CS230

Multitask Deep Learning for Depth Estimation and Object Classification

A. Nemi Nayak
Department of Physics
nayak03@stanford.edu

Claire Du
Department of Mechanical Engineering
clairedu@stanford.edu

Victor Paul Portmann
Department of Mechanical Engineering
vpp@stanford.edu

1 Introduction

Monocular depth estimation, the process of extracting a pixel-wise depth map from a single RGB image, is a commonly studied problem in deep learning. We aim to investigate transfer learning strategies for depth estimation and object classification. Importantly, we train on Redwood 3D Scan, a dataset featuring commonly encountered objects collected by civilians equipped with RGB-D cameras; we feature a simpler, easier model for non-ideal, real-world environments. We hope it will be useful for everyday cases where complete knowledge of the surrounding objects is crucial.

2 Dataset and Features

We use the Redwood 3D Scan Dataset [3], a public domain dataset of RGB-D sequences collected by civilians featuring daily objects in various environments. RGB-D data has 3 channels for the RGB image and a fourth channel for the depth map, which associates each pixel with a value measured by the camera in millimeters. Redwood 3D specifies 320 object classes in total, but we used only 3 classes in our model. For faster development, we resize the 640x480-pixel images depth maps to 96x128 pixels using nearest-neighbor interpolation. We trained on a subset of the total dataset amounting to around 4500 total examples, with 300 examples used for development, 400 examples used for testing, and the rest for training.

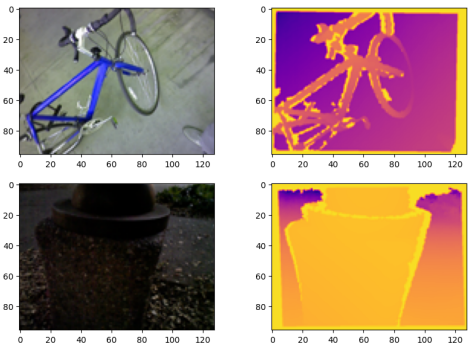

Figure 1: RGB and D channels, separated

We employ the data augmentation scheme discussed in Alhashim and Wonka [1], which horizontally flips images with a probability of 0.5 and swaps the R and G channels of the image with a probability of 0.25. Vertical flips are not used as they would create invalid depth maps, and rotations are not used as they would create unusable values in the depth map arrays.

3 Methods

Applying the transfer learning approach proposed by Alhashim and Wonka [1], we use pretrained weights from a pre-existing robust image classification model such as DenseNet as the base for a model that performs monocular depth estimation. The theory is that the feature-learning capabilities of these models will provide our model with a solid starting point for the depth estimation task. Our project focuses on two main goals. Firstly, we note that Alhashim and Wonka

CS230: Deep Learning, Fall 2024, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

were able to create substantial results using DenseNet169 as their backbone but since then, newer, smaller and more efficient models have emerged [6] and we would like to see if these models, which are better at object classification, would also provide strong backbones for the depth estimation task. Secondly, we aim to perform some multitask learning analysis by seeing if unlocking and fine-tuning later layers of the base model will result in significant changes to either our object classification or depth estimation heads.

Our model uses a U-Net style encoder-decoder architecture. We use the TensorFlow approach from Alhashim [2] as a guideline to create the framework for the new model, including the ability to swap and sample different backbones. Using the base model as the encoder, we implement a custom decoder/upsampling block with the option to form skip connections. In our final implementation, we create skip connections to the first four blocks and simple upsampling in the fifth and last block. For our depth estimation head, we use a loss function with three weighted terms: mean absolute error (MAE) term (Eq. 2), a gradient loss (Eq. 3), and a structural similarity index (Eq. 4) (SSIM) term. The equations for the terms are as given below:

$$L(y, \hat{y}) = L_{MAE}(y, \hat{y}) + L_{grad}(y, \hat{y}) + L_{SSIM}(y, \hat{y}) \quad (1)$$

$$L_{MAE} = \frac{1}{n} \sum_{n=1}^{\infty} |y_i - \hat{y}_i| \quad (2)$$

$$L_{grad} = \frac{1}{n} \sum_{i=1}^{n} |g_x(y_i, \hat{y}_i)| + |g_y(y_i, \hat{y}_i)| \quad (3)$$

$$L_{SSIM}(y, \hat{y}) = \frac{1 - SSIM(y, \hat{y})}{2} \quad (4)$$

For our object classification head, we use a categorical cross-entropy loss.

