Fully Autonomous Grasping Using Vision

George Maratos

Advisor: Dr. Brian Ziebart

Secondary Commitee Member: Dr. Xinhua Zhang

Abstract

Fully autonomous grasping is a planning task. A machine senses objects in its environment, and predicts the configuration of its end effector so that it can grasp the object. A grasp here is defined as a stable envelopement, by the machine's end effector, so that it can manipulate the object for simple tasks like picking and placing. When considering a fully autonomous setting, where the machine has no external help, grasping can be divided into three separate tasks. First the machine must be able to detect that an object is present in its field of view, then it must detect a location on the object which is most suitable for grasping, and finally it must position its end effector so that it can successfully execute the grasp.

In this project, I explore these three areas of fully autonomous grasping where the machine's only sensing capabilities are 2-D images from a camera. I design an architecture which can learn how to find grasps on the image plane, and train it on the Cornell Grasping Dataset. Given the difficulty of the task and the small size of the dataset, I employ techniques to extend the utility of the limited data like pretraining and image augmentations. I also explore the task of object detection on the Cornell Dataset. The tasks described above are difficult, and since there is already work that successfully solves the task in certain settings, I investigate if the task can still be solved using a lower complexity model. A lower complexity would be beneficial because most state of the art models, as far as I know, have very large architectures that require large amounts of computing power or a GPU to solve in reasonable time.

Contents

1 Introduction

In the robotic grasping problem the goal is, given an object, select a grasp configuration such that the object can be restrained and manipulated to some desirable end. Finding such configurations is difficult because of the multi-modal nature of the input and the fact that there can be more than one suitable grasping location, which means the machine must choose one grasp from many optimal ones to attempt. Some of the earliest reviews of algorithms for grasping [27, 1], shows that the premliminary work involved solving unconstrained linear programming problems using objectives that measure dexterity and grasp quality. A grasp algorithm in this context is one that is able to calculate the stability or equilibrium of its grasp, and they are collectively refered too in the review as *robot grasp synthesis algorithms*.

The review by Sahbani *et al.* [23], makes a distinction between methods that model the kinematics and dynamics of a grasp, like the synthesis algorithms, and methods that mimic human grasp strategies or learn from data. The former called the analytic methods and the latter empirical. They divide the analytical techniques into force closure methods and task oriented. The authors determine that force closure is able to find stable grasps but are usually not task oriented, and the task oriented strategies tend to be computationally intensive. The empirical methods on the other hand can model task oriented features, but struggle to generalize well to new objects.

Bohg *et al.* [2] observe that grasping methods typically aim to address the following object types: fully known, familiar, or fully unknown. When considering the empirical methods, fully known represents objects that have been seen in the training data before and the grasping problem reduces to locating the object and applying a similar grasp to those from experience. The familiar objects are assumed to have some matching characteristics to objects from the training data, like shape, color, and texture. Familiar objects introduce a new challenge of calculating similarity between objects so that the appropriate grasp can be determined. On the other hand, grasping algorithms will have no experience to utilize when approaching fully unknown objects. Methods of this category typically rely on finding structures in the features to synthesize a grasp.

The focus of this work is to build empirical models for grasp synthesis using the Cornell Grasp Datset [11], and evaluating them on familiar and fully unknown objects. Section 2 will describe previous methods for grasp synthesis. Section 3 will focus on analysis of the dataset and a description of the task. Section 4 will contain the experimental section. Finally, Section 5 is the conclusion and future works.

2 Existing Grasp Synthesis Methods

2.1 Analytical Methods

The earliest mention of qualities of a successful grasp, to the author's knowledge, is from *Kinematics of Machinery* by Franz Reuleaux [22]. In it, constant forces are applied to an object and it is considered constrained if sensible external forces can be balanced. When the object is in equilibrium, which occurs if the above conditions are met, then force closure occurs. Nguyen *et al.* [17], explored the notion of force closure and developed algorithms for computing the set of all force closure grasps on simple shapes. This work is extended by Ferrari *et al.* [5], to calculate a grasp quality criteria. The criteria is measured as the ratio between the force applied externally and by the fingers. The best grasp is determined by solving an optimization problem that minimizes the finger force but still can maintain force closure against a large external force.

Due to the multimodal nature of the task, it could be useful to model various features of the object being grasped. In the work by Zhang *et al.* [31], the authors explore bayesian filtering in $G-SL(AM)^2$. They build probabilistic models that model various features: the shape of the, object, mass, and friction coefficient for example. The application of this is for when you have input data blackout, which could occur if the arm during operation occludes the visual sensors.

