CS172 Computer Vision I:
Single Image Depth Estimation

Yang Hongdi
2019533234
yanghd@shanghaitech.edu.cn

Abstract

This project reproduces the depth estimation ConvNet propsed in Revisiting Single Image Depth Estimation: Toward Higher Resolution Maps with Accurate Object Boundaries(CVPR 2018) by using ResNet18 and L1 loss.

1. Introduction

In the paper, the depth estimation ConvNet consists of four modules: an encoder (E), a decoder (D), a multi-scale feature fusion module (MFF), and a refinement module (R). In the paper, ResNet50 is used for encoder, for simplicity and time concern, ResNet18 is used for encoder in this project. In section 3.2, the paper propses some l2 loss functions, for simplicity, l1 loss function is directly used in this project.
To build point cloud, Open3d library in python is used.

2. Algorithm Description

2.1. ConvNet Structure

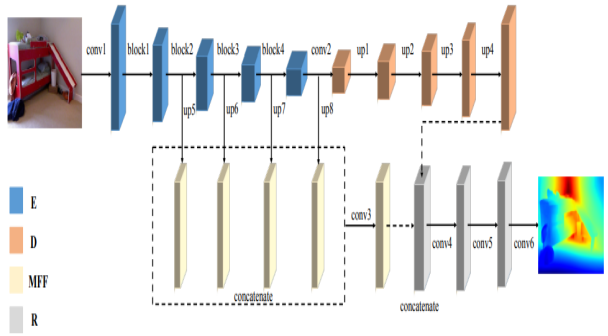


Figure 1. ConvNet Structure

As the above image shows, the ConvNet contains 4 modules. To fit the size of output, the input would be a RGB image with 304 x 228 pixels, and the output will be a single channel depth image with 152 x 114 pixels.
Each module will be introduced in the following sections.

2.2. Encoder

In this project, encoder is an ResNet18 with a little change.

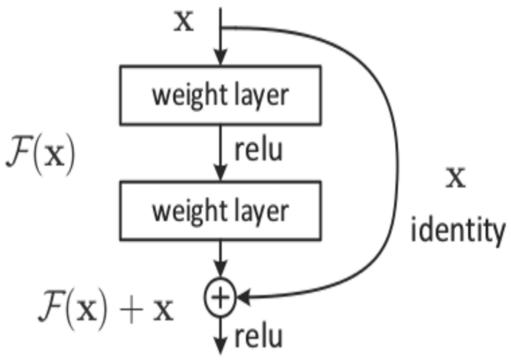


Figure 2. ResNet Block

ResNet consists of blocks like above image shows. For each block, there are two weight layer for convolution, the result $F(x)$ will plus the original x if the input size matches the output size. And if they do not match, input will be downsampled to match the size of output. As the following image shows.

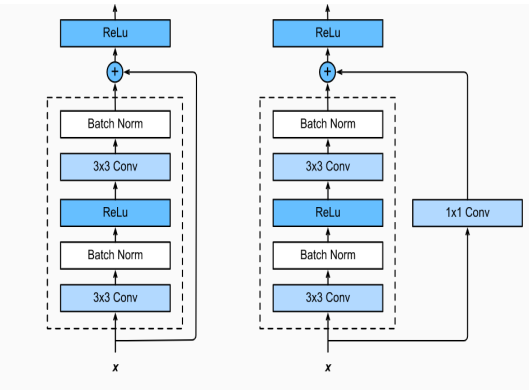


Figure 3. ResNet Implementation

So ResNet18 is a ConvNet consists of 4 of these blocks, along with some pooling and finally a fc layer. To use ResNet18 as an encoder, we do not need the average pool and fc layer at last. The architechture of ResNet18 is shown as below.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Figure 4. ResNet18 Architecture

2.3. Decoder

For the decoder, it converts the last 1/32 scale feature to get a 1/2 scale feature, so basically, it consists of one convolution layer and 4 upsampling layer. In each upsampling layer, the channels would reduce to 1/2 scale while the size would increase to double scale.

