Low-Latency Monocular Depth Estimation

Reese Haly
rjhaly@wpi.edu
Worcester Polytechnic Institute
Worcester, Massachusetts, USA

Joshua Malcarne
jrmalcarne@wpi.edu
Worcester Polytechnic Institute
Worcester, Massachusetts, USA

1 INTRODUCTION

As deep learning algorithms become increasingly more capable at solving complex tasks in a short amount of time, the desire to achieve accurate real-time inference on 30fps or higher has gained in popularity. One such task is monocular depth estimation(MDE), which has been a well documented challenge in the field of computer vision. MDE attempts to infer depth information from a single RGB image using contexts and details from within the image[6]. MDE is a key task in fields like augmented reality, autonomous driving, and robotics.

In an attempt to achieve accurate real-time MDE, we started with MiDaS v2.1 small, a pre-trained encoder-decoder model known for its accuracy and efficiency[7]. We then pruned the model's weights to various degrees to decrease its computational complexity. After pruning, the estimated model parameters, RMSE, and MAE were measured. Additionally, the unpruned MiDaS model was deployed onto an NVIDIA Jetson Nano and inference time, RMSE and MAE were measured.

In Section 2, we discuss the proposed application of our project by giving an overview of the depth estimation problem domain, the MiDaS depth estimation network, and the KITTI Depth Dataset used in our research. In Section 3, we cover our proposed methodology, including our approach to pruning, fine tuning, and model deployment on an NVIDIA Jetson Nano. In Section 4, we evaluate our results from sparsifying the models and model deployment. We discuss the limitations and challenges our team faced in Section 5, with a focus on our issues deploying a PyTorch model on a Jetson Nano, attempting to transfer the model to TFLite, failing to fine tune the MiDaS model or calculate loss from output predictions, and shelving efforts to quantize the model due to a lack of time. Finally, we wrap up the paper with our conclusion in Section 6 and each of our teammates' contributions to the paper in Section 7.

2 PROPOSED APPLICATION

In this section, we will provide an overview of depth estimation and monocular depth estimation in Section 2.1. We will then discuss the model we chose for our research in Section 2.2 before closing with a discussion of our chosen data set in Section 2.3.

2.1 Depth Estimation

Depth information is widely used in applications like robotics, augmented reality, and autonomous driving. These fields require accurate depth information to ensure a good understanding of their spatial environments. To achieve this, there are several approaches to calculating depth information.

Specialized hardware can be used to accurately measure depth. LIDAR and TOF RGB-D are able to measure depth within millimeters. However, these specialized sensors can have a limited range and are susceptible to inaccuracies in non-ideal conditions like reflective surfaces[6]. To circumvent these shortcomings, two methods have been introduced to estimate depth information without the use of dedicated hardware: stereo vision and monocular depth estimation.

Stereo vision harnesses the power of two or more RGB cameras to perceive depth by triangulating disparities between corresponding points in paired images. By capturing a scene from slightly different angles, stereo vision algorithms can calculate depth information through tiny variations in pixels between each image[9]. However, stereo vision requires the setup and calibration of multiple cameras. Furthermore, stereo vision has difficulty calculating depth for textureless or highly reflective surfaces where it may be harder to match pixels between images.

Monocular depth estimation takes a single RGB image stream as input and uses deep learning models to infer depth information from the contexts within the image. MDE does not require any specialized hardware or setup to infer depth information, requiring only an RGB image stream and a computer to run inference[6]. However modern MDE is able to predict depth information with high accuracy. It is for this reason that MDE has become a widely studied topic in computer vision.

While MDE has become increasingly accurate, there is still work to be done to lower inference time and computation complexity. Large deep learning models require high computational power to compute which exceed the computation resources available on most edge devices. This can be an issue for tasks like augmented reality which may want to calculate depth information in real time at 30fps with limited computational capabilities. Several models have been introduced to attempt real-time MDE for on-device inference such as FastDepth[11] and GuideDepth[8], although with decrease in accuracy as a trade-off for decreased inference time.