The simulation software *GraspIt!* [15], was developed with the goal of aiding research in robotic grasping. It has a wide set of features such as, many different types of hand models, collision detection, and grasp quality evaluation. It could be used in setting where expensive robotic hardware is unavailable or used in an algorithm that evaluates grasps in the simulation before choosing the best one for a physical robot. The software was applied in work done by Miller *et al.* [16], where they designed a grasp planning algorithm that modeled objects as shape primitives and generated poses based off the primitives.

2.2 Empirical Methods

The empirical methods involve techniques that implement learning algorithms that model the grasping problem from data. This section will only discuss the non-deep learning methods, deep learning is discussed in another section.

The simulator Graspit! [15] enabled researchers to collect synthetic data for grasping, one example is by Pelossof *et al.* [18]. The authors generated synthetic data by subsampling the parameters from generated grasps and they trained an SVM-RBF regressor with the goal of predicting grasp quality. They generated grasps by fixing certain joints, for example to guarantee the palm runs parallel with the surface of the object, and allowing others to be sampled randomly within a range. Using their procedure, the authors were able to generate a dataset with approximately 80% of the grasps having a non-zero graspability score. Graspability being calculated in the same way as previous works [5].

Robots that are predicting grasps will not always have a complete model of the input, for example visual sensors might not be able to view an object from all directions. Kamon *et al.* [9] attempt to predict grasps from only 2-D images of the object. They solve two problems simultaneously: choosing grasping points, and predicting the quality of a grasp. About 2000 grasping examples were collected, which were used for prediction during testing. Features are extracted directly from the image, for example center of mass, and were used to calculate the quality of a grasp. The authors claim that grasp quality can be predicted reliably using only local features.

In some settings, a full 3-d model of the object to be grasped will be unavailable. The work by Saxena *et al.* [24, 25] address this. Given a set of images of the object in 2-d, it is possible to infer the location of it in 3-d. Their approach invovles modeling the probability that a grasp is present in the center of an image patch. Using synthetic data, they learn a set of parameters via MLE using logistic regression. The features are extracted from the image set and correspond to the following: edge filtering, texture masking, and color masking. To predict a grasp a MAP estimate is calculated for selecting the most probable grasp. In preceeding work [26], they address the issue that predicting a "grasp point" does little to model the physical limitations of the robotic grasping platform. They further incorporate depth maps and generate a new set of features that account for other properties: quatities representing the envelopement of the object by the hand, quantities that represent the distribution of the mass of the object in relation to the hand, and quantities that roughly calculate the shape of the object and the alignment of the hand in relation to this shape. In extending their previous work, their solution is two-fold with a model for predicting the location of a grasp and another model for predicting the probability that the robotic arm will be able to achieve the grasp using the extra features as predictors.

Learning a grasping point in the image plane may be easier for learning algorithms but they do not model the gripper configuration completely. The work by Jiang *et al.* [8], learn a rectangle configuration that can better represent a parallel gripper plate end effector. A rectangle, in this case, has 5 parameters which correspond to the 2-d coordinate of the top left corner of the rectangle, the height and width, and the angle between the first edge and the x-axis. They define a score function, which is a linear combination of features, that represents the quality of a grasp rectangle. The features are calculated from the pixels within the rectangle, and consist of 17 filters and 5-bin histograms as a set of quick features and a collection of more advanced non-linear features are used as well when doing the final inference. SVM's have been used in the past as a way to rank the quality of a grasp [10], in this work inference is done in a two step process with two separate SVM ranking models. The first model ranks using the features that are easy to compute, and this prunes the space of possible rectangles. The second ranks the rectangles chosen by the first model. For calculating the rectangles on an angle, they rotate the image in discrete intervals of 15° before calculating the features.

One approach to leveraging empirical data is to apply KNN similar to previous works [32, 3]. The objects are represented geometrically and a good grasp is modeled based on the quality of grasps from previous examples in the same coordinate position. A simple discriminator is used to compute a binary score for a grasp from the dataset and a new example. It compares the coordinates from both grasps and returns 0 if they contain grasps, and returns 1 if they do match. Using this metric, they model the probability of a good grasp occuring at a coordinate by considering the ratio of good grasps and bad grasps in the set of nearest neighbors.

Some early approaches try to combine visual features with haptic feedback, [19, 4]. Finding parameters for high quality models in this case is done using Expectation Maximization, learning and execution stage are not separated. Visual features are used to recommend grasp configurations, and from these configurations the machine will probe the surface of the object with its end-effector until a stable grasp is found. This work represents a critical shift where a grasping algorithm does not explicitly represent objects geometrically but instead sythensize a grasp from incomplete images and feedback of the object.

Many works try modeling the probability that a grasp lies in some coordinate in the image plane. Another approach is to consider grasp synthesis as a planning problem, which can be about finding a configuration for the machine in environments of high uncertainty. Sensors could give a partial view of the object, for example the front of cup, and the machine must be able to quantify this uncertainty and act based off this information. A machine that can recognize the uncertainty in a grasp can decide if it has enough information to attempt the grasp or if it needs to probe the environment for further data. In Hisiao *et al.* [7], the authors partition the configuration space of the robot into a discrete set of commands which act as states in a Partially Observable Markov Decision Process.

