LightDepth: Lightweight Monocular Depth Estimation

CS 7180 Advanced Perception Project Report

Author: Sukhrobbek Ilyosbekov
Date: October 17, 2025

Abstract

This project presents LightDepth, a simple lightweight monocular depth estimation model inspired by Depth Anything V2, LightDepth implements a simplified encoder-decoder architecture using ResNet18 as the backbone, achieving competitive performance with significantly fewer parameters (14.3M vs 24.8M). The model was trained on the NYU Depth V2 dataset using L1 loss with automatic mixed precision training. Evaluation results demonstrate that LightDepth achieves an RMSE of 2.948 meters on the test set, comparable to the more complex Depth Anything V2 model (RMSE: 2.645), while maintaining a 42% reduction in model parameters. The project successfully demonstrates that effective depth estimation can be achieved with simpler architecture, making it more accessible for resource-constrained devices.

Introduction and Prior Work

Monocular depth estimation - predicting the 3D structure of a scene from a single RGB image - is a one of the core problems in computer vision with applications in robotics, autonomous driving, augmented reality, and 3D scene reconstruction. This project was inspired by **Depth Anything V2** (Yang et al., 2024), a state-of-the-art monocular depth estimation model that achieves robust generalization across diverse scenes.

LightDepth is a lightweight depth estimation model. While Depth Anything V2 demonstrates impressive performance, its complex architecture can be challenging for learning about depth estimation. LightDepth addresses this by providing a simplified, easy-to-understand implementation that captures the core concepts of monocular depth estimation without the overwhelming complexity of state-of-the-art models. By using a straightforward encoder-decoder structure with fewer components and clearer code organization, LightDepth serves as an educational project for understanding how depth estimation networks work.

Methods

Model Architecture

LightDepth implements a U-Net style encoder-decoder architecture optimized for depth estimation:

The architecture consists of three main components:

1. **Encoder**: ResNet18 backbone for feature extraction

2. **Decoder**: Progressive upsampling with skip connections

3. **Head**: Final prediction layer with positive depth constraints

1. Encoder: ResNet18 Backbone

The encoder uses a pretrained ResNet18 (ImageNet weights) modified for feature extraction at multiple scales:

Layers:

- **Layer 0**: Initial convolution (3→64 channels) + BatchNorm + ReLU

- **Layer 1**: MaxPool + First residual block (64→64 channels)

- **Layer 2**: Second residual block (64→128 channels)

- **Layer 3**: Third residual block (128→256 channels)

- **Layer 4**: Fourth residual block (256→512 channels)

The encoder outputs 5 feature maps at different spatial resolutions:

- f_0: [64, H/2, W/2]

- f_1: [64, H/4, W/4]

- f_2: [128, H/8, W/8]

- f_3: [256, H/16, W/16]

- f_4: [512, H/32, W/32]

2. Decoder: Progressive Upsampling

The decoder progressively upsamples features while integrating skip connections from the encoder:

Architecture:

Stage 1: [512, H/32, W/32] + skip[256] → [256, H/16, W/16]

Stage 2: [256, H/16, W/16] + skip[128] → [128, H/8, W/8]

Stage 3: [128, H/8, W/8] + skip[64] → [64, H/4, W/4]

Stage 4: [64, H/4, W/4] + skip[64] → [32, H/2, W/2]

3. Prediction Head:

[32, H/2, W/2] → Conv(3×3) → ReLU → Conv(1×1) → ReLU → Output (1, H, W)

The final ReLU ensures positive depth predictions.

CLI commands

The project includes PDM scripts accessible via **pyproject.toml** and these commands are used for training and model evaluation. Also, there is a **config.yaml** file used for modifying parameters for train script

- **pdm train** command executes training for NYU Depth V2 dataset using default parameters from the **config.yaml** and epoch 50.
- **pdm eval** command executes a model inference on the NYU Depth V2 test dataset folder where it contains images for evaluation. It will show metrics results such as RMSE, MAE, absolute relative error, and squared relative error.
- **pdm infer** command executes a model inference for a single image and saves output in the **output** folder with depth image and numpy array file.
- **pdm compare** command executes a script **compare_models.py** which runs both LightDepth and DepthAnything V2 small models and compares results. It also produces samples of 5 image pairs (pair of 4 images: input, ground truth, DepthAnything V2 and LightDepth results) in the output folder.