2.2 MiDaS v2.1 Small

When considering different MDE networks, we chose to go with MiDaS due to its versatility when selecting datasets. Originally from the research paper "Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer", MiDaS is a state-of-the-art MDE network designed to be trained on various different depth estimation datasets[7]. To achieve this, the network was created in a way that handles the various ground truth representations from these differing datasets. The MiDaS network implements scale- and shift-invariant losses, allowing normalization of network predictions via a global average of prediction scale, shift, and magnitude. The normalization is implemented within the MiDaS loss function itself using a "multi-scale, scale-invariant gradient matching term", thereby alleviating the issue on unknown or varying scale across the ground truth depth maps at training time. The network was trained on the datasets DIML Indoor, MegaDepth, ReDWeb, WSVD, and 3D Movies, and it was tested on the datasets

DIW, ETH3D, Sintel, KITTI, NYUDv2, and TUM-RGBD. Tests on these datasets indicated that both MiDaS and MiDaS Small outperformed other state-of-the-art MDE models in terms of zero-shot performance [7].

We considered borrowing three different MiDaS models for this research project: MiDaS v2.1 Small, MiDaS v3 Hybrid, and MiDaS v3 Large. Based on outputs from Pytorch summaries, MiDaS v2.1 Small has a total of 16,600,929 parameters and an estimated memory overhead of 867.02 MB, MiDaS v3 Hybrid has a total of 120,753,921 parameters and an estimated memory overhead of 2184.29 MB, and MiDaS v3 Large has a total of 341,258,433 parameters and an estimated memory overhead of 3385.56 MB. Each of the models contains various layers including Conv2d, BatchNorm2d, ReLU6, Sequential, Residual, InvertedResidual, Identity, and a few custom convolutional layers. In all three models, convolutional layers make up the vast majority of the model's parameters. Due to the memory overhead of MiDaS v3 Large and MiDaS v3 Hybrid, we were unable to load the models into memory and perform inference. As a result, we chose to work with MiDaS v2.1 Small.

2.3 KITTI Depth Dataset

We chose the KITTI-Depth validation dataset for training and validation. KITTI is a widely implemented dataset for depth training and has been used to train popular models such as PixelFormer[1] and VA-DepthNet[5]. KITTI-Depth is a massive dataset that contains over 93,000 RGB and depth image pairs[10]. It is a dataset primarily created to train deep learning models for automated driving as it includes outdoors scenes such as cities, roads, and campuses. To ensure that we were able to store the whole dataset, we implemented the KITTI-Depth validation dataset which only consists of over 4,000 manually selected image pairs. We chose to use the validation set over the test set as the ground truth images were withheld from the test set for industry benchmarking purposes.

3 METHODOLOGY

In this section, we will discuss our approach to making the MiDaS model we selected "on-device". Specifically, we will discuss our efforts to reduce model size, memory overhead, FLOPs, and inference time. This will begin with an overview of the pruning and sparsification techniques we explored in Section 3.1. Following this, we will provide insight into our attempt to fine tune the model in Section 3.2. Finally, we will discuss deployment of the model on an NVIDIA Jetson Nano in Section 3.3.

3.1 Pruning

In order to reduce the number of parameters in the MiDaS v2.1 Small model and subsequently decrease the number of FLOPs required for inference, we implemented both channel pruning and fine grained pruning. Pruning reduces the number of parameters within a model by setting the parameters' values to zero or removing them from calculations entirely, dependent on the pruning technique. Channel pruning accomplishes this for entire channels within a network's structure, whilst fine grained pruning only prunes individual weights. Because channel pruning removes entire channels in structures, it leads to a significant speedup and reduction in number of FLOPs in the pruned network at the cost of faster degradation of network accuracy. This is relative to fine grained pruning, which results in forward/backward pass speedups with much lesser degradation in network accuracy, but does not affect the number of FLOPs or the memory overhead of the model without specialized hardware [3].