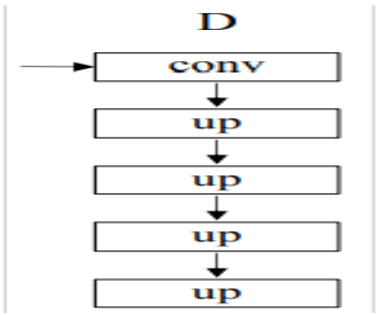


Figure 5. Decoder Structure

Table 1. Decoder: Sizes of output features, and input/output channels of each layer using ResNet18

Layer	Output Size	Input\C	Output\C
conv2	10 x 8	512	256
up1	19 x 15	256	128
up2	38 x 29	128	64
up3	76 x 57	64	32
up4	152 x 114	32	16

The decoder module will output a 152 x 114 16 channel image, and it will be concatenated with the output of MFF module.

2.4. Multi-scale Feature Fusion Module

In MFF module, it upsamples features extracted from Encoder module to be size of 152 x 114 and then fuse them together.

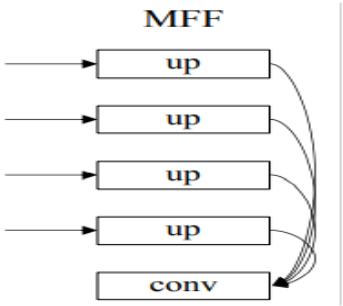


Figure 6. MFF Structure

In each upsampling layer, the output channel will be 16. For the last convolution layer, the output channel would be 4 x 16 = 64.

Table 2. MFF: Sizes of output features, and input/output channels of each layer using ResNet18

Layer	Output Size	Input\C	Output\C
up5	152 x 114	64	16
up6	152 x 114	128	16
up7	152 x 114	256	16
up8	152 x 114	512	16
conv3	152 x 114	64	64

The MFF module will output a 152 x 114 64 channel image, and it will be concatenated with the output of Decoder module.

2.5. Refinement

In the refinement module, the outputs of D and MFF are first concatenated, then refined by three convolution layers.

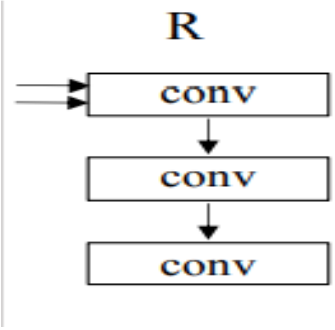


Figure 7. Refinement Structure

The input will be a 152 x 114 image with 16 + 64 = 80 channels. After 3 convolution layers, it will get a final output of size 152 x 114 and one channel of depth estimation, which would be the final prediction result.

Table 3. Refinement: Sizes of output features, and input/output channels of each layer using ResNet18

Layer	Output Size	Input\C	Output\C
conv4	152 x 114	80	80
conv5	152 x 114	80	80
conv6	152 x 114	80	1

2.6. Point Cloud

For point cloud construction, some basic functions of Open3d is used. To build point cloud, for each pixel $p = (u, v)$, we need to compute its coordinate in camera coordinate system, i.e $P = (X, Y, Z)$, as we know

$$Zp = Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = KP$$

and we know $Z = d$,by solving the equation, we can get

$$X = Z(u - c_x)/f_x$$
$$Y = Z(v - c_x)/f_y$$

by using the 3d coordinates, we can construct point cloud.

3. How to run code

1. download nyu dataset
2. (optional) if to use pretrained ResNet18, download it from https://download.pytorch.org/models/resnet18-5c106cde.pth and put it in the same folder with train.py
3. python train.py
4. after training is done, run test with command python test.py
5. To build point cloud, run point cloud with python pointcloud.py

4. Result Analysis

4.1. Training Loss

In this project, we use Adam optimizer with an initial learning rate of 0.0001, and reduce it to 10% for every 5 epochs. torch.nn.L1Loss() is directly used as loss function. A total num of 20 epochs are trained, in eaech epoch, the batch size is 8. The loss in each epoch is shown as below.

