CS230

Object Localization with CNN

Holly Liang
Electrical Engineering
Stanford University
xuejiao@stanford.edu

Zixuan Zhou
Electrical Engineering
Stanford University
zixuan95@stanford.edu

Yuwen Zhang
Materials Science and Engineering
Stanford University
yuwen17@stanford.edu

Abstract

Humans need to use vision to localize objects before grasping, in order to roughly estimate the localization of the object. Thus, it is logical to incorporate vision and vision-based localization in the design of robotic systems. In this project, we trained CNN-based neural network models to predict the 3D location of the target object from RGB-D image inputs. We propose 3 models: 2D VGG model, Voxel model, and colormap parallel model. Comparing the results, we found colormap parallel model to perform the best.

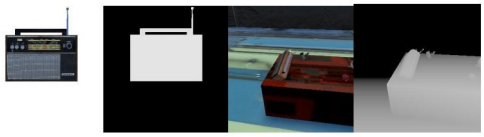
1 Introduction

When training a robot to perform what may seem like simple tasks such as pick-and-place, an important task is to visually perceive the object and extract the localization information, specifically, the target object's position relative to the robot's hand. Since it is easy to build a correspondence between the location of robot's camera and robot's hand, the major task would be to find the location of the target object in the coordinates of the camera. Our goal in this project is to tackle this important task of extracting localization information from the robot's camera (eyes).
The input of our model is an RGB-D image output from a (synthesized) depth camera, and the output is a (x, y, z) 3D location of a chosen object. Deep learning has been proved to have magical power for computer vision tasks. However, compared to RGB image data, large-scale RGB-D image dataset is much harder to find. Thus, using RGB-D image for localization is a new subject that few has touched. In this paper, we propose three VGG16-based models to extract features from RGB-D images and make localization predictions.

2 Related work

Since 2D image training has been widely done, some works applied transfer learning to train 3D CNN models, among which [2] proposed a hybrid 2D/3D convolutional neural network that can be initialized with pre-trained 2D CNNs. Besides, some works attempted to design a better CNN structure to introduce depth information. [13] is based on parallel neural network structures, and [14] introduced a scale-adaptive regression model, which adjusts the filter size according to the depth values from the depth image. [7] propose an encoder-decoder type network, where the encoder part is composed of two branches of networks that simultaneously extract features from RGB and depth images, and fuse depth features into RGB feature maps as the network goes deeper. Apart from that, many works [7] [8] performed semantic segmentation, which complements object detection, and can yield reliable object perception.

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



(a) (b) (c) (d)

Figure 1: (a). RGB image of target item, (b). depth image of target item, (c). RGB image sample from dataset, (d). depth image sample from dataset.

3 Dataset and Features

A. Dataset preparation

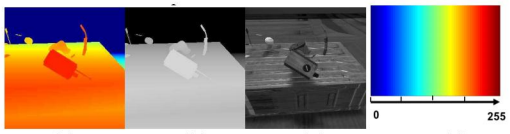
We explored the 18,000-pictures dataset generated by Unity3D simulation platform, provided by Stanford AI Lab and Deepdroid. In this dataset, each sample consists of a $299 \times 299 \times 3$ RGB image and a $299 \times 299 \times 1$ depth image. On each image, there are 10-20 different items, and all of the items belong to a 55 items itemset. The true position (x, y, z) of each data point, that is, the center of each item that appeared in the image, is specified in the names of the image files (RGB and depth). We parsed the file names into 3D labels of the data point, which we use to calculate loss for the network. Not having pixel-wise or boxing labels is a huge challenge in our work, which later on limited our options in model selection.
To simplify the problem, making it easier to train, instead of localizing all 10-20 objects that appeared in an image, such as the one in Figure 1(c) and (d), we are approaching the problem by localizing one item at a time. Currently, we are training the model to localize an old radio, shown in Figure 1(a),(b). In our project, for a specific object, we built our model on a training set of 3000 samples, and an additional 300 random samples for each development and test sets respectively.