4 Discussion

4.1 Backbone Experiments

Our backbone experiments were conducted using our smaller development set of 300 image-map pairs. For our depth estimation head, we use two metrics that correspond to two of the three terms in our loss function, MAE (as is standard for regression tasks) and SSIM (as is standard for computer vision tasks). For our object classification head, we use the standard classification metrics of categorical accuracy, precision, and recall.

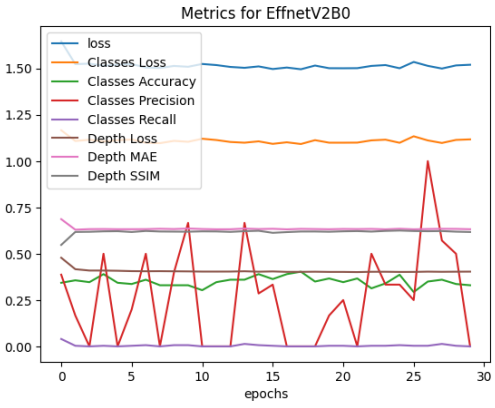

Figure 2: Metrics plateau for EfficientNetV2B0

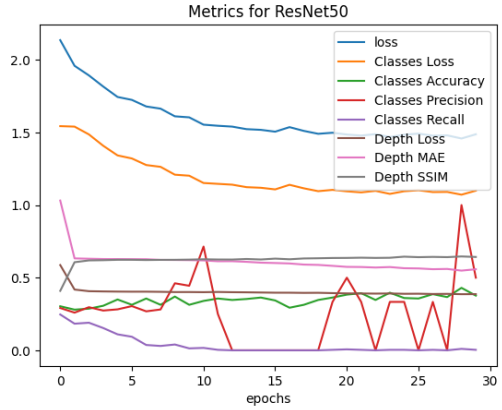

Figure 3: Metrics plateau for ResNet50

We were hopeful about EfficentNetV2B0 as a workable base model, as EfficientNet models are more compact and have less parameters than other models. The base model also performs better than both ResNet and DenseNet on standard object classification datasets [6]. Theoretically, as its weights have optimized feature recognition, it should be able to work as a backbone for our monocular depth estimation task. However, we found the contrary, as even after rigorous hyperparameter adjustments, the model with EfficientNet as the base refused to converge meaningfully (Figure 2).