These commands use Python scripts defined in the **scripts** folder. The full list of command parameters is documented in these files (train.py, infer.py, compare_models.py and eval.py).

Results

The model was evaluated on standard error metrics such as RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), AbsRel (Absolute Relative Error), and SqRel (Squared Relative Error). All metrics are computed only on valid depth pixels as indicated by the ground truth mask. The pretrained model (**best_model.pth**) was trained on the NYU Depth V2 dataset with default parameters (image width = 640, image height = 480) and 50 epoch. It took approximately 4 hours on RTX 5080 16 GB VRAM GPU. The training script supports AMP optimization to speed up training.

Model evaluation

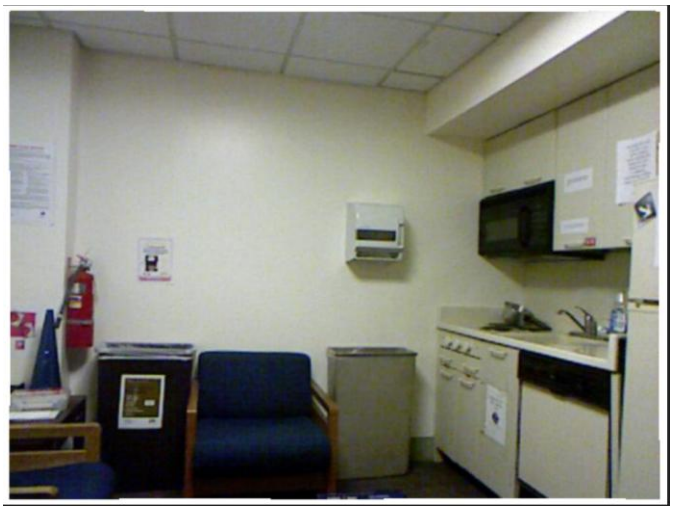
Model evaluation results compared to DepthAnything V2 small model (the lower results, the better). The NYU V2 test dataset containing 654 samples was used for evaluation.

Metric	**LightDepth**	**Depth Anything V2**	**Winner**
Parameters	14,330,369	24,785,089	LightDepth
RMSE	2.9477	2.8074	DepthAnything V2 (5.0%)
MAE	2.6758	2.2360	DepthAnything V2 (19.7%)
Absolute Relative Error	0.9724	1.0272	LightDepth (5.3%)
Squared Relative Error	2.6063	3.7849	LightDepth (31.1%)
Total inference time	5 seconds	18 seconds	LightDepth (72.2%)

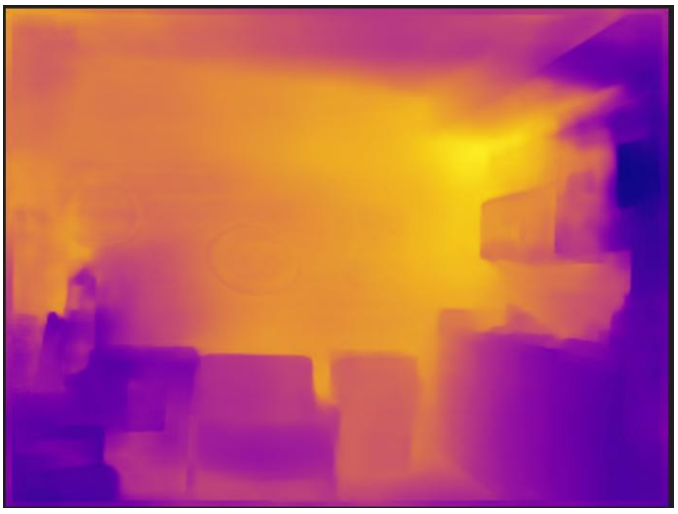
Sample Results:

Below are samples of depth estimation results from NYU V2 dataset images, compared to results from LigthDepth and DepthAnythingV2 to a ground truth.