When performing pruning on the MiDaS model, we used off-the-shelf pruning functions from the torch.nn library. For channel pruning, we found the magnitude of each Conv2d layer and added it to a list, sorted least to greatest. We then truncated the list according to the desired percentage of layers to prune. For each layer in the list, we then used torch.nn's prune_l1.unstructured() function call to prune the entire layer by passing a pruning percentage of 1.0 (100%) to the function. We channel pruned the model for sparsity percentages .02, .04, .06, .08, .10, .12, .14, .16, .18, and .20. For fine grained pruning, we called the prune_l1.unstructured() function on all Conv2d layers in the model, passing the desired pruning percentage to the function. We fine grained pruned the model for sparsity percentages 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, .90, and 0.99.

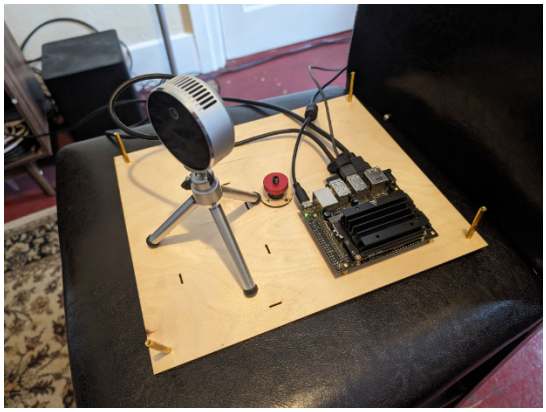
3.2 Fine Tuning

After sparsifying a network, it is common practice to fine tune the model in order to maintain similar accuracy despite the reduction in the number of significant weights [3]. In accordance with this, we attempted to fine tune the MiDaS v2.1 Small model after performing both fine grained pruning and channel pruning. To do this, we passed images to the model in batch sizes of two (the largest we could achieve with our GPU's memory restrictions). These images are preprocessed according to transforms predefined by MiDaS, including converting the images to three-channel RGB, resizing the images to (256, 256), converting the image data to a tensor, and then normalizing the tensor based on a specific standard deviation and mean. The ground truth depth maps are similarly converted to single-channel grayscale, resized to (256, 256), and converted to a tensor. After feeding the preprocessed images into the model and making a forward pass, we took the predictions from the model and found the MSE loss between the predictions and the ground truth. We then updated the model weights in a backward pass according to the loss.

3.3 NVIDIA Jetson Nano

To measure the effectiveness of our pruned model, we created the following experimental setup: we loaded each model onto an NVIDIA Jetson Nano, a small yet capable device designed for AI experimentation. Attached to the Jetson nano was an Intel Realsense L515 Lidar camera. The L515 is an RGB-D Lidar camera able to take incredibly precise depth images with an accuracy of 14mm up to 9 m^2[4]. Because of this, we were able to align the RGB data stream to the depth data stream, then use the RGB frames as input to the models After measuring inference time for a set of frames and taking the average inference time, we could compute the decrease in inference time of the pruned model when compared to the original model. Then using the lidar frames as a ground truth, we were able to measure both RMSE and MAE to calculate the accuracy drop of the pruned model.

Figure 1: Image of Intel Realsense L515 Lidar camera attached to NVIDIA Jetson Nano



4 EVALUATION

In this section, we discuss the results of our implementation efforts. We begin in section 4.1 with the RMSE and MAE results of our pruned models. Then in section 4.2, we discuss our on-device testing pipeline and present the baseline RMSE, MAE, and inference time of the unpruned MiDaS model.

