MCEN 5228: Project 2 - Coded Deep Depth

Rama Chaganti
rama.chaganti@colorado.edu
Team Roronoa
using 1 late day

Srikanth Popuri
srikanth.popuri@colorado.edu
Team Roronoa
using 1 late day

***Abstract*—The project focuses on simulating coded image generation and deep depth estimation using monocular RGB images and depth maps. The first part involves simulating coded images by applying a phase mask's Point Spread Function (PSF) to RGB images and corresponding depth data. The second part evaluates monocular depth estimation methods using datasets like NYUv2 and UMDCodedVO-LivingRoom, testing UMDCodedVO-DiningRoom datasets with both AiF and coded images. Quantitative and qualitative evaluations include Rel-Abs and RMSE metrics and color schemes.**

I. Introduction

Monocular depth estimation and coded imaging are crucial in computer vision, particularly in applications requiring depth-aware data from 2D images. This project utilizes two main phases to develop a depth estimation pipeline: simulating coded images through phase masks and training depth estimation models on coded and pinhole camera models. Results are evaluated to assess the reliability of the coded imaging approach. The project explores these methods' effectiveness, precision, and generalizability across different data distributions. link to code part 2, link to code part 1

II. Phase 1

A. Part 1: Coded Image Generation

This phase involves generating a simulated coded image from a given RGB image, ground truth depth data, and a Point Spread Function (PSF) associated with the phase mask. The process consists of several steps to load and preprocess the input images, apply the PSF, and save the resulting coded image.

1) Methodology: A phase mask's PSF is applied to RGB images and corresponding depth maps to generate coded images. The data includes an RGB image, a metric depth image, and a downloadable PSF file from CodedVO's repository. Using these, the method blurs the original RGB image based on the PSF and depth information. Below are the steps.

a) GetDepthLayers: The function get_depth_layers is designed to discretize a continuous depth map into predefined depth layers. This process quantizes depth values to facilitate depth-aware processing. Given an input depth map tensor and a predefined set of depth layer thresholds, the function produces a stack of binary masks, each representing pixels within a specific depth range.

The function operates as follows:

- *Depth Quantization:* Using `torch.bucketize`, each depth value in `depth_map` is assigned to a discrete depth layer based on the defined thresholds in `depth_layers`. The result is a quantized depth tensor.
- *Binary Masks Creation:* For each depth layer, a binary mask is generated, where pixels falling within that specific depth layer are marked as `1.0` and others as `0.0`. These masks are appended to a list of tensors.
- *Layer Stacking:* Finally, all binary masks are stacked along a new dimension, producing a 3D tensor where each slice represents a specific depth layer. This stacked tensor is then returned as the output.

This function allows depth information to be segmented and processed layer by layer, which is essential for coded imaging applications that rely on depth-dependent processing.

b) SinglePSFConvolution: The function performs a convolution operation between an input image and a specified Point Spread Function (PSF) for a single depth layer and color channel. This approach allows depth-dependent convolution to simulate depth-aware blurring effects in the image.

The function works as follows:

- *Depth and Channel Selection:*
  The function takes `depth_idx` and `channel_idx` as input parameters, which specify the particular depth layer and color channel of the PSF to be used for convolution.
- *Convolution Operation:*
  The `torch.nn.functional.conv2d` function is used to perform the convolution. The selected PSF layer is applied to the input `image` tensor, with padding specified by the `padding` parameter to control the output size.

The function outputs a tensor representing the result of convolving the image with the selected PSF, retaining the input image's shape of (N, C, H, W), where N is the batch size, C is the number of channels, and (H, W) are the height and width of the image. This depth-specific convolution is crucial in generating realistic depth-dependent blur effects.

c) Linear Convolution: The `linear_convolution` function performs a depth-dependent convolution on an input image by using a set of Point Spread Functions (PSFs) corresponding to different depth layers, enabling a depth-aware blur effect. The function applies specific PSFs at different depth levels across the image, thus producing a realistic blur effect that varies with depth.