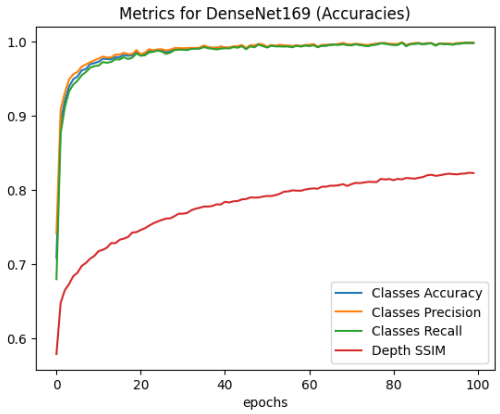

Figure 4: Increasing metrics for DenseNet169

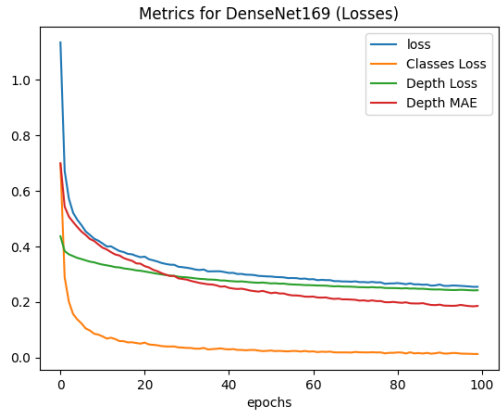

Figure 5: Decreasing metrics for DenseNet169

We experimented with a sliding scale of values in learning rate (0.1 to 1e-5 as well as some dynamic learning rate schedulers) and the number of filters applied in five different convolutional transpose operations (16, 32, 52, 64, and 128). We also experimented with sending different blocks from the original model forward to the upscaling blocks during the skip connections, cleaning the data by removing depth maps that were not properly indicative of their images, and unfreezing the whole network and allowing all trainable parameters to be adjusted during the training process, among other strategies. In all cases, all metric values quickly hit the same plateau that can be seen in Figure 2 and refused to move. Earlier, we had developed a very small fully convolutional network to act as a proof of concept for our task, and that toy network had better performance than what we were seeing from EfficientNet, so we knew we could do better.

We also tested briefly using ResNet50 as a model, but the metrics on that model plateaued as well (Figure 3), although at different values from the plateaus of EfficientNet. Some time was spent performing hyperparameter adjustments on this model as well.

Reluctantly, we reverted to using DenseNet169 as our base model, and to our surprise, the model surpassed the EfficientNet plateau almost immediately and loss values continued to decrease.

4.2 The Usable Model

The DenseNet backbone model was trained on our full training set for 100 epochs with a batch size of 16 because it was the batch size that ran the fastest. A learning rate of 0.0001 was chosen as it was the largest learning rate that did not overshoot the minimum, and after testing 16, 32, 52, and 64 filters, 52 filters yielded the best results. The metrics of the final model are shown in Figures 4 and 5, and the final metric values are summarized in Figure 8.

We then evaluated the model on a test set of around 350 image-map pairs. As expected with a transfer-learned model like this one, the classification head consistently yields good accuracy values on both the train and test sets, plateauing in the 97-98% range. For the depth maps, the results are less consistent between train and test set cases, as the model was able to fit to the training set with an MAE range of 0.17-0.24 meters, while the test set MAE was 0.44 meters. This suggests that the model may have overfit the training distribution. Nonetheless, when visually inspecting the depth maps yielded from both train and test sets, the model consistently performs well on both sets in understanding and accurately predicting the depths of prominent features in the image (Figure 7). Visually inspecting images between classes, the model does better on the "trash container" class than any other class, likely because those images are much simpler than the rest of the set. The model performs particularly poorly on images in the "car" category wherein the image contains a car wheel (Figures 9 and 10 are provided in the appendix).

