CS172 Computer Vision I
Homework4: Depth Neural Network

WU Daqian
2018533262
wudq1@shagnahitech.edu.cn

1. Introduction

In this homework, I constructed and trained a neural network to perform depth estimation on images. This paper follows the work of NYU Depth[1] and is also tested on the NYU Depth dataset[1].

2. Approach

The original network is made of two component stacks. A coarse-scale network first predicts the depth of the scene at a global level. This is then refined within local regions by a fine-scale network. Both network are applied on the original input with the output of the first network concatinated on the first layer of the fine-scale network so that he local network can edit the global prediction to incorporate fine-scale details.[1] However, in this homework, my approach would be simplier and would only consider the first coarse-scale network and try to enhance training.

2.1. Net Structure

The original coarse-scale network uses five layers of convolution to extract features and two fully connected layer to generate the ouput coarse depth. This is extremely similar to AlexNet which has five feature extraction layer and three fully connected layer as classification. Therefore, the backbone chosen for this hoemwork is AlexNet, on which slight adjustment is made to fit the input-output scale. And dropout layers are also reconsidered due to overfitting problem.

Here is the structure:

- Convolution layer 3 input and 64 output channel with kernel size of 11×11, stride 4 and padding 2
- Relu acvtivation
- Maxpooling of 3×3 and stride 2
- Convolution layer 64 input and 192 output channel with kernel size of 5×5, stride 2 and padding 2
- Relu acvtivation
- Maxpooling of 3×3 and stride 2
- Convolution layer 192 input and 384 output channel with kernel size of 3×3, stride 1 and padding 1
- Relu acvtivation
- Convolution layer 384 input and 256 output channel with kernel size of 3×3, stride 1 and padding 1
- Relu acvtivation
- Convolution layer 256 input and 256 output channel with kernel size of 3×3, stride 1 and padding 1
- Relu acvtivation
- Adaptive Average Pool with output of (6,8)
- Flattern layer
- Dropout
- Fully connected $(256 * 6 * 8)\times4096$
- Relu acvtivation
- Dropout
- Fully connected 4096×4096
- Relu acvtivation
- Fully connected $4096\times(74*55)$

3. Data

The Data used to train and test the network is the NYUDepth[1]. The NYU Depth dataset is composed of 464 indoor scenes, taken as video sequences using a Microsoft Kinect camera as in Figure1. The raw data needs alignment and masking the missing depth, whats more, areas with infinte depth such as outdoor and windows also needs masking. Due to these reasons and that the raw dataset is too large(400G+) to compute on personal computers, the network is only trained on the 1449 label data provided in NYUDepth[1].

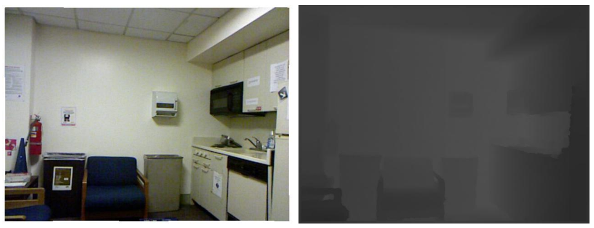
Figure 1. Image on the left is the input which is single view of a scene and image on the left is the target image, which originally in depth, and obtained by normalizing on depth and scaling by 255

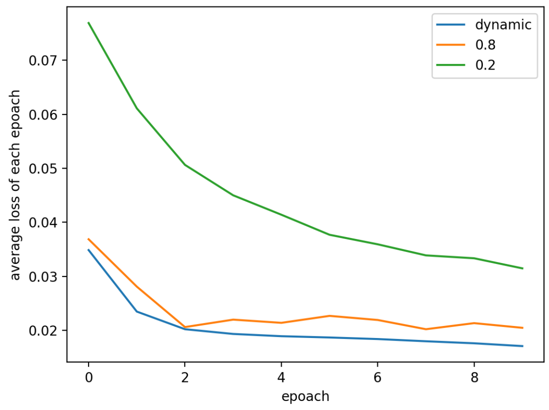

Figure 2. Average loss of each epoch

3.1. Augmentation

Using labled data will cause problem of its own. The 1449 images is clearly not enough to even train the coarse-scale network, I can only try to imporve on the data and make augmentations. The easy approach I took is to filp the image left to right and upside down and also do the same opertation on the target depth image.

