Monocular Depth Estimation with Object Detection for low End Computing Device

Sahin Hossain Chowdhury

A Dissertation Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology/ Doctor of Philosophy

Department of Smart Mobility

June, 2023

Declaration

I declare that this written submission represents my ideas in my own words, and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

_____

(Sahin Hossain Chowdhury)

(SM21MTECH12002)

Approval Sheet

This thesis entitled Monocular Depth Estimation with Object Detection for low End Computing Device by Sahin Hossain Chowdhury is approved for the degree of Master of Technology from IIT Hyderabad.

_____

Dr. R Prasanth Kumar

_____

Dr. C P Vyasarayani

_____

Dr. Sireesh Saride

Acknowledgements

I would like to express my sincere gratitude and appreciation to the following individuals who have been instrumental in the successful completion of this project.

First and foremost, I would like to extend my heartfelt thanks to my family for their unwavering support, understanding, and encouragement throughout this journey. Their love and belief in me have been a constant source of motivation, and I am deeply grateful for their presence in my life.

I would like to express my deepest gratitude to my guide, Dr. R. Prasanth Kumar, for his invaluable guidance, expertise, and mentorship. His profound knowledge, patience, and insightful feedback have been instrumental in shaping this project and pushing me to achieve my best. I am truly grateful for the opportunities he has provided me and the trust he has placed in my abilities.

I would also like to extend my appreciation to my friends and colleagues who have been a constant source of support and encouragement. Their invaluable insights, brainstorming sessions, and collaborative efforts have enriched my understanding and propelled me forward.

I would like to acknowledge the contributions of the research community whose work has laid the foundation for this project. Their groundbreaking research, published papers, and open-source implementations have been invaluable resources that have guided and inspired my work.

Lastly, I would like to express my heartfelt thanks to all the anonymous participants and contributors to the various datasets and tools used in this project. Their efforts and generosity in sharing their work have been crucial in the development and evaluation of this project.

I am indebted to all these individuals and organizations for their contributions, and I am grateful for the opportunity to work on this project.

Abstract

Monocular depth estimation and object detection are fundamental tasks in computer vision that provide crucial information about the spatial layout and the presence of objects in an image. This study presents a novel approach that combines these two tasks to achieve a comprehensive understanding of the scene using a single monocular image.

The proposed framework utilizes a CNN-based encoder-decoder architecture for monocular depth estimation, which learns to predict the depth map of the scene. Simultaneously, an object detection model, such as YOLO or SSD, is employed to identify and localize objects within the image. The depth estimation and object detection networks share a common backbone, enabling feature sharing and efficient computation.

Experimental results on benchmark datasets demonstrate the effectiveness of the proposed approach. The integrated monocular depth estimation with object detection achieves superior performance compared to individual tasks, enabling a more comprehensive understanding of the scene. The combined model provides accurate depth estimation and precise object localization, making it valuable for applications such as autonomous driving, robotics, and augmented reality.

In conclusion, the fusion of monocular depth estimation with object detection presents a powerful framework for scene understanding. By leveraging the complementary information from both tasks, the proposed approach enables enhanced perception and interpretation of visual data, contributing to advancements in various computer vision applications.

Contents

Chapter 1

Introduction

1.1 Background

Monocular depth estimation and object avoidance are critical components of autonomous systems, such as self-driving cars and drones. Depth estimation involves inferring the distance of objects in a scene from a single 2D image or video, while object avoidance focuses on identifying and navigating around potential obstacles to ensure safe and efficient navigation.

Traditionally, depth estimation relied on expensive and complex hardware setups, such as stereo cameras or depth sensors, which limited its applicability in real-world scenarios. However, recent advancements in computer vision, specifically in deep learning techniques, have enabled accurate depth estimation using monocular images. This has opened up new possibilities for a wide range of applications where depth perception is crucial.

Object avoidance is another vital aspect of autonomous systems. Whether it's navigating through a busy street or avoiding obstacles in a cluttered environment, the ability to perceive and react to objects in the surroundings is crucial for safe and reliable operation. By combining depth estimation with object detection and tracking, intelligent systems can make informed decisions to avoid potential collisions and ensure smooth navigation.

