Learning to Grasp Objects in Highly Cluttered Environment

Cognitive Robotics - Practical 2

1st Mo Assaf
University of Groningen
Artificial Intelligence, MSc.
Groningen, Netherlands
s4051378

2nd Daniel Skala
University of Groningen
Artificial Intelligence, MSc.
City, Groningen, Netherlands
s3953602

CONTENTS

***Abstract*—In this paper, we explain how we implemented GGCNN in the PyBullet simulator, compared it to GR_ConvNet model, and how we conducted the experiments for the object grasping assignment. Furthermore, we present the results of the experiments and the lessons learned during the assignment. While both models are competetive regarding their speed, GR_ConvNet significantly outperforms GGCNN in all three scenarios (pile, packed, and isolated).**

***Index Terms*—GR_ConvNet, GGCNN, grasping, cluttered environment, CNN**

I. INTRODUCTION

In the first assignment, we learned about various hand-crafted and deep-learning approaches for 3D object recognition. In this assignment, we will study how to use perception to assist the robot in manipulating objects in a highly cluttered environment. Toward this goal, we will learn about two deep-learning approaches for object grasping. The first approach is based on the approach proposed by (Kumra, Joshi, & Sahin, 2019) (GR-ConvNet) and the second one is based on the approach proposed by (Morrison, Corke, & Leitner, 2018) (GGCNN). We will experiment with these methods to grasp objects in three scenarios: isolated, packed, and pile scenarios.

Object manipulation in a highly cluttered environment is a challenging problem. The robot has to be able to detect the objects in the environment, segment them from the background, estimate their poses and then perform a suitable grasping motion. Possible applications for object manipulation tasks are, for example, domestic tasks such as putting objects in a bin or sorting objects from a pile, or industrial tasks such as packing and unpacking boxes. The challenge arises from the fact that the environment is highly cluttered. This means that the objects are close to each other and the robot has to be able to detect and segment them from the background. Furthermore, the objects might be occluded by other objects, which makes it even more difficult to estimate their poses. Therefore, the robot needs to be able to reason about the environment and the objects in it in order to perform the manipulation task successfully.

In this assignment, we will use the PyBullet robotics simulator to simulate the object manipulation task. The simulator allows us to reset the environment to different configurations, which is useful for testing the methods. We will use the UR5e robot arm with a two-fingered gripper to perform the manipulation tasks. The robot is equipped with an RGB-D camera used for perception. The goal of the assignment is to experiment with the two deep-learning methods for object grasping and compare their performance in different scenarios. We will also investigate the usefulness of using deep-learning methods for object grasping in general-purpose tasks.

A. Structure of the report

In the Related work section, we explain the inner working and implementation of both GR_ConvNet and the GGCNN networks. In the Methodology section, we describe how we

implemented GGCNN in the provided code and how the experiments were conducted. In the Results section, we present six figures that compare the performance of the two networks. Finally, we conclude the paper in the Conclusions section.

II. Related work

In this work, we will be comparing two convolutional neural networks; the state-of-the-art GR_ConvNet and the GGCNN network. We chose GGCNN mainly due to its speed despite knowing that GGCNN performs much worse than GR_ConvNet.

A. GR-ConvNet

The GR-ConvNet is a (Kumra et al., 2019) is a generative deep neural network architecture that is designed and trained to predict the grasp point (i.e., angle, quality, and width) given the depth and colour map output from an RGB-D camera. The camera data is then normalised, resized, cropped. The model incorporates an arbitrary choice for n channels. This gives removes input constraints and allows multi-modal inputs. The architecture begins with convolution blocks that is extracts all the relevant features follow by a batch normalisation. While it is true that more convolutions can improve the accuracy, it can also be prone to vanishing gradients and can cause the model to become over-fitted. To mitigate this issue, residual blocks have skip connections from the beginning to the end allowing the model to learn the identity model if necessary. Lastly, the data is encoded into 128 56×56 filters which are interpreted by carrying out transpose convolutions which upscale the shape back to the original image size. The model finally outputs the quality, angles ($cos(2\theta)$, $sin(2\theta)$), and the width.

The model was trained to maximise the log likelihood applied to the Cornell grasp dataset (Morrison et al., 2018). To evaluate the results, the intersection over union (IoU) as a metric. The model was able to achieve an accuracy 94.6% accuracy on the Cornell grasp dataset and has an inference time of $20ms$. The model was also tested on a robot on 10 different positions and orientations and the robot was able to perform 334 successful grasps out of 350 trials.

B. GGCNN

The Generative Grasping Convolutional Neural Network (GG-CNN) is a small neural network that does not rely on sampling of grasp candidates, but directly generates grasps on a pixel-wise basis (Morrison et al., 2018). Thanks to its small scale, GGCNN has a state-of-the-art speed performance and it can generate grasps in as little as 19 milliseconds. This makes it a great candidate for comparison with GR_ConvNet (due to its comparable speed performance).