4.1 Sparsifying the Model

After performing channel pruning on the original MiDaS v2.1 Small model, we achieved the approximate sparsities indicated in Table 1. Similarly, after performing fine grained pruning on the original MiDaS v2.1 Small model, we achieved the approximate sparsities indicated in Table 2. Table 1 also records the RMSE and MAE loss for the channel pruned models, while Table 2 records the RMSE and MAE loss for the fine grain pruned. Because our attempt to fine tune the pruned models was unsuccessful for reasons discussed in Section 5.3, the losses were calculated using the pruned models with no fine tuning applied. Finally, example outputs from the channel pruned models can be seen in Figure 2, while example outputs from the fine grain pruned models can be seen in Figure 3.

Sparsity	Num. Parameters (M)	RMSE Loss	MAE Loss
0.0 (baseline)	16.6	482.88	408.20
.02	16.3	482.88	408.20
.04	15.9	184.51	166.50
.06	15.6	177.07	165.24
.08	15.3	178.24	170.55
.10	14.9	114.84	114.23
.12	14.6	112.42	111.80
.14	14.3	113.16	112.69
.16	13.9	56.54	56.11
.18	13.6	1.33	1.33
.20	13.3	1.33	1.33

Table 1: Sparsities, parameters, and loss for channel pruned models

Sparsity	Num. Parameters (M)	RMSE Loss	MAE Loss
0.0 (baseline)	16.6	482.88	408.20
.10	14.9	475.89	402.20
.20	13.3	452.37	388.17
.30	11.6	296.49	265.65
.40	9.96	274.13	263.05
.50	8.3	148.50	141.23
.60	6.6	63.27	59.45
.70	5.0	30.24	28.45
.80	3.3	0.86	0.83
.90	1.7	1.92	1.91
.99	0.17	0.58	0.57

Table 2: Sparsities, parameters, and loss for fine grain pruned models

As can be seen in Table 1 and Table 2, the RMSE and MAE loss metrics decrease as sparsities increase, despite the pruned models not being fine tuned. This is a strong indication that our calculation of loss during inference is handled incorrectly. These numbers for loss are thus inaccurate, and cannot be used to draw any substantial conclusions. The issues calculating loss are further expanded upon in Section 5.3.

Figure 2: Inference at several levels of channel pruned sparsity

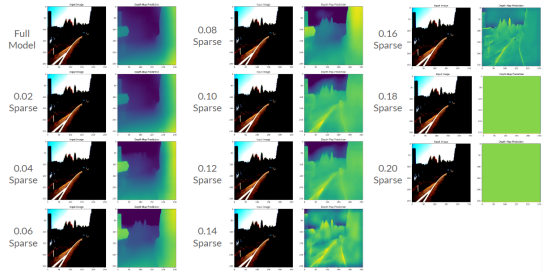
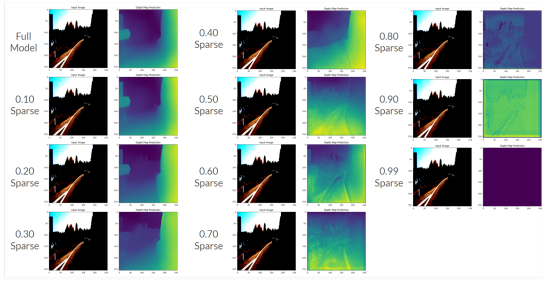


Figure 3: Inference at several levels of fine grain pruned sparsity



Despite the major flaw in the loss metrics displayed in Table 1 and Table 2, we observe a gradual degradation in the example outputs shown in Figure 2 for channel pruned models and Figure 3

for fine grain pruned models as sparsity increases. This indicates that our pruning methods are successful in sparsifying the model. Although the pruned models were not fine tuned, we thus felt that these models were acceptable to attempt to deploy for the sake of gathering latency metrics. However, this was barred by a conversion limitation discussed in Section 5.1 and Section 5.2.

