CS230

Deep Learning to Predict Successful Grasp Configurations for a Robotic Manipulator

Aliyah Smith
Department of Aeronautics & Astronautics
Stanford University

Leila Taleghani
Department of Aeronautics & Astronautics
Stanford University

Abstract

Modern robots must be able to do a wide variety of tasks in changing environments and situations. One important task is to be able to pick up, manipulate, and place objects in the robot's environment, which is largely dependent on the robot's ability to grasp items. Robot grasping is a widely-studied research area due to its inherent difficulty and involvement of various facets of robotics, such as perception, control, and planning. Building upon the work of Douglas Morrison, Peter Corke, and Jürgen Leitner at the Australian Centre for Robotic Vision, we sought to train a convolutional neural network (CNN) model to learn successful grasps from a series of images. The dataset, known as Cornell Grasping Dataset, is a widely used dataset for grasp prediction and consists of nearly 900 depth images of various three-dimensional objects on a flat surface. By changing the network structure and tuning the hyperparameters, our model achieved a grasp success rate of 68.5%.

1 Introduction

Robotic grasping is an important and widely-researched subfield within robotic manipulation that involves planning, control, and perception. Generally, robots must be able to grasp and manipulate all objects in its environment, which may vary in size, shape, weight, and material. While there have been many improvements in this field, robots still have a long way to go before they are able to successfully grasp *any* object and more generally, before they are able to mimic the fluidity of human hand grasping. As mentioned previously, there are many facets of robotic grasping. One challenging aspect of robotic grasping can be formulated as a hand-eye coordination problem, where the robot takes in image data of the objects and identifies successful grasps to manipulate those objects. In this project, we chose to focus on this challenge.

Using the Cornell Grasping Dataset, we investigate the problem of identifying successful robotic grasps solely on images of three-dimensional (3D) objects placed on a flat surface. The dataset consists of over 800 images of the different objects [1]. Furthermore, this dataset of images has been used to train a Generative Grasp Convolutional Neural Network (GG-CNN) by Douglas Morrison, Peter Corke, and Jürgen Leitner from the Australian Centre for Robotic Vision. We seek to improve this pretrained model to achieve even better performance for a single parallel jaw gripper.

2 Related work

As stated previously, robotic grasping is a popular area of research; furthermore, many researchers have investigated the problem of identifying grasps from images. For instance, Levine et al. developed a model to predict robot grasp prediction based on images from their Google Brain Robotics Data -

CS230: Deep Learning, Spring 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Grasping Dataset [6]. In their work, they develop a convolutional neural network architecture that can predict grasp configurations for soft and hard objects. Specifically, they tested two experimental scenarios: grasping with replacement (the object is placed back into the bin after grasping) and without replacement (the robot removes the objects from the bin). Mahler et al. used the Dex-Net 4.0 Dataset to train a CNN model to successfully grasp objects in a cluttered work environment [7]. More specifically, the robot was tasked to clear a heap of up to twenty-five 3D objects. This model achieved a success rate of 95% on a physical robot. However, this work is designed for an ambidextrous robot, or a robot with two grippers: a parallel-plate gripper and a suction gripper. When only the suction gripper is used, the robot is able to perform successful grasps 80% of the time. This framework is best used for applications where two grippers are possible.

2.1 Prior Work using the Cornell Grasping Dataset

Lenz et al. proposed a model consisting of two deep neural networks for identifying successful grasps from RGB-D images in the Cornell Grasping Dataset [5]. The first deep network worked quickly to prune out unlikely grasps, while the second had more features, ran more slowly, and evaluated the remaining candidate grasps. The model achieved a 74% success rate, and only considered local features, which makes it more difficult to encode imperative information that may not be obvious from an image, like grasping a knife by its end instead of by the blade [5]. Kumra et Kanan also utilized the Cornell Grasping Data to train a CNN model to predict successful grasps from RGB-D images. The entire model consisted of one deep CNN to extract the key features from the image and a shallow CNN to predict the corresponding grasp for the object in the image [4]. This model achieved a 89% success rate. Park et al. combined robotic grasp detection and object detection using a single deep neural network. This model achieved a 74% success rate on the Cornell Grasping Dataset [9]. Johns et al. approached this problem from a different perspective by formulating a grasp function, which predicts a score for every candidate grasp [2]. They use this function along with a pose uncertainty function to determine grasps when the gripper's pose is uncertain. A CNN was trained using training data generated through a physics simulation [2].