2.3 Deep Learning Methods

The Deep learning methods can be considered a subset of the empirical methods that has shown promise for many computer vision tasks, see the review by Voulodimos *et al.* [29] for an overview of recent advances. As described above, some approaches to grasp synthesis involve the usage of visual features. One challenge in tasks of computer vision is the selection of powerful and discriminative features, and deep learning methods are useful for learning features from the data as opposed to hand-crafted ones.

Using the configuration in [8], where the grasp is represented as a rectangle, the task can be formulated in two ways. The first where the machine is asked to evaluate if a given rectangle contains a grasp, which is known as recognition. The other is detection, where the machine is asked to identify all possible grasps from an image. The work by Lenz *et al.* [11], demonstrated an approach to grasp detection which involved a two step cascaded approach. First, they employ a shallow network to learn features that will discriminate good grasp locations from bad. The network is shallow so that fast detection can occur. Then, they use a deeper network for selecting the best grasp from the set of grasps proposed by the first network.

In the work done by [33, 30], the authors design a real-time method for predicting poses of a parallel gripper plate on RGB and RGB-D images. The extend the idea of anchor boxes, mostly used in object detection, and add anchors not just for the position in an image but also the angle orientation of the rectangle. They design a network that uses a ResNet50 backbone, and apply a small convolutional filter to the features for grasp detection and orientation of the end effector. They also use the Cornell Dataset and employ similar techniques for image augmentation.

A final approach discussed here is the one by Levine *et al.* [12], and Pinto *et al.* [20]. The approach is different than others by learning how to grasp through trial and error. Through thousands of trials, the robots in both works were able to devise a strategy for grasping objects. The method for learning was done via Reinforcement Learning, and the data collection step was fully autonomatic requiring little human intervention.

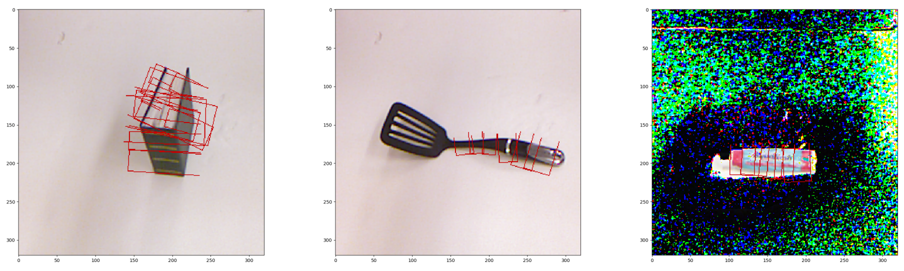
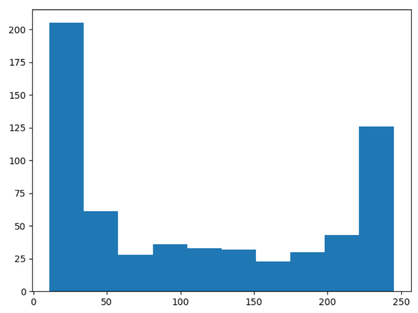
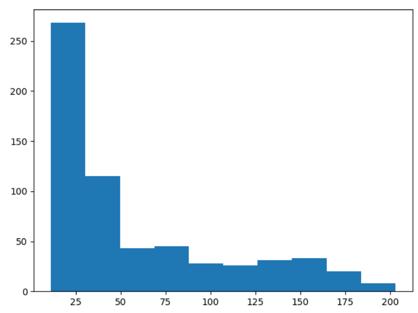


Figure 1: ground truth and images in the dataset, the right most image has background subtraction



(a) Old Backgrounds



(b) New Backgrounds

Figure 3: The Change in the Mean Pixel Distribution using the New Backgrounds

3 Data Analysis

The dataset is approximately 880 images, which contain common household and office objects. The entire dataset consists of about 10.4 GB. Each image has one object, centered in the image, placed on a white table. Every object has a set of grasping rectangles, that are defined by the position of the corners. Most objects have around 5 rectangles, but there are outliers with over 20. See the histogram showing the distribution of rectangles per image in Figure 13.

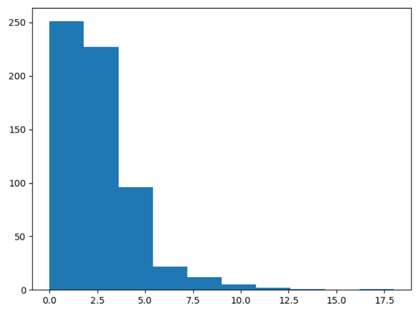
We also observe that the distribution of the rectangles apears to be normally distributed (see Figure 15) and also that width is also normally distributed with length being closer to a half normal. The other offsets are more uniform. See Figure 2 for detailed information about general information of the dataset.