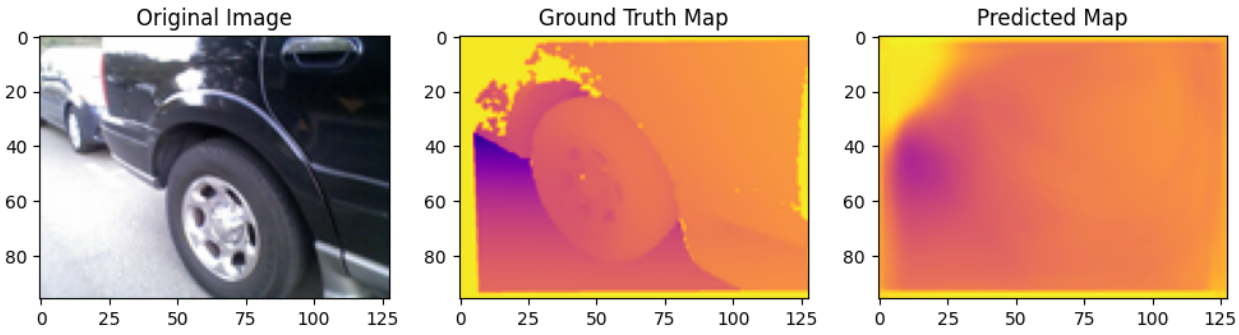

Figure 6: Output after 30 epochs of training

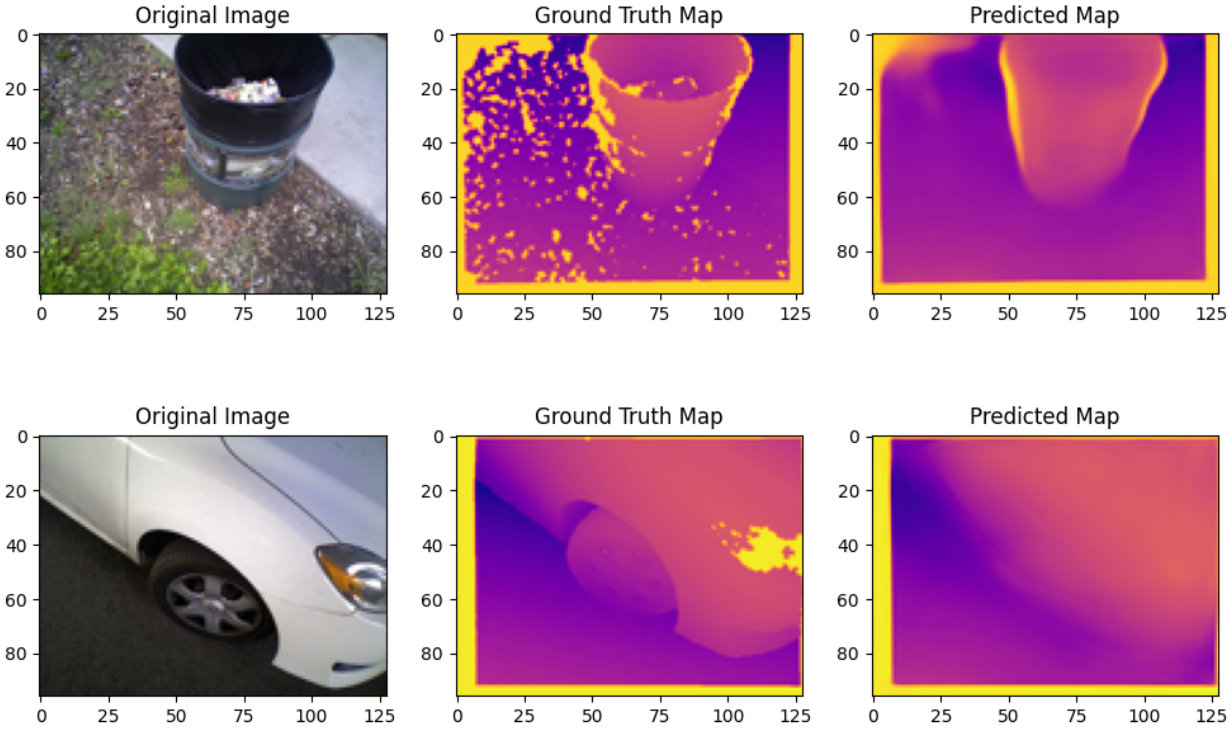

Figure 7: Output after 100 epochs of training

4.3 Fine Tuning Experiments

After training the model, we wanted to see if unfreezing and allowing the model to train the upper layers of the base model would result in significant changes, preferably improvements. However, unlocking the upper layers seemed to confuse and overcomplicate the model. Accuracy metrics on the classification head took longer to converge, and the depth map MAE was only able to reach 0.37 meters after 50 epochs of training. We believe that more investigation is required in this respect, perhaps more training or unlocking less layers.