1.2 Motivation

The motivation behind this project stems from the increasing demand for robust and efficient autonomous systems that can operate in dynamic and unpredictable environments. The ability to estimate depth accurately from a single camera and effectively avoid obstacles is essential for enhancing the safety, reliability, and autonomy of such systems.

By developing advanced monocular depth estimation techniques, we can reduce the reliance on expensive and bulky depth sensors, making autonomous systems more affordable and accessible. Additionally, the integration of object avoidance strategies with depth estimation algorithms can significantly improve the ability of autonomous systems to understand and respond to their surroundings, enabling them to operate in complex and dynamic real-world scenarios.

Furthermore, the advancements in this field have the potential to revolutionize various industries. From self-driving cars that can navigate congested city streets to drones that can safely fly in complex environments, the applications are numerous and far-reaching. By contributing to the research and development of monocular depth estimation and object avoidance techniques, we can drive innovation and pave the way for the widespread adoption of autonomous systems in various domains.

In summary, the combination of monocular depth estimation and object avoidance has immense potential to enhance the capabilities of autonomous systems, making them more efficient, safe, and adaptable. This project aims to contribute to this field by exploring novel algorithms and methodologies for accurate depth estimation and effective object avoidance, thereby advancing the state-of-the-art and enabling the realization of intelligent autonomous systems.

1.3 Problem Statement

The problem addressed in this research project revolves around monocular depth estimation and object avoidance in autonomous systems. While depth estimation from a single camera has made significant progress with the advent of deep learning techniques, there are still challenges to overcome in terms of accuracy, robustness, and real-time performance. Additionally, integrating depth estimation with object detection and tracking to enable effective object avoidance presents its own set of complexities. Therefore, the problem statement focuses on improving the accuracy of monocular depth estimation and developing efficient strategies for object avoidance to enhance the capabilities of autonomous systems.

Depth estimation with object detection and tracking to enable effective object avoidance presents its own set of complexities. Therefore, the problem statement focuses on improving the accuracy of monocular depth estimation and developing efficient strategies for object avoidance to enhance the capabilities of autonomous systems

1.4 Research Objective

The research objectives of this project are designed to address the problem statement and contribute to the field of monocular depth estimation and object avoidance. The specific objectives include:

1. Investigating existing monocular depth estimation techniques: This objective involves conducting a comprehensive literature review to understand the state-of-the-art methods for monocular depth estimation. By analyzing existing approaches, their strengths, limitations, and performance characteristics, this research aims to identify areas for improvement and novel directions to explore.

2. Developing advanced depth estimation algorithms: Building upon the existing techniques, this objective focuses on proposing novel deep learning architectures and algorithms for monocular depth estimation. The goal is to improve accuracy, robustness, and efficiency while considering real-time performance constraints. These algorithms may incorporate advanced concepts such as attention mechanisms, multi-scale information fusion, or geometric priors to enhance depth estimation quality.

3. Evaluating and comparing depth estimation algorithms: This objective involves designing rigorous evaluation protocols and datasets to quantitatively assess the performance of the developed algorithms.

Comparative analyses will be conducted to benchmark the proposed methods against state-of-the-art approaches. The evaluation metrics may include accuracy, completeness, computational efficiency, and generalization across different scenarios and datasets.

4. Investigating object detection and tracking techniques: To enable effective object avoidance, this objective focuses on exploring object detection and tracking algorithms suitable for autonomous systems. Various approaches, including traditional computer vision methods and deep learning-based techniques, will be investigated and evaluated based on their detection accuracy, tracking robustness, and computational efficiency.

5. Fusion of depth estimation and object detection: This objective aims to develop strategies for integrating depth estimation and object detection to facilitate accurate and reliable object avoidance. Different fusion techniques, such as sensor fusion or feature-level fusion, will be explored and evaluated in terms of their effectiveness in identifying and avoiding obstacles in real-time scenarios.

6. Designing and evaluating object avoidance algorithms: This objective involves designing and implementing intelligent algorithms for object avoidance based on the fused information from depth estimation and object detection. The developed algorithms will consider factors such as the dynamics of the environment, the speed of the autonomous system, and the trajectory planning to ensure safe and efficient navigation. The performance of these algorithms will be evaluated through simulations and real-world experiments.