B. Domain Randomization

The dataset was processed using domain randomization [3], which adds different simulated light sources and hues to the generated images. As such, our model can better fit to images taken in real-life situations in the future.

C. Color Mapping

In our third model (Color-mapping Parallel Model), we processed the depth image (shown in Figure 2(b)) using color-mapping strategy, to convert it into RGB $(299 \times 299 \times 3)$ image using a JET color matrix. As shown in Figure 2(d), the JET matrix transitions from blue to red, with green, yellow and orange in between. According to this matrix, we convert depth information (0 - 255) to corresponding colors to get the color-mapped image. In this converted image, objects with smaller depths will be more red and objects with higher depths will be more blue (shown in Figure 2(a)).
The pre-trained VGGNet (excluding the top fully connected layers) is skilled in detecting basic edges of RGB image. In this way, depth information is represented as colors and hence can be captured by VGGNet.



(a) (b) (c) (d)

Figure 2: (a). color-mapped depth image, (b). original depth image, (c). color image. (d).JET matrix for color-mapping

4 Methods

A. Model 1: 2D VGG Model

We built our first model with VGG16. The first challenge we encountered was to add the 4th channel to the input of VGG16. As we know, the original VGG16 from Keras package takes in images with 3 channels(RGB). We treated the depth information as the "fourth color" and added an additional convolutional layer to convert the 4-channel input images into three channels. The resulting network is shown in Figure 3. We input a $299 \times 299 \times 4$ image file, run through a convolution layer with filter size = 3, add padding and reshape to feed it into the Keras VGG16 model. After VGG16, we added a few more fully-connected layers to generate an output of estimated (x, y, z) location of the target object.

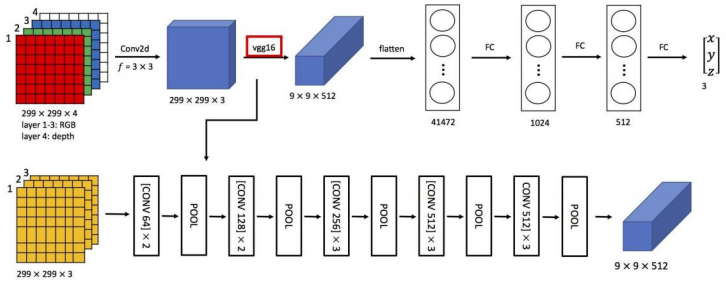


Figure 3: Illustration of model combining depth information into 2D VGG.

B. Model 2: Voxel Model

We then switched to preprocessing the depth information to produce a spatial 3D voxel representation combining depth and RGB information, so that after the 2nd layer, we can continue with the original 2D VGG model. The 3D voxel representation is created with the same height and width as the original image, and with depth determined by the difference between the maximum and minimum depth values found in the images[2]. Each RGB-D pixel of the image is then placed at the same positions in the voxel grid but at its corresponding depth, as illustrated in Figure 4[2]. We then quantize our depth values into 10 intervals, and then feed our input into VGG net. To explain further, we chose one-hot encoding as the way to discretize the depth values. i.e. for each pixel, if d = 10, the generated matrix contains all zeros except the first interval.

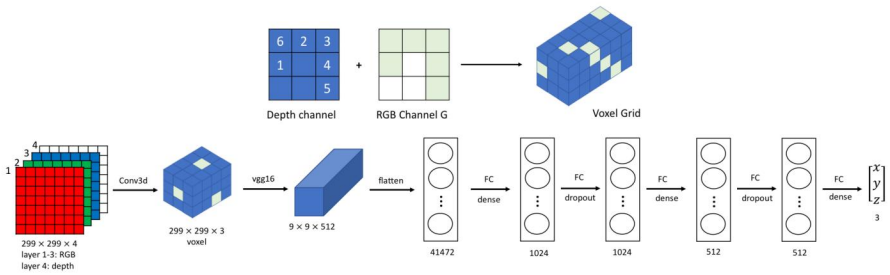


Figure 4: Illustration of Model combining depth information into 3D voxel VGG.