Our work is based off of the model developed by Morrison et al, whose goal was to develop a smaller model that is able to predict accurate grasps without long computational times. Their model directly takes in a depth image to generate a grasp map, which consists of the grasp quality, grasp width, and grasp angle for every pixel in the image [8]. Using this framework, they were able to achieve 88% accuracy on objects that are moved during the grasp attempt, 81% accuracy in cluttered environments, and 83% accuracy on previously unseen objects with adversarial geometry. The motivation of this project was to alter the model architecture of the pretrained GG-CNN model to improve the accuracy rate when trained on the Cornell Grasping Dataset.

3 Dataset and Features

The Cornell Grasping Dataset consists of 885 RGB-D images of three dimensional objects, which contain human labels for positive and negative grasps. In the entire dataset, there are 2909 negative grasps and 5110 positive grasps [8]. Generally, a successful grasp is denoted by a rectangle on the image, which coincides with the rotation and position of a parallel-plate gripper. The mapping from a rectangle to the grasp map (G_θ) involves updating sections of the training image using an image mask based on the center third of the grasping rectangle. For more information on this mapping, we refer to [8].

Due to data preprocessing in the code from the pretrained model, there was no need for additional preprocessing for this dataset. Furthermore, the data was initially split into an 80/20 training and validation set. However, the training/validation split is a tunable hyperparameter. More discussion on hyperparameters can be found in the Experiments section. A few examples from the data are shown in Figure 1 [5].

Figure 1: Example Images from the Cornell Grasping Dataset [5]

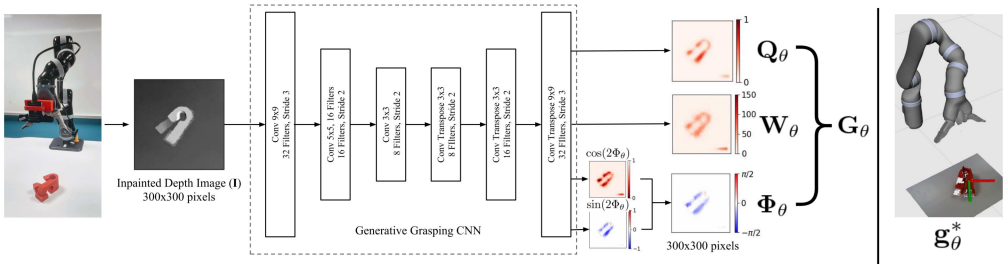

Figure 2: Network Architecture for the Pretrained GG-CNN model. The input to the convolutional neural network is the depth image, which passes through three 2D convolutional layers and three 2D transpose convolutional layers. The output of the model is the grasp quality (Q_θ), grasp width (W_θ), and grasp angle (Φ_θ), which are then used to determine the best grasp. [8]

4 Methods

4.1 Model

The model used in this project was developed by Morrison et al. While it is possible to train the model from the beginning, we chose to begin with a pretrained model, which consists of the network weights after 23 epochs of training, to save time due to training. The architecture of the pretrained model by Morrison et al. can be seen in Figure 2. This model consists of three two-dimensional convolutional layers, and three two-dimensional convolutional transpose layers. There are four output layers, one for the grasp quality (Q_θ), one for the grasp width (W_θ), and two which together determine the grasp angle (Φ_θ). The best grasp (g_θ^*) is then determined from the grasp image based on these three outputs.