7. Analysis and discussion of results: This objective focuses on analyzing and interpreting the results obtained from the evaluation of the developed algorithms. The strengths, limitations, and potential areas of

improvement will be discussed, along with insights into the performance trade-offs and practical implications for real-world deployment of autonomous systems.

By addressing these research objectives, this project aims to advance the state-of-the-art in monocular depth estimation and object avoidance, providing valuable insights and practical solutions to enhance the capabilities of autonomous systems in terms of perception, decision-making, and navigation.

1.5 Scope

The scope of this research project encompasses the development and evaluation of monocular depth estimation algorithms and object avoidance strategies for autonomous systems. The focus is on leveraging deep learning techniques, along with traditional computer vision methods, to improve the accuracy, robustness, and real-time performance of depth estimation from a single camera. Additionally, the project involves investigating object detection and tracking algorithms and integrating them with depth estimation to enable effective object avoidance. The evaluation of the proposed algorithms will be conducted using various datasets and performance metrics to assess their effectiveness and compare them against state-of-the-art approaches. The project aims to contribute to the field by exploring novel methodologies and providing insights into the challenges and potential solutions for monocular depth estimation and object avoidance.

1.6 Limitations

Despite the extensive research and development in monocular depth estimation and object avoidance, there are certain limitations that need to be acknowledged. These limitations include:

1.Accuracy trade-offs: While the aim is to improve the accuracy of monocular depth estimation, there may be inherent limitations due to the absence of explicit depth information from a single image. Depth estimation from monocular images is an inherently ill-posed problem, and achieving perfect accuracy may be challenging, particularly in complex scenes with occlusions, texture less regions, or highly reflective surfaces.

2.Generalization across diverse scenarios: The proposed algorithms and strategies may exhibit varying performance across different scenarios and datasets. Achieving robustness and generalization to handle diverse environmental conditions, lighting variations, and object appearances can be a complex task. The project will strive to address these challenges, but achieving universal applicability may not be feasible within the scope of this research.

3.Computational constraints: Real-time performance is crucial for practical implementation in autonomous systems. However, the computational requirements of advanced depth estimation algorithms and object detection/tracking techniques can be significant. Striking a balance between accuracy and computational efficiency is a challenge that needs to be considered in this research.

4.Hardware limitations: The research project primarily focuses on the algorithmic aspects of monocular depth estimation and object avoidance. However, the performance and limitations of the hardware used, such as the camera sensor quality, processing units, and memory resources, can influence the overall system performance. While these hardware

considerations are essential, they are outside the direct scope of this research project.

5.Simulated and controlled environments: The evaluation and testing of the proposed algorithms and strategies will be performed using various datasets and simulated or controlled environments. While this allows for controlled comparisons and performance analysis, the real-world deployment and performance of the developed algorithms may differ due to factors such as weather conditions, dynamic objects, or unexpected scenarios.

Chapter 2

Literature Review

2.1 In the paper titled "Unsupervised Monocular Depth Estimation with Left-Right Consistency" by Clément Godard et al the authors propose an unsupervised learning method for monocular depth estimation. The key idea of this approach is to leverage left-right image consistency to infer accurate depth information without requiring ground truth depth annotations.

The unsupervised framework is based on training a deep neural network to predict depth maps from a single input image. The network takes as input a single image and learns to estimate depth by leveraging the left-right consistency between stereo image pairs. The left-right consistency assumption states that if a point in one image is projected onto the other image, the corresponding point should have similar appearance and depth.

To enforce this left-right consistency, the authors introduce a novel photometric loss function that compares the reconstructed right image with the actual right image. By minimizing this loss, the network learns to estimate depth maps that are consistent with the appearance of the corresponding points in the stereo image pair.

The authors also propose an additional geometric loss term that encourages the predicted depth maps to be consistent with the scene geometry. This loss term penalizes depth discontinuities and encourages smoothness in the estimated depth maps.

Experimental results demonstrate that the proposed unsupervised approach achieves competitive performance compared to supervised methods that rely on ground truth depth annotations. The method is evaluated on various benchmark datasets, and the results show accurate depth estimation in challenging scenarios.

