CS230

SuctionNet: An end to end model for suction

Nikita Demir
Department of Computer Science
Stanford University
ndemir@stanford.edu

Jose Giron
Department of Computer Science
Stanford University
jmgiron@stanford.edu

Mauricio Wulfovich
Department of Computer Science
Stanford University
mauriw@stanford.edu

Abstract

Suction-based end effectors are becoming a common tool for automated robots to lift and carry objects. However, a common challenge with these systems is selecting optimal suction grasp locations. Most current systems use a multi-step process to perform this analysis. In this paper, we build on previous work on RGB-D image segmentation to propose a new model for suction location based on RGB-D images. The model we propose is an FCN composed of two encoders, one which takes as input RGB images and one which takes 3-dimensional representations of depth images, and a single decoder network which takes the average of the encoder network's outputs as input to produce an output segmentation. We propose a new method called "mid-fusion" to enable skip-connections which, to the best of our knowledge, hasn't been tried in dual-network RGB-D CNNs.

1 Introduction

In order for robots to robustly perform useful tasks in the real-world – from stocking grocery shelves to assembling complex machinery – they must be able to interact with varied previously-unseen objects. A crucial and necessary first component of such tasks is for robots to automatically detect the best places to grab these items. In this paper, we worked with a synthetic dataset provided by Nimble.ai created to train robots that use suction grasping. The input data consists of randomized RGB-D images of objects labeled with their optimal suction location. After training, our neural network model then predicts optimal suction grasp labels on such unseen images.

Due to a lack of accessible pre-trained models on related RGB-D tasks, we did transfer learning using weights from the VGG-16 network trained on the ImageNet challenge. Since such weights, however, are built to work with three channel, rather than four channel images, we used an architecture that takes in the depth and RGB components of our images separately. We pre-process our depth data using depth-jet encoding to spread out its information over three layers and then run the 'encoding' portion of our model on it in parallel with the RGB data. We then combine the information gleaned from both streams, and then finally 'decode' the data using a Fully Convolutional Network architecture to output labels that are the same size as our original images.

2 Related work

DexNet 3.0 by Mahler et al (1), was a highly influential paper on suction grasping which created the first major dataset on the subject. This model works by creating a point cloud on an object based on the depth data, and then individually evaluating each point to determine the quality of the grasp. Mahler et al. introduced the idea of a Grasp Quality CNN (GQ-CNN) to evaluate the grasp quality of a single point. The model would return in the end a map of the quality of every point in the point cloud to determine what a good suction space is. While this approach delivered good results, we were able to achieve this task in an entirely end-to-end model.

As part of our literature review, we also looked at other tasks using RGB-D images. Wang et al. (3) developed a novel approach to using depth-data which uses the depth dimension to create weights for pixel importance in convolution and pooling layers. This approach worked well in a segmentation task and we thought it could also be used successfully in our problem. The proposed approach adds an extra weighting term to the convolution and avg. pooling operation based on the difference in depth between a point and the center of the kernel (at that step). This adds essential geometric information to the operations that isn't contained in RGB data. An attractive element of this approach is that it doesn't require extra parameters to train on depth data, which most other approaches do. However, we had difficulties working with their repository and in the end, had to abandon this approach.

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

To the best of our knowledge, Gupta et al. (7) was the first to propose using two separate CNNs, one of which was trained on RGB data and another on depth data. They then took the concatenation of this output as input to an SVM for object detection. Based on these results, they train an RCNN (Regions with CNN features) as described in Girshick et al. (8) on a semantic segmentation task. Gupta also introduced the concept of turning depth data into a 3-dimensional representation which they call HHA. This enable CNNs to be pre-trained on large datasets of RGB images like ImageNet and then fine-tuned with the task specific data. Gupta found that knowledge learned by the networks on RGB data transferred well to interpreting HHA images. This paper introduced key innovations like using 3-dimensional depth representations and training separate networks for depth and RGB data. However, the model we develop is different in that it uses parallel FCNs instead of RCNNs to develop an end-to-end model for image segmentation, skipping the object detection step. Furthermore, instead of using the more complex HHA encoding, we achieved good results with the simpler jet encoding.