The depth-aware convolution process can be mathematically represented by the following linear model:

$$b(\lambda) = \sum_{k=0}^{K-1} \text{PSF}_k(\lambda) * I_k(\lambda) + \eta \quad (1)$$

where $b(\lambda)$ is the final blurred image at wavelength λ, $I_k(\lambda)$ represents the input RGBD image quantized into K discrete depth layers, and $\text{PSF}_k(\lambda)$ denotes the Point Spread Function applied at each depth layer k. Here, $k = 0$ corresponds to the furthest layer, and η represents additional noise or error. This equation illustrates how each depth layer is convolved with a specific PSF and then summed to generate the final depth-aware effect.

The function proceeds through the following key steps:

- *Depth Mask Generation:* Using the `get_depth_layers` function, binary masks are created from the `depth_map` based on the predefined `depth_layers`. Each binary mask identifies pixels that belong to a specific depth layer, facilitating the selective application of PSFs.
- *Channel-wise Convolution:* For each depth layer, depth-dependent convolution is applied separately to the red, green, and blue channels of the input image. The process involves:
  1) Extracting each color channel from the input image.
  2) Applying convolution using `torch.nn.functional.conv2d` with the appropriate PSF for each color channel.
  3) Multiplying the resulting convolved output with the corresponding binary depth mask.
- *Summing Across Depth Layers:* To form the final depth-aware effect, the convolved results across all depth layers are summed for each color channel. This ensures that each channel of the output image reflects the combined effects of depth-specific blurring.

The final output of `linear_convolution` is an image tensor of the same shape as the input, but with depth-dependent blur effects integrated across each channel. This depth-specific processing makes the approach suitable for applications that require realistic depth-based image processing. For more detailed notation and background, refer to Equation 1 in the cited work.

In this project, the functions `capture_image` and `process_folder` are already provided to facilitate depth-dependent convolution and batch processing of images and depth maps, respectively.

2) Results: The output is a coded RGB image that captures the depth-dependent blur effect. The generated image is visually examined to ensure the coded effect aligns with the depth variations in the scene. The above functions were applied to two datasets: `LivingRoom1` and `nyu_data`. The following results were obtained:

- *LivingRoom1 Dataset:* The function processed 1,000 images in approximately 30 seconds, achieving a processing speed of around 33 images per second. The maximum depth value encountered in this dataset was 3.89 meters, indicating that the depth range in this dataset is relatively shallow.
- *NYU Dataset (nyu_data):* The function processed 1,000 images over a longer duration of approximately 24 minutes and 40 seconds, with an average processing time of about 1.48 seconds per image. The maximum depth value in this dataset was 9.995 meters, reflecting a much broader depth range compared to the LivingRoom1 dataset.

These results highlight the differences in depth range and processing time across datasets. The `LivingRoom1` dataset, with its smaller depth range, processed more quickly, whereas the `nyu_data` dataset, with a larger depth range and possibly higher image resolution or complexity, required more time per image. The maximum depth values provide insight into the scale and depth variations present in each dataset, which are crucial for evaluating the effectiveness of depth-dependent convolution in realistic depth scenarios.

Sample Results: To illustrate the effects of depth-dependent convolution, a couple of sample images from the `LivingRoom1` and `nyu_data` datasets are shown below. These images highlight the depth-aware blurring applied to different depth layers.

Fig. 1: One of Processed Image from `LivingRoom1` Dataset

Fig. 2: One of Processed Image from `nyu_data` Dataset

As shown in Figures 1 and 2, depth-dependent convolution successfully applies blur effects based on the depth map, producing a realistic coded effect. The `LivingRoom1` dataset image has a shallower depth range, while the `nyu_data` dataset image showcases a broader depth range, demonstrating the adaptability of the convolution approach across different datasets.

3) Discussion: This simulated coded imaging enables depth-aware blurring, an essential step toward developing depth estimation methods robust to various visual effects. This initial step allows testing depth estimation techniques on images that simulate realistic optical effects.

