High Quality Monocular Depth Estimation Paper Re-implementation

David Wang
EECS,
University of Michigan
Ann Arbor, MI, USA
tawei@umich.edu

Yatharth Chhabra
EECS,
University of Michigan
Ann Arbor, MI, USA
ychhabra@umich.edu

Zareef Raiyan Safdar
EECS,
University of Michigan
Ann Arbor, MI, USA
szareef@umich.edu

1. Introduction

Depth prediction is a process whereby the distance of an object from a reference point is determined. Usually, the reference point is the camera plane because visual data is collected using video cameras. Once video footage is collected from sensors, it is stored as a sequence of static RGB images. Thus, depth prediction usually involves determining the distance of each pixel in an RGB image from the camera plane.

Depth prediction is an essential task in many application domains. In self driving cars, depth prediction can determine the distance of pedestrians to the car in order to avoid collisions. Additionally, robots need to predict depth in order to operate in a 3-dimensional world. Furthermore, depth prediction is a required component of many video games, particularly virtual and augmented reality. If technology is to replace the need for humans in certain fields, it needs to have similar sensory capabilities. Depth prediction is one such task that is performed by humans and needs to be emulated by machines. Computer vision techniques are well suited to solving the task of depth prediction because it is a form of image processing in which numerical labels are assigned to each pixel. Deep Learning based methods have been recently used due to it being close to real time speeds in processing RGB images to produce depth maps. Convolutional neural networks in specific are a common deep learning tool that can learn shared features across images, and are thus useful for generating depth map images.

We looked at two papers [1], which we will be referring to as the Alhashim-Wonka paper in this report. We mainly tried to implement the Alhashim-Wonka paper, which used the encoder-decoder structure to perform depth prediction.

In this report, we use an encoder-decoder machine learning model to perform depth prediction. We use the Densenet-169 model pretrained on the ImageNet dataset in conjunction with an upscaling decoder trained on the NYUv2 dataset to generate depth heatmaps. The depth maps are evaluated using both qualitative and quantitative test metrics to determine model performance.

2. Approach

In this section, we describe the architecture of the encoder-decoder model and specify the training and testing details. We split our approach into three sections: Model Architecture, Training and Loss and Data Augmentation.

2.1. Model Architecture

We used the encoder-decoder architecture, as shown in Figure 1 below, from the Alhashim-Wonka paper [1]. It uses transfer learning for the encoder in which features are learned from a larger dataset and then transferred to the depth prediction task. An RGB image is inputted to the encoder, which encodes a feature vector that is upscaled by the decoder. The encoder is a DenseNet-169 model as shown in Figure 2 below from PyTorch pretrained on the ImageNet dataset. The last fully connected layer of the encoder is removed to allow the learned features to be transferred to the decoder. We use the first four blocks of the Densenet-169 model, from which we also extract some layers for the decoder. The decoder is a series of layers which involve upsampling, concatenating with some of the layer outputs from the encoder and convolution layers. The specific implementation details for both encoder and decoder are reported in Tab. 1.

A change that we made from the Alhashim-Wonka paper is that we decreased the input sizes by 4, which we adapted by downsampling the input before entering the encoder and adding some since due to our limited hardware resources, it got harder to train. Also we had to change our loss function to reduce the size of the ground truth maps.

Table 1. Model Architecture, reused and modified Table 5 from [1]

Layer	Output (c,h,w)	Function
Input	3, 480, 640	
Downsampling *	3, 120, 160	MaxPool2D 4x4 to decrease size of input
CONV1	64, 60, 80	DenseNet CONV1
POOL1	64, 30, 40	DenseNet POOL1
POOL2	128, 15, 20	DenseNet POOL2

POOL3	256, 7, 10	DenseNet POOL3
...	...	...
CONV2	1664, 3, 5	Convolution 1 × 1of DenseNet BLOCK4
UP1	1664, 6, 10	Upsample 2 x 2
PAD1 *	1664, 7, 10	Padding the top by 1 with 0s to match original dimensions
CONCAT1	1920, 7, 10	Concatenate POOL3
UP1-CONVA	832, 7, 10	Convolution 3x3
UP1-CONVB	832, 7, 10	Convolution 3x3
UP2	832, 14, 20	Upsample 2 x 2
PAD2 *	832, 15, 20	Padding the top by 1 with 0s to match original dimensions
CONCAT2	960, 15, 20	Concatenate POOL3
UP2-CONVA	416, 15, 20	Convolution 3x3
UP2-CONVB	416, 15, 20	Convolution 3x3
UP3	416, 30, 40	Upsample 2 x 2
CONCAT3	480, 30, 40	Concatenate POOL3
UP3-CONVA	208, 30, 40	Convolution 3x3
UP3-CONVB	208, 30, 40	Convolution 3x3
UP4	208, 60, 80	Upsample 2 x 2
CONCAT4	272, 60, 80	Concatenate POOL3
UP4-CONVA	104, 60, 80	Convolution 3x3
UP4-CONVB	104, 60, 80	Convolution 3x3
CONV3	1, 60, 80	Convolution 3x3

* Not used in the non-downsampled model.

