EECS 442 Project Report: Monocular Image Depth Prediction via Encoder-Decoder-based Deep Learning

Huijie Zhang, Yiting Zhang, Xiaoyang Qian
University of Michigan
{huijiezh,yitzhang,braxq}@umich.edu

1. Introduction

Computer vision is a popular subject nowadays. One of the crucial subdivision tasks it is trying to solve is Depth prediction, which is what we are doing in this project. For depth prediction, essentially what we are doing is: input a 2-D RGB image and output an estimation of depth prediction for every pixel on that image with respect to the position of the camera plane. In the ideal situation, having multiple images of the same scene from different perspectives help constrain the problem; the problem becomes more defined through finding local correspondences. When this problem is not constrained: in other words, having only single image from a specific view point, it can be very ambiguous to solve this problem due to the global and local uncertainties. However, solving depth prediction helps understanding the 3D geometry relationships in a scene. Additionally, the results from depth prediction could be used to help other applications in CS, robotics, etc. For example: 3D-rendering and self-driving cars.

The major task we were trying to accomplish was solving monocular depth prediction via deep learning in two different ways: DenseNet and MiniNet based Deep Neural networks. The problem is solvable because we iteratively run the models and try to minimize the loss between the predicted depth map and ground-truth depth map by improving the gigantic set of model weights. As mentioned above, our work is basically inspired by papers which introduce DenseNet[5] and Mininet[6].

2. Approach

2.1. DenseNet based Deep Neural Network

Regarding the model architecture: our DenseNet-based deep neural network is composed of a DenseNet encoder and bilinear upsampling decoder. Fig. 2 explains the skeletons of the encoder we use. This structure may not seem too novel, because the essence of this model is in each Dense block where the output map from each layer (1x1 Conv and 3x3 Conv in the table) is forwarded and concatenated, so that more features could be learned. A more illustrative way could be a plot on the same paper of Densely Connected Convolutional Networks: what this graph means mathematically is given an input x and series of functions("Conv" or Dense layers). Suppose the functions are $f_1, f_2, f_3...$, then passing these three functions would yield a result in the form: $[x, f_1(x), f_2(x, f_1(x)), f_3(x, f_1(x), f_2(x, f_1(x))]$. This mathematical form is intuitive in showing how the feature

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48

Figure 1. **Architecture of DenseNet for imagenet.** Note that the "conv" in the table is equivalent to a set of BN-RELU-Conv.[5]

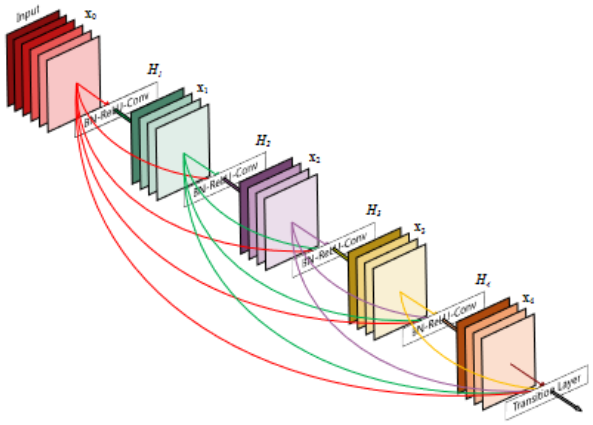

Figure 2. **Detailed illustration of modules in the Dense Block.**[5]

outputs from the previous functions helps training in the structure of DenseNet. After downsampling the input RGB image using the encoder, features after each dense block is passed onto the decoder matching the correct shape, which is very similar to the feature concatenation in U-net[9]. Then this becomes a chain of: concatenation, up convolution, and bilinear upsampling with leaky ReLU inserted as the activation function in the decoder part. In the end, additional bilinear upsampling and convolutions are deployed to match the input RGB shape (Height and width).

2.2. MiniNet based Deep Neural Network

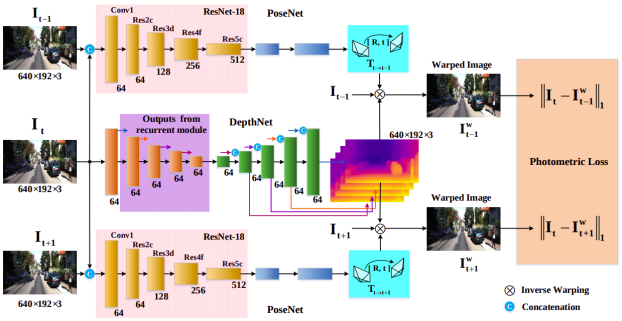

Figure 3. **Architecture of MiniNet, that is composed of a DepthNet and two shared-weight PoseNets.** The DepthNet only takes a single still image as input, while the two shared-weight PoseNets take the frame pair as input. The value below each feature map rectangle denotes the channel number, and the smaller height rectangle is the half size of the preceding one[6].