	Objects	Images	Rectangles
Train	168	617	3567
Val	24	81	456
Test	52	185	1082
Totals	244	883	5105

Figure 2: Summary Counts of the Dataset

The dataset does have a set of labels for classification. There are 93 classes in total, with 78 classes being represented in the training data. The classes are more relevant for the object detection task, as the model for grasping does not learn or use class data explicitly. As such, it will be discussed more then.

The method used in this paper employs an anchor box based detection model. The input is divided into a grid of anchors. For each of these anchors, a prediction is made whether or not a grasp lies inside the anchor box or not. At the same time, the model predicts where the grasp lies as an offset of the top left corner of the anchor box. See Figure 5 for a visual representation of what a prediction looks like. Selecting the appropriate anchor box width is important, because since the model makes only one prediction per anchor box there could be a case where the dataset will have two rectangles that fit in a single anchorbox. Here an anchor box is a square with a side length of 32 pixels. This is


(a) Train

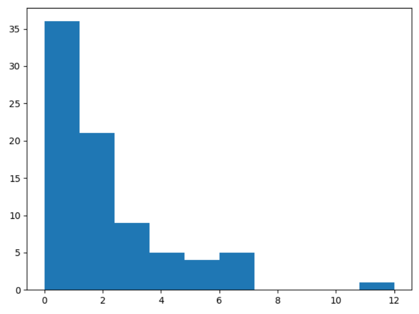

(b) Val

Figure 4: The Distribution of Number of Overlaps per Image on the Training and Validation Sets

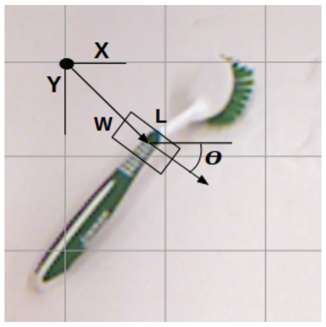


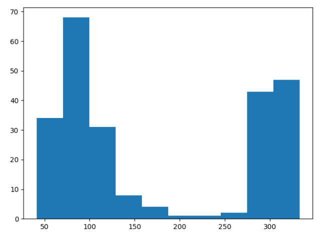
Figure 5: The image is partitioned into an anchor box grid. If a grasp lies in an anchor box then its position in the box is defined by (X, Y, L, W, and θ)

chosen because of constraints imposed by the backbone of the network. See Figure 4 for a distribution of the number of overlapping rectangles per image, given that anchor size is 32.

3.1 Data Preprocessing

Given that the dataset is small, in comparison with other datasets for deep learning, it is important to perform data augmentation to prevent overfitting. Following Zhang *et al.* [30] the augmentation done to each image is the following, a random flip horizontally, a random flip vertically, a random rotation of the image about the center ranging from $-30°$ to $30°$. The background image is also subtracted to reduce noise. Finally, the image pixels are normalized to a range of $[0, 1]$.

There are cases where the background image subtraction does not fit well with the original image, and to evaluate the quality we observe the mean pixel value of an image. As a general rule for evaluating the quality of a subtraction, if an image has the background removed, then the mean pixel value would be low as the majority of the pixels have a value of 0. If we use the annotations in the dataset, the average mean pixel value will be approximately 112.11 when sampled from the training set, with a standard deviation of 87.62. Since there could be many mislabels in the backgrounds, a procedure was developed with searches for the background image that finds the lowest mean pixel value for each image. Using this method, the mean drops to 58.51 with a standard deviation of 49.52. Since the background images are virtually the same, being a white table with no object, we hypothesize that there are some mistakes in the annotations, or that variations in the lighting between shots have impacted the accuracy of the background images. See Figure 3 for details on the shift in mean pixel value.


(a) Area Distribution of YOLO Bounding Boxes (square root)

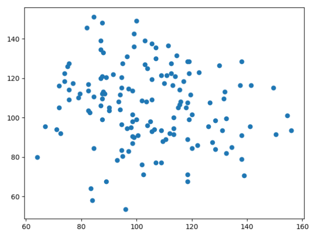

(b) Center of Bounding Boxes from Inference with Yolo (filtered)

Figure 6: Observations from the Object Detection Experiment

4 Object Detection

Grasp detection can be seen as a subset of object detection. As stated earlier, detecting the object to be grasped can be helpful in finding the optimal grasp location. A proposed approach to grasp detection is to use a standalone object detection model to filter out background noise, then apply a more expensive algorithm for detecting grasps and finding end-effector orientation. We explore how one could apply an object detection module to the Cornell Grasping dataset.

