edge information. The `gqcnn` package used for grasp sampling only considers first order gradients at sampled edge pixels when computing antipodal grasps. This makes the estimation of surface normals extremely sensitive to measurement noise. Using the curvature information at the edges, smoothed by a local Gaussian filter, would considerably improve the robustness of this estimate.

In this project, we have discussed a bimodal approach to grasp execution: we split the task into an analytical grasp generation step and a learning based discrimination step. While neural-networks have traditionally excelled as discriminators, some of the cutting-edge work in the field today [8] is aimed at training them as generative models — Generative Adversarial Networks (GANs).

[1] **RAN**dom **SA**mple **C**onsensus — technique for minimising the effect of outliers.

An alternative approach to ours could involve training GANs to learn *one-shot* optimal grasp planning from raw RGB-D images. Along with the `gqcnn` package, we have provided all the tools necessary to achieve such a setup. From training a discriminator on analytical metrics, to sampling new depth-images directly from object meshes, and augmenting with grasps sampled on pre-existing images. What remains is to assemble these tools into an end-to-end pipeline, and utilise each at the task it performs best. Neural-networks after-all, are nothing more than tools that must be wielded effectively to maximise their utility.

Bibliography

[1] Spy in the wild. URL `http://www.bbc.co.uk/programmes/p04px5zw`. Accessed on 23/08/2017.

[2] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2308–2315. IEEE, 2010.

[3] Spotmini. URL `https://www.bostondynamics.com/spot-mini`. Accessed on 23/08/2017.

[4] Darpa robotics challenge (drc) (archived). URL `https://www.darpa.mil/program/darpa-robotics-challenge`. Accessed on 23/08/2017.

[5] Amazon picking challenge. URL `http://www.robocup2016.org/en/events/amazon-picking-challenge/`. Accessed on 23/08/2017.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[7] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6): 82–97, 2012.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[9] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[10] Edward Johns, Stefan Leutenegger, and Andrew J Davison. Deep learning a grasp function for grasping under gripper pose uncertainty. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4461–4468. IEEE, 2016.

[11] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 348–353. IEEE, 2000.

[12] Domenico Prattichizzo and Jeffrey C. Trinkle. *Grasping*, pages 671–700. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-30301-5. doi: 10.1007/978-3-540-30301-5_29. URL `https://doi.org/10.1007/978-3-540-30301-5_29`.

[13] Bhubaneswar Mishra, Jacob T Schwartz, and Micha Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2(1):541–558, 1987.

[14] P Somoff. Uber gebiete von schraubengeschwindigkeiten eines starren korpers bieverschiedener zahl von stuz achen. *Zeitschrift fur Mathematic and Physik*, 45:245–306, 1900.

[15] Steve Jacobsen, E Iversen, D Knutti, R Johnson, and K Biggers. Design of the utah/mit dextrous hand. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1520–1532. IEEE, 1986.

[16] Rui Li, Robert Platt, Wenzhen Yuan, Andreas ten Pas, Nathan Roscup, Mandayam A Srinivasan, and Edward Adelson. Localization and manipulation of small parts using gelsight tactile sensing. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3988–3993. IEEE, 2014.

[17] Bruno Rubinger, Paul Fulford, Loris Gregoris, Clement Gosselin, and Thierry Laliberté. Self-adapting robotic auxiliary hand (sarah) for spdm operations on the international space station. *Proceedings of I-SAIRAS, Quebec, Canada*, 2001.

[18] Matthew T Mason, J Kenneth Salisbury, and Joey K Parker. Robot hands and the mechanics of manipulation, 1989.

[19] Matthew Mason. The mechanics of manipulation. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 544–548. IEEE, 1985.

[20] Ch Borst, Max Fischer, and Gerd Hirzinger. Calculating hand configurations for precision and pinch grasps. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 2, pages 1553–1559. IEEE, 2002.

[21] Jeffrey Coates Trinkle. The mechanics and planning of enveloping grasps. 1987.

[22] Giuseppe Carbone. *Grasping in robotics*, volume 10. Springer, 2012.

[23] WT Townsend. Mcbindustrial robot feature articlebarrett hand grasper. *Industrial Robot: An International Journal*, 27(3):181–188, 2000.

[24] Yumi - creating an automated future together. URL `http://new.abb.com/products/robotics/industrial-robots/yumi`.

[25] Menglong Guo, David V Gealy, Jacky Liang, Jeffrey Mahler, Aimee Goncalves, Stephen McKinley, Juan Aparicio Ojea, and Ken Goldberg. Design of parallel-jaw gripper tip surfaces for robust grasping. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2831–2838. IEEE, 2017.

[26] The shadow robot company dexterous hand. URL `https://www.shadowrobot.com/products/dexterous-hand/`. Accessed on 23/08/2017.