The MiniNet is based on the work of J. Liu et al. that is composed of a DepthNet and two shared-weight PoseNets as illustrated in Fig 3. The DepthNet takes the target image to predict the depth map, while the PoseNets take the adjacent two frames for camera ego-motion estimation. In the training stage, three consecutive video frames I_{t-1}, I_t and I_{t+1} are fed into the MiniNet, where the middle frame I_t is marked as the target image and the rest are source images. While in the inference stage, only the DepthNet is remained for single image depth prediction. In this project, we first implement a DepthNet and test the prediction performance when using the DepthNet alone. Then, we add up the PoseNet to the DepthNet and compare the result with the one without using the PoseNet.

2.2.1 Structure of the DepthNet

The structure of the DepthNet is illustrated in Fig. 4 that consists of a recurrent module-based encoder and an upsample blockbased decoder.

The encoder part consists of a standard convolutional layer and a recurrent module. The first layer is a standard

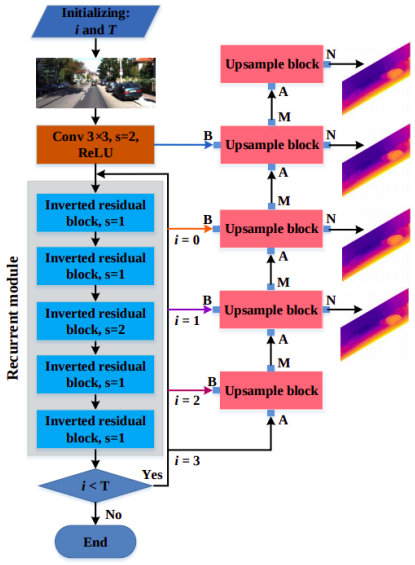

Figure 4. **Schematic diagram of the DepthNet in the MiniNet.** The output feature maps from the first convolutional layer and the recurrent module will be skip-connected to the corresponding upsample blocks in the manner of concatenation. The DepthNet iteratively uses the recurrent module to generate multi-scale feature maps. i and T indicate the iteration time and total iteration number, respectively. s denotes the stride number of convolutional layer. The multi-scale disparity predictions will be bilinearly upsampled to the same spatial resolution of the input RGB image[6].

3×3 convolution with a stride of 2 followed by ReLU activation. The output channel number c of the first layer is empirically set to 64. The recurrent module is built upon the inverted residual block of MobileNetV3[3], which is composed of five inverted residual blocks, where the middle block has the stride of 2 and the rest have that of 1. Thus, the size of features will be halved via each iteration of the recurrent module. Each inverted residual block is composed of a pointwise convolution with ReLU6, a depthwise (Dwise) convolution with ReLU6 and the stride of 1 or 2, a Squeeze-and-Excitation (SE) block[4] and a pointwise linear convolution.

The decoder part is based on several up-sample blocks. The up-sample block is composed of three residual DSconv blocks[7], Nearest-upsample, concatenation, and sigmoid operations. These residual DSconv blocks are plug-in replacement of the standard convolutions, which consist of depth-wise and point-wise convolutions with the shortcut connection between the input and output[7].

2.2.2 Structure of the PoseNet

The architecture of PoseNet is shown in Figure 3. We use ResNet as the backbone, following by a 1×1 convolution

with 6 output channels, corresponding to 3 Euler angles and 3-D translation to represent the transformation matrix $T_{t-1,t}$. Finally, a global average polling is applied to aggregate predictions among pixels. Based on the camera intrinsic matrix I, we could project the target image's depth D into a 3D point cloud. After applying the transformation matrix $T_{t-1,t}$ and projecting them back to the pixel space, we could get a color prediction about the source image.

2.3. Optimizer

The training of this network utilizes Adam optimizer with weight decay equal to 1×10^{-5} and learning rate set to 5×10^{-4}. This is due to the fact that Adam converges faster and usually works well with default parameters. However, while Adam plays a major role in this part of training, SGD optimizer is tested on its training performance as a minor character too.