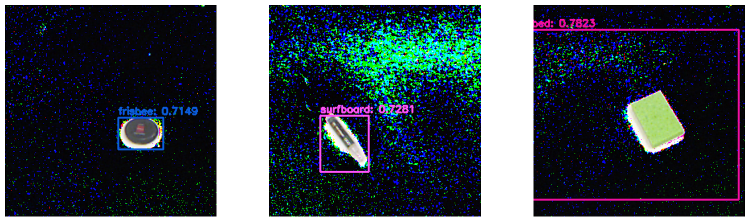
We consider one state of the art model YOLO [21], which represents a class of architectures that take a fully convolutional approach to object detection. It has the advantage over other high performance models like Faster-RCNN [6] of being faster because there is no region proposal stage. Visual features are convoluted sequentially and the output of the network is a tensor. The postion of the values in the tensor are important as they encode information about the location of predictions on the input image plane.

For our analysis, we use a pretrained model that was trained on MSCOCO [13]. Which is a general purpose, and widely used, dataset for object detection. A question arises of whether or not MSCOCO will perform well in this setting. If the model observes many objects that were not present in MSCOCO, its performance might suffer. We can analyze the precision of the bounding boxes during inference, and how close the class labels are in both datasets.

Measuring the precision of the bounding boxes is difficult because the Cornell dataset does not have any annotations for object detection, but we can still get some rough estimates by understanding how the data is configured. All images are, for the most part, centered on a white table and never take up more than roughly a third of the center of the image. So by looking at the distribution in position of the centers of the bounding boxes and how large the area of the bounding boxes are we can deduce how well YOLO performs in this new setting.

Starting with examining the class name overlap. YOLO was trained on a set of 81 classes, and after running a naive search of simply matching names with Cornell we can observe that there are about seven classes total that have explicit matches.

The distribution of the areas was calculated and shown in Figure 6 a. What can be observed is a natural split around the bounding box area (square root) of 200. When the examples with areas larger than 200 are filtered out, the total number of detections is 145 compared to the original 239 on the training set. The positions of the bounding box centers (on the original 320x320 image can be found in Figure 6 b. The positions are mostly centered, if we consider being centered as existing in the center $1/3$ of the image. Although, the author admits that further analysis might be necessary. See Figure 7 for examples of detections.



(a) Good (b) Wrong Class (c) Bad Precision

Figure 7: The Three Examples of Detections When Using YOLO on Cornell

5 Rectangle Grasp Model

A grasp is defined as a rectangle, oriented in the image plane. A robot with parallel gripper plates for end effectors would be able to line up its plates parallel with the short ends of the rectangle (width) and close its gripper along the long edge (length). This type of end effector can be better understood as a pinching action. The grasping model is designed to predict the location of all the rectangles that would result in a successful grasp. It does this by dividing the image into a grid of anchor boxes, then makes a prediction for each anchor box. A prediction involves the detection of a grasp existing in an anchor box, and the five grasping parameters necessary for configuring the end effector.

See Figure 8 for a diagram of the architecture. The model consists of two major parts. The first being the backbone which extracts visual features from the data, which is VGG16, [28], using batch normalization and has approximately 14 million parameters. The pretraining was done on ImageNet for the task if classification. The final piece is a single convolutional layer with seven filters, which represent the five grasping parameters x, y, l, w, θ and two scores for graspability. Figure 5 shows how the five grasping parameters are used to calculate the location of the grasp. The output of the model is a tensor of shape (Batch Size) x 7 x (num of Anchors in Width) x (num of Anchors in Height). During training the ground truths are matched with the corresponding vector of 7 values in the output tensor. To perform inference, the model predicts whether a not a grasp exists in an anchor by comparing the predicted scores for graspability and non-graspability and whenever the graspability score is larger than the non-graspable it is considered a positive prediction.

The loss function has two components, the first being the binary cross entropy The cross entropy represents the loss measure for grasp detection. Detection ground truth, in this case, is simply 1 if there exists a grasp and 0 if it does not. The grasp configuration loss is calculated by a Smooth L1 Loss between the predicted 5 values and the calculated ground truth. The total loss is inspired by the work by Zhang *et al.* [30] who use a similar metric. The final loss can be calculated as follows.

$$loss = 2 * CrossEntropy(s_p, s_t) + smooth\_l1\_loss(\{t_x, t_y, t_w, t_l, t_\theta\}, \{x_t, y_t, w_t, l_p, \theta_p\})$$

Where s_p is the network's confidence that a grasp exists and s_t is the binary ground truth.$t_x, t_y, t_w, t_l, t_\theta$ are the predicted configurations and $x_t, y_t, w_t, l_t, \theta_t$ is the ground truth. Cross entropy is scaled to emphasize the importance of detection, which is arguably the harder part of the task. The predicted parameters were encoded the following way. The ground truth is also encoded in the following format.

