Final Project – Monocular Depth Estimation

Pranay Katyal, Anirudh Ramanathan

Problem Statement

Monocular Depth Estimation from Drone Camera Images

Overview

- The goal of this project is to estimate depth from images captured by a monocular camera onboard a drone. Monocular depth estimation is a challenging problem in computer vision, which this project will address using deep learning methods.
- Reproduce the results from the paper:
  - M. Folder et al., "Parallax Inference for Robust Temporal Monocular Depth Estimation in Unstructured Environments", 2022.
    - This technique is unique because it **fuses sequential camera images** with corresponding **camera translation and rotation measurements (from GPS/IMU)** to **produce metric depth estimates**. The authors have open-sourced their code, which is implemented in TensorFlow.

Goals

- The project has two main phases:
  - **Synthetic Data Training:** We will first follow the instructions in the official GitHub repository to train the network on the **MidAir dataset**, a **synthetic dataset**. The objective is to **replicate the performance** reported in the paper.
    - https://midair.ulg.ac.be/
  - **Real-World Data Fine-Tuning:** Next, we will use the **pre-trained model** and **fine-tune** it on the **UseGeo dataset**, which consists of **real-world images**. The final objective is to predict depth for images in the UseGeo dataset and compare your predictions to the ground-truth depth maps.
    - https://github.com/3DOM-FBK/UseGeo/tree/master

Dataset - MidAir Synthetic Dataset

- **Mid-Air**, The *Montefiore Institute Dataset of Aerial Images and Records*, is a multi-purpose synthetic dataset for low altitude drone flights. It provides a large amount of synchronized data corresponding to flight records for multi-modal vision sensors and navigation sensors mounted on board of a flying quadcopter. They also provide Train and Test Data.
- Due to the sheer size of their dataset, we had to select a fragment (**Trajectory0000**) one we want to download. Their download procedure works as follow:
  - Select the desired visual sensors;
  - Select the desired training data, *i.e.* trajectories and climate setups;
  - Select desired benchmarks;
  - Submit the form.
- Image Type Options:
  - Left RGB, Right RGB, **Down RGB**, Surface Normals, **Depth**, **Stereo Disparity,** Stereo Occlusion, Semantic Segmentations.

Dataset - UseGeo Real Dataset

- **UseGeo,** is the new large-scale real-world set of data Depth estimation, Multiview 3D Reconstruction, etc.
- Number of Available Datasets
  - **3 Total Datasets** available.
  - Approximately **100GB worth of Data**.
  - **Images**, **Depths**, Dense point clouds.
  - **Image Size** : 7952 x 5304 px
  - **Camera Poses** : X0,Y0, Z0, Omega, Phi, Kappa, **Camera Intrinsics.**
- Flight Specifications
  - **Area Covered:** 1100 × 650 meters
  - **Average Flight Height:** 80 m above ground
  - **GSD (Ground Sampling Distance):** approx. 2 cm
  - **Image Overlap:** 80% (forward) – 60% (side)
  - **LiDAR Point Cloud Density:** ~50 points/m²