Overall, the paper presents a novel unsupervised approach for monocular depth estimation that exploits left-right image consistency. By leveraging this consistency, the proposed method achieves accurate depth estimation without the need for ground truth annotations,

demonstrating the potential of unsupervised learning techniques in the field of depth estimation.

2.2 The paper titled "Depth Estimation via Affinity Learned with Convolutional Spatial Propagation Network" by Fei Yin et al. presents a novel approach for monocular depth estimation using a convolutional spatial propagation network. The proposed method leverages a spatial propagation module to capture long-range dependencies and improve the accuracy of depth predictions.

The core idea of the approach is to exploit the relationships between pixels within an image to enhance depth estimation. The spatial propagation module is designed to propagate information from neighbouring pixels to refine the depth predictions. It employs convolutional operations to capture and propagate the affinity between pixels, allowing for the exchange of information and the modelling of contextual relationships.

The network architecture consists of an encoder-decoder structure, with the spatial propagation module integrated into the decoder. The encoder encodes the input image into multi-scale feature maps, capturing different levels of information. The decoder utilizes these feature maps and combines them with the propagated information from the spatial propagation module to generate refined depth predictions.

To train the network, a loss function based on the pixel-wise differences between the predicted depth maps and ground truth depth maps is employed. Additionally, the authors introduce a novel affinity loss that encourages the propagation module to learn meaningful relationships between pixels, further improving the accuracy of the depth estimates.

Extensive experimental evaluations on benchmark datasets demonstrate the effectiveness of the proposed method. The results showcase that the convolutional spatial propagation network achieves superior performance compared to existing state-of-the-art approaches. The method successfully captures long-range dependencies and accurately predicts depth maps, even in challenging scenarios with occlusions and texture less regions.

In summary, the paper presents a convolutional spatial propagation network for monocular depth estimation. By utilizing a spatial propagation module, the approach effectively captures long-range dependencies and improves the accuracy of depth predictions. The experimental results validate the effectiveness of the proposed method, showcasing its superiority over existing approaches in terms of depth estimation accuracy.

2.3 The paper titled "Revisiting Single Image Depth Estimation: Toward Higher Resolution Maps with Accurate Object Boundaries" by Iro Laina et al. introduces a deep learning framework for high-resolution depth estimation from a single image. The proposed approach combines global and local information and incorporates object boundary detection to enhance the accuracy of depth maps.

The main objective of the paper is to address the challenge of obtaining high-resolution depth maps with precise object boundaries, which are crucial for various applications such as augmented reality and autonomous navigation.

The framework consists of two main components: a global depth prediction module and a local refinement module. The global depth prediction module leverages a deep convolutional neural network to estimate an initial depth map at a coarse resolution. This module captures global context information and provides an initial depth estimation for the entire scene.

To enhance the accuracy of the depth maps, the local refinement module utilizes object boundary detection. It employs an additional network that focuses specifically on detecting object boundaries in the input image. The detected boundaries are then used to guide the refinement of the depth maps by incorporating local information. This enables the model to improve depth estimates around object boundaries, where accurate depth estimation is particularly challenging.

The proposed framework is trained using a combination of supervised and unsupervised learning. The supervised training utilizes ground truth depth maps to guide the network optimization, while the unsupervised training leverages photometric and geometric constraints

to enforce consistency between the predicted depth maps and the input image.

Extensive experimental evaluations are conducted on benchmark datasets, demonstrating the effectiveness of the proposed approach. The results show that the framework achieves higher-resolution depth maps with improved accuracy, especially around object boundaries. The approach outperforms previous methods and provides state-of-the-art performance in terms of depth estimation quality.

In summary, the paper presents a deep learning framework for high-resolution depth estimation from a single image. By combining global and local information and incorporating object boundary detection, the proposed approach achieves improved accuracy in depth maps, particularly around object boundaries. The experimental results highlight the superiority of the framework, demonstrating its potential for various applications that require high-resolution depth estimation

2.4 In the paper "Depth from Videos in the Wild: Unsupervised Monocular Depth Learning from Unknown Cameras" by Tinghui Zhou, et al., the authors propose an unsupervised learning approach for monocular depth estimation using videos captured from unknown cameras. The goal is to learn depth information without relying on ground truth depth annotations.