Based on the Gupta et al. architecture, Eitel et al. (2) created a network for object detection based on RGB-D images. They train two separate CNNs, one for depth encoded in a 3D format (both Jet and HHA encodings are used, with Jet achieving the best results) and another for RGB images. The output of these is concatenated and then fed into a fully connected neural network to create an output distribution. While not an image segmentation task, this paper inspired our approach to merge the output of the two CNNs before passing them through a network which produces the output. Unlike Eitel et al., we don't concatenate the output of the two networks, but instead average it, a step we call "mid-fusion", before passing it through our decoder network. This motivated our double encoder with one joint decoder approach.

The final paper we looked at was Long and Shelhamer et al. (9), which also looks at the task of semantic segmentation. Long and et al. create a similar network to ours, and introduce FCNs intialized with common CNN weights like VGG and AlexNet for RGB-D image segmentation, which are then fine-tuned on the target task. Based on Gupta et al., they also use a parallel network structure and the HHA representation of depth images. However, unlike our model, they perform late-fusion, training the two FCNs in parallel end-to-end and then summing the predictions of both networks at the end. The approach we take is simpler, using two encoder CNNs pretrained on VGG, like Long and Shelhammer et al., but then one single decoder network to upsample the image. The key step here is mid-fusion, which is the averaging of the skip connections from each network, which are then passed on to one decoder network that up samples these intermediate outputs to produce an output of the same size as the input image.

3 Dataset and Features

Due to the difficulty in collecting and labelling a large dataset of images for this task, our project partner, nimble.ai, provided us 480 randomly simulated examples. Each example consists of an rgb image, a depth image, and a label image, which contains the target suction spots. The images are generated by dropping randomly selected 3D objects on a simulated tray using a physics simulator. Each of the objects has labels for correct suction spots. Those per-object labels are used to create the example label image.

Research on using generated images has shown promising results, with James et al attaining 70% zero-shot grasp success on unseen objects simply training on such data, and later attaining 91% accuracy after fine tuning on a batch of only 5,000 real-world grasps (6).

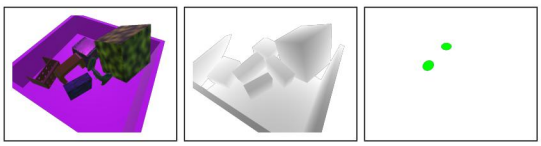

Figure 1: *Left:* RGB image *Center:* Depth image *Right:* Correct output

In Figure 1 above, we present a sample from our dataset. The top left and right images correspond to the RGB and depth input to our system, respectively. On the bottom lies the correct output, which is the coordinates of surfaces with high potential for suction grasping (the coordinates are labeled in green).

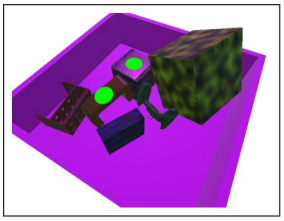

Figure 2: Correct output overlayed on RGB image

3.1 Processing

The original images are all of RGB type and are of shape 800x600 pixels. We reshape all the images using Pillow to 224x224 pixels to make training faster. This specific shape was chosen because it's the one VGG originally takes in. We then strip the RGB images of the alpha dimension, and take only the first dimension from the depth and label images. We divide the RGB images by 255 to restrict every pixel between 0 and 1. We also preprocess the label so that every pixel is either 0 (not a good suction point) or 1 (good suction point).

For our depth images, studies (2) (7) have shown that colorizing them can provided a significant performance boost. That is, spreading the depth information over three RGB channels performs much better than encoding the depth as greyscale. Given that the VGG network from which we get our pretrained weights is designed for RGB images, the encodings better allow our network to learn useful features. We used the Depth jet encoding which has been shown to outperform more complex and computationally costly methods like HHA (2). The Jet encoding maps every pixel in the depth image to a color value. Nearer pixels (smaller depth values) map to red, over green, all the way to further pixels which map to

green. Finally, we divide by 255 to bound each pixel's value between 0 and 1.

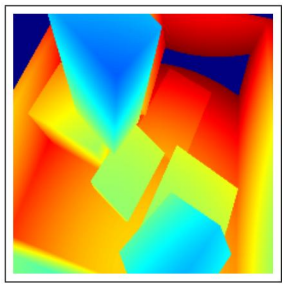

Figure 3: Example of processed, jet-encoded depth image

4 Methods

4.1 SalNet

Our first approach originates from viewing the problem as an image saliency detection task. In image saliency detection a model seeks to output a saliency map that highlights the most important parts of an input image. Importance is determined in different ways ranging from pixel uniqueness to probability of visual human attention. This later probability metric is an explored Deep Learning problem that closely matches our input-output pairs, so we implement the two neural models from the seminal work in the field by Pan, McGuinness et al (6).