2.4. Loss Function

The idea of training is using the loss between the depth map output and the ground-truth to optimize, with the ideal situation to find the global minimum in the loss contours. In this project, three different loss functions: mean-square-error(MSE) loss, L1 loss and self-defined scale-invariant loss function based on paper[1] are applied for training the model. Particularly, scale invariant loss is illustrated as following: $L(y, y^*) = \frac{1}{n}\sum_i d_i^2 - \frac{\lambda}{n^2}(\sum_i d_i)^2$ where $d_i = log(y_i) - log(y_i^*)$, y_i and y_i^* are pixel values in two images and $\lambda \in [0, 1]$[1], this loss balances between the L2 loss and the exact scale-invariant loss, which is ideal for this scenario.

3. Experiments

3.1. Computational resources

In this depth prediction project, we all used google colab and pytorch framework to do the works. With the aid of colab Pro, we obtained the access of more powerful GPU, more workers and larger memory to store bigger data sets; colab Pro helped us boost our training speed and find problems faster.

3.2. Metrics

In this project, we consider to evaluate the performance of the model from two aspect, error and accuracy respectively.

For the error part, we would follow the standard evaluation tools like relative absolute error (Abs Rel), relative squared error (Sq Rel), root mean squared error (RMSE) and root mean squared log error (RMSLE)[10]. The following is the list of all metrics we would deploy to evaluate the performance:

Abs Rel: $\frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{d}_i - d_i|}{d_i}$

Sq Rel: $\frac{1}{N}\sum_{i=1}^{N}\frac{||\hat{d}_i - d_i||^2}{d_i}$

RMSE: $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{d}_i - d_i)^2}$

RMSLE: $\sqrt{\frac{1}{N}\sum_{i=1}^{N}||log_{10}(\hat{d}_i) - log_{10}(d_i)||^2}$

where $\hat{d}_i$ and d_i are predicted depth and ground-truth depth at pixel $i \in Test\_image\_j$ respectively; N is the total number of pixels in $Test\_image\_j$.

For the accuracy part, we evaluate it within a threshold: the percentage of $\hat{d}_i$ s.t. $\delta_j = \max\left(\frac{d_i}{\hat{d}_i}, \frac{\hat{d}_i}{d_i}\right) < 1.25^j$, where $j = 1, 2, 3$.

3.3. Qualitative results

Shown as Fig. 5, our model can catch details from the ground truth maps and generate similar results. The contour of objects is also clear in the prediction results.

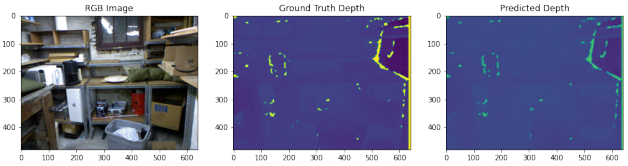

Figure 5. **Example of a depth map prediction result.**

3.4. Quantitative results

The quantitative results are shown in Table. 1, in which we compare the performance of different models and parameters. Here adaptive learning rate means that there exists a 1% - 4% decay on the learning rate after each epoch. Two scales of the dataset are used for experiment, the standard size dataset includes 3500 pairwise images and the large size dataset includes 50000 pairwise images.

From the results, it can be found that the DenseNet trained by MSE loss and adaptive learning rate for 100 epochs on the standard size of dataset has the best performance in both error and accuracy metrics.

3.5. Data

We use the NYU dataset [8] for our training and evaluation. It is a dataset containing some daily scene like bathroom, bedroom, living room and so on. The raw dataset contains the raw RGB and Depth images and accelerometer dumps from kinect.

4. Implementation

4.1. Data Preparation

We use around 50000 pairwise images as our data. Because the data from kinect is not synchronized, we should first align them following the timestamps. 50000 images would take a large amount of memory to store. Rather than

Table 1. Comparison of monocular depth prediction results on NYUv2 dataset, where adaptive learning rate means that there exists a 99% - 96% decay on the learning rate after each epoch. Two scales of the dataset are used for experiment, the standard size dataset includes 3500 pairwise images and the large size dataset includes 50000 pairwise images.