The approach leverages both photometric and geometric constraints present in video sequences to infer depth. The key idea is to exploit the consistency between consecutive frames in terms of pixel intensities and camera motion. By analyzing the changes in pixel intensities and corresponding camera motion between frames, the network can infer the underlying scene geometry and estimate depth.

The proposed method consists of two main stages: depth initialization and depth refinement. In the depth initialization stage, a depth network is trained to predict initial depth maps based on a photometric loss that measures the discrepancy between synthesized and real images. These synthesized images are created by warping neighbouring frames to a target frame using estimated camera motion. The photometric loss

encourages the network to produce depth maps that are consistent with the observed image sequences.

In the depth refinement stage, the initial depth maps are further refined using a combination of geometric and photometric losses. The geometric loss enforces consistency between the estimated depth and the relative camera poses estimated from the video sequences. The photometric loss is computed based on the photometric consistency between synthesized and real images after warping. By jointly optimizing the geometric and photometric losses, the network learns to refine the initial depth estimates, producing more accurate depth maps.

The authors evaluate their approach on various challenging datasets, including the KITTI and Make3D datasets, which contain videos captured by different cameras in diverse environments. The experimental results demonstrate that the proposed unsupervised method achieves competitive performance compared to supervised methods that rely on ground truth depth annotations. The approach is capable of estimating accurate depth maps from videos captured by unknown cameras, highlighting its potential for real-world applications.

Overall, this paper presents a novel unsupervised learning approach for monocular depth estimation from videos captured by unknown cameras. By leveraging photometric and geometric constraints, the proposed method enables the learning of depth information without the need for explicit supervision. The experimental results demonstrate the effectiveness of the approach in estimating accurate depth maps and showcase its potential for applications in real-world scenarios.

2.8 The paper "Real-Time Monocular Object Detection and Avoidance for Small Unmanned Aerial Vehicles" by Matteo Fumagalli, et al. addresses the challenge of real-time monocular object detection and avoidance specifically tailored for small unmanned aerial vehicles (UAVs). The authors propose a lightweight detection algorithm and a path planning strategy to enable safe and efficient UAV navigation in dynamic environments.

The key contributions of the paper are as follows:

1. Lightweight Detection Algorithm: The authors introduce a lightweight object detection algorithm suitable for resource-constrained UAV platforms. The algorithm leverages deep learning techniques to detect objects of interest in monocular images captured by the UAV's onboard camera. The focus on lightweight design ensures real-time performance and efficient execution on UAV hardware.

2. Real-Time Object Detection: The proposed detection algorithm is optimized for real-time performance, making it suitable for dynamic environments where timely detection and avoidance are critical. By efficiently processing images in real-time, the UAV can detect objects in its vicinity and respond quickly to potential obstacles.

3. Path Planning for Object Avoidance: The paper also presents a path planning strategy specifically tailored for small UAVs. The path planning algorithm takes into account the detected objects and generates trajectories that allow the UAV to avoid collisions while optimizing for efficiency and smooth navigation. The proposed approach enables safe and autonomous operation in cluttered environments.

4. Experimental Evaluation: The authors provide a comprehensive experimental evaluation of the proposed system. They validate the effectiveness and real-time performance of the object detection algorithm using various datasets and real-world scenarios. The evaluation includes quantitative metrics such as detection accuracy, computational efficiency, and response time.

5. Real-World UAV Implementation: The proposed system is implemented and tested on a small UAV platform, showcasing its practicality and applicability. The authors provide insights into the challenges faced during the implementation process and discuss the system's performance and limitations in real-world scenarios.

Overall, this paper contributes to the field of monocular object detection and avoidance for small UAVs. By proposing a lightweight detection

algorithm and a path planning strategy, the authors enable real-time object detection and safe navigation for UAVs operating in dynamic environments. The experimental evaluation and real-world implementation provide evidence of the system's effectiveness and demonstrate its potential for practical applications in UAV operations.