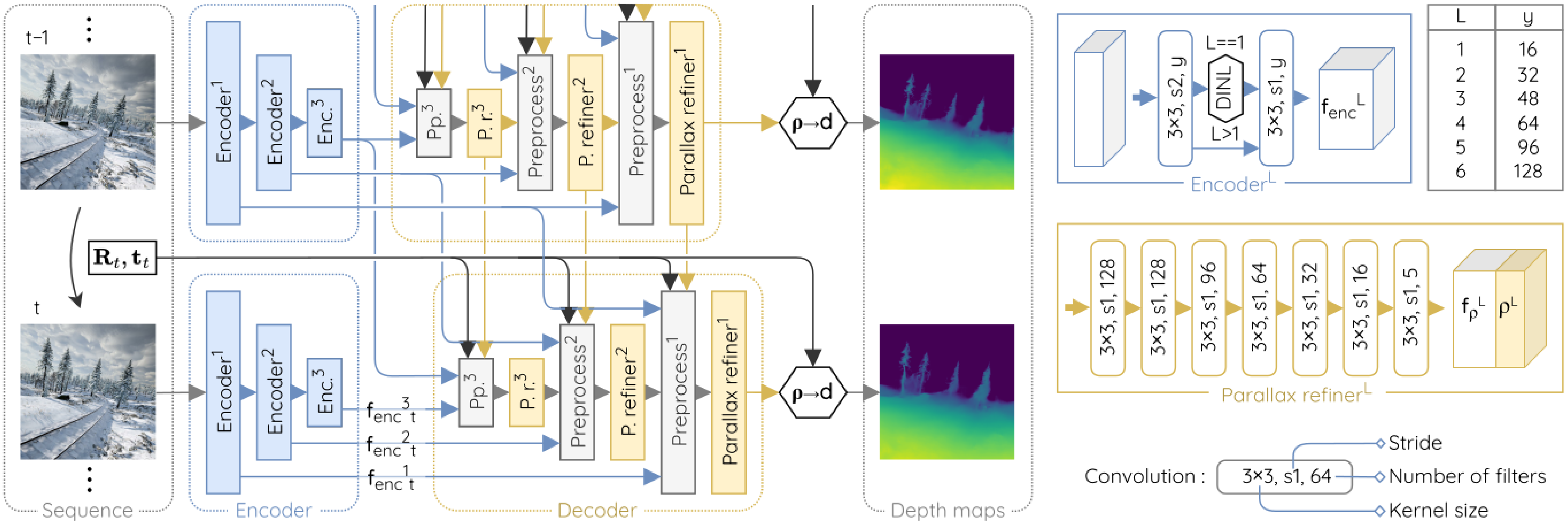
Dataset - Summary

- **MidAir:** 550 synthetic aerial images, relative poses
  - Gained from Trajectory0000 Alone!
- **UseGeo:** 828 real drone images, absolute GPS poses
  - All 3 Datasets on GitHub combined.
- **Train/Validation Split:** 80/20 for UseGeo
  - We split across each Dataset rather than a 2:1 Training and Eval split.

M4Depth Architecture

- **Feature Pyramid Encoder**
  - Extracts hierarchical features at 6 resolution levels
  - Uses Domain-Invariant Normalization (DINL)
- **Temporal Depth Estimator**
  - **Temporal Recurrence:** Warps features from previous frames using camera motion.
  - **Cost Volume:** Correlates features across frames to find correspondences.
  - **Depth Refinement:** Multi-scale processing for accurate predictions
- **Parallax to Depth Conversion**
  - Converts network output (parallax/disparity) to metric depth.
  - **Critical:** Uses camera motion (translation & rotation) via parallax2depth().
  - Formula: depth = f(disparity, camera_motion, intrinsics)
- **Key Insight:** Model fundamentally depends on accurate pose information!

M4Depth Architecture



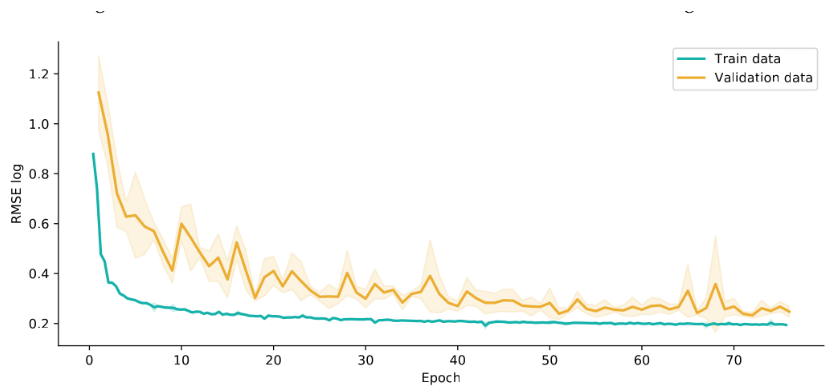
Phase 1 – Results for Pretrained Model Performance on MidAir

- Evaluation Setup:
  - Pretrained M4Depth model (71 epochs on MidAir training data)
  - Evaluated on MidAir trajectory0000 validation set (550 samples)
  - Same domain as training (synthetic aerial imagery)
- Performance Metrics:

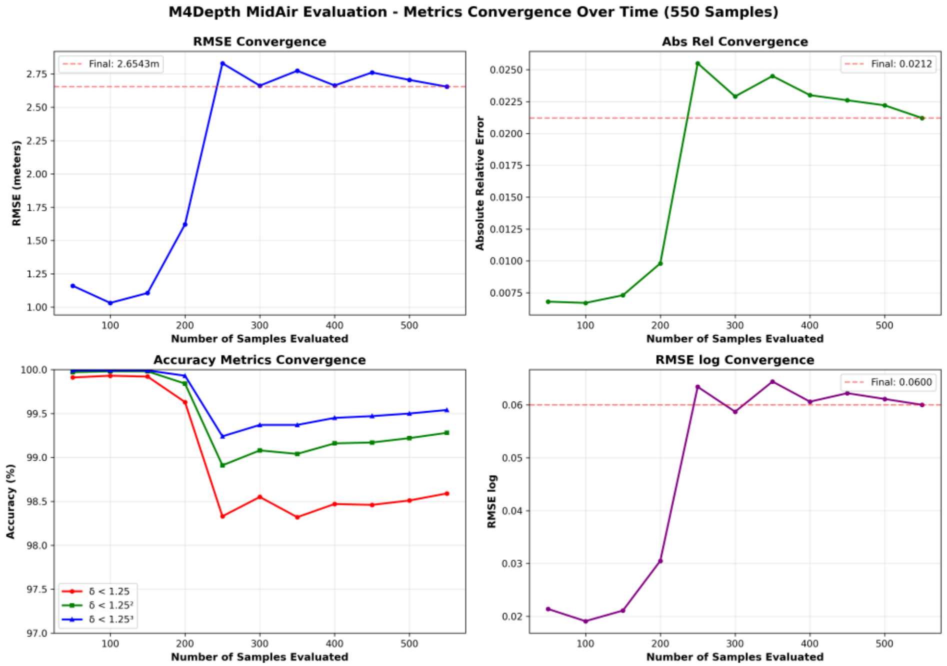
Metric	Value	Interpretation
RMSE	2.65m	Average depth error
Abs Rel	0.021	2.1% relative error
$\delta < 1.25$	**98.6%**	Near-perfect accuracy

Phase 1 – Results for Pretrained Model Performance on MidAir

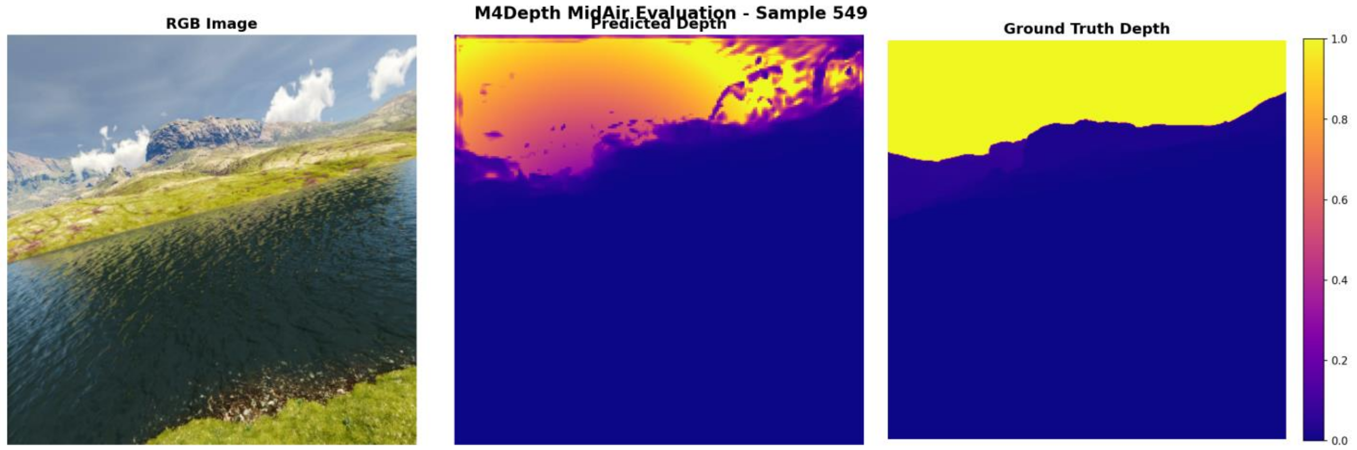
- Training Plot:

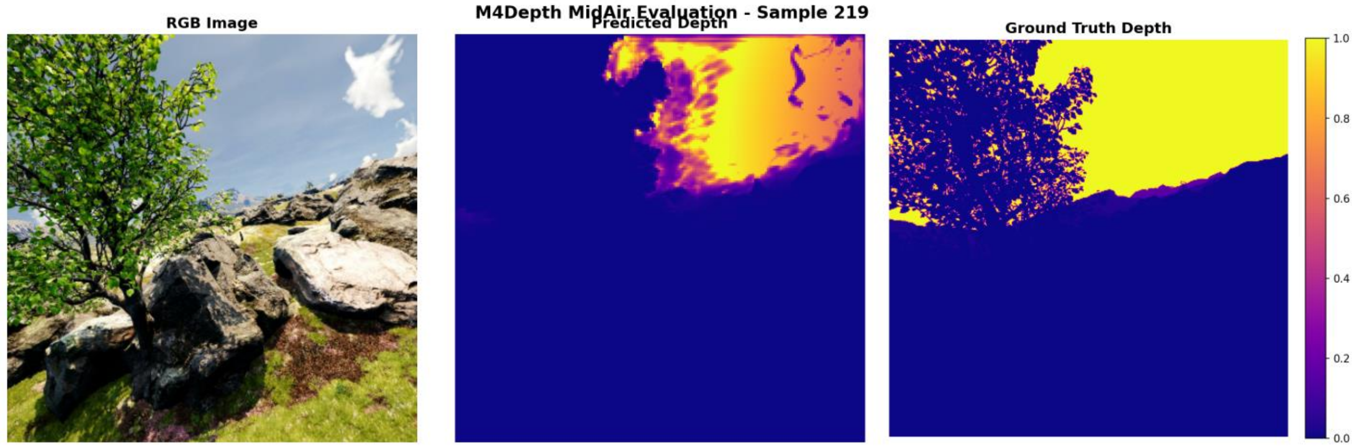


- Phase 1 Evaluation:



Phase 1 – Results - 2 of the 10 Samples





- Each row: RGB | Predicted Depth | Ground Truth
- The base model – from their **Checkpoint 71**
- Good performance!

Phase 2 – Results for Training on Real-World UseGeo Data

- Evaluation Setup:
  - Custom DataLoader for UseGeo (828 real drone images)
  - 80/20 split: 661 train, 167 validation
  - 146 epochs, ~3 hours training
  - **Note:** Pretrained checkpoint failed to load → trained from scratch
- Observations:
  - Dramatic metric improvement
  - Training converged smoothly (see curves)
  - **BUT:** Visual inspection revealed problems...
- Performance Metrics:

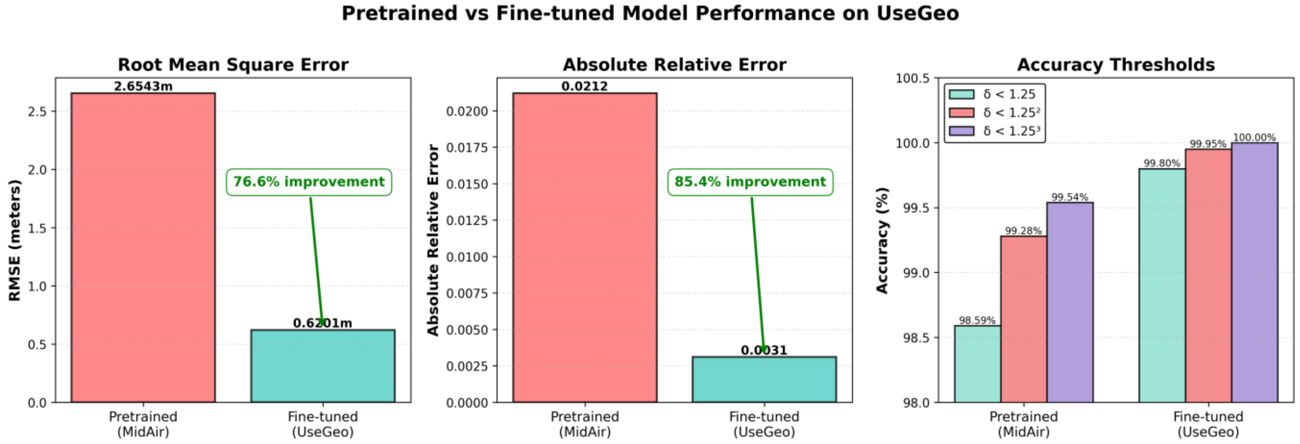
Metric	Pretrained (MidAir	Fine-tuned (UseGeo	Improvement
RMSE (m)	2.6543	0.6201	76.6% ↓
Abs Rel	0.0212	0.0031	85.4% ↓
$\delta < 1.25$ (%)	98.59	99.80	1.21% ↑
$\delta < 1.25^2$ (%)	99.28	99.95	0.67% ↑
$\delta < 1.25^3$ (%)	99.54	100.00	0.46% ↑