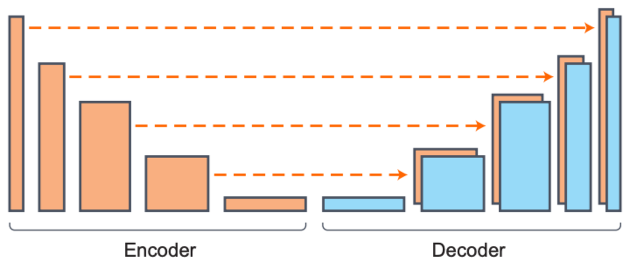

Figure 1. Model Architecture, reused Figure 2 from [1]

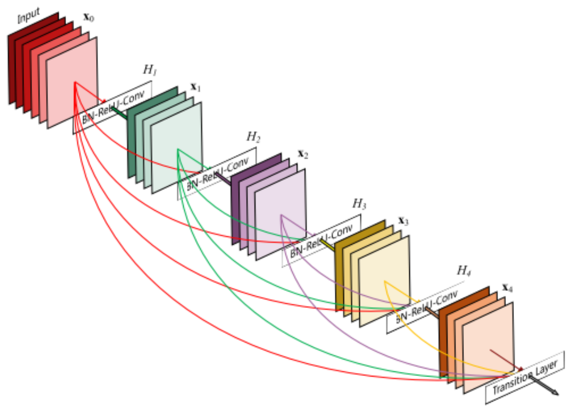

Figure 2. DenseNet 169 Architecture [2]

2.2. Training and Loss

For training, we used the AdamW optimizer with a learning rate of 0.0001 and a batch size of 8 images of size 480 x 320 pixels. We tried to understand whether our model was learning properly based on a running loss over 1159 training examples and after more than 30 hours of training we ended up at a loss of 39.931 after 236 epochs as described by the loss function above. We looked at the paper implementation which reached a good set of results after a million iterations but that was infeasible for us.

The loss function we used is identical to the one in the Alhashim-Wonka paper, which has three parts - a Depth loss, a Gradient loss, and SSIM Loss. The Depth Loss is an L1 Loss based on the predicted and labelled depth map values. The Gradient Loss is an L1 Loss based on the x and y gradients of the predicted and the labelled depth map values. The SSIM Loss uses the structural similarity metric between the predicted and the labelled depth map values.

2.3. Data Augmentation

We also followed Alhashim-Wonka's practice in data augmentation [1]. According to the paper, it could reduce overfitting and lead to better generalization performance [1]. We horizontally flipped 50% of the images in our dataset. We also switched the RGB channels for 25% of the images.

3. Experiments

3.1. Data

We used the NYU Depth v2 labeled dataset - particularly due to the ease of reproducing the results in the Alhashim-Wonka paper.

The NYU Depth v2 labeled dataset consisted of 1449 indoor images and depth maps. Each image is a 3-channel matrix of size 3 x 480 x 640 pixels. The value for each pixel is an integer between 0 and 255 as a RGB representation of the color. Each depth map is a 1-channel matrix of size 480 x 640 pixels. The value for each pixel is a float between 0.0 and 10.0, representing the depth in meters. Since each picture is taken indoors in a regular sized room, the depth between 0.0 and 10.0 meters makes sense.

We split the entire dataset into the training, validation, and testing set. They consist of 80%, 10%, and 10% of the original dataset.

3.2. Qualitative Metrics

The qualitative metric we chose were mean SSIM of the depth maps and the mean F1 score of the edge maps. We chose these two metrics used by the paper to better compare our result to theirs.

The first qualitative metric, SSIM, is a commonly used metric for measuring the similarity between 2 images. Its ability to detect changes in image degradation makes it a good metric, as it wouldn't easily classify our blurry prediction as highly similar to the ground truth label.