The first SalNet model we implement is a shallow ConvNet inspired by the AlexNet architecture for image classification but with 3 convolutional and maxpool layers rather than 5. The output is then passed through 2 fully-connected layers and the sigmoid function to arrive at pixel-wise probabilities. Although this architecture is rather shallow in layers, due to the fully connected layers the final parameter count is about 64 million. In addition because this model is trained entirely from scratch we found it difficult to train.

The second SalNet model we implement is the deeper ConvNet with pretrained VGG layers. The architecture consists of three VGG pretrained convolutional layers mixed with maxpool layers followed by 5 convolutional layers of various filter number and size and finally passed through a single deconvolutional layer plus sigmoid to arrive at a saliency map of the original image size. Although this model is partially pretrained and contains fewer parameters at about 26 million we also found it unwieldy to train despite extensive parameter and architecture search.

Unfortunately, neither of the two SalNet approaches produced significant learnings. After an extensive hyperparameter and architecture search, which we discuss in Experiments, we decided to reasses our problem and find alternative solutions.

4.2 FCN-8s

After reexamining the literature, we saw more similarities with our task and image segmentation. Image segmentation is the task of separating out an input image into parts that belong to different objects or classes. Deep Learning segmentation is similar to saliency in that the output is a pixelwise prediction probability, but different in that the output is simply a per pixel classification rather than a mix between regression and binary classification. We implement a Fully Convolutional Network approach to image segmentation described in Long and Shelhamer et al (9)

In particular, we implement the VGG-FCN-8s model described in Long and Shelhamer et al (9) VGG-FCN-8s, hereinafter referred to as FCN-8s, takes the pre-trained layers from VGG, removes the classification layer and replaces the fully connected layers with convolutional layers. Since the network lacks any densely connected layers it has the advantage of working with any sized inputs without needing to be retrained. The first 7 trainable layers are taken from VGG after which a single 1x1 convolution is applied instead of a fully-connected layer. If this intermediate output is though of as an encoding of the information in the original image then the following layers constitute the decoder part of the network. The encoding is passed through several upsampling layers using learnable transpose convolutional layers. At each step of upsampling the output is summed with an earlier output from the VGG layers creating a skip connection.

We want to explain in more detail the deconvolutional layers as they are key operations in our network. A deconvolutional layer simply performs the opposite of a convolutional operation by both upsampling the image and passing a learnable filter of parameters over it. If one thinks of the convolutional operation as a matrix product involving a special matrix created from unrolling the filter then the deconvolution operation can be thought of as a matrix product between the transpose of a filter matrix and the flattened deconvolutional layer input.

We also want to highlight the importance of the skip connections in FCN-8s as they are the key difference between our successful models and the unsuccessful SalNet models. Generally, adding skip connections to neural networks allows for gradient information to reach deeper layers in a neural network by providing a sort of "highway" for the gradient to pass along and bypass intermediate layers. In addition, during the forward step layers can just default to the identity function by setting their learned parameters to zero. These advantages allow for training of much deeper networks such as ResNet-101. While these advantages definitely contributed to our FCN-8s's ability to train better, the skip connections in this model also allows the network to combine information from earlier finer layers with the deeper coarse layers to produce more rich segmentations during upsampling. These earlier layers contain information about edges and edge combinations that especially in our case should intuitively improve our model's ability to detect suction locations by providing structural information of the objects.

4.3 RGB-D FCN

Our RGB-D FCN was based based on our FCN-8 architecture but uses two encoders and one decoder. One encoder takes RGB images as input and the other takes Jet encodings of depth images. Each of the encoders has the same archi-

tecture as in the FCN-8s network and is initialized using the VGG-16 network's weights. After initializing each of the decoders with the VGG weights, we use the output of the 3rd, 4th and 7th layers for each network. We average each corresponding layer output from each of the two networks to create one unified representation of the 3rd, 4th and 7th layers. We then pass these as input to the decoder.

The decoder is the same as the one for the FCN-8 network, but using the averaged skip-connections. The output of the network for each pixel is passed through a sigmoid function, We trained using binary cross entropy loss as our loss function and used the Adam algorithm. We used the default 0.9 as beta 1 and 0.999 as beta 2. We also used a learning rate of 0.0001.

The model is constructed such that both encoders and the decoder can be trained in parallel, end-to-end.