WPI

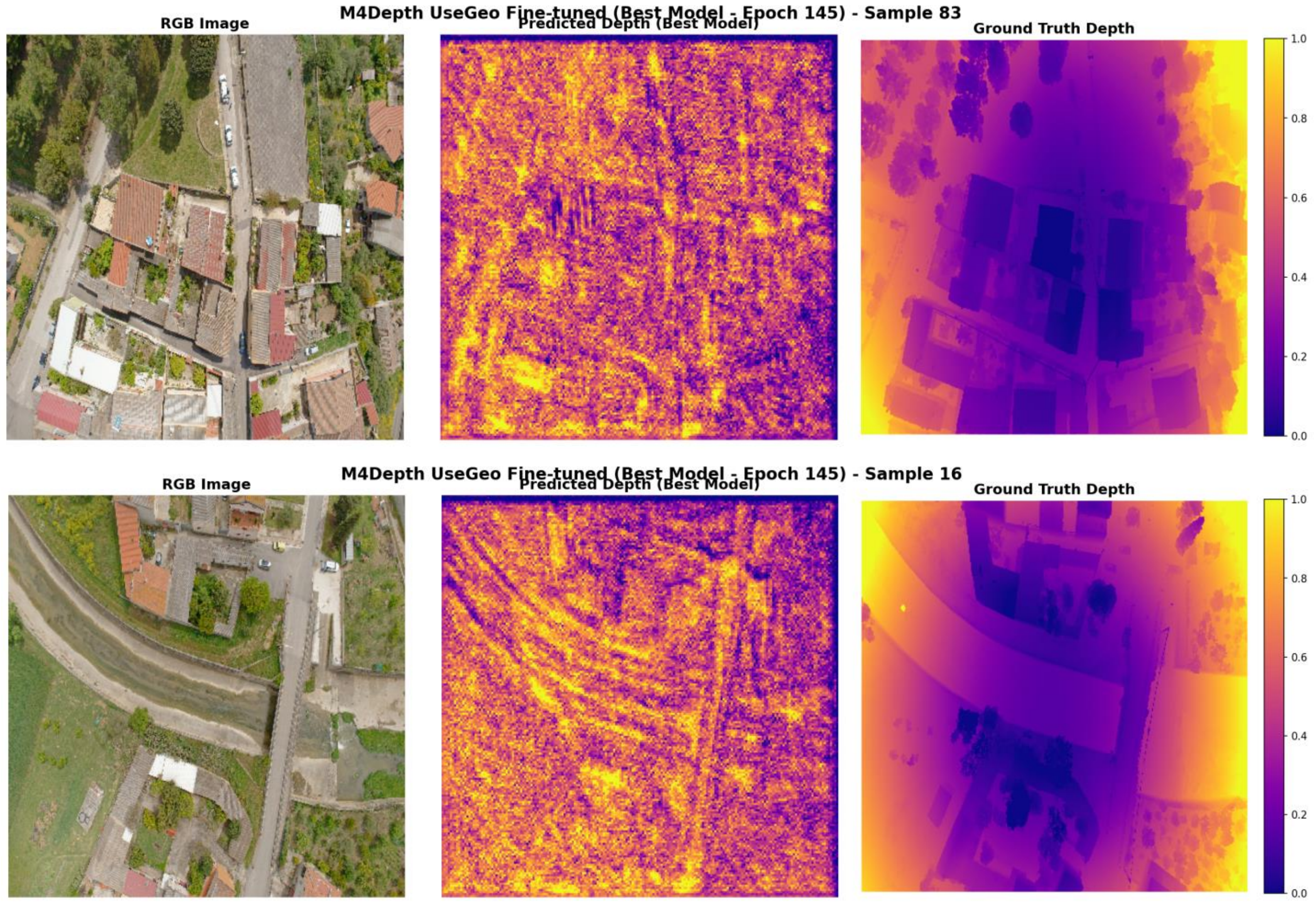
Phase 2 – Results for Training on Real-World UseGeo Data