The second qualitative metric, edge maps F1 scores, checks whether the edge map between our prediction and the ground truth label matches up. For depth maps, edges mark sharp changes in depth. It is justified for us to compare these edge maps to ensure the prediction and ground truth label share the same outline. Since the majority of each depth map consists of non-edge pixels, the proportion between edge pixels and non-edge pixels is unbalanced. F1 score helps us overcome this imbalance between the proportion of these two classes.

Below is an example of a ground truth depth-map and its corresponding edge map. The blurriness of the depth map is due to resizing.

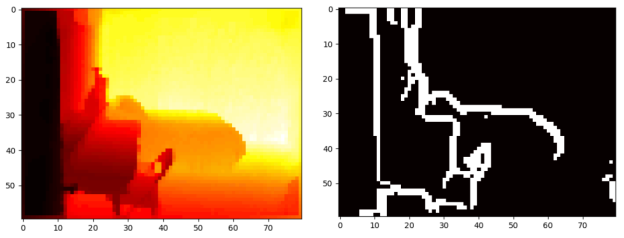

Figure 3. A ground truth depth-map and its corresponding edge map

3.3 Loss

Both training and validation loss are calculated using the same method described in the approach section. The figure below shows the training and validation loss throughout all 236 iterations. Training loss begins at a high value of 909.332, drops to below 200 for the majority of the epochs, and ends at the value of 39.931. Validation loss begins at 3.76, drops below 1.5 for the majority of the epochs and ends at 1.066.

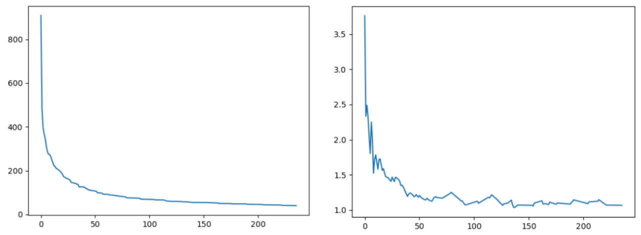

Figure 4. Training Loss (Left) and Validation Loss (Right)

3.4. Qualitative Results

Table 2. mSSIM and F1 Results

Method	mSSIM ↑	F1 ↑
Alhashim-Wonka	0.968	0.519
Ours	0.974	0.316
Random Guesses	0.745	0.206

Our qualitative results for both mSSIM and F1 are better than Random Guesses, and our model achieved a higher mSSIM than even the Alhashim-Wonka paper.

Random guesses are obtained by generating depth maps filled with random values between 0.0 and 10.0, the range of the ground truth labels.

3.5. Quantitative Metrics

The qualitative metric we chose were average relative error (rel), root mean squared error (rms), average (log10) error, and threshold accuracy (δi) for thresholds 1.25, 1.252, 1.253. Not only were these metrics used by Alhashim-Wonka, they were also used in previous works mentioned in their comparison table [1]. It shows that these metrics do hold significance in quantitative evaluation. We also chose these metrics to compare our model's performance quantitatively that of Alhashim-Wonka's.

3.6. Quantitative Results

Table 3. Quantitative Results 1

Method	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$
Alhashim- Wonka	0.846	0.980	0.996
Ours	0.326	0.598	0.790
Random Guesses	0.126	0.255	0.389

Table 4. Quantitative Results 2

Method	rel ↓	rms ↓	log10 ↓
Alhashim- Wonka	0.103	0.390	0.043
Ours	0.435	1.591	0.185
Random Guesses	1.527	3.897	0.412

As listed in the quantitative results, our model performed better than Random Guess on all of the metrics (average relative error, root mean squared error, average, threshold accuracy). However, we were not able to reproduce the performance achieved by the Alhashim-Wonka paper due to lack of computational resources. Nevertheless, the approach described in this report is a promising avenue for additional research.

3.7. Single Image Training Results

Since we couldn't fully train our model, we decided to attach some sample results from two models that was trained on a single image, the same image in both but one was downsampled by a factor of 4 and the other one wasn't downsampled at all.