3.2. Training

I tried three different traings with, most importantly, changing and tuning the leraning rate. I trained the network with: i) fixed learing rate of 0.2, ii) fixed learning rate of 0.8, iii)dynamic learning rate droping from 0.8 to 0.08 in ten epochs.

The average loss for each training is as Figure2 shows.

We can see that when the learning rate is small(0.2), the average loss is droping but get too slow in the end due to lack of dataset, and when the learning rate is large(0.8), the loss drops quickly but bounces fluctuently and could not continue to drop. As for the dynamic loss rate, it is pretty leasing but the loss is still too larege due to lack of dataset.

4. Results

As in Figure3 the results are pretty poor. We can see a

Figure 3. From left to right: lr = 0.2, lr = 0.8, dynamic

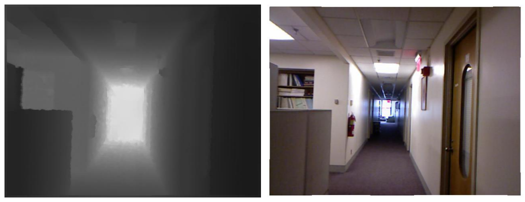
Figure 4. Input and output for section **4.1**

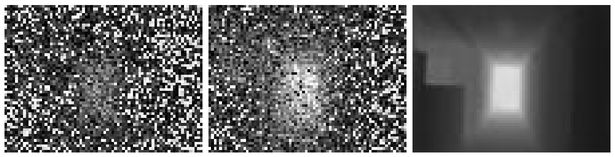
Figure 5. From left to right: reapating 100, 300, 1000 times

slightly smoothing effect from the change of learning rate, but still far from good enough. What's interesting is that the three images are quite similar, this has me thinking, what did the network learn? My theroy is that the lower part of the image is usually closer to the camera and upper part is further. This is actually true, since lower part is often the floor and other closer object and upper part is usually further in sight.

4.1. A test on one picture

To see if the net work can actually work, and for fun, I tested the network by train on a single pair of image and target. The results are as shown in Figure5. It is pleasing to see that the network is actually working.

5. Vishualize

5.1. Point Clound Reconstruction

If my network could produce a relative good result of depth estimation, we can then use this depth information together with the original input rgb picture to form a point cloud. Here are some examples(Figure6-9) of point cloud using given data in NYU Depth[1].The package used is open3d Python api[2].

6. Code

- **images/**: rgb images for input
- **idepths/**: grayscale images as target
- **output/**: used to save the output of the net

