Point Cloud Grasp Classification for Robot Grasping

Navjot Singh
Stanford University
navjot@stanford.edu

Zachary Blum
Stanford University
zblum25@stanford.edu

Neethu Renjith
Stanford University
neethur@stanford.edu

Abstract

This paper aims to improve the grasping capabilities of robotic arms by using a neural network that classifies a potential grasp of an arbitrary object as a high quality or low quality grasp, given a direct point cloud of the grasp. Inspired by the recent successes of new architectures such as PointNet++ [2], KD-Networks [5], and Dynamic Graph Convolutional Neural Networks [8] that are able to classify objects directly through their point clouds, this project aims to leverage the PointNetGPD [10] pipeline and compare the performance of these new architectures, and modified versions of the architectures, to the task of classifying point clouds of robotic grasps.

1 Introduction

The task of robotic grasping on novel and varied general objects is a very prominent field of research and a challenging engineering problem [4]. Recently, research has been focused on using point clouds, 3D representations of the points on an object (which is very closely associated with how the objects are represented by the sensors themselves), for certain computer vision tasks. Advances in point cloud based neural networks [3, 2, 5, 8] have allowed neural networks to directly take in the 3D information provided by point clouds to more accurately and robustly understand the physical environment provided to them. This paper is looking to leverage these recent advances to better solve the problem of finding good grasp poses for any arbitrary object. To directly use the information provided by point clouds, PointNet [3] was developed as an end-to-end neural network solution that directly consumed point clouds and output classification labels. Current state of the art technologies for classifying the quality of different potential grasps use 3D CNNs which work on RGBD images. Others have implemented Grasp Pose Detection (GPD), such as [6], using a reduced version of point clouds to sample different potential grasps on the point cloud of the object.

The first major paper to integrate a more advanced point cloud architecture such as PointNet directly into the Grasp Pose Detection algorithms, was a paper titled "PointNetGPD: Detecting Grasp Configurations from Point Sets" [10]. PointNetGPDs have a higher rate of success while being lighter and more robust to noise than compared to the CNN-based GPD models, or the early point cloud-based GPD models.

Even more recently, and in the general field of computer vision (not specifically in the robot grasping field), new point cloud-based architectures have been developed that perform better than PointNet for object classification on the standard ModelNet40 dataset. This paper aims to understand if these new architectures (and modified versions of the architectures), including PointNet++ [2], KD-Networks [5], and Dynamic Graph Convolutional Neural Networks (DGCNNs)[8], can similarly be applied to the problem of grasp pose detection and outperform PointNetGPD.

2 Baseline : PointNetGPD

PointNetGPD [10] provides a pipeline to generate and evaluate grasp congurations. Several feasible grasp configurations are generated by sampling using meshes. The grasp candidates represented by the point cloud within the closing area of the gripper is passed into the PointNet architecture. PointNet predicts the grasp quality of the candidate which acts as the evaluation metric and classifies them as good or bad candidates. As it uses point clouds directly, it could mostly sustain the geometric information of the original point cloud and infer grasp quality more efciently than methods that first convert point clouds to meshes or voxels. The method does not employ any hand-crafted features and so generalizes well on new objects. This architecture acts as the baseline for our work. As a simple attempt to improve the model we incorporated three more fully connected layers at the end

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

before the softmax layer. The original model was designed with the ModelNet-40 dataset in mind with a 40 class classification. We hypothesized that with just 2 classes to output, having more layers that more gradually reduced the width down to 2 would perform better. With this in mind, we added a 128, 64, 32 width layers at the end of PointNet to call it PointNetDeeper.

3 Extensions and variations on PointNet

3.1 PointNet++

In PointNet++ [2], the point cloud is rst partitioned into overlapping sets by the distance metric of the underlying space. Local features extracted from small neighborhoods using pointNets are further grouped into larger units and processed to produce higher level features. This process is repeated to obtain the features of the whole point set, i.e, the architecture uses pointNet recursively on smaller regions. While PointNet uses a single max pooling operation to aggregate the whole point set, pointNet++ builds a hierarchical grouping of points and progressively abstract larger and larger local regions along the hierarchy. PointNet++ performed better than PointNet on the ModelNet-40 dataset, so we hypothesized this similar improvement in learning can be transferred over to our grasp dataset.

