Evaluation of "Learning to Recover 3D Scene Shape from a Single Image"

LEN BAUER

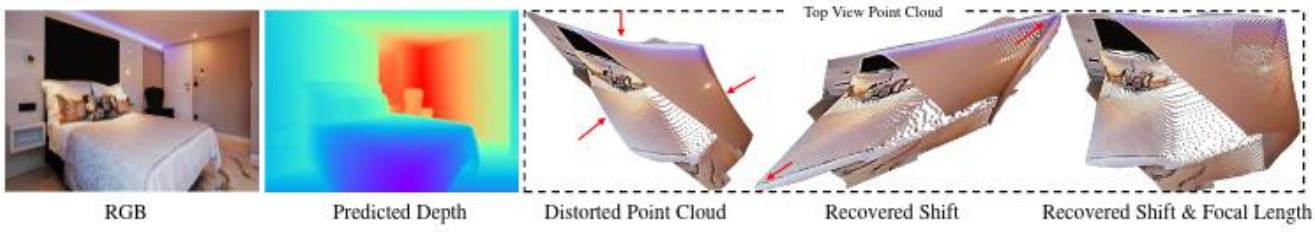

Fig. 1. 3D scene structure distortion in point clouds. While the depth map is correct, the 3D shape suffers from distortions due to unknown depth shifts and focal length. This method recovers these parameters to correct the distortions.

Recent advancements in monocular depth estimation have improved the performance of predicting depth from a single image. However, accurately recovering 3D scene shapes remains challenging due to depth shifts and unknown camera focal length. The paper "Learning to Recover 3D Scene Shape from a Single Image"[7] introduced a two-stage framework to address these issues, using a convolutional neural network (CNN) for depth prediction and point cloud encoders for refining depth shift and focal length. Additionally, novel loss functions were proposed to enhance model training on diverse datasets. This method achieves state-of-the-art performance on unseen datasets. Notably, this method only uses a single image to predict depth and a 3D point cloud.

Unfortunately, the paper was released in 2021, and their codebase has not been updated since 2022, making it outdated. This document explains how to use the model today, more than two three after its release, and summarizes my experimental research on its robustness. First, let's summarize the paper's contributions and model construction.

ACM Reference Format:
Len Bauer. 2024. Evaluation of "Learning to Recover 3D Scene Shape from a Single Image". 1, 1 (June 2024), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

3D scene reconstruction from a single image is a key task in computer vision, typically approached via methods such as SLAM or SfM, which rely on multiple views or frames. Monocular depth estimation offers an alternative but faces difficulties in accurately reconstructing 3D shapes due to unknown depth shifts and camera parameters.

This work proposes a two-stage approach: a depth prediction module (DPM) that predicts depth maps with unknown scale and shift, and a point cloud module (PCM) that refines the 3D reconstruction by predicting depth shifts and focal length adjustments. The method leverages point cloud networks and new loss functions to improve generalization and depth prediction accuracy.

Author's address: Len Bauer.

ACM XXXX-XXXX/2024/6-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

Monocular depth estimation has made significant strides, primarily through data-driven approaches using diverse datasets. Methods using stereo images and videos provide depth supervision up to scale and shift, leading to the development of losses invariant to these transformations. However, these methods do not address shape distortions caused by depth shifts and unknown focal lengths.

3D reconstruction from a single image has been explored for various objects and scenes but often relies on specific priors or assumptions. This work differs by proposing a fully data-driven approach applicable to diverse scenes.

3 METHODOLOGY

The proposed framework comprises two main components: a depth prediction module and a point cloud module.

3.1 Depth Prediction Module

The DPM is a CNN trained on mixed datasets to predict depth maps up to an unknown scale and shift. To handle diverse data sources with varying depth ranges, an image-level normalized regression loss (ILNR) and a pair-wise normal regression loss (PWN) are introduced.

