Robotic Grasping Detection

Yunchu Zhang
yunchu@ucla.edu

Zhi Li
zhili@ucla.edu

Abstract

The goal of our project is to infer grasping location given an object for robotic grippers by using deep learning algorithms. ResNet based neural networks is built and implemented by using dataset from Cornell robot learning group. We also implement simple Variational Autoencoder for data augmentation. We compared the performance on architecture based on ResNet18 and ResNet34 with augmented data. The results show that our model is overfitting. We may either use larger datasets(like Jacquard) or propose new model method(thoughts from Yolo/fast rcnn)for further research to solve the problem.

1. Introduction

Robotic grasping is a challenging problem for many years. One research direction for this problem is to convert the perception aspect into a detection problem where, given noisy, partial view of the object from a camera, the goal is to infer the top locations where a robotic gripper could be placed.(See Figure 1). Our project was inspired by [1] and our goal for this project is not only to infer a viable grasp, but to infer the optimal grasp for a given object that maximizes the chance of successfully grasping it.

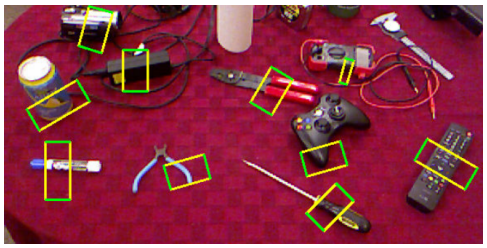


Figure 1. Rectangles Guiding Grippers[1]

1.1. Method

We use transfer learning method to load famous Deep convolution neural network ResNet as our main architecture and make pixel information (3*480*640 matrix) as our input. Also, we grab the mainly part of ResNet and change the back part (1000 output linear full connected layer) to 3*(4*2) linear module. Thus, the output could be 3 possible grasping boxes coordinates. The Resnet Network architecture and our algorithm are as follow, and we tried refined 18 and 34 layers resnet respectively with VAE or not. To simplify the algorithm, we choose MSE loss between label and output matrix,which will be easier to implement but hard to train. We also defined Accuracy function based on rectangle's IOU(intersection-over-union).

ResNet

Architectures for ImageNet:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2.x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3.x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4.x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5.x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)

Figure 2. Resnet net framework

1.2. Dataset

Our dataset is provided by Robot Learning Lab at Cornell University.`http://pr.cs.cornell.edu/grasping/rect_data/data.php`. The dataset consists of 885 images of 280 different objects with several images taken of each object in different orientations or poses. Each image was labeled by ground truth rectangles for grasping. Each rectangle in the label file is represented by 4 lines and each line contains the x and y coordinate of a vertex of that rectangle separated by a space. We reshape the labels and make sure there are total 3 ground truth grasping rectangles for each image.

1.3. Metrics for Detection

We compare the predicted rectangle for each image with the set of ground truth rectangles. We use the common bounding box evaluation metric of intersection divided by union - i.e. $Area(G \cap G^*)/Area(G \cup G^*)$ where G and G^*