C. Model 3: Colormap Parallel Model

As shown in Figure 6, in this model we are still using VGG16 model (upper) to process the RGB image. In addition, we make use of the depth image by color-mapping it into another RGB image using JET colormap. The resulting RGB image is then processed by another VGG16 model (lower) in parallel. We then merge and flatten the two VGG outputs into one, and then feed into 3 FC(Fully Connected) layers. First FC layer is tanh with 1024 nodes, second FC layer is relu with 512 nodes,

and third FC layer is relu with 3 nodes. We started by building all three FC layers with relu. However, the loss exploded and we tried 1) change first FC layer from relu to tanh, 2) add dropout layers between neighbor FC layers to make sure that the weights are spread out evenly among all the nodes and further regularize the model. And 3) use Adam update method instead of Momentum.

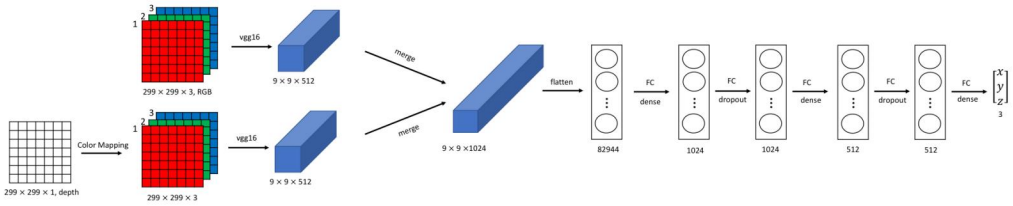


Figure 5: Illustration of Model color-mapping depth information into parallel VGG.

5 Experiments/Results/Discussion

A. Training strategy

Training Environment. We built our instance on: 1) Google Cloud Platform (GCP), with n1-highmem-8 (8 vCPUs, 52 GB memory). We divided our data into mini-batches of size 32 to improve the efficiency. In this setup, our entire dataset is too big to fit into the GCP memory.
2) Amazon Web Services (AWS), with p2.xlarge instance (1 GPU, 4 vCPUs, 144 GB memory). With this setup we can fit our entire dataset into AWS memory. However, we still divided the dataset into the same mini-batches to keep our experiments consistent.
Initialization. Transfer learning is applied to train the model. Weights for VGGNet are copied from RGB network trained on ImageNet. The weights of fully connected layers and modified convolution layers are initialized with He initialization.
Batch size. The batch size for gradient descent is 32. In order to maximize computation efficiency, a batch size of a power of 2 is preferred. Moreover, since the matrix of our image is of large size and the computation ability available to us is limited, small batch size selection is further justified.
Learning rate. The learning rate is 0.0001 for first 7 epochs, and followed by 0.00001 towards the end. As 10 epochs are trained, using 2 different values for this parameter is simple and good enough. To explain further, the learning rate for the latter epochs is relatively small, since we need to perform fine-tuning of the model for second stage to obtain both high efficiency and good convergence.
Update Method. Adam Adaptive Moment Estimation[11] is an optimization method that computes adaptive learning rates for each parameter. In practice, when compared to other adaptive learning algorithms, Adam shows the advantages of faster convergence and more efficient learning. It can also correct common problems in other optimization techniques like diminishing of learning rate, slow convergence speed and large fluctuations in the loss function caused by update of high variance parameters. In our project, we also tried Momentum gradient descent, but for Model 3, it may cause vanishing gradient, and Adam optimizer solved this problem. In our models, we compute the decaying averages of past and past squared gradients m_t and v_t respectively as follows (we set the parameter $\beta_1 = 0.9$ and $\beta_2 = 0.999$.
Loss function. We compare the predicted location with the label and calculate a cost function using the mean square error (mean of total Euclidean distance):