[27] Minzhou Luo, Tao Mei, Xiaohua Wang, and Yong Yu. Grasp characteristics of an underactuated robot hand. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2236–2241. IEEE, 2004.

[28] Andrew T Miller, Steffen Knoop, Henrik I Christensen, and Peter K Allen. Automatic grasp planning using shape primitives. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1824–1829. IEEE, 2003.

[29] Andrew T Miller and Peter K Allen. Graspit! a versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine*, 11(4):110–122, 2004.

[30] Ying Li, Jiaxin L Fu, and Nancy S Pollard. Data-driven grasp synthesis using shape matching and task-based pruning. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):732–747, 2007.

[31] Corey Goldfeder, Matei Ciocarlie, Jaime Peretzman, Hao Dang, and Peter K Allen. Data-driven grasping with partial sensor data. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1278–1283. IEEE, 2009.

[32] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[33] Corey Goldfeder, Matei Ciocarlie, Hao Dang, and Peter K Allen. The columbia grasp database. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1710–1716. IEEE, 2009.

[34] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[35] Jeannette Bohg and Danica Kragic. Learning grasping points with shape context. *Robotics and Autonomous Systems*, 58(4):362–377, 2010.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[37] David Tidd. Grasping - 3d grasp quality computations. 2013. URL `http://people.duke.edu/~kh269/teaching/b659/David-Grasping2.pptx`.

[38] Beatriz León, Antonio Morales, and Joaquin Sancho-Bru. *Robot Grasping Foundations*, pages 15–31. Springer International Publishing, Cham, 2014. ISBN 978-3-319-01833-1. doi: 10.1007/978-3-319-01833-1_2. URL `https://doi.org/10.1007/978-3-319-01833-1_2`.

[39] Zexiang Li and S Shankar Sastry. Task-oriented optimal grasping by multifingered robot hands. *IEEE Journal on Robotics and Automation*, 4(1):32–44, 1988.

[40] Carlo Ferrari and John Canny. Planning optimal grasps. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2290–2295. IEEE, 1992.

[41] Daniel Seita, Florian T Pokorny, Jeffrey Mahler, Danica Kragic, Michael Franklin, John Canny, and Ken Goldberg. Large-scale supervised learning of the grasp robustness of surface patch pairs. In *Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), IEEE International Conference on*, pages 216–223. IEEE, 2016.

[42] Neurons, nerve tissues, the nervous system. URL `http://biomedicalengineering.yolasite.com/neurons.php`. Accessed on 23/08/2017.

[43] Michael A. Nielsen. Neural networks and deep learning. URL `http://neuralnetworksanddeeplearning.com/index.html`. Accessed on 23/08/2017.

[44] Suzana Herculano-Houzel and Roberto Lent. Isotropic fractionator: A simple, rapid method for the quantification of total cell and neuron numbers in the brain. *Journal of Neuroscience*, 25(10):2518–2521, 2005. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.4526-04.2005. URL `http://www.jneurosci.org/content/25/10/2518`.

[45] G Gybenko. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[46] Paul John Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Doctoral Dissertation, Applied Mathematics, Harvard University, MA*, 1974.

[47] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10): 947–954, 1960.

[48] Arthur E Bryson. A gradient method for optimizing multi-stage allocation processes. In *Proc. Harvard Univ. Symposium on digital computers and their applications*, page 72, 1961.

[49] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

[50] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[51] D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[52] Victor Genin. Optimizations of gradient descent. URL `http://dsdeepdive.blogspot.com/2016/03/optimizations-of-gradient-descent.html`.

[53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[55] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3357–3364. IEEE, 2017.

[56] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39): 1–40, 2016.

[57] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

[59] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

[60] Baxter collaborative robots for industrial automation. URL `http://www.rethinkrobotics.com/baxter/`. Accessed on 23/08/2017.

[61] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, page 0278364917710318, 2016.

[62] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *arXiv preprint arXiv:1703.07326*, 2017.

[63] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*, 2017.

[64] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[65] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems*, pages 64–72, 2016.

[66] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[67] Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2161–2168. IEEE, 2017.

[68] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. Leveraging big data for grasp planning. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4304–4311. IEEE, 2015.

[69] Russell Smith. Open dynamics engine. URL `http://www.ode.org/`. Accessed on 23/08/2017.

[70] George F Jenks. The data model concept in statistical mapping. *International yearbook of cartography*, 7(1):186–190, 1967.

[71] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[72] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[73] Eamonn Keogh and Abdullah Mueen. *Curse of Dimensionality*, pages 257–258. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_192. URL `https://doi.org/10.1007/978-0-387-30164-8_192`.

[74] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[75] Lih-Yuan Deng. The cross-entropy method: a unified approach to combinatorial optimization, monte-carlo simulation, and machine learning, 2006.

[76] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.