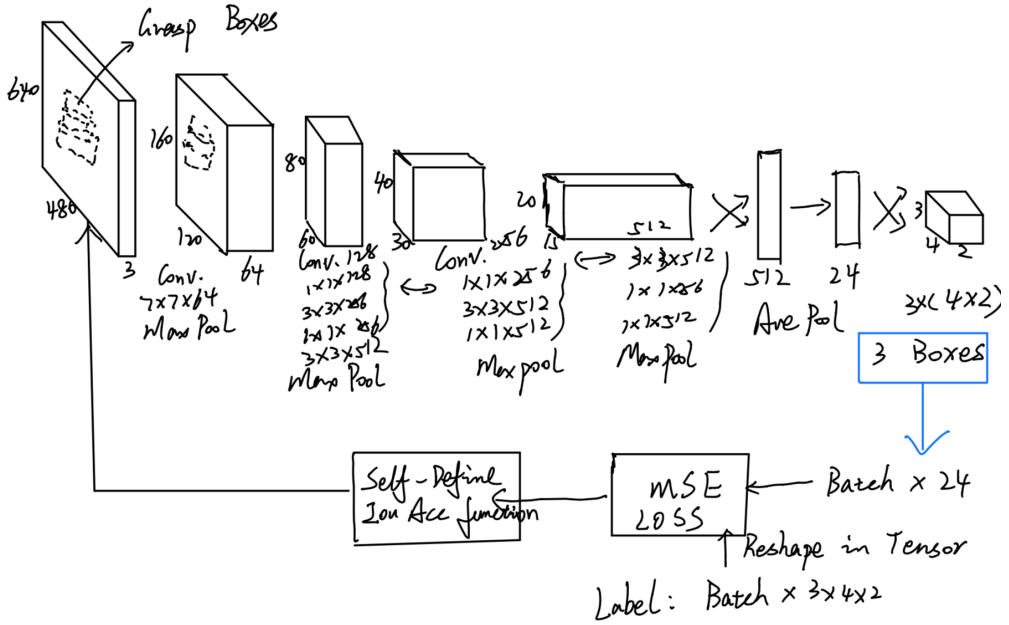


Figure 3. Grasp net framework

denote the predicted rectangle and ground truth rectangle. We consider a prediction to be correct if it scores at least 50% by this metric.

1.4. Experiment

1.4.1 data augmentation

We doubled our dataset by using Variational AutoEncoder(VAE). In VAE, we build 2 loss functions for accuracy of neural network and trade off between latent vectors and standard normal distributions. We simply implement MSE loss function for the model accuracy and implement Kullback-Leibler Divergence(KLD) for measuring similarity between the 2 distributions. The structure is shown in the extra page. By using VAE we enlarge our dataset from 885 to 1770 images with the same labels. The architecture for VAE is shown in Figure 4

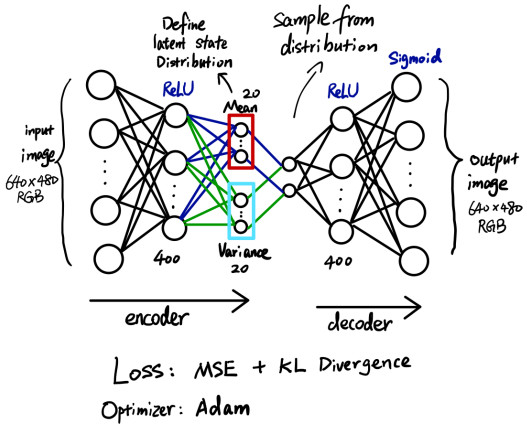


Figure 4. VAE Structure

1.4.2 Comparable experiments

We compared refined architecture based on ResNet34, ResNet34 with VAE, ResNet18 and ResNet18 with VAE.

Table 1. Accuracies for Different Experiments

ResNet18/34	Train Acc %	Val Acc %	Test Acc %
18 no VAE	97.66	52.86	50.87
18 with VAE	96.81	60.32	52.63
34 no VAE	98.12	55.65	50.65
34 with VAE	98.24	58.85	52.96

2. Results

We run our code on PC with i7-7700K CPU, Nvidia GTX 1070 and 16 GB RAM. We use tensorboard to record training loss/acc and validation loss/acc and print out the test acc. For each training, it nearly cost 5,6 hours to train on GPU. The results are shown in the Table 1: The results shows that the model is overfitting. There is no obvious improvement when we change to ResNet34 for this problem. Also the results implies that with data augmentation(implemented by VAE), the validation accuracy improved a little and also the test accuracy. However, we still

got overfitting results.

The accuracy criteria we set is the rectangles' IOU is greater than 0.5. This means that the model should at least predict half of rectangles correctly compared to the ground truth rectangles in given image. This may be strict for the given problem. We may easy the criteria to perhaps 0.25 for further study.

Figure 5 and Figure 6 show the loss and accuracy changes for ResNet18 with and without data augmentation and Figure 7 and Figure 8 show the loss and accuracy for ResNet34 with and without data augmentation. The validation loss is high and validation accuracy cannot improve by simply changing learning rate.

Due to limited dataset, we had not achieved ideal results(overfitting). We tried to solve this problem by enlarging the dataset. The reason we chose VAE for data augmanetation not traditional ways is that our problem is not just a classification problem. Relabeling the augmented data(e.g. crop image) requires lots of work.(Labels in dataset is assigned based on coordinates in 640*480 images). So we applied VAE to keep the images in same dimension but slightly different in pixel values. In this way we could keep the label for each scenario and increase the number of images for training. The VAE is built by fully connected neural network. This requires lot of computational resources due to the size of the images.

In Figure 9, it shows that larger learning rate maybe better for our model to fit the model and reduce loss quickly since our MES loss want model to learn every rectangle's coordinates accurately,which will result huge loss and need to decrease fast at start and a relatively small learning rate at mid stage.