Structure	Loss	Epoch	Learning Rate	Dataset Size	Error (lower is better)				Accuracy (higher is better)		
					Abs Rel	Sq Rel	RMSE	RMSLE	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
DepthNet	MSE	100	constant	standard	0.039	0.012	0.125	0.040	0.968	0.985	0.996
DepthNet	MSE	100	adaptive	standard	0.039	0.012	0.125	0.039	0.969	0.986	0.996
DepthNet	Scale-Invariant	100	adaptive	standard	1.471	2.219	1.518	0.393	0.005	0.013	0.022
DepthNet	MSE	50	adaptive	standard	0.071	0.019	0.169	0.055	0.946	0.977	0.992
DepthNet	MSE	15	adaptive	large	0.091	0.038	0.240	0.078	0.906	0.943	0.959
DenseNet	MSE	15	constant	standard	0.050	0.017	0.147	0.049	0.947	0.981	0.995
DenseNet	MSE	22	constant	standard	0.050	0.015	0.141	0.047	0.949	0.981	0.996
DenseNet	MSE	45	adaptive	standard	0.032	0.011	0.120	0.038	0.966	0.988	0.997
DenseNet	MSE	100	adaptive	standard	**0.025**	**0.008**	**0.101**	**0.032**	**0.974**	**0.992**	**0.998**

load them together into RAM, we use pytorch dataset module to design a high performance, multi-process dataloader. The data is stored in the disk, and the dataloader would create multi-subprocess to accelerate data loading.

4.2. DenseNet based Deep Neural Network

The implementation of this part was built from scratch and was guided by the Densely Connected Convolutional Networks paper[5]. Since the structure of DenseNet[5] is so complex that it seems wise to make things module-wise. Here are the logistics of my net structure: firstly, there should be Dense Blocks[5] and Transition Blocks[5] that constitutes the bones of the net. Dense Blocks are consisted of many Dense layers of user's configuration. Since we are doing depth-prediction and for the sake of same output size, we removed the Classification layer that flattens the output, on the basis of this, we add upsampling encoder to work with this DenseNet based decoder. A scale-invariant loss was also implemented[1].

4.3. MiniNet based Deep Neural Network

The implementation of the MiniNet as well as the DenseNet is based on the work of J. Liu et al.[6]. The output depth map of their network is 1/4 of the original image size. In our work, we add two more upsample blocks in the decoder to let the output size match the ground truth depth map. This modification is benefit for generating more accurate depth map. Besides, it is also more convenient to compute error between the prediction and the ground truth.

5. Pitfalls

The first pitfall in the training process is the usage of scale-invariant loss. Despite this loss is nice in the way of combining the nice properties of two losses, we ignore a huge issue during training: the pixel values. During the start of training process, many of the pixel values in the predicted depth map may not be well defined yet, even missing values could present. Especially near object boundaries, windows and specular surfaces[1], this leads to our models not converging well, stuck at some point during training while using scale-invariant loss.

The quality of dataset could be another troublesome issue in training. When plotting out the ground truth depth maps for many images, there are many evident sections colored in yellow caused by reflection or edge problems. Learning these unwanted noises could be potentially harmful to our final network predictions.

Additionally, we naively contributed the reasons of our networks not performing good enough to the reason of smaller dataset (of around 3500 labelled images). Since larger sample size usually means lower variance among the distribution of sample, we suppose that better models could be learned. We then try to train with dozens of thousands of labelled data. But the performance, surprisingly, degrades so fast that everything seems to be much worse than previous results, in the perspective of either training time and accuracy. Based on the work of J. Hestness et al.[2], in order to reach frontier targets defined by ml experts, datasets will need to grow 33-971 times, while models will need to grow 6.6-456 times to achieve target accuracy. This gives us a hint that a more complex network is required for a larger size of dataset.

While implementing the PoseNet, we find it is hard to vectorize the total process, especially about the process of differentiable image wrapping from PyTorch, and a for-loop version would be a disaster for training. Finally, we only successfully implement a Numpy version . We think the result turns into CUDA programming, however there is not much time to do this.

References

[1] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *CoRR*, abs/1406.2283, 2014. 3, 4

[2] Joel Hestness, Newsha Ardalani, and Greg Diamos. Beyond human-level accuracy: Computational challenges in deep learning. *CoRR*, abs/1909.01736, 2019. 4

[3] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019. 2

[4] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2019. 2

[5] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018. 1, 4

[6] Jun Liu, Qing Li, Rui Cao, Wenming Tang, and Guoping Qiu. Mininet: An extremely lightweight convolutional neural network for real-time unsupervised monocular depth estimation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 166:255–267, 2020. 1, 2, 4

[7] Marcelo Gennari do Nascimento, Roger Fawcett, and Victor Adrian Prisacariu. Dsconv: efficient convolution operator. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5148–5157, 2019. 2

[8] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 3

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. 2

[10] Guanglei Yang, Hao Tang, Mingli Ding, Nicu Sebe, and Elisa Ricci. Transformers solve the limited receptive field for monocular depth prediction, 2021. 3