As defined by Morrison et al., the grasp quality for a pixel is a binary value, which is set 1 for a positive grasp, and 0 otherwise. The grasp angle is a value in the range $[-\frac{pi}{2}, \frac{pi}{2}]$, which is later normalized to lie in the range $[-1, 1]$. Finally, the grasp width is the width of gripper, and has a maximum value of 150 pixels. This value is later normalized to lie in the range $[0, 1]$.

4.2 Loss Function

The loss function for each output is the mean squared error between the predicted output and the true value. For instance, the mean squared error for the grasp width is defined in the following way.

$$\mathcal{L}_{width} = \frac{1}{n}\sum_{i=1}^{n}(Width_{pred} - Width_{true})^2$$

The loss function can be defined similarly for the three other outputs. The total loss is the sum of the individual losses.

$$\mathcal{L}_{total} = \mathcal{L}_{pos} + \mathcal{L}_{width} + \mathcal{L}_{sin} + \mathcal{L}_{cos}$$

This was the original loss function defined in the pretrained model. We chose to keep this loss function because it is a commonly used function for object detection problems.

5 Experiments & Results

5.1 Hyperparameter Tuning

The main emphasis in this project was on the hyperparameter turning. We chose to focus on hyperparameter turning because (1) the code from the pretrained model had been easily set up to change these hyperparameters and (2) we initially wanted to improve performance while keeping architectural changes to a minimum. As previously mentioned, the pretrained model is small in comparison to other prior works, which helps reduce the computational cost.

The following hyperparameters were investigated: number of training epochs, batch size, batches per epoch, and the split ratio between the training and validation sets. While learning rate is often the first hyperparameter tuned when optimizing a model, the learning rate was not one of the tunable hyperparameters in the pretrained model. It is important to note that the initial inspection and analysis of the code from the pretrained model did not include a learning rate. More discussion on learning rate can be found in the Future Work section.

It also important to note that one epoch of training could take upwards of 1 to 2 hours. We chose to train each of these models for 1-3 epochs due to this computational cost. However, 1-3 epochs were enough to get a general sense of the trend of the loss.

The results from the initial hyperparameter tuning are summarized in Table 1 below.

Table 1. Results from Initial Hyperparameter Tuning.

Experiment	Epochs	Batch Size	Batches per Epoch	Split	Accuracy (%)
1	5	16	863	0.9/0.1	62.92
2	1	2	1596	0.875/0.125	0.00
3	5	13	1471	0.731/0.269	66.29
4	3	3	540	0.851/0.149	1.12

The conclusions from these initial experiments align with our intuition that accuracy increases when the model is trained for longer periods of time and when the training set is a higher majority of the data. To tune the hyperparameters even further, we implemented a cross entropy optimization to find the optimal parameters. A description of the cross entropy optimization process can be found in the Appendix. The range for the hyperparameters were 1-6, 500-2000, and 1-20, for the epochs, batches per epoch, and batch size, respectively. The results from this optimization are as follows:

- Epochs: 5
- Batches per Epoch: 1217
- Batch Size: 14
- Training/Validation Split: 81.5 / 18.5

This combination of hyperparameters was then used to train the model.

5.2 Architectural Changes

Training with the optimized hyperparameters led to overfitting. To prevent overfitting of the model, two dropout layers were added to the model architecture - one after the last Conv2D layer and one before the lat ConvTranspose2D layer. The final model architecture can be found in the Appendix. Table 2 shows the differences between the grasp prediction accuracy before and after the dropout layers were added. The dropout layers resulted in slightly better results, as the test accuracy was reduced and the validation accuracy was increased.

With the addition of the dropout layers, the model was able to achieve 68.53% accuracy. An example of a successful grasp is shown in Figure 3, which displays the RGB image, the depth image, the grasp image (G), and the grasp angle.