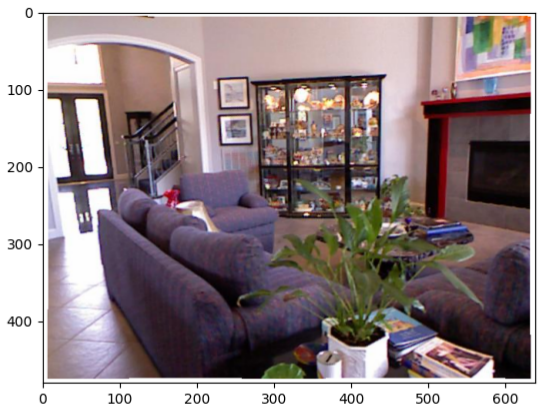

Figure 4. The Actual Image

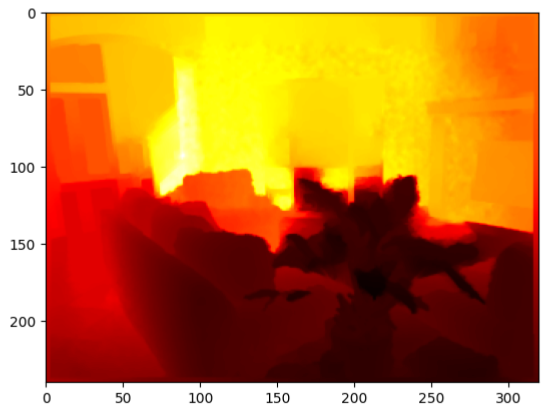

Figure 5. The Labelled Depth Map

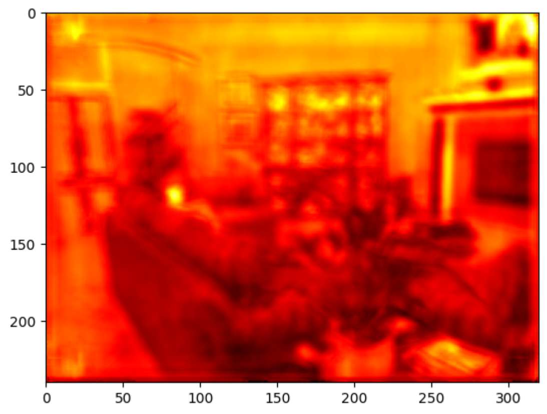

Figure 6. Result after training on the downsampled image for 8000 epochs

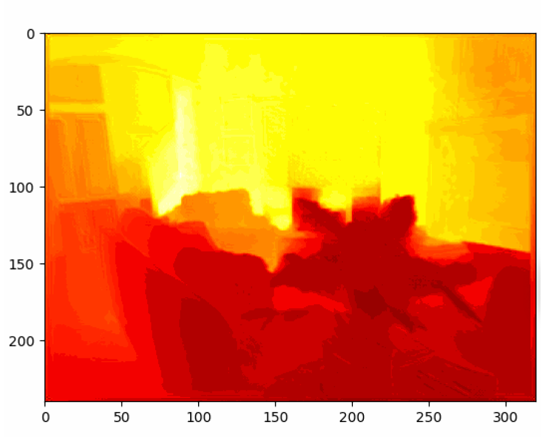

Figure 7. Result after training on the original image for 4500 epochs

4. Implementation

For our implementation of the paper, we used the PyTorch library. The data augmentation operations were accomplished using functions from the Numpy library. Although the Alhashim-Wonka paper [1] was used as guidance for implementation, all code was written from scratch without any usage of reference code from the paper.

We made a model class to train and test the model. For the encoder part of the model, we first downsampled the image by a factor of 4 using Max Pooling 2D and then we used a pretrained Densenet 169 from PyTorch, and we froze all of its parameters. We iterated through the Densenet model in our forward pass to collect the output of some layers for the decoder. Then we used various PyTorch objects such as Conv2D, Upsample and LeakyRELU to implement the decoder.

We made a DepthLoss class for implementing the loss. For the loss, we used functional objects from PyTorch for our forward passes and also a library called pytorch_msssim to implement SSIM loss.

We mainly followed section 4.3 of the Alhashim-Wonka paper to implement our quantitative and qualitative metrics [1]. Most could be implemented with simple Pytorch operations. The edge maps were obtained using the sobel operation found in kornia.filters.sobel.

One key change that we made the metrics was changing the threshold of the edge maps from 0.5 to 0.2. We discovered that if we set 0.5 as our threshold, almost none of the edges from a ground truth depth map would remain. By setting 0.2 as our threshold, the resulting edge map was more representative of the visible edges in a depth map.

References

[1] Alhashim and P. Wonka, “High Quality Monocular Depth Estimation via Transfer Learning”, *Arxiv.org*, 2019. [Online]. Available: https://arxiv.org/pdf/1812.11941.pdf. [Accessed: 27-Apr- 2021].

[2] G. Huang, Z. Liu and L. van der Maaten , “Densely Connected Convolutional Networks”, *Arxiv.org*, 2018. [Online]. Available: https://arxiv.org/pdf/1608.06993.pdf. [Accessed: 27-Apr- 2021]