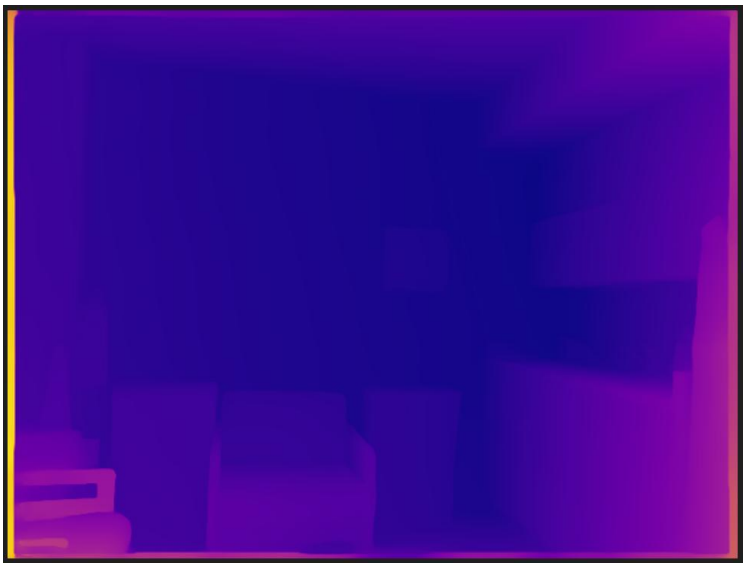
Input (nyu2_test/00000_colors.png):



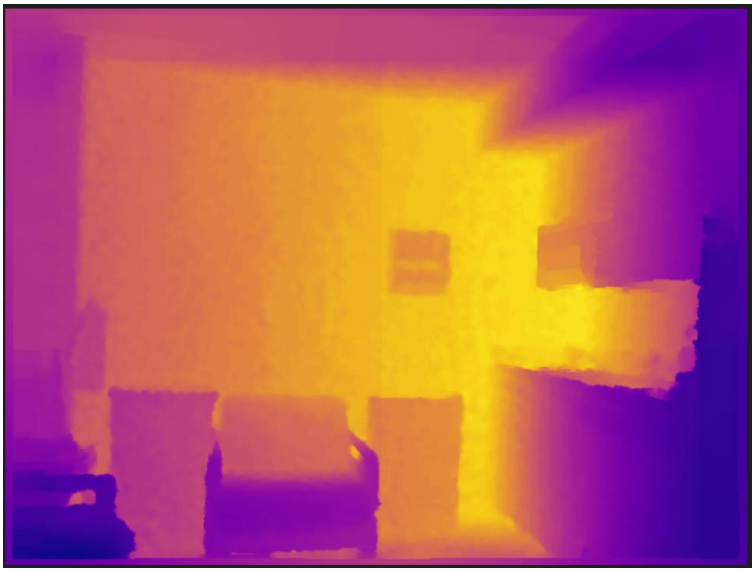
LightDepth:



DepthAnything V2:



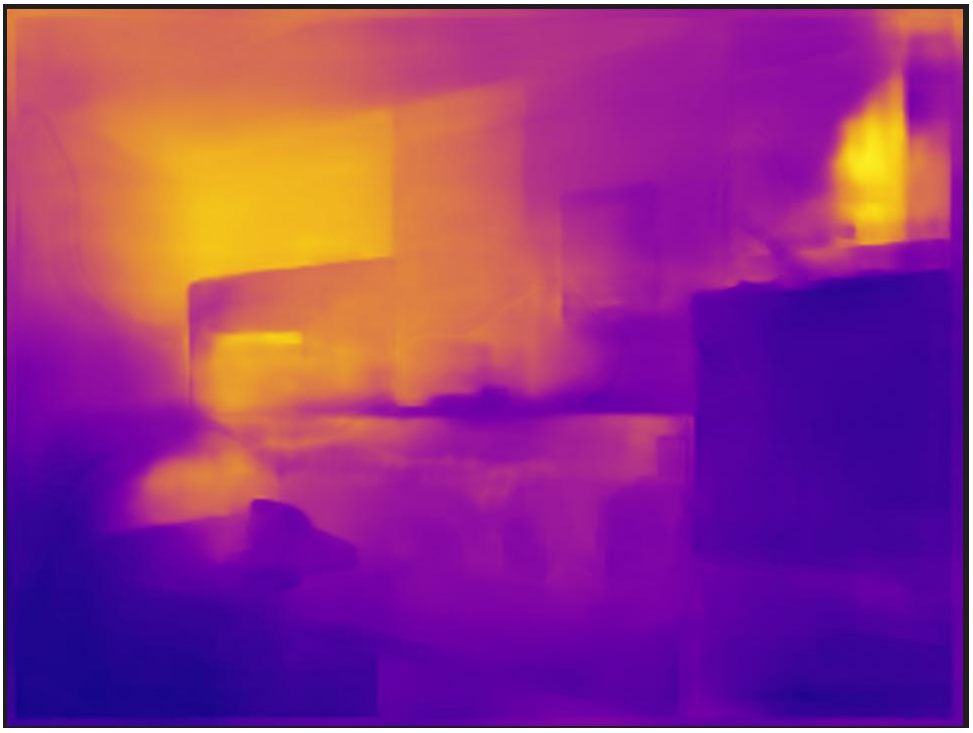
Ground truth:



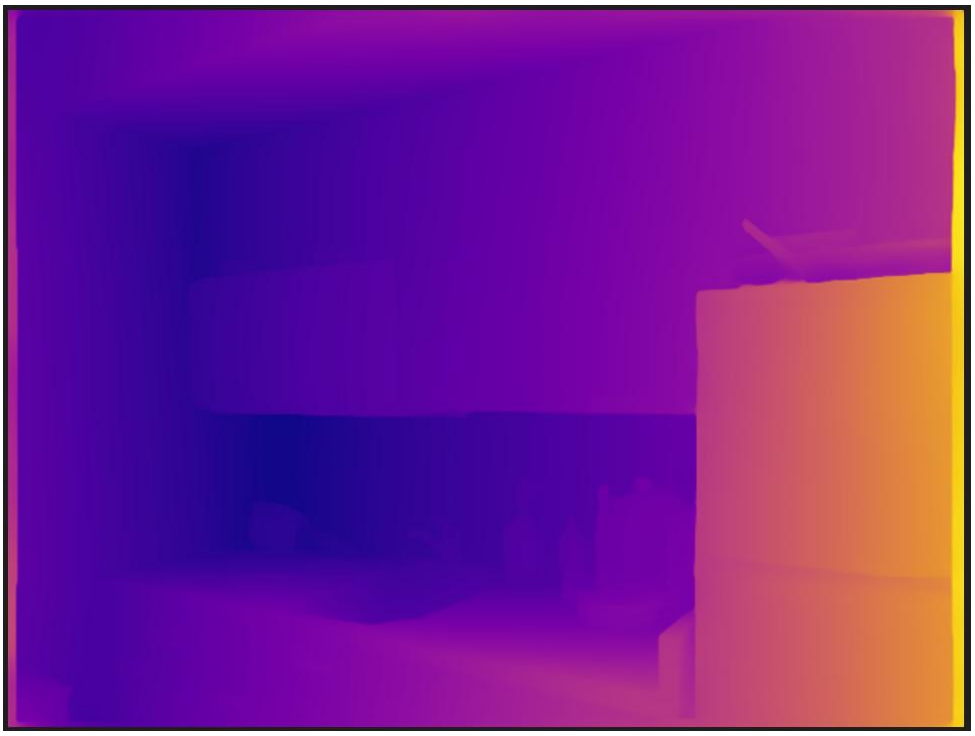
In first image results, we can see that our LightDepth generated depth estimation closer to ground truth compared to DepthAnythingV2. The inference time was significantly short compared to DepthAnythingV2 while producing closer results to ground truth image.

Input (nyu2_test/00001_colors.png):