3.1.1 Image-Level Normalized Regression Loss. ILNR normalizes depth maps using trimmed statistics to reduce outlier effects, defined as:

$$L_{ILNR} = \frac{1}{N}\sum_i |d_i - d_i^*| + \left|\tanh\left(\frac{d_i}{100}\right) - \tanh\left(\frac{d_i^*}{100}\right)\right| \quad (1)$$

where d_i is the predicted depth, and d_i^* is the ground truth depth normalized using trimmed mean and standard deviation.

3.1.2 Pair-Wise Normal Regression Loss. PWN improves local geometric features by leveraging depth edges and planes. Normals, being complementary to depth, enhance the overall prediction quality.

3.2 Point Cloud Module

The PCM refines the initial 3D reconstruction by predicting depth shift and focal length adjustments. Using a pinhole camera model, the unprojection from 2D coordinates to 3D points is:

$$\begin{aligned} x &= \frac{u - u_0}{f} \cdot d \\ y &= \frac{v - v_0}{f} \cdot d \\ z &= d \end{aligned} \tag{2}$$

where (u_0, v_0) are the optical center coordinates and f is the focal length. The point cloud networks, trained on synthetic 3D data, predict the depth shift and focal length scaling factor, reducing shape distortions.

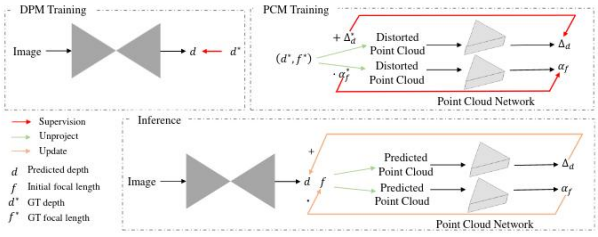

Fig. 2. Method Pipeline. The depth prediction model and point cloud module are trained separately and combined during inference to predict depth, depth shift, and focal length for accurate 3D reconstruction.

4 EXPERIMENTS

The proposed method is tested on nine unseen datasets, achieving state-of-the-art performance in zero-shot dataset generalization. The depth prediction model and point cloud reconstruction module effectively recover accurate 3D shapes from single images.

5 CONCLUSION

This paper presents a novel framework for monocular 3D scene shape estimation, addressing the challenges of depth shifts and unknown focal lengths. By integrating depth prediction with point cloud refinement and introducing robust loss functions, the method significantly improves 3D reconstruction accuracy in diverse scenes. Future work could explore further enhancements in model training and application to real-world scenarios.

6 RUNNING THE MODEL

As the paper was written in 2021 and the last commit to the GitHub repository[8] was 8 months ago, the project appears to be end-of-life and is no longer maintained. This raises significant challenges in running the model. This section outlines the steps to get the model running today for those who wish to try it.

Even though the authors' work is impressive, they did not provide a well-maintained codebase, which has made working with their code quite challenging. Users frequently report issues with compatibility and training data.

6.1 Training Data

First, obtaining the training data is problematic. The authors provided it on a Chinese platform, Baidu, which requires creating an account, downloading a custom client, and possibly making a payment. This process is inconvenient and considered unreliable by many users. Some users have criticized Baidu as a scam and expressed frustration over the lack of a more accessible hosting provider[2].

Fortunately, a user uploaded the data to a more accessible platform, which can be found here: https://huggingface.co/ffranchina/LeReS/tree/main. As the project is at its end-of-life, the author did not highlight this alternative, making it difficult to find. I discovered this link completely hidden in an issue only after hours of trying to download the weights from Baidu, effectively wasting time.

After successfully mastering this step, it shouldn't be too hard to finally run the model, right? Unfortunately, users frequently encounter issues with certain libraries not working with their code anymore. Someone also made a pull request this January, claiming to fix "some" compatibility issues, but it was never accepted and, in any case, did not work for me.

6.2 Torchsparse

What is the problem? The issue is that LeReS depends heavily on torchsparse, a project created and maintained by MIT students for efficient training and inference for sparse convolution on GPUs. Following the LeReS installation guide, they used torchsparse v1.2.0, while the current version is v2.1.0. Another example of dependency issues is that LeReS also used very old versions of torchvision and PyTorch, while their installation guide uses the up-to-date libsparsehash, which is not compatible with these older versions.