Architecture

RGB-D FCN

RGB Encoder	Depth Encoder	
input (224 x 224 RGB image)	input (224 x 224 Jet image)	
conv3-64 conv3-64	conv3-64 conv3-64	Layer 1
maxpool = layer 1 output		
conv3-128 conv3-128	conv3-128 conv3-128	Layer 2
maxpool = layer 2 output		
conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256	Layer 3
maxpool = layer 3 output		
conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	Layer 4
maxpool = layer 4 output		
conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	Layer 5
maxpool = layer 5 output		
conv1-4096	conv1-4096	Layer 6
conv1-4096	conv1-4096	Layer 7
Decoder		
input (avg layer 7 output)		
conv1-3		Layer 8
deconv4-512		Layer 9
Add avg layer 4 output		Skip connection
deconv4-256		Layer 10
Add avg layer 3 ouptut		Skip connection
deconv16-4		Layer 11

4.4 Loss Functions

We used two loss functions during our project, mean-squared-error and binary cross entropy.

Mean-squared-error is defined for a single image by the following formula:

$$\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2$$

where $\hat{Y}_i$ is the probability of pixel i being a good suction spot outputted by the network and Y_i is the correct label, which is either 0 or 1. The difference between the network output and the correct label is calculated at every pixel and then squared. The results of these operations are then averaged over all N pixels in the image. This is done for every image and defines the loss.

Binary cross entropy is defined for a single example as follows:

$$\sum_{i=1}^{N} -Y_i \log(\hat{Y}_i) - (1 - Y_i)\log(1 - \hat{Y}_i)$$

where $\hat{Y}_i$ is the probability predicted by the network of pixel i being a good suction spot, and Y_i being the ground truth. This is calculated across all N pixels in the image. Note that because the output of the network is passed through a sigmoid function, $\hat{Y}_i$ will always be bounded by 0 and 1. Cross entropy loss is the standard loss used for classification tasks, which we do on a per pixel basis.

5 Experiments/Results/Discussion

5.1 Experiments

For all models weights were initialized with Xavier initialization and biases with 0. For standardization every pixel input was divided by 255. and all ground truth label pixels were transformed to either 0 or 1. We ran the shallow SalNet with the authors' hyperparameter choices of 0.03 initially for the learning rate and decreased over 1,000 epochs to 1e-4. We used Stochastic Gradient Descent with Nesterov momentum. Using binary cross entropy loss we confirmed their findings of vanishing gradients and saw that our model failed to lower training loss even on a single example. So, we switched to using the MSE / Euclidean loss as recommended by the authors. Although this led to our initial training loss decreasing, we found that the training loss quickly plateaued and the output strangely converged to vertical lines. Faced with this high bias problem, we extensively searched for hyperparameters including learning rate, weight decay, fully connected layer sizes, number of filters, and learning rate decay but we were not able to overcome the plateau.

We then transitioned to the deeper SalNet model in the hopes of achieving lower bias with a larger network. We also initially ran the model with the authors' hyperparameters of $0.1/(224 \times 224 \approx 1e-7$ and halved it every 100 iterations. We used standard L2 weight regularization and froze the VGG weights. Unfortunately, this model also encountered a plateau and failed to lower bias. We experimented with higher and lower hyperparameter values as well as different regularization weights. We used a small batch size of 2 to improve stochasticity and used MSE / Euclidean loss to combat vanishing gradients as well as gradient clipping for exploding gradients. Despite these searches our model at best converged to a 0.5 output for all pixels in the input image. We theorized that either the model did not contain enough representational power in its structure for our task, or we simply did not run it for enough epochs to see a breakthrough in training. Either way, we pivoted to FCN-8s because of its incredible success.

We first ran two FCN-8s models separately on rgb images and jet-encoding depth images. We set our batch size to 4, dropout probability of 0.5, and performed a random search

over possible learning rates. We settled on 5e-5 as the learning rate for both of them. We experimented with decreasing the learning rate so that every couple of epochs it would be halved, but found that this only slowed initial training. Our minibatch size was set to balance between not being so high that we ran into out of memory issues on the AWS GPU instances and not being so low that training was slow.

We only had enough time to run RGB-D FCN with the same hyperparameters. We trained all FCN models on artificially augmented datasets to increase our overall dataset size. We incorporated image mirroring, random cropping, shearing, and rotation.