2.9 The paper "Learning Monocular Reactive UAV Control in Cluttered Natural Environments" by Antoni Rosinol Vidal, et al. tackles the challenge of monocular reactive UAV control in cluttered natural environments. The authors propose a learning-based approach that allows autonomous unmanned aerial vehicles (UAVs) to navigate and avoid obstacles in real-time.

The main contributions of this paper are as follows:

1. Learning-Based Approach: The authors propose a learning-based method for UAV control, which allows the UAV to react and navigate in cluttered natural environments using only monocular vision. The approach leverages deep learning techniques to learn the mapping between visual input from the monocular camera and appropriate control commands for obstacle avoidance.

2. Reactive Control: The proposed method enables reactive control, meaning that the UAV continuously perceives its surroundings and adjusts its control actions in real-time based on the perceived obstacles. This reactive behaviour ensures the UAV can respond quickly and effectively to dynamic changes in the environment, enabling safe navigation.

3. Cluttered Natural Environments: The focus of this paper is on cluttered natural environments, where obstacles can have complex shapes, sizes, and appearances. The proposed learning-based approach

aims to handle such challenging scenarios by learning from a diverse set of training data and capturing the variations present in cluttered environments.

4. Real-Time Obstacle Avoidance: The learning-based approach allows the UAV to perform obstacle avoidance in real-time. By leveraging the learned mapping from visual input to control commands, the UAV can autonomously navigate and avoid obstacles without the need for explicit environment modelling or pre-determined paths.

5. Experimental Evaluation: The authors provide an experimental evaluation of the proposed method, demonstrating its effectiveness in cluttered natural environments. The evaluation includes real-world scenarios where the UAV successfully avoids obstacles in real-time using only monocular vision.

By presenting a learning-based approach for monocular reactive UAV control in cluttered natural environments, this paper contributes to the field of autonomous UAV navigation and obstacle avoidance. The focus on real-time obstacle avoidance using monocular vision makes it suitable for practical applications where UAVs operate in dynamic and complex environments.

Chapter 3

Data Loading and Preparation

3.1 Different Dataset Available

3.1.1. NYU Depth V2:

- Description: The NYU Depth V2 dataset consists of RGB-D images captured by a Kinect camera. It contains 464 indoor scenes with diverse layouts and objects.
- Data: Each scene provides synchronized color and depth images, along with intrinsic and extrinsic camera parameters. The depth maps are provided at the same resolution as the RGB images.
- Annotations**:** The dataset includes pixel-level labeling of object instances, semantic labels, and 3D bounding boxes for objects**.**
- Applications: NYU Depth V2 is commonly used for depth estimation, semantic segmentation, object recognition, and scene understanding tasks in indoor environments.

3.1.2. KITTI:

- Description: The KITTI dataset is widely used for research in autonomous driving. It includes high-resolution images and accurate depth measurements collected from a car-mounted sensor suite.
- Data: The dataset provides various data modalities, including grayscale images, stereo image pairs, lidar point clouds, and accurate GPS/IMU data.
- Annotations: KITTI offers annotations for different tasks, including object detection, tracking, semantic segmentation, and depth estimation.
- Applications: KITTI is primarily used for developing algorithms related to 3D object detection, tracking, scene understanding, and autonomous driving.

3.1.3 Make3D:

- Description: The Make3D dataset focuses on outdoor scenes and contains stereo images along with corresponding manually-generated depth maps.
- Data: The dataset provides a collection of outdoor images captured from a calibrated stereo camera setup.
- Annotations: The depth maps in Make3D are generated manually by human annotators based on the stereo image pairs.
- Applications: Make3D is used for evaluating and benchmarking stereo matching and monocular depth estimation algorithms in outdoor environments.

3.1.4. ScanNet:

- Description: The ScanNet dataset is a large-scale indoor dataset that includes RGB-D images and 3D reconstructions of various scenes.
- Data: ScanNet captures indoor scenes using depth sensors, providing RGB images along with aligned depth maps.
- Annotations: The dataset includes per-pixel semantic annotations, instance-level annotations, and reconstructed 3D models.
- Applications: ScanNet is utilized for tasks such as scene understanding, semantic segmentation, 3D reconstruction, and depth estimation in indoor environments.