Table 2. Grasp prediction accuracy (GPA) during and after training without and with dropout. Optimized hyperparameters: 5 batches, 4217 batches per epoch, batch size of 14, and split of 81.5%/18.5%.

Time of Validation	GPA (w/o Dropout)	GPA (w/ Dropout)
Epoch 1	0.80%	0.00%
Epoch 2	53.81%	30.12%
Epoch 3	61.04%	45.38%
Epoch 4	62.24%	51.40%
Epoch 5	69.47%	64.25%
Post-Training Validation	66.29%	68.53%

6 Conclusion & Future Work

Our model was not able to achieve comparable results to prior works. While many prior works were able to achieve a successful grasp rate greater than 70%, our model falls short with a maximum accuracy measure of 68.5%. A major setback in this work was due to accessibility and compatibility issues with online grasping datasets towards the beginning of this project, which resulted in delays in implementation.

If more time was allotted for this project, performance of the model could be improved with longer training time, additional hyperparameter tuning using a more extensive grid search, and perhaps a bigger model. However, it is important to note that a bigger model would result in a higher computational cost. Additionally, the introduction and adjustment of a learning rate could have a positive influence on the model performance. Thus, we would also work to implement this adjustment in the model with additional time.

7 Contributions

Both Leila and Aliyah worked on the dataset acquisition and analysis. All decisions i.e. changes to the model structures and decision of the hyperparameter tuning approach were made together. Due to computational limitations, Leila took lead on implementing the hyperparameter optimization and changes to the network architecture. Aliyah took lead on the reports and presentation.

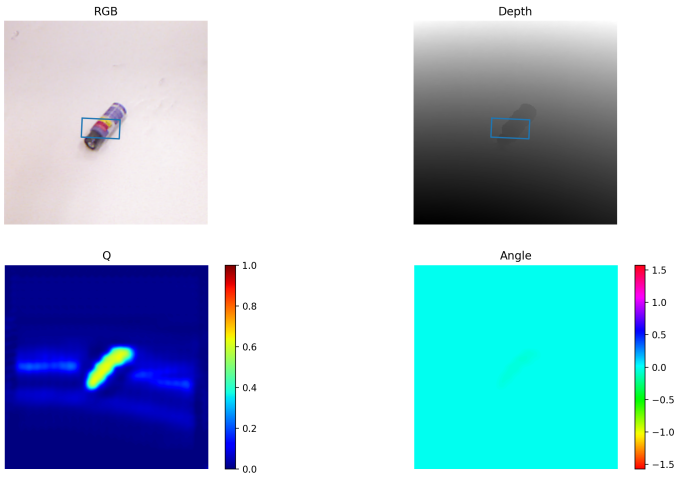

Figure 3: Qualitative Results of a Successful Grasp.

References

[1] *Cornell Grasp Dataset¶*. URL: https://www.kaggle.com/oneoneliu/cornell-grasp.

[2] E. Johns, S. Leutenegger, and A. J. Davison. "Deep learning a grasp function for grasping under gripper pose uncertainty". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 4461–4468. DOI: 10.1109/IROS.2016.7759657.

[3] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization*. The MIT Press, 2019. ISBN: 0262039427.

[4] S. Kumra and C. Kanan. "Robotic grasp detection using deep convolutional neural networks". In: (2017), pp. 769–776. DOI: 10.1109/IROS.2017.8202237.

[5] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps". In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 705–724. DOI: 10.1177/0278364914549607. eprint: https://doi.org/10.1177/0278364914549607. URL: https://doi.org/10.1177/0278364914549607.

[6] Sergey Levine et al. "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection". In: *CoRR* abs/1603.02199 (2016). arXiv: 1603.02199. URL: http://arxiv.org/abs/1603.02199.