3.2 KD-Networks

The KD-networks deep network [5] uses kd-tree structure constructed for 3D point clouds to form the computational graph, and to compute a sequence of hierarchical representations in a feed-forward bottom-up fashion. A kd-tree is constructed recursively in a top-down fashion by picking the coordinate axis with the largest range of point coordinates, and splitting the set of points into two equally-sized subsets and then recursively going through each of them. Given an input kd-tree T,Kd-network computes a vector representation v_i associated with each node of the tree, using :

$$v_i = g\left(W_{d_i}^{l_i}\left[v_{c1(i)}; v_{c2(i)}\right] + b_{d_i}^{l_i}\right) \tag{1}$$

where g is a non-linear function; $W \;\; b$ are the weights; d_i is the coordinate axis of splitting and subscript c represents the children of the node i . Just as CNNs share parameters for localized multiplications (convolution kernels) across different spatial locations, Kd-net also shares the multiplicative parameters W_i^j, b_i^j across all nodes at the tree level j. The network structure makes it invariant to rotations and partially invariant to spatial jitter. These properties make it ideal for handling point cloud information. Kd-Net of depth 10 has been proven to outperform CNNs and sometimes performs even better than PointNets, using benchmark data .This architecture has not been used for grasp classification yet.

3.3 DGCNN

Much research in the last few years has gone into using graphs to extrapolate convolutional neural networks (CNNs) to point cloud data. Two examples of this are regularized graph CNNs (RGCNNs) [11] and dynamic graph CNNs (DGCNNs) [8]. While both RGCNN and DGCNN use dynamically-updated graph convolutions, the specific edge convolution used in the DGCNN paper allows it to achieve a higher classification accuracy than RGCNN (at the expense of a larger forward time). Both RGCNN and DGCNN have a shorter forward time than PointNet and PointNet++, but require a larger model size [11]. An additional advantage that DGCNN has over PointNet and PointNet++ is that it examines local geometric neighborhoods of each point, which allows it to utilize local features when performing classification, without sacrificing robustness to permutations of the points [8]. The dynamic nature of DGCNNs allows the neighborhood graph to be updated each layer, rather than having one fixed graph for the entire network. An example graph convolution is shown below in Figure 1 from [8].

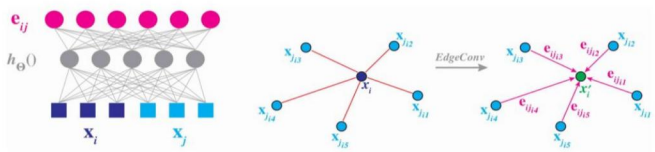

Figure 1: Given two points xi and xj, the edge eij is calculated using a function h which incorporates both local and global geometric features. With the connected graph (shown in the middle), the edge convolution (EdgeConv) operation is performed in which a filter is convolved over each of the edges.

4 Dataset and Features

Our project used the YCB object dataset [1] which includes 3D scans and RGBD pictures of 59 everyday objects. Many pre-processing steps were taken to convert this data into the grasp data needed for our experiments (the conversion is based on that done in [10]). First, we created a point cloud file for each object from the RGBD scans, and created a mesh file from the 3D scans. An example object representation after this conversion, is shown below in Figure 2

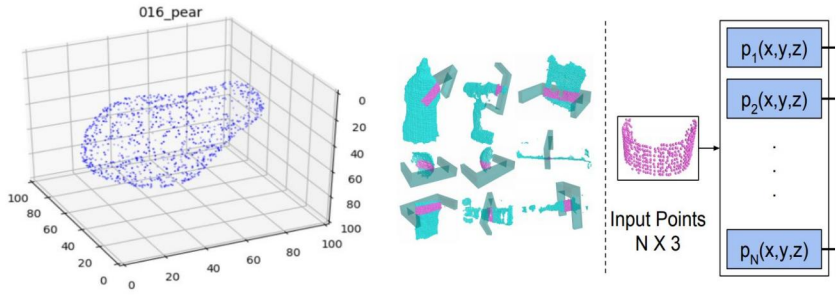

Figure 2: (a) Example point cloud object representation, (b) Point clouds of objects shown in blue with the projection of the gripper shown in pink. (c) The point cloud of the projected gripper for one object. (d) The input to the neural networks is the x, y, and z values for each of the points in the projected gripper point cloud.

Given each object's mesh file, random antipodal grasps are sampled, and a numerical score (for quality of the robot grasp) is calculated using the equation below. In the equation, the quality depends on the state of the object (which is a combination of its pose and its mass, friction, and geometry properties) and the grasp g (which represents the robot gripper's orientation and pose). The two metrics used are force-closure (a binary variable based on the coefficient of friction that determines force stability) and Grasp Wrench Space (which relates to the radius of the grasp) [9]:

$$Q(s,g) = \alpha * Q_{fc}(s,g) + \beta * Q_{gws}(s,g)[10] \tag{2}$$

Given the projection area of the robot gripper, each grasp is projected onto the object point cloud to create the grasp point cloud, which is the input to the neural network (see Figure 2 below, from [10]).

Using the grasp quality equation above, the ground truth for grasps with scores above 0.6 were labeled a class of 1 (high quality grasp) and the ground truth for grasps below 0.6 were labeled a class of 0 (low quality grasp).