5 Next Steps

In short, our framework showed that while DenseNet169 seems to be a good backbone for simple monocular depth estimation models, EfficientNet and ResNet50 did not seem to be good candidates for the same task.

We propose two key next steps for this project. We speculate that since EfficientNet is heavily optimized for image recognition specifically, it may result in features that are less transferable to regression tasks like depth estimation. Looking into why exactly this may be the case is an interesting next question. However, the image recognition head did not converge on the EfficientNet model as expected for an image recognition model, which leads us to suspect that something may have gone wrong with our implementation. We would need to look into this further as well.

Metric	Train	Test
Total Loss	0.2517	0.4561
Classes Loss	0.0124	0.0833
Depth Loss	0.2393	0.3722
MAE	0.1817	0.4318
SSIM	0.8235	0.6841
Accuracy	0.9984	0.9777
Precision	0.9984	0.9777
Recall	0.9981	0.9777

Figure 8: Metrics

Another next step would be to exchange the object classification head for a semantic segmentation one. We chose to add an object classification head to our current model because we wanted to work with civilian-collected data points, and the data set that we were working with only provided single-class markers. Training similar models on datasets like CARLA [8] (wherein both depth and segmentation data are provided) will allow us to properly investigate the multi-task nature of our problem.

Github Repository

`https://github.com/nayak-03/CS230-Project`

References

[1] Alhashim, Ibraheem and Wonka, Peter. (2019). *High Quality Monocular Depth Estimation via Transfer Learning.* Available at: https://arxiv.org/abs/1812.11941.

[2] Alhashim, Ibraheem. (2019). *DenseDepth: High Quality Monocular Depth Estimation via Transfer Learning.* Available at: https://github.com/ialhashim/DenseDepth. Accessed: 2024-11-15.

[3] Choi, Sungjoon, Zhou, Qian-Yi, Miller, Stephen, and Koltun, Vladlen. (2016). *A Large Dataset of Object Scans*. Available at: https://arxiv.org/abs/1602.02481.

[4] Wang, Zhou, Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). *Image quality assessment: from error visibility to structural similarity.* IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600–612. doi: 10.1109/TIP.2003.819861.

[5] Yang, Lihe. (2024). *Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data.* Available at: https://github.com/LiheYoung/Depth-Anything. Accessed: 2024-11-15.

[6] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." International conference on machine learning. PMLR, 2019.

[7] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[8] Team, CARLA. “Carla.” CARLA Simulator, carla.org/. Accessed 3 Dec. 2024.

[9] Tensorflow. “Tensorflow/Tensorflow: An Open Source Machine Learning Framework for Everyone.” GitHub, github.com/tensorflow/tensorflow. Accessed 3 Dec. 2024.