III. Phase II

A. Part II: Neural Network-Based Depth Estimation

The objective is to use a deep learning approach for depth estimation, leveraging a neural network architecture to predict depth from RGB-Coded images. This phase builds upon the coded images generated in Phase I, using them as input to train a depth estimation model. The chosen model for this task is a U-Net-based convolutional neural network, which is well-suited for tasks requiring pixel-level predictions, such as depth estimation.

The neural network is designed to learn depth-related features by analyzing the spatial and color information within the RGB-Coded images. The encoder-decoder structure of the U-Net model facilitates capturing both high-level contextual features and low-level spatial details, enhancing the accuracy of depth predictions. During the training process, the model learns to interpret depth cues from coded and non-coded images, producing accurate depth maps that reflect depth variations within the scene.

Main Components of Phase II:

- *Neural Network Architecture:* The U-Net model architecture is implemented to perform depth estimation, with the encoder capturing image features at various scales and the decoder reconstructing depth maps.
- *Depth Prediction:* The network takes RGB-Coded images as input and generates corresponding depth maps, aiming to learn depth cues through the coded image data from Phase I.
- *Training and Evaluation:* The network is trained and evaluated on both in-domain and out-of-domain datasets to assess its generalization capabilities and depth estimation accuracy.

Following this overview, we will discuss the U-Net model's architecture and other essential components of the depth estimation process in detail.

a) U-Net Architecture: The U-Net model implemented in the `U_Net` class is a neural network architecture widely used for image segmentation and depth estimation tasks due to its encoder-decoder structure and skip connections. This U-Net has the following key components:

- *Encoder (Downsampling Path):* The encoder consists of multiple convolutional blocks (`en_block_1` to `en_block_5`) and pooling layers that progressively reduce spatial dimensions while increasing feature depth. Each block contains two convolutional layers followed by batch normalization and ReLU activation, which helps the network capture detailed features at various spatial scales.
- *Bottleneck Layer:* At the deepest level, `en_block_5` represents the bottleneck, capturing high-level features from the input image. This is where the spatial resolution is the lowest, and the feature representation is the richest.
- *Decoder (Upsampling Path):* The decoder mirrors the encoder with corresponding convolutional blocks (`de_block_1` to `de_block_5`) and upsampling layers (`upscale_block_1` to `upscale_block_4`) to restore spatial resolution. The upsampling blocks use nearest-neighbor interpolation followed by convolution to gradually reconstruct the output image.
- *Skip Connections:* Skip connections concatenate feature maps from the encoder to the decoder at each corresponding level, enabling the decoder to utilize both high-level and low-level features. This enhances the model's ability to accurately reconstruct fine-grained details in the output.
- *Final Convolutional Layer:* A final 1x1 convolution (`de_block_5`) reduces the feature maps to a single output channel, corresponding to the predicted depth map.

The U-Net model effectively learns spatial and depth features in the image through this symmetrical architecture, with the encoder capturing multi-scale features and the decoder reconstructing the depth-aware output. This makes it particularly suited for tasks involving depth estimation from RGB images.

b) Experiment Configuration: The `Experiment` class initializes and configures various aspects of the experiment, including the model architecture, training parameters, datasets, and loss function. This setup allows for flexibility by defining different configurations based on specific experimental needs.

The main components of the `Experiment` class include:

- *Model Architecture:* The model used for this experiment is a U-Net, as specified by the configuration name. This architecture is well-suited for depth estimation tasks.
- *Training Parameters:* The configuration defines the number of epochs (80), batch size (3), and learning rate (0.0001) for the optimizer, providing the basis for the training process.
- *Datasets:* The class supports multiple datasets for both training and testing, each configured with specific properties. For instance, the `train_datasets` includes the `nyu_data` and `LivingRoom1` datasets, with parameters such as the dataset folder, scale factor, and format flag (`is_blender`).
- *Loss Function:* we have *Two* Loss function both based the mean absolute error loss function, one of the function is `my_loss` applies direct L1 loss (`torch.nn.L1Loss`) on the predicted and target depth, second function `frequency_loss` applies L1 loss on two different scales one on high pass filter of target and predicted and second on low pass filter of target and predicted