$$Loss = (x - \hat{x})^2 + (y - \hat{y})^2 + (z - \hat{z})^2$$

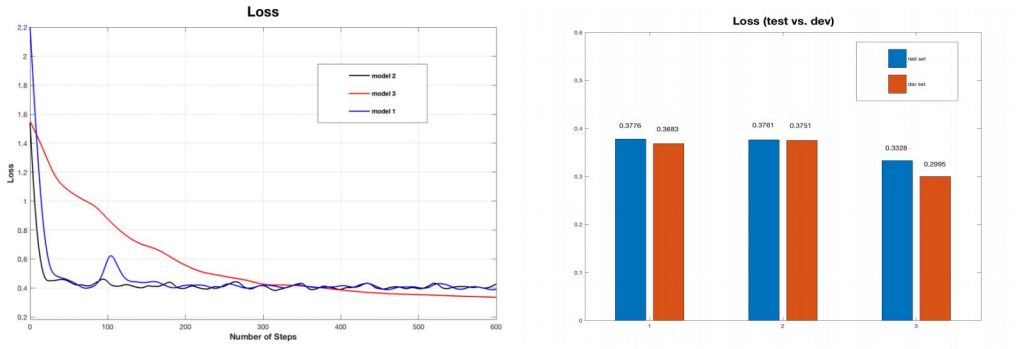
Regularization. The dropout method is applied as regularization techniques for our three models, which can reduce overfitting in neural networks by preventing complex co-adaptations on training data [11]. In this training process, drop probability is equal to 0.5, which maximizes the number of randomly-generated network structures, and prevents the neural network from being over-dependent on a specific set of neurons.

B. Result and Evaluation

Error Analysis. Our evaluation metric is the Euclidean distance between the predicted location and ground truth location:

$$D = \sqrt{(x - \hat{x})^2 + (y - \hat{y})^2 + (z - \hat{z})^2}$$

After running the model for 20 epochs, we plot the training loss(see Fig 7). We fitted the curve to smooth out the noise, and we can see how training error goes down for all three models. Training loss of Model 3 decreases the slowest but ends up in the lowest training error.
Both Model 1 and Model 2 converged after 1 2 epochs. The fast convergence is expected since the main part of both models are VGGNet with fixed weights. In both models, the neural networks need to learn very straight forward parameters for FC layers and input layers. Model 3, however, needs to merge two VGGNets, therefore introducing more complexity especially when dropout layers are used. On the good side, a more complex neural network can always capture more information. For Model 1, a single convolutional layer was used to convert the depth information, which roughly adds the depth and RGB information together, and hence the error is relatively large. In Model 2, even though the design of 3D-space voxel is reasonable, depth information was quantized into 10 intervals and large quantization might have been introduced. At the end of the day, Model 3 achieves the best performance, which coincides with our intuition. The color-mapping strategy converts the depth images into pseudo-RGB images, so that the network can capture more depth information. Pre-trained VGG can then extract basic features from these RGB images very well. To analyze further, while the result of Model 2 for regression strategy did not perform the best, similar voxel strategy in [2] achieved very high-accuracy in classification problems. We believe that this strategy would work better if we had pixel-wise labels so that we can use softmax instead of linear regression. We also plotted the test set vs. dev set error. The dev set error seems to be very close to the training set error. However, we do suffer from having a very small dev/test set, which created a relatively small difference in our results. Our resulting error of <0.3 indicates that the model works well, since the view of the camera spans roughly from -3 to 3.



(a) (b)

Figure 6: (a). Training loss for different models (blue: model 1, red: model 2, black: model 3). (b). Test loss vs. dev loss (blue: test set, red: dev set).

Attention Map. Keras-Vis is used to visualize layers inside Neural Network. We plotted the attention map through backward propagation from output layer to input layer of Model 2 (Voxel Model), as shown in Figure 8. Since Model 2 input was a 10-grid segmentation of the depth image incorporated with RGB image, the attention map we got was also a 10-layer attention map. We can roughly see the shape of the items and the table from all 10 layers. Moreover, the closer the object is, the more clear the edges will be, and thus the larger the depth value will be returned. In Figure 8, it is expected that the right most (closest) layer is brighter, and sees more information because it is the closest layer. In this case, we can be sure that the depth information was seen and understood by our model. Unfortunately, due to a bug of Keras-Vis, we cannot visualize the layers for parallel model.