- Training Plot:

- Phase 2 Evaluation:



Phase 2 – Results - 2 of the 10 Samples



- Each row: RGB | Predicted Depth | Ground Truth
- Tuning didn't work so we trained from scratch using 80-20 split.
- Model **Checkpoint 145** used
- Okayish performance!
  - 2 main reasons
    - Low amount of Data
    - Model Architecture issues possibly.

The Problem: Pose Format Incompatibility

- What We Discovered:
  - Depth predictions were **200 million meters** instead of 60-140m!
- Root Cause: Pose Representation Mismatch
- Scale difference: ~2,000,000×
- Impact on M4Depth:
  - `parallax2depth()` expects relative motion (meters)
  - Received absolute GPS coordinates (hundreds of kilometers!)
  - Internal calculations: depth = f(disparity, translation)
- Result: Predictions in millions of meters

Dataset	Pose Format	Translation Values
MidAir	Relative motion	0-20 meters
UseGeo	Absolute GPS	~498,340 meters

WPI

The Paradox: Good Metrics, Bad Predictions

- Metrics Computed on Wrong Tensors
  - Evaluated on **internal network representations**
  - NOT on final metric depth after `parallax2depth()`
  - Network outputs had correct *relative patterns.*
- Training in Log-Space
  - Loss: log(200M) - log(100) = finite, trainable gradient
  - Model learned consistent patterns despite wrong scale
  - Log-space masks absolute magnitude errors
- Pattern Matching Without Scale
  - Network learned: "building closer than Roads"
  - But absolute values: millions of meters!
  - Relative ordering correct, absolute scale wrong

Debugging Attempts: What We Tried

- **Solution 1: Pose Conversion Tool**
  - Created `convert_usegeo_poses.py`
    - Converted absolute GPS → relative motion
  - Algorithm: `t_rel = R_prev^T × (t_curr - t_prev)`
  - **Result:** Mean 16.7m, median 12.8m (correct range!)
- **Solution 2: Fine-tune with Relative Poses**
  - Loaded MidAir checkpoint + relative pose CSVs
  - **Result:** Training crashed at batch 47 with NaN loss
- **Solution 3: Lower LR + Gradient Clipping**
  - Learning rate: 0.0001 → 0.00001 (10× lower), and Added gradient clipping (clipnorm=1.0)
  - **Result:** Still crashed at batch 47 with NaN
- **Solution 4: Train from Scratch with Relative Poses**
  - Violates assignment: must use pretrained weights
  - **Result:** Doesn't work well due to lack of data 350GB.

WPI

Lessons learnt

- Transfer Learning Requires Complete Data Compatibility
  - Not just image domain adaptation
  - ALL input representations must match
  - Pose format as critical as visual features.
  - Hidden dependencies in pretrained weights
- Visual Validation is Essential
  - Metrics can mask fundamental failures
  - Always inspect actual outputs, not just numbers
  - Check value ranges for physical plausibility
- Systematic Debugging Reveals Root Causes
  - Traced through: metrics → depth values → pose format
  - Multiple solution attempts confirmed hypothesis

Conclusions & Future Directions

- Does pre-training on synthetic MidAir help, hurt, or have no effect on real UseGeo images?
  - We Cannot Definitively Conclude Transfer Learning Effectiveness
    - Pose format incompatibility **prevented** proper transfer
    - Phase 2 trained **from scratch**, not fine-tuned
  - What We Can Say:
    - **Pose representation compatibility** is a prerequisite for transfer
    - **Transfer learning requires** matching ALL input formats, not just images
    - **From-scratch training worked**, showing architecture is sound
- **Expectation**:
  - IF pose formats matched → pretrained weights likely would help (Similar scene types, motion patterns, depth estimation task) just like it did when we did Transfer Learning on CNN images.

Thank You!