$$t_x = \frac{x - x_a}{w_a} \qquad t_w = \log(\frac{w}{w_a}) \qquad t_\theta = \frac{\theta}{2\pi}$$

$$t_y = \frac{y - y_a}{l_a} \qquad t_l = \log(\frac{l}{l_a})$$

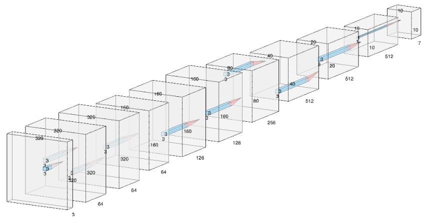


Figure 8: Architecture of VGG16 plus head. The predictions are extracted from the final tensor

6 Experiments

6.1 Hardware and Software Used

See Figure 9 for details about the machine used for training.

All work was done using Python 3, and the major libraries were Numpy, OpenCv, and Pytorch. Numpy and OpenCV were used for image processing and augmentation, and all Deep Learning Modules were done in Pytorch, with GPU support. The pretrained networks also come from Pytorch, specifically from the torchvision library. All code and documentation can be found at `https://github.com/gpmaratos/rgrasp.pytorch`. The documentation is generated using Doxygen.

CPU	Intel Xeon E5-2697 @ 2.60 GHz
RAM	157 GB DDR4
GPU	NVIDIA GeForce GTX 1080 Ti 12 GB

Figure 9: Machine Specifications

6.2 Results

The prediction space for the Rectangle Grasp Model is very large, there are 100 anchors with the majority of them not containing a grasp on average. This makes training difficult as the model has a tendency to optimize towards predicting negative for all anchors. There were a couple ways chosen to mitigate this, the first was to weigh the classes in the cross entropy calculation in favor of the positive class. In this case, we chose a scale of 1 to $\frac{3}{2}$. The next strategy was to use a technique called Hard Mining [14], which involves sampling specific anchor points instead of using the whole prediction space. The sampling ratio was 3 to 1 in favor of the negative class. The model's backbone was pretrained on ImageNet as discussed earlier, but these weights did not remain fixed throughout training and thus were fine-tuned for the grasping task.

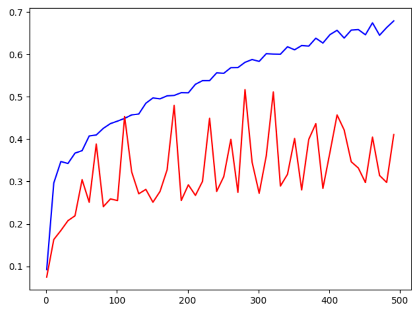
For measuring the performance of the detection, a simple binary metric was used. Either an anchor did or did not contain a grasp. Precision, recall, and F1 Score were calculated based on the network's performance using this metric. Since the network outputs two scores, one for graspability one for non-graspability, whenever the score for graspability was larger it was considered a positive prediction for the network. See Figure 11 for results on the training and the validation set for 500 epochs of training. For results on how the cross entropy and smooth l1 loss change during training see Figure 14. The test results can be found in Figure 10 along with the final results from the training and validation sets. For examples of predictions by the model see Figure 12.

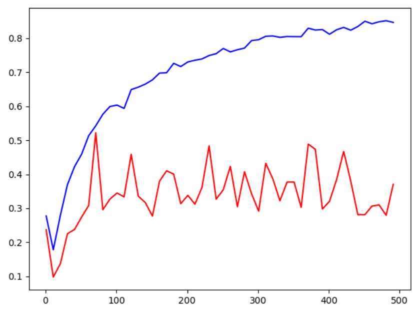
	Train	Val	Test
DL	0.256	0.622	0.622
RL	0.11	0.24	0.24
Precision	0.680	0.41	0.38
Recall	0.847	0.37	0.31
F1	0.753	0.38	0.34
Theta	0.014	0.022	0.017

Figure 10: Summary of Rectangle Grasp Model Performance

6.3 Conclusion

An explanation for why the peformance is poor (besides the overfitting that is clearly occuring after epoch 100) on detection, might be because the performance metric might be too strict. In most


(a) Precision


(b) Recall

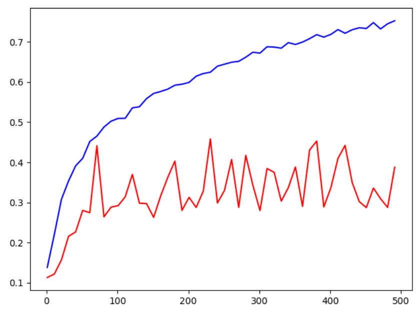

(c) F1 Score

Figure 11: Metrics on Grasp Detection for the Rectangle Grasp Model over 500 Epochs for Train (blue) and Val (red)

object detection settings, and in [33, 30], measuring the intersection over union of the bounding boxes is used. This is because in many cases, the model might predict a box that encapsulates most of the object but is missing a small portion. In this scenario the metric used here would consider that incorrect because it does not overlap completely. It is also worth noting the performance of the regressor on selecting the angles for orienting the rectangle. By observing the angle score (represented as theta in the figure) which is one component of the smooth l1 loss, we can see that on the test set the predicted angle was off by on average 3°.