GGCNN follows a similar concept of taking an impainted depth image (as opposed to multi-channel image in GR_ConvNet) and mapping it to the required angle, quality, and width parameters representing different grasp points. Therefore, the model learns a pixel-to-pixel mapping. The model takes in an inpainted normalised depth image and outputs the required grasping parameters. The generative architecture includes an information bottleneck by applying convolutions that reducing the dimensions and the degrees of freedom. Similar to GR-ConvNet, the encoded information is then interpreted and used to generate the output by applying transposed convolutions. The model is lightweight and contains 62,420 parameters.

Despite its small size, the model is able to generalize well and achieve accuracy of 87% even in cluttered environments on which the network was not train. For isolated objects, the experimenters performed 4 grasps on 10 test objects and achieved accuracy of 100%. Lastly, an accuracy of 81% was achieved in dynamic cluttered environment where the objects are in motion.

III. Methodology

In this work, we use a simulated environment to test the performance of two models. The simulation contains a table with objects placed in the middle, and a robot arm with a sensor. The robot must successfuly grasp the objects on the table and carry them to the bin to clear the table. The robot arm's trajectory is configured by providing the grasp points map using the quality, angle, and width output of a given model. The simulated robot has a simulated sensor that reads the depth image of the objects on the table which generates RGB color and depth data. This data is fed into a model and the robot plans the trajectory based on the output of the model. We evaluate the performance of the model based on the success rate of grasping, manipulation, and objects landing in the bin. An object placed in the bin implies successful grasping and manipulation.

A. Code implementation

The simulated environment was built using the python library PyBullet. To implement the GGCNN network we had to rewrite the predict function in grasp_generator.py and along with some modifications in simulator.py.

In simulator.py we added another condition $network\_model == "GGCNN"$ in which case we set the image size to 224, and we set the network path to the path of our GGCNN model. The model was located in trained_models in "ggcnn_weights_cornell/ggcnn_epoch_23_cornell". We also append the path to sys.path.

To successfully implement GGCNN we first had to understand how the network works and what is its domain. In contrast to GR_ConvNet, GGCNN only accepts the depth image and not the four-channel RGBD image. To do this, we had to create a DepthImage and discard the negative values by calling the inpaint() function. Next, we normalize the image by calling the normalize() function. We then convert the image

to Tensor and call .to(self.device) function. The full code is depicted in III-A.

```
dimg = DepthImage(depth)
dimg.inpaint()
dimg.normalise()
depth_img = dimg.img[np.newaxis, :, :]
depth_img = torch.Tensor(depth_img)
depth_img.to(self.device)
```

Once the GGCNN network is prepared for the inference (*torch.no_grad()*) we perform the forward pass, find *pixels_max_grasp* just like in GR_ConvNet and finally post-process the network's output. The output is stored as grasp quality, angle and image width. For implementation see III-A

```
p, c, s, w = self.net.forward(depth_img)
pos_output = p
cos_output = c
sin_output = s
width_output = w
conv = self.PIX_CONVERSION
pix_max_grasp = int(self.MAX_GRASP * conv)
qual, ang, width = self.post_process_output(
    pos_output,
    cos_output,
    sin_output,
    width_output,
    pix_max_grasp
)
```

After both networks are ready to be run, we performed a number of experiments to test the performance of the networks. Together, we conducted 6 experiments; each network's performance in the pile, packed, and isolated scenarios.

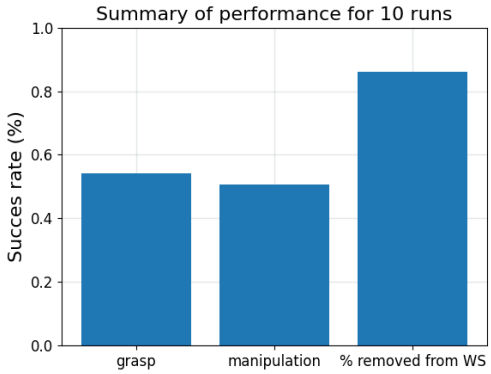
IV. Results

A. Discussion of results

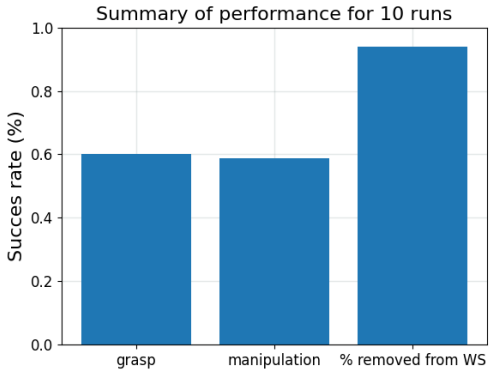
1) Pile: The Pile scenario was the most difficult for both models. We can observe the values in 1a and 2a - while GR_ConvNet led to 52% of successful grasps, 50% in manipulation, and 87% of objects removed from the workspace, GGCNN scored only 25% in grasping, 24% in manipulation, and 58% in objects removed from the workspace.

GGCNN performs significantly worse in the Pile scenario.