In order to augment the data, each grasp point cloud was randomly rotated in addition to applying Gaussian noise. Given the 59 objects and all of the grasps (and augmented grasps created), we had a total of 58,560 distinct gripper point clouds. We split the data into 48,960 point clouds in the training set, 4800 point clouds in the validation set, and 4800 point clouds in the test set.

5 Methods and experiments :

For this paper, in an attempt to improve the baseline pointNetGPD architecture, we modified the architectures of PointNet, PointNet++, KD-Networks, and DGCNN and trained and tested them on the above grasp data. The code was implemented in PyTorch [7]. For our binary classification task of low quality grasp (0) and high quality grasp (1), we used un-weighted negative log likelihood loss. All the models were trained through mini-batch gradient descent with Adam's optimizer and a learning rate that decayed by half every 30 steps. We used mini-batches of size 16 to speed up learning. Accuracy is chosen as the primary evaluation metric to compare the different variations of architecture we have implemented. While we understand that a confusion matrix is useful to identify misclassifications in a classification problem, for point cloud projections of grasps, a confusion matrix would not provide much useful information. Additionally, since the number of training and test examples in each class are equal, we found that accuracy was a better metric than other evaluation metrics such as precision and recall.

5.1 PointNet Deeper

Figure 3: (a) Training accuracy with epoch (b) Test accuracy with epoch

The PointNet Deeper architecture was able to converge to almost the same level of accuracy as the baseline model, and the training time to the original model was also similar. Batch size and learning rate were the only hyperparamters tuned against the validation set.

5.2 PointNet++

Figure 4: (a) Training accuracy with epoch (b) Test accuracy with epoch for single scale PointNet++, r=0.2, 0.4

The PointNet++ architecture was tuned with the number of scales for the radius as well as the size of each radius. The multiscale model was 6 times larger and thus trained much slower. Overall, the best performer among the pointNet++ architecture was the simpler single scale model with radii of 0.2 and 0.4 for the first and second layers, respectively.

5.3 KD-Networks

Randomized kd-tree networks were used as they were reported to be more accurate. A quick implementation of the kd-net was carried out and ran for a fraction of the total data available. 5 shows that this led to overfitting to the training data set; the training accuracy went up while the test accuracy went down.

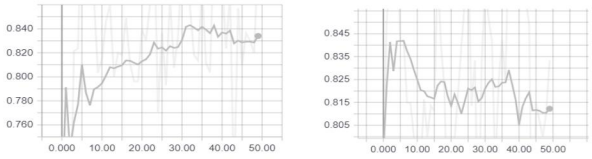

Figure 5: (a) Training accuracy with epoch (b) Test accuracy with epoch. Depth =11

This was remedied by training the network on the entire data set. The accuracy is seen to go up with each passing epoch though not by much as seen in 6. We get an accuracy of 82.13% for depth of 11. The kd-net was also trained with higher depth of 15. The number of non-leaf nodes in the kd-tree increases exponentially with the depth, so higher depths were not attempted.The higher depth resulted in an accuracy of 84.3% . From 6 we can see that the deeper kd- net though initially had a higher accuracy tends to overfit to the training data, so using a deeper model would not be useful.

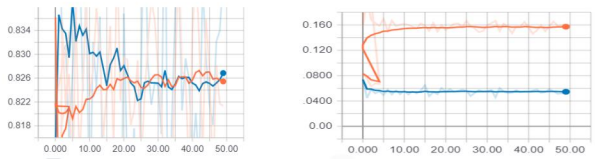

Figure 6: (a) Test accuracy with epoch (b) Test loss with epoch. (Orange : depth =11, Blue :depth 15)

5.4 DGCNN

In order to integrate the DGCNN architecture into the end-to-end deep learning model for grasp classification, one main architecture change was initially made. While the original DGCNN architecture from [8] output a Bx2 matrix of class scores (where each row represented one input point cloud from the batch of size B, and each column represented the probability of belonging in class 0 or 1, respectively), in order to use the negative log likelihood loss, we added a LogSoftmax layer (which output a Bx2 matrix of log probabilities of belonging to class 0 and 1). The architecture is shown below in Figure 7. We kept the same edge convolution and multilayer perceptron layers (along with the associated batch normalization in order to speed up learning and have slight regularization).