5.2 Results

Models	Intersection over Union
Shallow SalNet	0.0
Deep SalNet	0.0
RGB FCN-8s	0.15
Depth FCN-8s	0.12
RGB-D FCN-8s	0.13

Table 1: Final results of different models

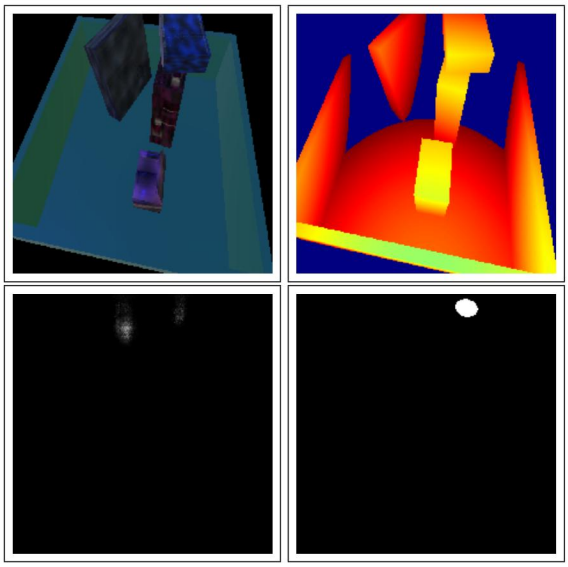

Figure 4: *Top left:* RGB input. *Top right:* Depth input. *Bottom Left:* Our predictions. *Bottom Right:* Ground truth.

5.3 Discussion

We were pretty impressed with our overall results. The best outputs capture the general location of the ground truth labels pretty well. We found that one limitation of our data is that points are sometimes not defined for objects which probably should have them. This led to our model outputting seemingly correct predictions to the naked eye but contradictory with our ground truth labels. Thus lowering our overall Intersection over Union.

We were surprised to see poorer results from the RGB-D FCN-8s, but we suspect this may be because we did not provide it enough time to fully train. In addition, we found that after a certain point the learning rate may have been too high. We encountered variance issues that we tried to combat by using dropout as well as data augmentation.

We chose to use Intersection over Union as our main metric because it captures our end goal very well. IoU is defined as the amount of area that two classifications (in our case the ground truth suction labels and our predicted suction labels) intersect divided by the union of their areas.

6 Conclusion/Future Work

In this paper, we created a single end-to-end system to find optimal suction points based on RGB-D data. We propose a new model for RGB-D segmentation which we train and test on a generated dataset of RGB-D images of random objects. Our final model is able to create labels on

Due to time constraints, there were items of work that we were unable to explore. The first is running and fine-tunning our current model with real RGB-D images. We believe that we can get better results on generated images with more training time, but if this model is to be used in production, further training on real world conditions is necessary. We would also like to experiment with more methods of mid-fusion than simple averaging to hopefully leverage more information from each of the data sources. Specifically, we'd like to experiment with methods that include learnable parameters in merging the outputs from the two encoder networks. Finally, we'd experiment with manual learning rate decay to counter instances where training began to slow down or diverge.

7 Github Repository

https://github.com/N-Demir/RobotSuctionAI

8 Contributions

Nikita: Pre-processing, SalNet's, RGB FCN-8s, Depth FCN-8s, Research. *Jose:* Depth-Aware CNN (didn't work), Preprocessing, RGB-D FCN, Metrics, RGB-D image segmentation research. *Mauricio:* Pre-processing, Jet encoding, depth FCN-8, Data augmentation, research

Thank you to Nimble.ai for providing the dataset

References

[1] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, Ken Goldberg. Dex-Net 3.0: Computing Robust Vacuum Suction Grasp Targets in Point Clouds using a New Analytic Model and Deep Learning. CoRR, vol. abs/1709.06670 (2017)

[2] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, Wolfram Burgard. Multimodal Deep Learning for Robust RGB-D Object Recognition (2015)

[3] Weiyue Wang, Ulrich Neumann. Depth-aware CNN for RGB-D Segmentation (2018)

[4] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia

Hadsell, Konstantinos Bousmalis. Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks (2018)

[5] N. Sunderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, et al. The limits and potentials of deep learning for robotics (2018)

[6] Junting Pan, Kevin McGuinness et. al. Shallow and Deep Convolutional Networks for Saliency Prediction (2016)

[7] Saurabh Gupta, Ross Girshick, Pablo Arbel, and Jitendra Malik. Learning Rich Features from RGB-D Images for Object Detection and Segmentation (2014)

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation (2013)

[9] Jonathan Long, Evan Shelhamer, Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation (2014)