Additional Figures

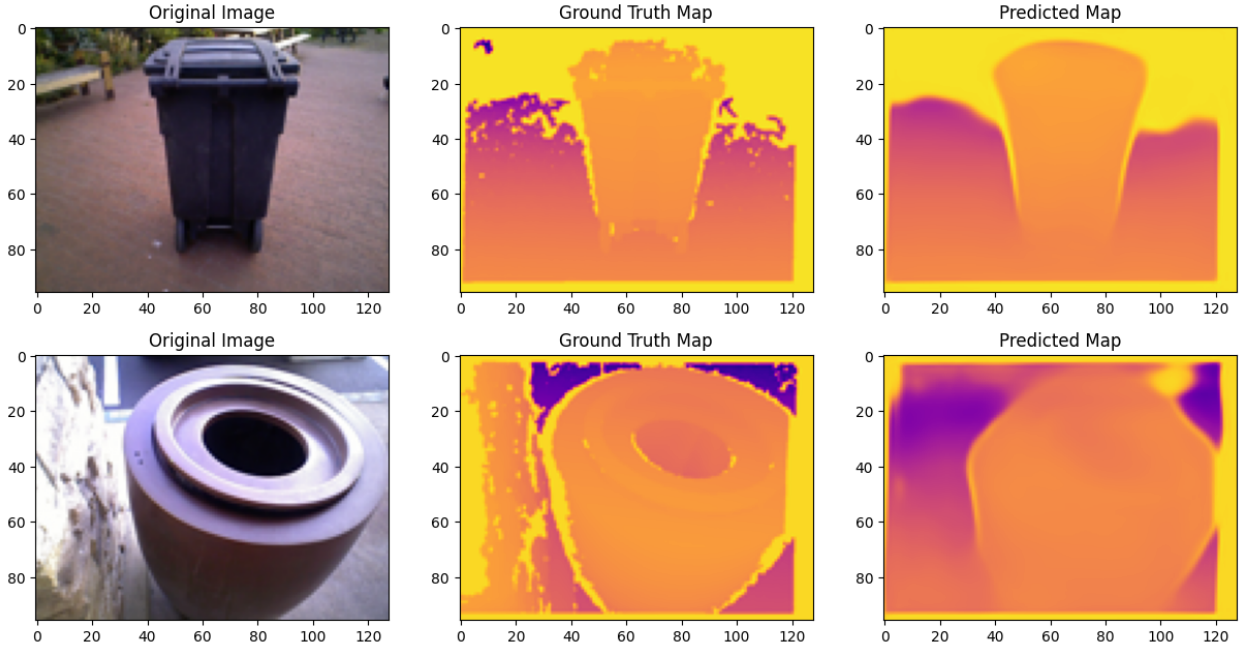

Figure 9: Model does well on simple-shaped trash cans

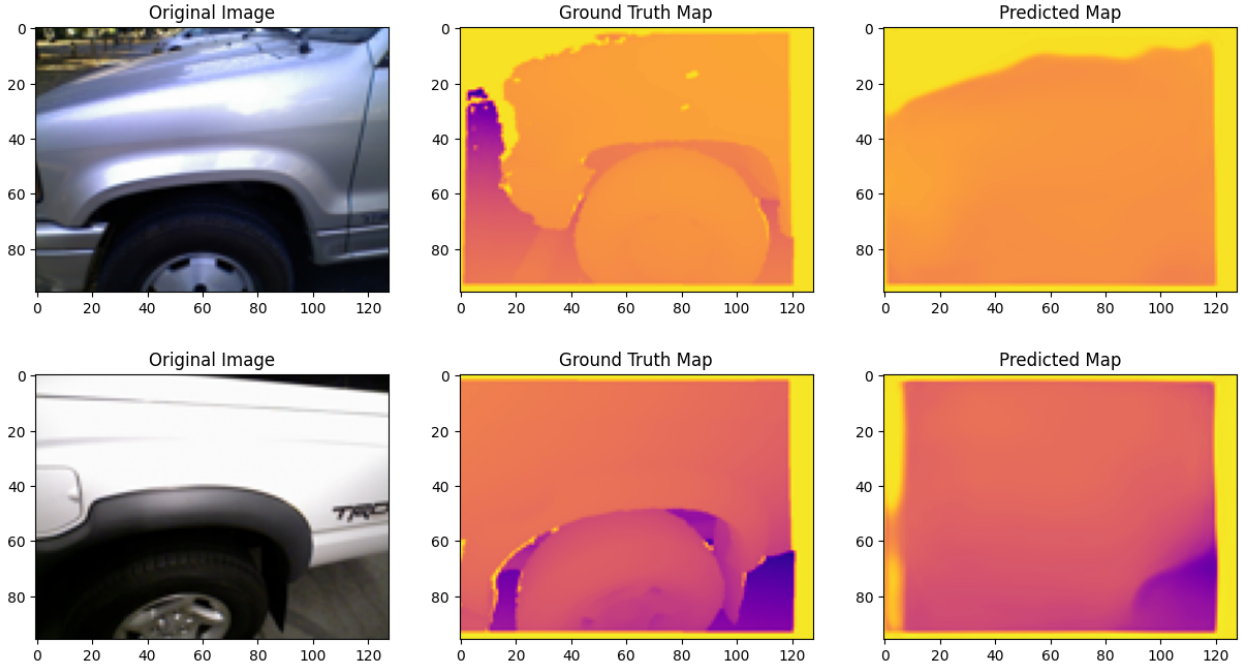

Figure 10: Model performs poorly on more complex images