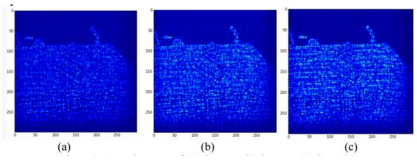


Figure 7: Attention map of (a). layer 1, (b). layer 5, (c). layer 10.

6 Conclusion/Future Work

Since we do not have pixel-wise training data, we built our model training on object-center-labeled data. we can make roughly correct predictions by incorporating depth information as a different colored image. Achieving average euclidean distance of 0.3 out of a view scope of [-3,3], our model works with little error, and should be good enough for a robotic hand to find the rough direction to go. Not having a perfect result is okay since the robot can capture more close-up images and perform on-the-fly calculations when actually carrying out a grasping task.
In order to make our predictions more accurate, however, we will need either pixel-wise training data and change our model into Yolo, or simply have more labelled data to train on. Besides that, we can now try the third model we proposed on a real robotic system, and see if the localization can still perform well. If not, we might need to build another model for simulation-to-real conversion.

7 Contributions

Simon Kalouche from Stanford AI lab and Deepdroid provided the dataset.
We worked together on literature review, model selection, and studying all the details about VGG model. We designed the 3 models together and outside of teamwork, Holly did report documentations, AWS setup and model training, Yuwen did output data analysis, graph generation and poster documentation, and Zixuan worked on data preprocessing and model training, as well as GCP/AWS setup.

GitHub repo: $https://github.com/hollylxj/cs230_localization.git$

References

[1] Karen Simonyan, Andrew Zisserman. (2014) "Very Deep Convolutional Networks for Large-Scale Image Recognition." *Computer Science.*
[2] Saman Zia, Buket Yuksel, Deniz Yuret et al. "RGB-D Object Recognition Using Deep Convolutional Neural Networks. (2017)" *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*
[3] Tobin, Josh, et al. (2017) "Domain randomization for transferring deep neural networks from simulation to the real world." Intelligent Robots and Systems (IROS), *IEEE/RSJ International Conference on. IEEE.*
[4] Redmon, Joseph, Anelia Angelova. (2015) "Real-time grasp detection using convolutional neural networks." *Robotics and Automation (ICRA), 2015 IEEE International Conference on. IEEE.*
[5] Gupta, Saurabh, et al. (2014) "Learning Rich Features from RGB-D Images for Object Detection and Segmentation: Supplementary Material.".
[6] He, Ruotao, Juan Rojas, and Yisheng Guan. (2017) "A 3D Object Detection and Pose Estimation Pipeline Using RGB-D Images." *arXiv preprint arXiv:1703.03940.*
[7] Hazirbas, Caner, et al. (2016) "Fusenet: Incorporating depth into semantic segmentation via fusion-based cnn architecture." *Asian Conference on Computer Vision. Springer, Cham.*
[8] Schwarz, Max, et al. (2016) "RGB-D object detection and semantic segmentation for autonomous manipulation in clutter." *The International Journal of Robotics Research.*
[9] Firman, Michael. (2016) "RGBD datasets: Past, present and future." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.*
[10] Eitel, Andreas, et al. (2015) "Multimodal deep learning for robust RGB-D object recognition." *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE.*
[11] Kingma, Diederik P., and Jimmy Ba.(2014) "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980.*
[12] Hinton, Geoffrey E., et al.(2012) “Improving nerural networks by preventing co-adaptation of feature detectors.” *arXiv preprint arXiv:1207.0580.*
[13] Gupta, Saurabh, Judy Hoffman, and Jitendra Malik.(2016) "Cross modal distillation for supervision

transfer."*Computer Vision and Pattern Recognition (CVPR). IEEE.*
[14] Porzi, Lorenzo, et al.(2017) "Depth-aware convolutional neural networks for accurate 3D pose estimation in RGB-D images." *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on. IEEE, 2017.*