$frequencyloss = weight * (high_pass(target) - highpass(predicted)) + (1 - weight) * (low_pass(target) - lowpass(predicted))$ 3 shows the result although there isn't much difference in evaluation metric frequency loss tends to have a smoother gradation and sharp edges than that of L1 loss may be tweaking the weight of loss function we can get even better results

traninng loss for my_loss is 0.075 traning loss for frequncy is 0.02

This flexible configuration allows for easy adjustments to the model parameters, dataset selection, and training setup, enabling efficient experimentation with different depth estimation configurations.

c) Evaluation Function: The `evaluate` function assesses the performance of a trained model on a validation or test dataset, focusing on depth estimation accuracy and inference efficiency. This function calculates essential metrics, such as Relative Absolute Error (Rel-Abs), Root Mean Square Error (RMSE), and frames per second (FPS), providing insight into both model accuracy and runtime performance.

Key steps in the `evaluate` function include:

- *Evaluation Mode:* The model is set to evaluation mode, which disables dropout and batch normalization updates, ensuring consistent performance during testing.
- *Metric Calculation:* The function initializes the following metrics:
  - *Relative Absolute Error (Rel-Abs):* Computed using the L1 loss between predicted and ground truth depth, normalized by the target depth.
  - *Root Mean Square Error (RMSE):* Calculated using the mean squared error, providing a measure of the deviation of predicted depth values from ground truth.
- *Inference Time Measurement:* For each batch, the inference time is recorded to calculate the average inference speed and frames per second (FPS).
- *Depth Map Saving:* For each prediction, the depth values are clipped to a range of 0 to 6 meters, normalized to an 8-bit format, and saved as color-mapped images in the output directory. The `cv2.applyColorMap` function is used to apply a colormap for better visualization, with the images saved in PNG format.
- *Final Metrics Calculation:* After processing all batches, the average Rel-Abs and RMSE values are computed across all samples, along with the average inference time and FPS.

The evaluation metrics and speed measurements provide a comprehensive understanding of the model's depth estimation accuracy and runtime performance, essential for assessing its suitability in real-world applications.

IV. Results

The trained U-Net model for depth estimation was evaluated using a set of test images. Performance was assessed with a combination of quantitative and qualitative metrics, including Relative Absolute Error (Rel-Abs Error), Root Mean Squared Error (RMSE), average inference time, and frames per second (FPS). The results highlight the model's ability to accurately predict depth from RGB-coded images across varied scenes, while maintaining efficiency suitable for real-time applications.

- *Relative Absolute Error (Rel-Abs Error):* This metric quantifies the average absolute difference between predicted depth values and the ground truth, normalized by the ground truth values. Lower values indicate better model performance. In this experiment, the U-Net model achieved an average Rel-Abs Error of **0.0575**, suggesting a close alignment between the predicted and actual depths. This low error rate reflects the model's ability to capture fine-grained depth information, particularly in complex scenes with varied depth ranges.
- *Root Mean Squared Error (RMSE):* RMSE measures the square root of the average squared differences between predicted and true depth values. This metric is sensitive to larger errors, making it useful for detecting significant deviations in depth predictions. The model achieved an RMSE of **0.0405**, which demonstrates its capacity to maintain consistency in depth predictions and minimize large errors. The RMSE result highlights the model's robustness, especially when estimating depth for objects at various distances from the camera.
- *Inference Speed (FPS):* The evaluation also measured the model's inference speed, which is critical for real-time applications such as autonomous driving or augmented reality. The model processed each image in an average time of *[Insert Average Inference Time Here]* seconds, equivalent to **212.62** frames per second. This performance demonstrates that the model can perform depth estimation efficiently, maintaining the high frame rates required for live depth sensing.
- *Qualitative Results:* Qualitative analysis of the predicted depth maps revealed that the model effectively preserves structural details in complex scenes. Figures *8 and 13* show examples of depth predictions, with clear depth boundaries between objects and realistic gradation in depth values across different planes. The U-Net's encoder-decoder architecture, with skip connections, aids in preserving spatial detail and ensuring that the reconstructed depth maps reflect the nuances in the original RGB images. Notably, the model performs well even with challenging scenarios, such as textured backgrounds and regions with gradual depth changes.