A direct comparison between maps computed on the same image, 30 and 100 epochs

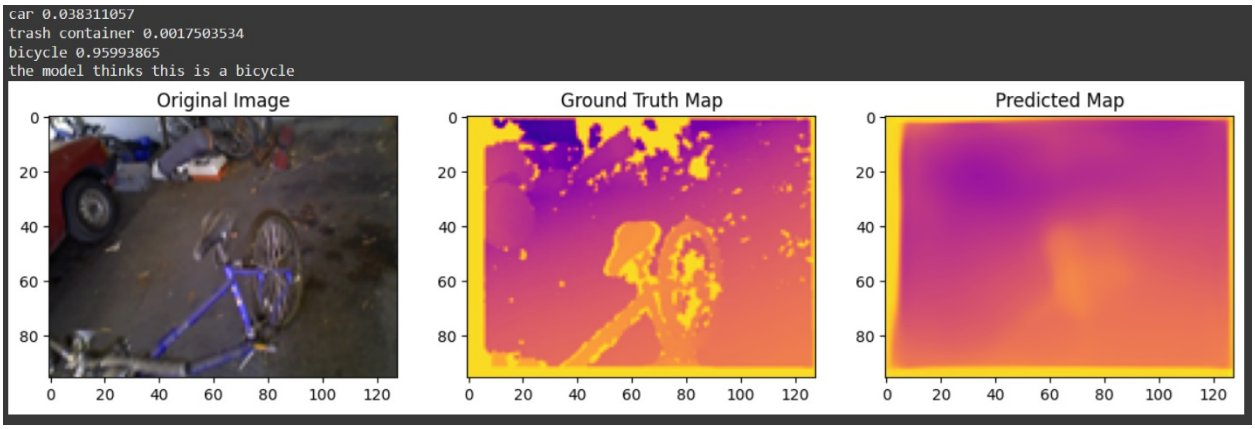

Figure 11: Bicycle after 30 epochs

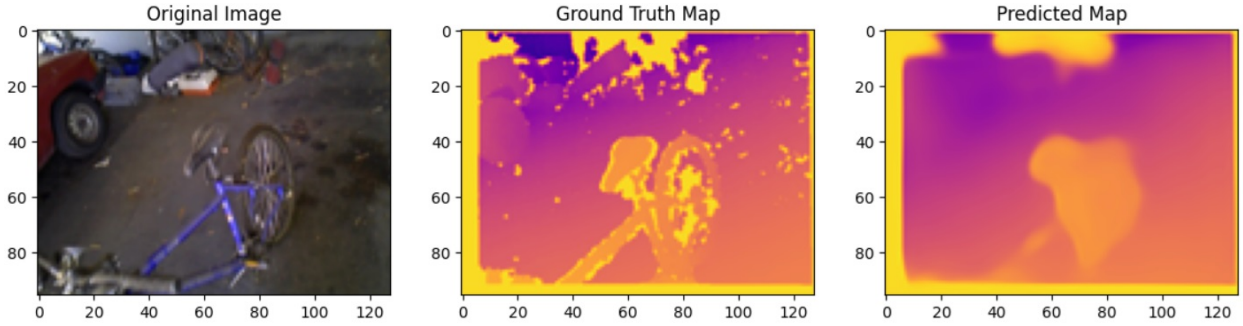

Figure 12: Bicycle after 100 epochs