Looking into torchsparse, there were two major updates that changed their API: the first between v1.2 and v1.4, and the second from v1.* to v2.0. Due to the extent of changes in the second update, v2.* cannot be used.

The problem with torchsparse is that its installation process is unreliable, especially for older versions. Torchsparse only provides pre-built packages for v2.*, so we must build older versions ourselves. They also provide an installation script for all versions, which initially worked for me but later became unavailable when their PyPI server was down and remains unavailable. They already provide an alternative, as this PyPI server is frequently down and unstable. This is a common issue reported by many users[5]. Fortunately, there is an alternative website (http://pypi.hanlab.ai/simple/torchsparse), though it is currently down, leaving no viable way to install torchsparse via the official script or any website. Reversing their install script and commit history for older versions was also not an option for me.

At this point, I regretted resetting everything, as the install script had worked a few days prior, but problems elsewhere forced me to restart. This illustrates the compatibility problems and why I criticized the authors for a weak codebase, as it depends on outdated software that seems nearly impossible to install.

Eventually, I found a way to download *some* old version, specifically v1.4.0, from Anaconda[6]. Although it is unclear why they never upgraded to newer versions, this method finally provided compatible software to run the code.

Here is the full installation process:

Oh wait, what about the API changes in version v1.4.0? Building LeReS with this version of torchsparse led to many compilation errors, requiring code adjustments to work with the "new" torchsparse version. I even attempted to build version v1.2 on my own, but they did not provide a makefile or adequate documentation, so I eventually gave up on that approach. You can find the final diff file attached, which includes the necessary changes to make the code compatible with the newer API[1]. But let's discuss what it does.

Checking out their latest branches tagged with v1.2.0 and v1.4.0 reveals what the obvious changes in their API were: Check out v1.2[3] and v1.4[4].

The first thing that appears is that they used before two different data structures for point cloud storage: torchsparse.SparseTensor and torchsparse.PointTensor. Afterwards they completely removed PointTensors from torchsparse and this is e.g. one part we have to fix in LeReS to get it compiled.

Here's a summary of changes between torchsparse versions:

SparseTensor Data Structure

- **v1.2.0:**
  - **SparseTensor:** Assumes integer coordinates without duplicates.
  - **PointTensor:** Assumes floating-point coordinates with possible duplicates.
- **v1.4.0:**
  - **SparseTensor:** Coordinates are integer tensors with a shape of $N \times 4$, representing quantized x, y, z coordinates, and a batch index. Features are an $N \times C$ tensor.

Sparse Quantize

- **v1.2.0:** The `sparse_quantize` function converts floating-point point cloud coordinates to unique integer coordinates, removing duplicates.
- **v1.4.0:** The `sparse_quantize` function now accepts a voxel size and is used for voxelizing coordinates and removing duplicates. The result includes a tensor with integer coordinates and a tensor of features corresponding to these coordinates.

Sparse Collate

- **v1.2.0:** Use `sparse_collate_fn` to combine a list of SparseTensors into a batch.
- **v1.4.0:** `sparse_collate_fn` is still used to assemble a batch of SparseTensors but now also adds the batch dimension to coordinates.

Neural Network Interface

- **v1.4.0:** Introduces `spnn` for sparse neural network layers, with a similar interface to PyTorch's `nn`.

After adjusting the code to work with this newer API, I managed to get the model running and tested it.

However, fixing compatibility issues involved several steps. Notably, the code had to be modified to accommodate changes in the torchsparse API. Building the software with torchsparse v1.4.0 led to compilation errors, which required in-depth adjustments to the codebase.

For those facing similar issues, the final diff file that I applied is attached. It reflects the necessary changes to make the code compatible with the newer API[1].

Sorry for making this more technical and boring part so long, but this was literally the part I spend the most of the time on and in this case I really tried to get it done for more than a whole week and really invested a lot of time in it, so I didn't want to skip it here.