[7] Jeffrey Mahler et al. "Learning ambidextrous robot grasping policies". In: *Science Robotics* 4.26 (2019). DOI: 10.1126/scirobotics.aau4984. eprint: https://robotics.sciencemag.org/content/4/26/eaau4984.full.pdf. URL: https://robotics.sciencemag.org/content/4/26/eaau4984.

[8] Douglas Morrison, Peter Corke, and Jürgen Leitner. "Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach". In: *Proc. of Robotics: Science and Systems (RSS)*. 2018.

[9] D. Park et al. "A Single Multi-Task Deep Neural Network with Post-Processing for Object Detection with Reasoning and Robotic Grasp Detection". In: (2020), pp. 7300–7306. DOI: 10.1109/ICRA40945.2020.9197179.

8 Appendix

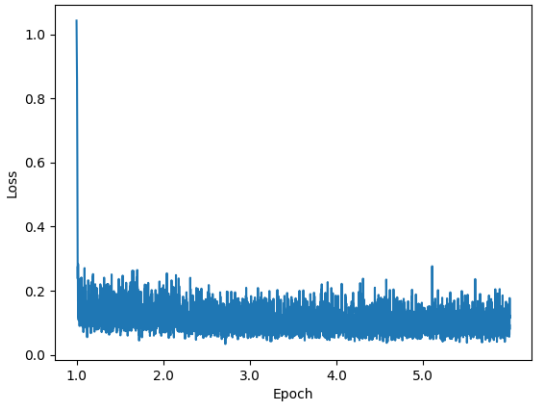

Figure 4: The Loss Curve for the experiment with added dropout layers, which resulted in a 68.53% grasp accuracy rate.

8.1 Cross Entropy Method

The cross-entropy method specifies a number of elite samples and fits a Gaussian Mixture Model to the elite samples. The hyperparameters are then re-sampled from the elite distributions and the algorithm iterates for a set amount of iterations.

For the Gaussian Mixture Model fitting, the elite samples are chosen according to a multivariate maximum likelihood estimate, as described in Figure 5.

We can use `Distributions.jl` to represent, sample from, and fit proposal distributions. The parameter vector θ is replaced by a distribution `P`. Calling `rand(P,m)` will produce an $n \times m$ matrix corresponding to m samples of n-dimensional samples from `P`, and calling `fit` will fit a new distribution of the given input type.

```
import Random: seed!
import LinearAlgebra: norm
seed!(0) # set random seed for reproducible results
f = x->norm(x)
μ = [0.5, 1.5]
Σ = [1.0 0.2; 0.2 2.0]
P = MvNormal(μ, Σ)
k_max = 10
P = cross_entropy_method(f, P, k_max)
@show P.μ

P.μ = [-6.13623e-7, -1.37216e-6]
```

Example 8.3. An example of using the cross-entropy method.

Figure 5: An example implementation of the cross-entropy method as described in Mykel Kochenderfer's *Algorithms for Optimization*[3]

$$\boldsymbol{\mu}^{(k+1)} = \frac{1}{m_{\text{elite}}} \sum_{i=1}^{m_{\text{elite}}} \mathbf{x}^{(i)} \tag{8.10}$$

$$\boldsymbol{\Sigma}^{(k+1)} = \frac{1}{m_{\text{elite}}} \sum_{i=1}^{m_{\text{elite}}} (\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k+1)})(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k+1)})^\top \tag{8.11}$$

Figure 6: Maximum likelihood estimate update equations for a multivariate normal distribution[3]

Model Architecture
1. Conv2D 9x9 (32 filters, Stride 3)
2. Conv2D 5x5 (16 filters, Stride 2)
3. Conv2D 3x3 (8 filters, Stride 2)
4. Dropout(0.25)
5. ConvTranspose2D 3x3 (8 filters, Stride 2)
6. ConvTranspose2D 3x3 (16 filters, Stride 2)
7. Dropout(0.25)
8. ConvTranspose2D 9x9 (32 filters, Stride 3)

Figure 7: Final Model Architecture with Dropout Layers.