Table 4. every epoch's Loss

epoch	loss	epoch	loss
0	0.136	10	0.0698
1	0.113	11	0.0686
2	0.107	12	0.0684
3	0.101	13	0.0689
4	0.0960	14	0.0678
5	0.0847	15	0.0675
6	0.0816	16	0.0682
7	0.0780	17	0.0681
8	0.0753	18	0.0681
9	0.0734	19	0.0679

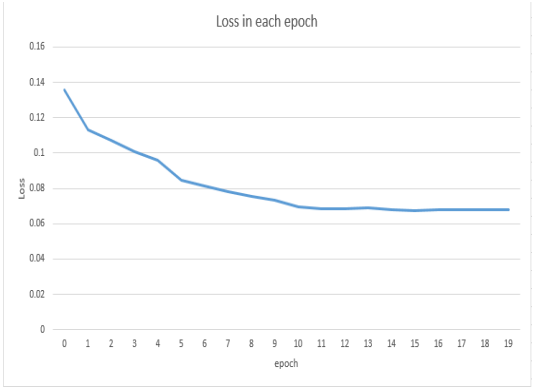


Figure 8. Loss in each epoch

We can see that at first the loss decreases as the epoch increases, and after about 12 epoch, the loss become stable.

4.2. Depth estimation Measures

To measure our depth estimation result, we use mean relative error (REL) and Mean log 10 error(log10).

- Mean relative error (REL) : $\frac{1}{T}\sum_{i=1}^{T}\frac{||d_i - g_i||_1}{g_i}$
- Mean log 10 error (log10) : $\frac{1}{T}\sum_{i=1}^{T}||log_{10}d_i - log_{10}g_i||_1$

After testing with nyu dataset, we get the REL and log10 as :

$$\text{REL} = 0.265, \log10 = 0.109$$

For comparison, the original paper using ResNet50 get the REL and log10 as :

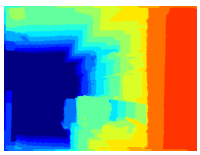
$$\text{REL} = 0.126, \log10 = 0.054$$

So the ConvNet in this project reaches about half performance of original paper.

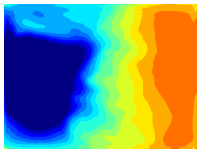
4.3. Depth Estimation Result

We use heat map to represent the result of our prediction. The cool-toned color represent deeper area and the warm-toned color present shallower area.

(a) original image



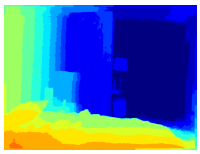
(b) ground truth



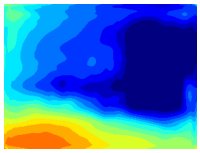
(c) our result

Figure 9. Depth prediction result

(a) original image



(b) ground truth



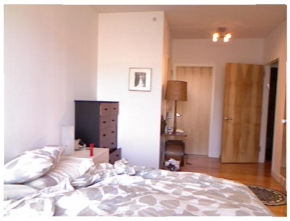
(c) our result

Figure 10. Depth prediction result

We can see that though we can't predict every object accurately, but relatively we can correctly predict the deeper area and the shallower area.

4.4. Point Cloud

Here we show the result of point cloud construction.



(a) original image

(b) ground truth

(c) our result

(d) original image

(e) ground truth

(f) our result

Figure 11. Point cloud Result

In our point cloud, there are some objects become really weird, but for the whole scene, the depth is relatively correctly predicted.

5. Important Pixels

Figure 12. A sofa image

Take this image as an example, the sofa's depth changes greatly. And for the sofa, we can detect its edges. So we can just predict its depth by applying functions on its edges. As the edge is a line in 2d coordinate system, it can make computation easier. Also, the depth would rather change along an edge or remain the same with an edge.
So to correctly predict how the depth change with edge can have the most impact on our prediction result, which make the pixels along the dege become the most important pixels.

6. References

1. https://iq.opengenus.org/residual-neural-networks/