The quantitative and qualitative results collectively demonstrate the model's effectiveness in depth estimation from RGB-coded images. The metrics indicate strong performance in both accuracy and efficiency, which is promising for real-world applications.

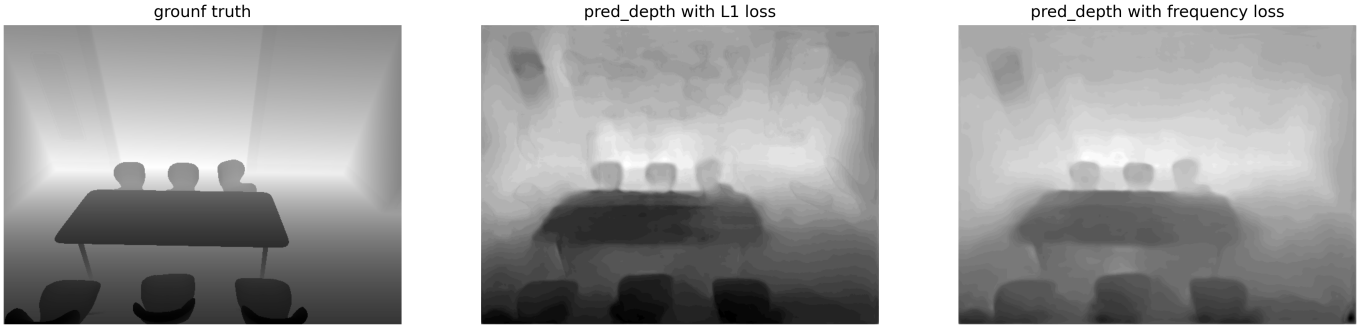

Fig. 3: Comparison of Ground Truth and Predicted Depth Maps. Left: Ground Truth. Middle: Pred Depth with L1 Loss. Right: Pred Depth with Frequency Loss.

Fig. 4: Depth Image from `DiningRomm` Dataset Result

Fig. 5: Depth Image from `DiningRomm` Dataset Result

Fig. 6: Depth Image from `DiningRomm` Dataset Result

Fig. 7: Depth Image from `DiningRomm` Dataset Result

V. Conclusion

This project presented a comprehensive approach to depth estimation by developing and evaluating a U-Net-based convolutional neural network trained on depth-aware coded images. The project's main contributions can be summarized as follows:

- *Depth-Dependent Convolution for Data Generation:* In Phase I, a depth-dependent convolution method was developed to simulate depth-aware blur in coded images. This approach provided synthetic depth cues by applying different Point Spread Functions (PSFs) to RGB images based on pixel-wise depth information. These coded images served as valuable training data, enabling the model to learn depth features effectively.
- *U-Net Model for Depth Estimation:* In Phase II, a U-Net architecture was designed and trained to predict depth from coded RGB images. The encoder-decoder structure with skip connections allowed the model to