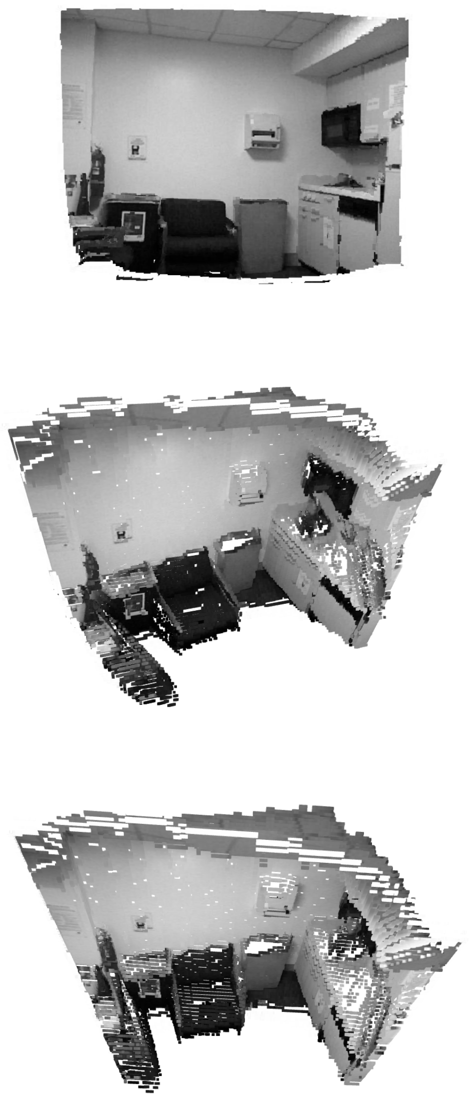

Figure 6. Point Clound Reconstruction 1,result

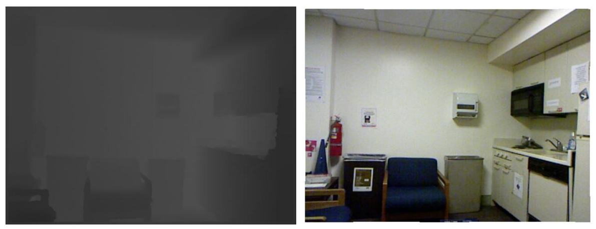

Figure 7. Point Clound Reconstruction 1 rgb and depth

- **AlexNet-dynamic.pth**: saved weights of the network parameters from dynamic learning rate training

- **AlexNet.py**: the network

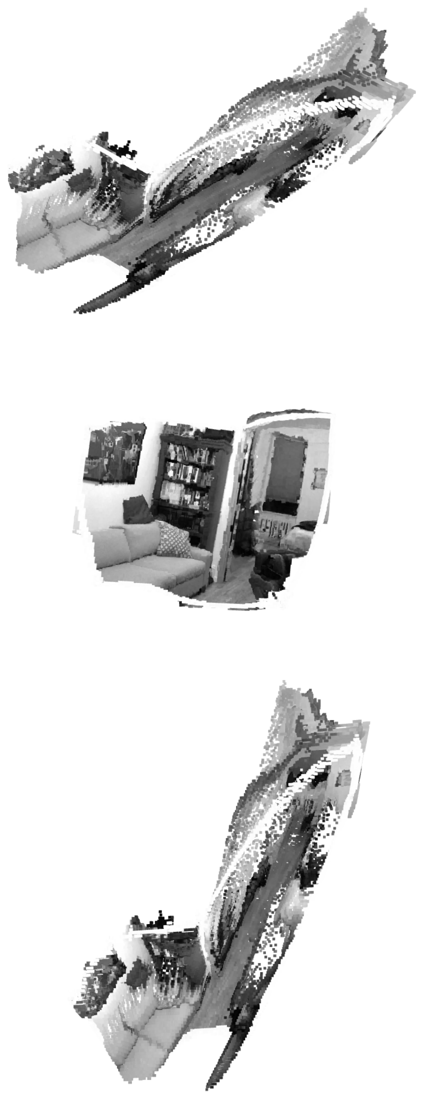

Figure 8. Point Clound Reconstruction 2,result

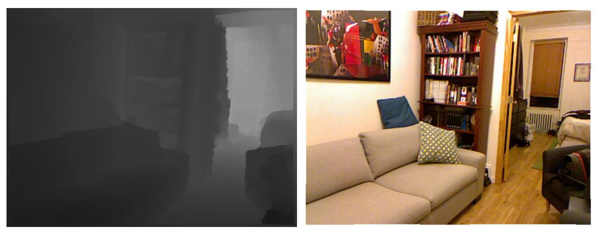

Figure 9. Point Clound Reconstruction 2 rgb and depth

- **train.py**: training process of the network

- **out.py**: using the save parameters to get ouput

- **utils.py**: uitility functions for reading, saving, showing and preprocessing images

- **pointcloud/**: code and input for point cloud generation

References

[1] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[2] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.