7 ROBUSTNESS

Here are the results, I found out during my "research" about it's performance. All in all the models deal quite well with images in usual settings. But as described, the model is strongly data driven and therefore there are some settings, where the model has problems to reliable estimate depth for example. I will argue with basic examples and the depth prediction module, but the most problems generalize also to the 3D point cloud construction module.

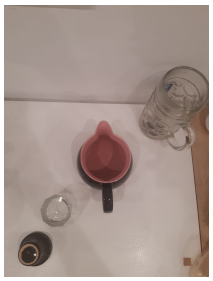

Fig. 3. Topview for setting in Fig 4

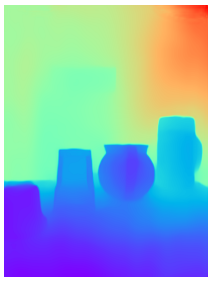

Fig. 4. Frontview for setting small to big

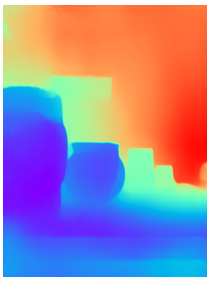

Fig. 5. Frontview for setting big to small

The first thing I noticed is, that it strongly relies on heuristics that hold for the most images. For example usually size corresponds to distance. This is why we can see in this experiment, that the prediction strongly relies on the size of the objects. On the left we can see, that the objects are ordered from smallest to largest and on the right (in Fig 5) from big to small and meanwhile also from close to more distanced. Note that the objects and also the wall in the background all have the same distance to the camera, but are ordered differently in both settings. On the left it detects only small depth differences while on the right the model overestimates the depth.

While for the first setting all objects are assigned similar depth due to increasing size, in the second setting the small objects appear to be way more behind than they actually are. Also it assumes for both settings, that the straight wall has more depth on the right, only because of the objects arranged into this direction with increasing depth. Again this effect is way stronger for the second setting, due to this size increase.

Also in real world a lot of objects actually have 90 degrees and its way more rare, that objects are no rectangles. In the figures above one can see, that it can successfully model the continuous depth of the wall on the left figure, as the left part (from left side the first side) of the rectangle appears to be bigger, while the right

Fig. 6. Frontview of rectangle

Fig. 7. Frontview of not-rectangular form

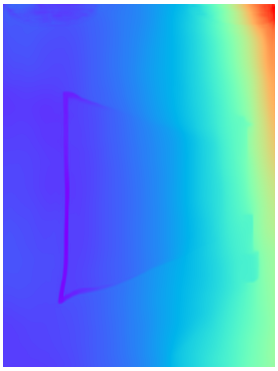

Fig. 8. View from left side for rectangle in Fig 6

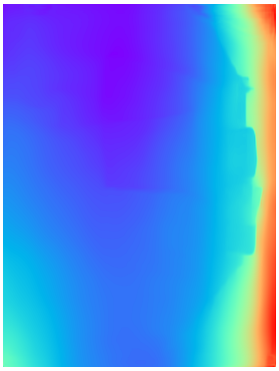

Fig. 9. View from left side for rectangle in Fig 7

example is constructed, s.t. it will seem to be a rectangle through viewing from the left side, so that it will model the depth of the drawing everywhere equally and then has hard boundaries when the drawing ends and the wall gets modeled with a lot of depth (red in the picture).

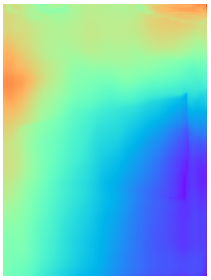

Fig. 10. Depthmap of Fig 7.

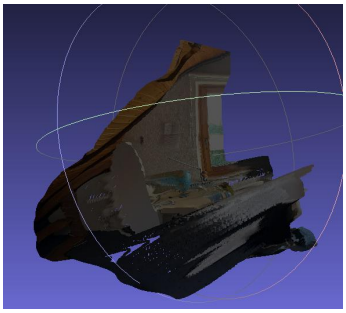

Fig. 11. Depthmap of Fig 7.