4.2 Model Deployment

After loading MiDaS V2.1 Small onto the Jetson Nano, we created a simple testing pipeline that would continuously receive RGB and depth frames from the L515 camera. Once the frames were received, they would be aligned to ensure pixel-wise calibration between the two images. Then the RGB image would be passed in as input to the MiDaS model and an estimated depth frame would be created. Inference time would be measured to calculate the latency of the model. Then both RMSE and MAE were calculated using the ground truth depth frame and the estimated frame. A mask was used to remove pixels that were marked as invalid in the ground truth frame from the accuracy calculations. After a specified amount of frames, the average inference time, RMSE, and MAE were calculated. The original MiDaS model achieved an average inference time of 1.22 seconds, an average RMSE of 11321.31, and an average MAE of 10471.45. The validity of these accuracy numbers will be discussed in section 5.3.

Figure 4: Ground truth depth images captured from L515 camera

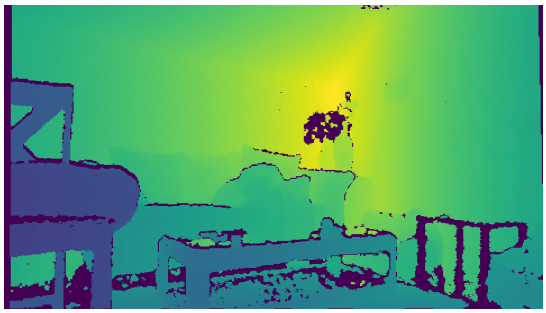


Figure 5: Depth image estimated from MiDaS V2.1 small

5 DISCUSSION, CHALLENGES, AND LIMITATIONS

In this section, we will discuss issues and challenges that we encountered during our efforts. In Section 5.1 we discuss deployment issues encounter with PyTorch models on the Jetson Nano. Following this, we review specific issues encountered involving the architecture of the MiDaS model and how it's loss function impeded progress in sections 5.2, and 5.3. Finally, we discuss challenges faced with implementing quantization and how we ran out of time to complete it in section 5.4.

5.1 Issues with PyTorch Libraries on Jetson Nano

While attempting to implement PyTorch on the Jetson Nano, we encountered multiple issues that prevented us from executing any PyTorch models on the device. Despite utilizing pre-built PyTorch binaries specifically distributed by NVIDIA for the aarch64 architecture of the Nano, we still faced operational difficulties. To avoid unproductive troubleshooting efforts, we transitioned to the TFLite version of MiDaS. This created another issue, however, as our pruning code was created for a PyTorch model and would be incompatible for a TFLite model. Although we lacked the time to redo our pruning and sparsification in TFLite, we believed this could be remedied by converting the pruned pytorch model into a TFLite model to be loaded onto the Nano.

5.2 Issues Transfering Model to TFLite

While attempting to convert our PyTorch model to a TFLite model using the ONNX (Open Neural Network Exchange) library. This library is capable of taking models such as ResNet50 from PyTorch or Tensorflow, and converting them to an intermediate .onnx format. This format can then be converted to a PyTorch or Tensorflow model as needed, including TFLite models [2]. Unfortuantely, ONNX does not yet have support for Squeeze version 13 and Unsqueeze version 13, thus making the library incompatable for use with the MiDaS model. As a result, we were ultimately unsuccessful at converting the PyTorch models to TFLite models, and were thus unable to deploy our pruned models on the Jetson Nano within the time we had remaining.

5.3 Issues Finetuning the MiDaS Model and Calculating Loss

Our fine tuning methodology discussed in Section 3.2 was grossly incorrect in accordance with the loss function of the MiDaS v2.1 Small model. While it is not uncommon for depth estimation networks to pass the ground truth as a single-channel gray scale image and calculate the MSE loss between this and the prediction, the MiDaS network was trained using an global average calculation of scale, shift, and magnitude to normalize the network's predictions before calculating the final loss, as discussed in Section 2.2 [7]. This naive error, made mainly due to our unfamiliarity with the problem domain, resulted in an unsuccessful effort to fine tune the model. In an attempt to remedy this, we searched for the original training pipeline for the MiDaS model. However, this training pipeline was

never released, thus making fine tuning the MiDaS model exceedingly difficult. For this same reason, we were unable to calculate the loss of the model, and all of our calculations of RMSE and MAE loss in section 4.2 are inaccurate and not representative of the model's performance.