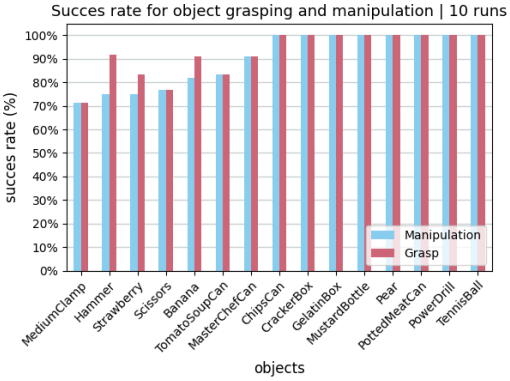
2) Packed: The Packed scenario led to an increase of the overall scores for both networks. GR_ConvNet led to 60% of successful grasps, 59% in manipulation, and 93% of objects removed from the workspace, GGCNN scored 31% in grasping, 31% in manipulation, and 73% in objects removed from the workspace. While the increase in grasps and manipulation is not significant for either network, the % of objects removed from the workspace is significantly higher for both networks.



(a) Pile scenario (GR_ConvNet)



(b) Packed scenario (GR_ConvNet)

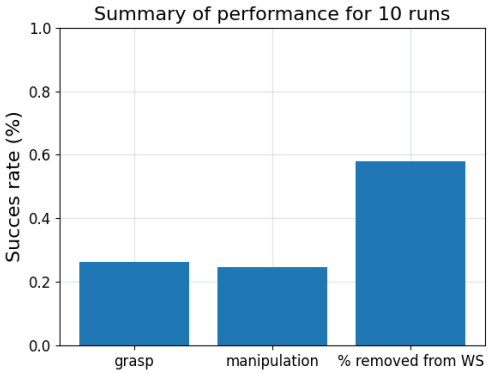


(c) Isolated scenario (GR_ConvNet)

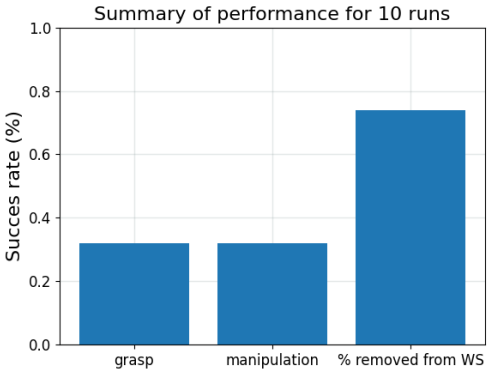
Fig. 1: Pile, packed, and isolated scenarios for GR_ConvNet

3) Isolated: GR_ConvNet in the Isolated scenario managed to grasp half of the objects with a success rate of 100% while the success rate for the other half is ranging from 70% to 90%. The most difficult object to grasp for GR_ConvNet was MediumClamp.

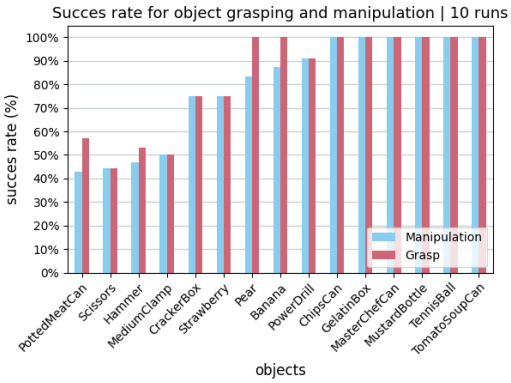
As expected, GGCNN performs worse than GR_ConvNet. The success rate of 100% applies to six out of fifteen objects. Other objects have a success rate ranging from 40% to 90%.



(a) Pile scenario (GGCNN)



(b) Packed scenario (GGCNN)



(c) Isolated scenario (GGCNN)

Fig. 2: Pile, packed, and isolated scenarios for GGCNN

The most difficult object for GGCNN is PottedMeatCan.

V. Conclusions

In this work we compared the performance of an existing deep learning model (GGCNN) and the default one (GR_ConvNet) in three scenarios. The results show that GR_ConvNet outperforms GGCNN in grasp classification. The performance of both models was evaluated and compared in three scenarios: Isolated, Pile, and Packed. The results show that GR_ConvNet outperforms GGCNN in all three scenarios. The Pile scenario is generally more difficult than the Packed scenario.

Potential future work is to make a large-scale comparison between other CNNs and see how they perform in the three scenarios regrading both speed and success rates.

VI. Contributions

The work was split equally between both authors. We both spent an equal amount of time writing the report and running the experiments. No issues of conflicts were encountered during the process.

References

Kumra, S., Joshi, S., & Sahin, F. (2019). *Antipodal robotic grasping using generative residual convolutional neural network.* arXiv. Retrieved from `https://arxiv.org/abs/1909.04810` doi: 10.48550/ARXIV.1909.04810

Morrison, D., Corke, P., & Leitner, J. (2018). *Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach.* arXiv. Retrieved from `https://arxiv.org/abs/1804.05172` doi: 10.48550/ARXIV.1804.05172