Additionally one can see, that the depth estimation model has problems to estimate depth, when we use something not rectangular and we can successfully trick the model into thinking the left part of the image has more distance than the right part, even if the wall is straight.

Also in Fig 11 we can see, that the 3D point cloud model has problems, to model a wall that has an angle of less than 90 degrees. It makes the edge round, as 90 degrees are way more often and also curved objects are more common than edges with different angles as in this case. Also the roof appears curved instead of straight.

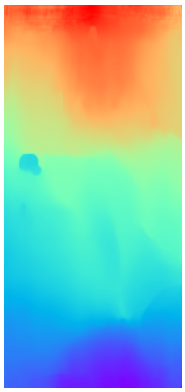

Fig. 12. Depth of Screenshot

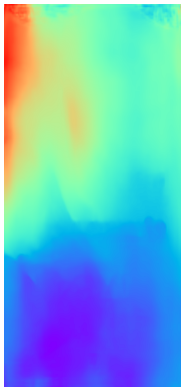

Fig. 13. Depth of Fig. 12 rotated by 180 degrees

The last thing I noticed was, that it also uses heuristics about light and image positioning. The depthpredictions seen here, are from some mobile phone screenshot, where no depth should be available. That the model tries to find some depth anyway is not spectacular, but it predicts in both cases, that the top has more distance. But the trick here is, that it's actually the same image just rotated with 180 degrees and therefore should predict usually the same depth but also rotated, if it would actually predict depth without prior knowledge on image structure. This is quite interesting, as it shows, that it strongly relies on the fact, that far away objects like the sky are usually on top of close objects.

One very last thing I noticed was, that the does not has a deeper under- standing on what objects are and how they are constructed, like a human has, but instead estimates depth without detecting objects. In this picture one can see a display that shows some picture.

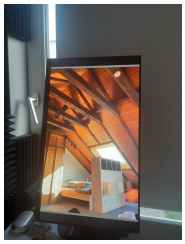

Fig. 14. Display with image

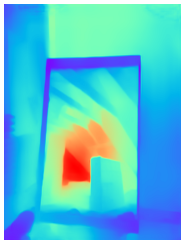

Fig. 15. Depth of Fig. 14

A human understands, that it is just an image, but the model still sees depth in the display due to the image and also creates the 3d

prediction point cloud way more behind the display. So we can see, that this model is not that robust against outliers and should not be used in critical systems or applications, as it can be tricked easily to predict depth at the wrong position.

REFERENCES

[1] Len Bauer. 2024. Code Changes. (2024). https://github.com/Len101218/Papers/blob/main/LeReS/changes.diff Accessed: 2024-06-03.

[2] Hugging Face. 2024. LeReS. (2024). https://huggingface.co/ffranchina/LeReS/tree/main Accessed: 2024-06-03.

[3] MIT Han Lab. 2021. torchsparse v1.2. (2021). https://github.com/mit-han-lab/torchsparse/tree/v1.2.0 Accessed: 2024-06-03.

[4] MIT Han Lab. 2021. torchsparse v1.4. (2021). https://github.com/mit-han-lab/torchsparse/tree/v1.4.0 Accessed: 2024-06-03.

[5] MIT Han Lab. 2024. torchsparse. (2024). https://github.com/mit-han-lab/torchsparse/issues/300 Accessed: 2024-06-03.

[6] Libraries.io. 2024. Download torchsparse v1.4. (2024). https://libraries.io/conda/torchsparse Accessed: 2024-06-03.

[7] Oliver Wang Simon Niklaus Long Mai Simon Chen Chunhua Shen Wei Yin, Jianming Zhang. 2020. Learning to Recover 3D Scene Shape from a Single Image. (2020). https://arxiv.org/abs/2012.09365 Accessed: 2024-06-03.

[8] Oliver Wang Simon Niklaus Long Mai Simon Chen Chunhua Shen Wei Yin, Jianming Zhang. 2021. AdelaiDepth. (2021). https://github.com/aim-uofa/AdelaiDepth Accessed: 2024-06-03.