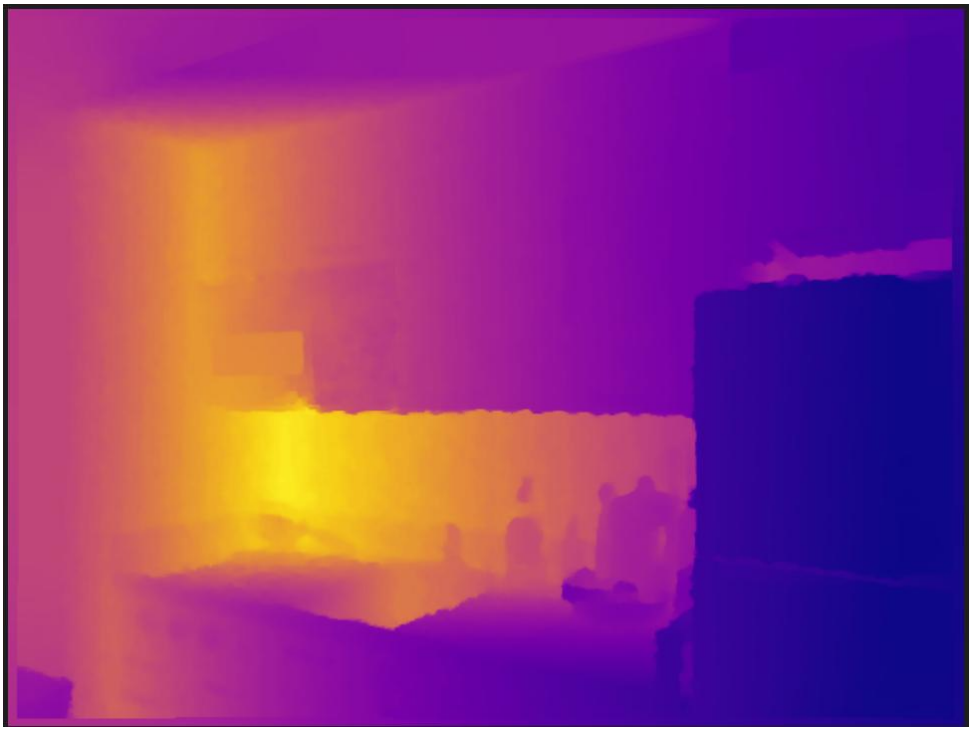
LightDepth:



DepthAnything V2:



Ground truth:



In the second image, we can observe different results, the DepthAnythingV2 does better job compared to LightDepth. It produces more accurate and sharp depths. While the LightDepth struggles to handle image edges especially the bottom right of the image.

Model strengths:

- **Small parameters:** The model uses significantly fewer parameters (42% reduction), while remaining comparable to DepthAnything V2.
- **Fast inference**: The model has significantly less inference time compared to DepthAnything V2 and can be used for resource constraint environments such as mobile devices.
- **Edge Preservation**: Skip connections effectively preserve fine-grained spatial details.

Model limitations:

- **Absolute Scale:** Slightly less accurate on absolute depth values compared to larger models.
- **Complex Scenes**: May struggle with very cluttered or textureless regions.
- **Fine Details:** Small objects may have less accurate depth boundaries.

Reflection

This project taught me a lot about how depth estimation actually works. I spent time studying the Depth Anything V2 model to understand how it predicts depth from regular photos. I learned some useful training tricks like using AMP to speed things up and reduce memory usage, and I saw firsthand how using a pretrained encoder really helps instead of training everything from scratch. The biggest challenge was working with limited computing power - I couldn't train on as much data or for as many epochs as I wanted to. If I had access to better hardware and more training time, the model would probably perform much better. Still, building this project from the ground up gave me a solid understanding of how these depth estimation models work and what goes into training them.

Acknowledgements

This project was completed with assistance from various resources:

- **PyTorch Documentation**: Essential for understanding framework capabilities. https://docs.pytorch.org/docs/stable/index.html
- **DepthAnything V2 Paper**: https://arxiv.org/abs/2406.09414
- **DepthAnything V2 Website:** https://depth-anything-v2.github.io
- **NYU Depth V2**: data set is comprised of video sequences from a variety of indoor scenes as recorded by both the RGB and Depth cameras. https://www.kaggle.com/datasets/soumikrakshit/nyu-depth-v2

- **PDM**: Setting up PDM for dependency management and development environment. https://pdm-project.org/en/latest

Project Extensions

- **Training Optimization with AMP**: I implemented Automatic Mixed Precision (AMP) training to optimize both memory usage and training speed. This required understanding float16/float32 precision trade-offs and implementing gradient scaling to prevent underflow issues. The AMP implementation reduced training time while maintaining model accuracy.
- **Comparative Evaluation Against DepthAnything V2**: I conducted a systematic comparison between LightDepth and Depth Anything V2 on standard metrics (RMSE, MAE, AbsRel, SqRel).
- **Multiple Output Formats and Visualization**: I extended the inference pipeline to support multiple depth visualization formats (grayscale, various colormaps like plasma, viridis, magma) and implemented both visual and numerical depth outputs.
- **Codebase Quality**: I implemented high-quality codebase that follows a clear structure **scripts/ src/** layout, uses **PDM** for managing dependencies, **Black, isort** for formatting.