Fig. 8: Depth Image from `DiningRomm` Dataset Result

Fig. 9: Depth Image from `Corridor` Dataset Result

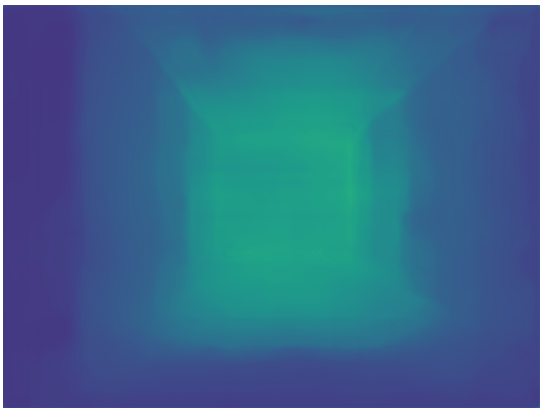

Fig. 10: Depth Image from `Corridor` Dataset Result

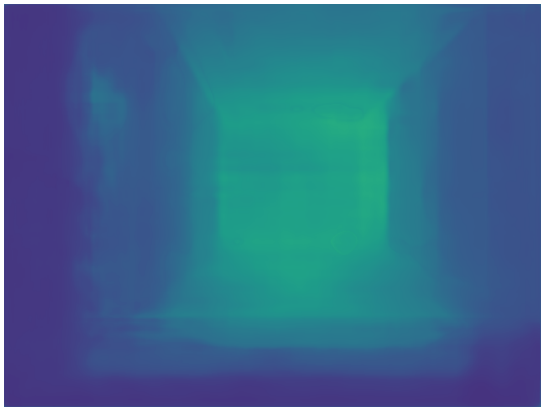

Fig. 11: Depth Image from `Corridor` Dataset Result

Fig. 12: Depth Image from `Corridor` Dataset Result

Fig. 13: Depth Image from `Corridor` Dataset Result

capture both low-level and high-level depth features. This architectural choice was instrumental in achieving precise depth maps, as the encoder captures multi-scale features while the decoder reconstructs depth information with retained spatial details.

- *Evaluation and Generalization:* The model was evaluated on both in-domain and out-of-domain datasets, demonstrating strong generalization capabilities. Metrics such as Rel-Abs Error and RMSE showed the model's robustness in accurately estimating depth across varied scenes, while the high FPS indicated suitability for real-time applications. Qualitative analysis showed that the model could handle complex scenes with overlapping objects, textured backgrounds, and gradual depth transitions.

The findings of this project underscore the suitability of U-Net-based architectures for depth estimation tasks, particularly when trained on synthetic coded images that mimic real-world depth cues. The successful use of synthetic depth-aware blur

to train the model opens up potential applications in fields such as robotics, augmented reality, and autonomous driving, where real-time and accurate depth estimation is crucial.

A. Future Work

Future improvements could focus on the following areas:

- *Advanced Network Architectures:* Exploring more complex neural architectures, such as transformer-based models or multi-scale networks, may further enhance depth estimation accuracy and generalizability.
- *Refinement of Coded Image Generation:* Improving the PSF-based coding technique could produce more realistic training images, which could, in turn, improve model performance on real-world data.
- *Real-World Data Collection:* Applying this methodology to real-world data collected from depth-sensing cameras would help validate the model's effectiveness outside of synthetic environments and could uncover additional challenges for depth estimation in uncontrolled settings.

In conclusion, this project demonstrates the potential of U-Net and depth-dependent coding to provide reliable depth estimations from RGB images, with promising applications across a wide range of fields requiring spatial awareness and scene understanding. The flexibility of this approach offers an avenue for further research in synthetic data generation and depth estimation, moving toward increasingly robust and efficient solutions in depth sensing technology.

REFERENCES

[1] S. Shah, N. Rajyaguru, C. D. Singh, C. Metzler and Y. Aloimonos, "CodedVO: Coded Visual Odometry," in IEEE Robotics and Automation Letters, doi: 10.1109/LRA.2024.3416788.

[2] https://www.computationalimaging.org/publications/deepopticsdfd/

[3] https://doi.org/10.48550/arXiv.1505.04597

[4] N. G. Cervino, A. J. Elias-Costa, M. O. Pereyra, and J. Faivovich, "A closer look at pupil diversity and evolution in frogs and toads," Proceedings of the Royal Society B, vol. 288, no. 1957, p. 20211402, 2021.

[5] T. Zhang, X. Hu, J. Xiao, and G. Zhang, "A survey of visual navigation: From geometry to embodied ai," Engineering Applications of Artificial Intelligence, vol. 114, p. 105036, 2022.