Because of the configuration of the network, the features for the detection and regression component are shared. This could cause a conflict of interest when the weights are being updated during training, and the features might be rendered suboptimal for one of tasks. It is worth investigating in the future if this is true, and if it is then designing architectures that use separate features for each. The task of grasp classification was not explored here, but might be worth investigating the performance of this type of architecture in a setting similar to [26]. The problem could be formulated as, given an image patch, determine if a grasp exists in the center of the patch or not. The same could be done for the regression task.

6.4 Acknowledgement

I would like to thank Dr. Brian Ziebart for all his help, advice, and computing resources while I worked on this project and for letting me be apart of his lab where I learned many valuable lessons. Also, I would like to thank Dr. Xinhua Zhang for being on my commitee and taking the time to review my work. Finally, the University of Illnois at Chicago for providing me with funding and the resources to work towards my goals.

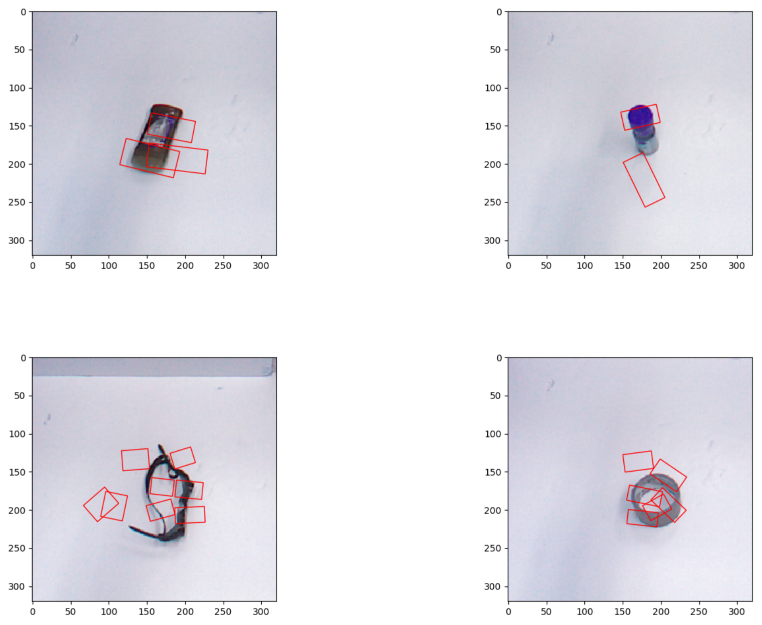
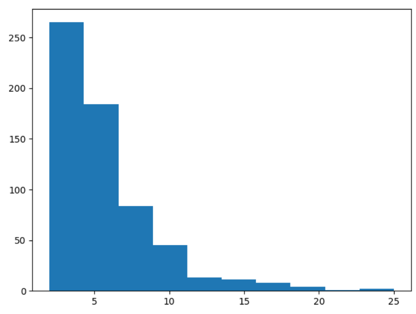


Figure 12: examples of inference done by the model


(a) Train

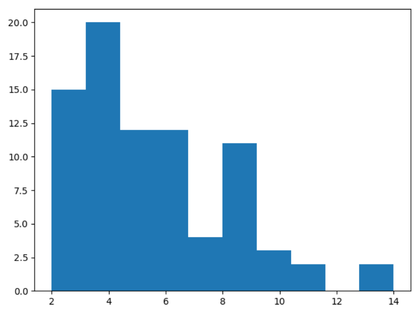

(b) Val

Figure 13: The Distribution of Rectangles per Image on the Training and Validation Sets

References

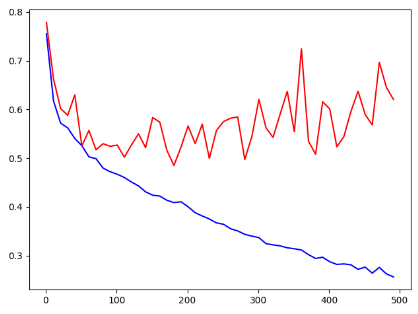
[1] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 348–353. IEEE, 2000.

[2] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2014.