To solve that problem, we utilize the scheduler module to decay LR by a factor of 0.9 in every 75 epochs,which will decrease the LR during training and learning more meticulously. In Figure 6 (b),we could see that it has a obvious step during 100 epoch and make the 'smooth' loss become drop again.

3. Discussion

From the result, we get over 50 percent accuracy in raw test data,which improves a lot after data augmentation. Also, we add scheduler module for learning rate to decay it with time going,which could make the model have good performance in later stage. In addition, the GPU training speed up our tuning process and make us change hyperparameters quicker based on loss and acc's trend.

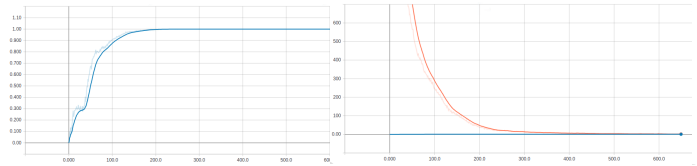
As the result shown, since our data scale is a little bit small, we use data augmentation method to overcome overfitting. For more improvement in performance, big data is needed. We have found another dataset called Jacquard with 20k pictures and standard labels. Since they do not reply to us for download link, we would try it in future research. Also the MSE loss for regression the 24 points is too simple and hard to tune the model. It is pretty hard to generate a model that have good performance with that way. Thus, we will try more logical algrithom such as yolo,yolo-v2 with anchors to search in picture first and then regress the rectangle's center x-y coordinates,width,height and rotation attitude. It is so regret that we did not have time to try those algorithms since the Cornell dataset is not so clean and we had to process raw data into a clean and usable form before improving training algorithms.

References

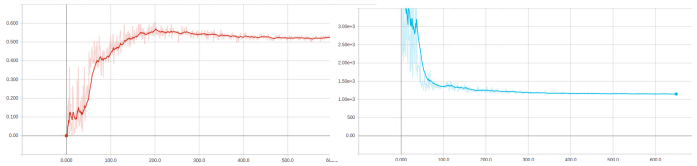
[1] Lenz, Ian, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps." The International Journal of Robotics Research 34, no.4-5 (2015): 705-724.

[2] Sergey levine, Peter Pastor."Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection."

[3] A. Depierre, E. Dellandrea, and L. Chen, Jacquard: A Large Scale Dataset for Robotic Grasp Detection in IEEE International Conference on Intelligent Robots and Systems, 2018.