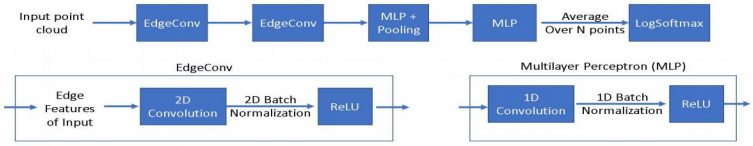

Figure 7: Architecture used for the DGCNN, with the definitions of the EdgeConv and MLP layers

For this first modification of DGCNN, we used an initial learning rate of 0.001 (as in [8]). This achieved a training accuracy of 0.8506 and a test accuracy of 0.8542 (and a test loss of 0.2977). In order to attempt to improve performance, we first changed the initial learning rate to 0.01. This achieved a training accuracy of 0.8223 and a test accuracy of 0.8242 (and a test loss of 0.393). Because the training and test accuracy for these was very similar to each other, but below the performance of the baseline PointNetGPD performance, this suggested that the issue was in the bias of the model, rather than the variance. In order to fix a model bias issue, one solution is to adjust the neural network (and specifically examine a larger network). Thus, we added an additional multilayer perceptron layer before the final averaging (and prior to the LogSoftmax). Using this new architecture (and the original starting learning rate of 0.001), we achieved training accuracy of 0.8609 and test accuracy of 0.8587 (and a test loss of 0.2760). As expected, this deeper version of DGCNN performed slightly better on the training and test sets than the shallower version.

6 Results, Conclusions and Future work :

The accuracies of the different models that we have explored and trained on the same dataset has been tabulated below:

Model	Test Accuracy (%)	Training Accuracy (%)
PointNet	87.35	88.02
PointNetDeeper	87.09	87.43
PointNet++	86.85	86.87
KD Net (depth = 11)	82.13	83.27
KD Net (depth = 15)	84.30	82.58
DGCNN (LR = 0.001)	85.42	85.06
DGCNN (LR = 0.01)	82.42	82.23
DGCNN Deeper (LR = 0.001)	85.87	86.09

Table 1: Accuracies of all the different variations

As can be seen from the table, PointNet performed the best. but the accuracies of the different models are comparable. This means that perhaps by adjusting the various hyperparamteres, the other models could potentially outperform the pointNet model. Different methods for grasp candidate generation was briefly explored during the course of this project but was not successfully implemented. This is something which could be further explored as right now, the candidates are being generated through a heuristic, which involves handcrafted features.

7 Appendices

This work was completed in partnership with Simon Kalouche of Nimble AI, who provided us with the initial project concept of working to improve robotic grasping using deep learning.

All code used in this project can be found at https://github.com/navjot95/pointNetGraspClassifier.git.

8 Contributions

Navjot worked on PointNetDeeper and PointNet++. Neethu worked on KD-Networks of varying depths. Zach worked on DGCNN (and its variations).

References

[1] Berk C. Calli; Arjun Singh; Aaron Walsman; Siddhartha Srinivasa; Pieter Abbeel; Aaron M. Dollar. *The YCB Object and Model Set: Towards Common Benchmarks for Manipulation Research*. URL: http://ycb-benchmarks.s3-website-us-east-1.amazonaws.com/. YCB Object dataset.

[2] Charles R. Qi; Hao Su; Kaichun Mo; Leonidas J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: NIPS, 2017. URL: https://arxiv.org/abs/1706.02413.

[3] Charles R. Qi; Hao Su; Kaichun Mo; Leonidas J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: CVPR, 2017. URL: https://arxiv.org/abs/1612.00593.

[4] Richard Hodson. "How robots are grasping the art of gripping". In: Nature, 2018. URL: https://www-nature-com.stanford.idm.oclc.org/articles/d41586-018-05093-1.

[5] Roman Klokov; Victor Lempitsky. "Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models". In: ICCV, 2017. URL: https://arxiv.org/abs/1704.01222.

[6] Andreas ten Pas; Marcus Gualtieri; Kate Saenko; Robert Platt. "Grasp Pose Detection in Point Clouds". In: Sage, 2017. URL: https://arxiv.org/pdf/1706.09911.pdf.

[7] "PyTorch". In:

[8] Yue Wang; Yongbin Sun; Ziwei Liu; Sanjay E. Sarma; Michael M. Bronstein; Justin M. Solomon. "Dynamic Graph CNN for Learning on Point Clouds". In: cs.CV, 2018. URL: https://arxiv.org/abs/1801.07829.

[9] David Kirkpatrick; Bhubaneswar Mishra; Chee-Keng Yap. "Quantitative Steinitz's Theorems with Applications to Multifingered Grasping". In: Discrete Computional Geography, 1992. URL: https://cs.nyu.edu/mishra/PUBLICATIONS/QuantSteinitz.pdf.

[10] Hongzhuo Liang; Xiaojian Ma; Shuang Li; Michael Gorner; Song Tang; Bin Fang; Fuchun Sun; Jianwei Zhang. "PointNetGPD: Detecting Grasp Configurations from Point Sets". In: ICRA, 2019. URL: https://arxiv.org/abs/1809.06267.

[11] Gusi Te; Wei Hu; Zongming Guo; Amin Zheng. "RGCNN: Regularized Graph CNN for Point Cloud Segmentation". In: cs.CV, 2018. URL: https://arxiv.org/abs/1806.02952.