[3] Matei Ciocarlie, Kaijen Hsiao, Edward Gil Jones, Sachin Chitta, Radu Bogdan Rusu, and Ioan A Şucan. Towards reliable grasping and manipulation in household environments. In *Experimental Robotics*, pages 241–252. Springer, 2014.

[4] Jefferson Coelho, Justus Piater, and Roderic Grupen. Developing haptic and visual perceptual categories for reaching and grasping with a humanoid robot. *Robotics and Autonomous Systems*, 37(2-3):195–218, 2001.

[5] Carlo Ferrari and John F Canny. Planning optimal grasps.


(a) Cross Entropy

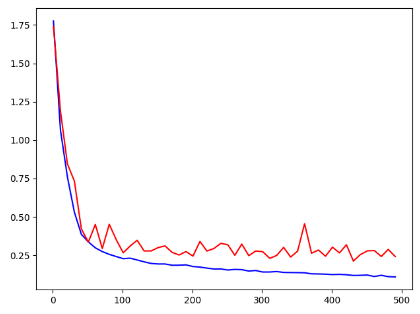

(b) Smooth L1 Loss

Figure 14: Loss Values over 500 Epochs of training. Training Loss (Red) and Validation Loss (Blue)

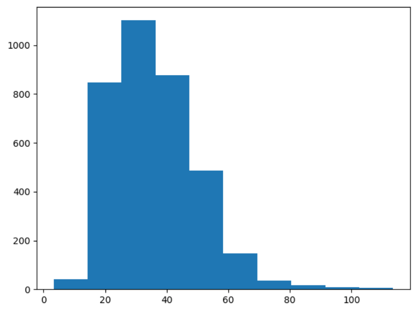


Figure 15: Distribution of bounding box areas

[6] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[7] Kaijen Hsiao, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Grasping pomdps. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4685–4692. IEEE, 2007.

[8] Yun Jiang, Stephen Moseson, and Ashutosh Saxena. Efficient grasping from rgbd images: Learning using a new rectangle representation. In *2011 IEEE International Conference on Robotics and Automation*, pages 3304–3311. IEEE, 2011.

[9] Ishay Kamon, Tamar Flash, and Shimon Edelman. Learning to grasp using visual information. In *ICRA*, pages 2470–2476, 1996.

[10] Quoc V Le, David Kamm, Arda F Kara, and Andrew Y Ng. Learning to grasp objects with multiple contact points. In *2010 IEEE International Conference on Robotics and Automation*, pages 5062–5069. IEEE, 2010.

[11] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015.

[12] 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*, 37(4-5):421–436, 2018.

[13] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[14] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[15] Andrew T Miller and Peter K Allen. Graspit! a versatile simulator for robotic grasping. 2004.

[16] Andrew T Miller, Steffen Knoop, Henrik Iskov Christensen, and Peter K Allen. Automatic grasp planning using shape primitives. 2003.

[17] Van-Duc Nguyen. Constructing force-closure grasps. *The International Journal of Robotics Research*, 7(3):3–16, 1988.

[18] Raphael Pelossof, Andrew Miller, Peter Allen, and Tony Jebara. An svm learning approach to robotic grasping. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3512–3518. IEEE, 2004.

[19] Justus H Piater. Learning visual features to predict hand orientations. 2002.

[20] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[22] Franz Reuleaux. Kinematics of machinery. page 169, 1876.

[23] Anis Sahbani, Sahar El-Khoury, and Philippe Bidaud. An overview of 3d object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60(3):326–336, 2012.

[24] Ashutosh Saxena, Justin Driemeyer, Justin Kearns, and Andrew Y Ng. Robotic grasping of novel objects. In *Advances in neural information processing systems*, pages 1209–1216, 2007.

[25] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[26] Ashutosh Saxena, Lawson LS Wong, and Andrew Y Ng. Learning grasp strategies with partial shape information. In *AAAI*, volume 3, pages 1491–1494, 2008.

[27] Karun B Shimoga. Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3):230–266, 1996.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[30] Hanbo Zhang, Xinwen Zhou, Xuguang Lan, Jin Li, Zhiqiang Tian, and Nanning Zheng. A real-time robotic grasp approach with oriented anchor box. *arXiv preprint arXiv:1809.03873*, 2018.

[31] Li Zhang and Jeffrey C Trinkle. The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing. In *2012 IEEE International Conference on Robotics and Automation*, pages 3805–3812. IEEE, 2012.

[32] Li Emma Zhang, Matei Ciocarlie, and Kaijen Hsiao. Grasp evaluation with graspable feature matching. In *RSS Workshop on Mobile Manipulation: Learning to Manipulate*, 2011.

[33] Xinwen Zhou, Xuguang Lan, Hanbo Zhang, Zhiqiang Tian, Yang Zhang, and Nanning Zheng. Fully convolutional grasp detection network with oriented anchor box. *CoRR*, abs/1803.02209, 2018.