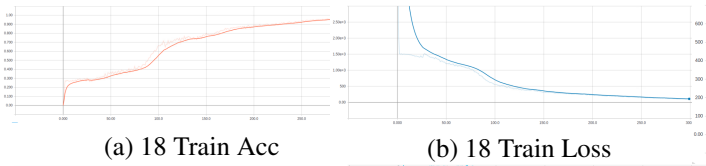


(a) 18 Train Acc (b) 18 Train Loss

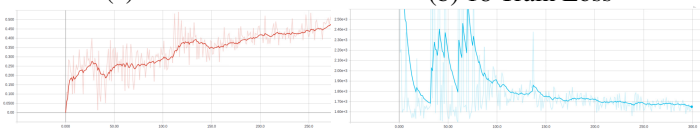


(c) 18 Val Acc (d) 18 Val Loss

Figure 5. Loss and Accuracy for ResNet18 with VAE

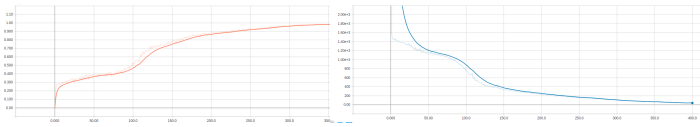


(a) 18 Train Acc (b) 18 Train Loss

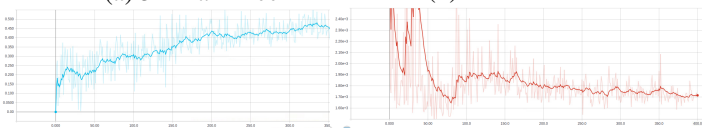


(c) 18 Val Acc (d) 18 Val Loss

Figure 6. Loss and Accuracy for ResNet18 no VAE

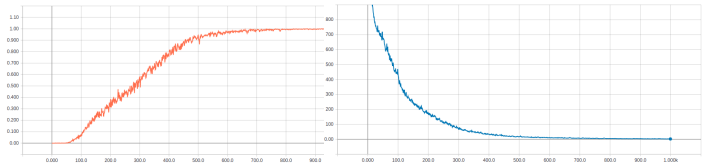


(a) 34 Train Acc (b) 34 Train Loss

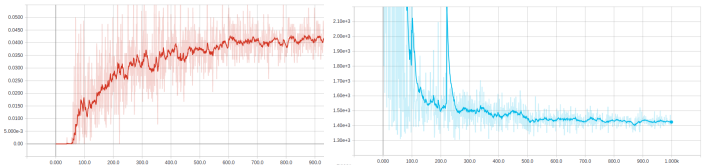


(c) 34 Val Acc (d) 34 Val Loss

Figure 7. Loss and Accuracy for ResNet34 with VAE

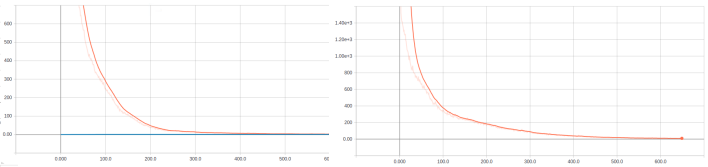


(a) 34 Train Acc (b) 34 Train Loss



(c) 34 Val Acc (d) 34 Val Loss

Figure 8. Loss and Accuracy for ResNet34 no VAE



(a) 18 loss with LR = 0.0035 (b) 18 loss with LR = 0.001

Figure 9. Loss Comparison on different Adam learning rate

The Experiment results is summarized in Table 2

Table 2. Accuracies for Different Experiments

ResNet18/34	Train Acc %	Val Acc %	Test Acc %
18 no VAE	97.66	52.86	50.87
18 with VAE	96.81	60.32	52.63
34 no VAE	98.12	55.65	50.65
34 with VAE	98.24	58.85	52.96

As shown below: After refining the architecture, resnet 18 and 34 will have the same output size (3*4*2), and layers, size information and activation functions details are shown in figures. Besides, in Pytorch library, the Resnet have been added batch-norm layer, which could improve the stability of network and overcome the bad initialization.

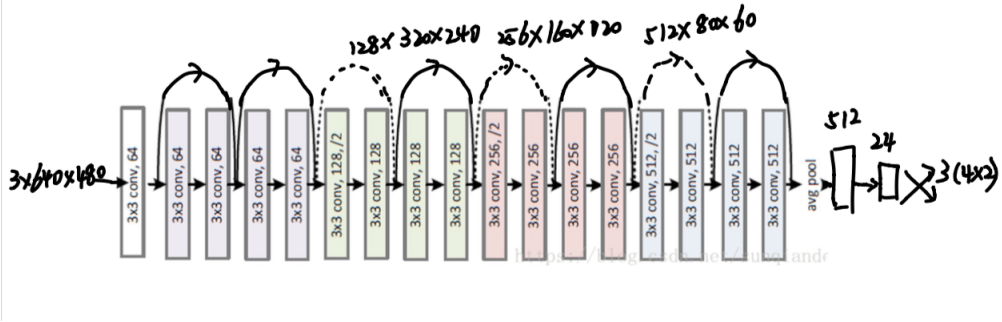


Figure 10. Refined resnet18 framework

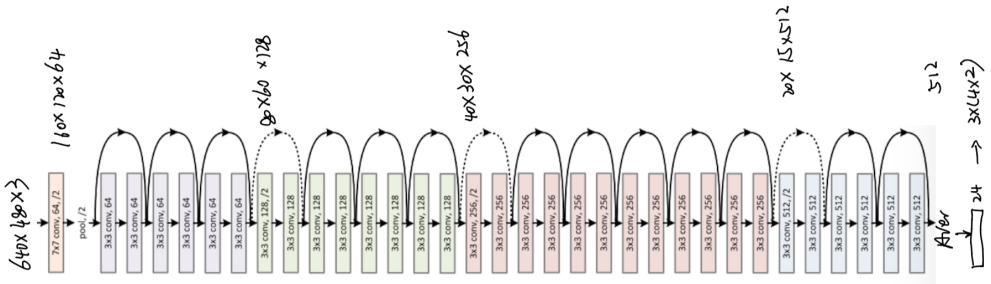


Figure 11. Refined resnet34 framework

```
(layer1): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (1): BasicBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

Figure 12. CNN layers detail