5.4 Quantization and Lack of Time

While attempting to remedy the rest of the challenges discussed in this section, we ran out of time to quantize the MiDaS model. We did attempt to use PyTorch's off-the-shelf quantization function torch.quantization.quantize_dynamic(), but were unable to produce a working result within a timely manner. Ultimately, we decided that continuing to attempt to remedy the above challenges were of greater importance, and we shelved the idea of quantizing the network for after we had solved the fine tuning, model loss, and Jetson Nano deployment issues.

6 CONCLUSION

In this study, we attempted to achieve real-time monocular depth estimation at 30fps on an NVIDIA Jetson Nano through the use of the MiDaS V2.1 Small model. We implemented both channel pruning and fine grained pruning on the MiDaS model to decrease the computational complexity, and subsequently decrease the inference time of the model. However, several challenges impeded our efforts. Most notably, the training pipeline for MiDaS has not been released to the public, therefore we were unable to fine-tune our pruned models, drastically lowering their accuracies. Moreover, the PyTorch incompatibilities with the Nano and the TFLite conversion incompatibilities with the pruned models interfered with our ability to load and test the pruned models using our on-device testing pipeline. In future research, we recommend exploring models other than MiDaS with up-front fine tuning capabilities, and we recommend implementing in TFLite from the beginning if working with a Jetson Nano.

7 CONTRIBUTIONS

The division of report work is as follows:

Reese Haly wrote 1, 2.1, 2.3, 3.3, 4.2, 5.1, 6

Joshua Malcarne wrote 2.2, 3.1, 3.2, 4.1, 5.2, 5.3, 5.4

REFERENCES

[1] Ashutosh Agarwal and Chetan Arora. 2022. Attention Attention Everywhere: Monocular Depth Prediction with Skip Attention. arXiv:2210.09071 [cs.CV]

[2] Junjie Bai, Fang Lu, Ke Zhang, et al. 2019. ONNX: Open Neural Network Exchange. https://github.com/onnx/onnx.

[3] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. arXiv:2102.00554 [cs.LG]

[4] Intel 2021. *Intel RealSense LiDAR Camera L515.* Intel. Rev. 003.

[5] Ce Liu, Suryansh Kumar, Shuhang Gu, Radu Timofte, and Luc Van Gool. 2023. VA-DepthNet: A Variational Approach to Single Image Depth Prediction. arXiv:2302.06556 [cs.CV]

[6] Armin Masoumian, Hatem A. Rashwan, Julián Cristiano, M. Salman Asif, and Domenec Puig. 2022. Monocular Depth Estimation Using Deep Learning: A Review. *Sensors* 22, 14 (2022). https://doi.org/10.3390/s22145353

[7] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. 2020. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. arXiv:1907.01341 [cs.CV]

[8] Michael Rudolph, Youssef Dawoud, Ronja Güldenring, Lazaros Nalpantidis, and Vasileios Belagiannis. 2022. Lightweight Monocular Depth Estimation through Guided Decoding. arXiv:2203.04206 [cs.CV]

[9] Beau Tippetts, Dah Jye Lee, Kirt Lillywhite, and James Archibald. 2016. Review of stereo vision algorithms and their suitability for resource-limited systems. *Journal of Real-Time Image Processing* 11, 1 (Jan. 2016), 5–25. https://doi.org/10.1007/s11554-012-0313-2

[10] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. 2017. Sparsity Invariant CNNs. In *International Conference on 3D Vision (3DV).*

[11] Wofk, Diana and Ma, Fangchang and Yang, Tien-Ju and Karaman, Sertac and Sze, Vivienne. 2019. FastDepth: Fast Monocular Depth Estimation on Embedded Systems. In *IEEE International Conference on Robotics and Automation (ICRA).*