3.1.5. Cityscapes:

-Description: The Cityscapes dataset was initially created for semantic segmentation tasks but also provides ground truth depth maps for urban street scenes.

- Data: The dataset offers high-quality RGB images captured in several cities, including diverse weather and lighting conditions.

-Annotations: Cityscapes provides pixel-level annotations for semantic segmentation, instance segmentation, and detailed annotations for object detection.

- Applications: Cityscapes is commonly used for urban scene understanding, semantic segmentation, object detection, and depth estimation in urban environments.

3.1.6. DIODE:

- Description: The DIODE dataset is a large-scale dataset that covers diverse scenes and provides both color and depth information.

- Data: DIODE contains RGB-D images captured from multiple viewpoints, allowing for a more comprehensive understanding of scenes.

-Annotations: The dataset includes pixel-level annotations for semantic segmentation and instance segmentation.

-Applications: DIODE is used for a wide range of tasks, including depth estimation, 3D reconstruction, scene understanding, and object recognition**.**

3.2 Why I Choose NYU Depth V2 Dataset:

3.2.1. Rich and Diverse Data: The NYU Depth V2 dataset provides a wide range of RGB-D images captured in indoor environments. It contains 464 scenes with diverse layouts, objects, and lighting conditions. This diversity ensures that your model can generalize well to different indoor scenes.

3.2.2. Ground Truth Depth Annotations: The dataset includes per-pixel depth measurements, providing ground truth depth maps for training and evaluation. Having access to ground truth depth annotations allows you to train and validate your model effectively, enabling accurate depth estimation.

3.2.3. Comprehensive Annotations: In addition to depth annotations, the NYU Depth V2 dataset also provides other annotations such as object boundaries, instance IDs, class labels, semantic segmentation, and 3D bounding boxes. These annotations enable you to explore and develop various computer vision tasks beyond depth estimation, such as object recognition, semantic segmentation, and scene understanding.

3.2.4. Benchmark for Comparison: The NYU Depth V2 dataset has been widely used in the research community, making it a standard benchmark for monocular depth estimation algorithms. By using this dataset, you can directly compare the performance of your model with existing state-of-the-art methods and contribute to the advancements in the field.

3.2.5. Availability and Accessibility: The NYU Depth V2 dataset is publicly available, making it easily accessible for researchers and developers. You can download the dataset and start working on your depth estimation project without significant barriers.

3.3 Different Data processing Techniques

Data preparation is an essential step in any machine learning project, including monocular depth estimation. It involves processing and organizing the dataset to ensure it is suitable for training and evaluation of your model. Here are some key aspects of data preparation for monocular depth estimation:

3.3.1. Data Cleaning: Inspect the dataset for any corrupt or incomplete samples and remove them if necessary. Check for missing depth maps or RGB images that don't align properly with their corresponding depth maps.

3.3.2. Data Split: Divide the dataset into training, validation, and testing sets. The training set is used to train your model, the validation set is used for hyperparameter tuning and model selection, and the testing set is used to evaluate the final performance of your trained model. Ensure a proper distribution of samples across these sets to avoid bias.

3.3.3. Data Augmentation: Apply data augmentation techniques to increase the diversity and variability of your training data. Common augmentations for monocular depth estimation include random cropping, flipping, rotation, scaling, and adding noise. These techniques help improve the generalization ability of your model and reduce overfitting.

3.3.4. Image Resizing and Normalization: Resize the RGB images and corresponding depth maps to a consistent size suitable for your model architecture. Normalize the pixel values of the RGB images and scale the depth values to a specific range, such as [0, 1] or [-1, 1].

3.3.5. Input-Output Pairing: Ensure that the RGB images and their corresponding depth maps are correctly paired, so that the model learns the relationship between them accurately. Verify that the RGB image and depth map filenames or file paths are properly aligned.

3.3.6. Data Loading and Batching: Implement an efficient data loading pipeline to load the data in batches during training. Consider using parallel processing and prefetching techniques to speed up the data loading process and optimize training performance.

3.3.7. Handling Class Imbalance: Depending on the dataset, you may encounter class imbalance in the depth values, where certain depth ranges are overrepresented or underrepresented. Apply appropriate strategies, such as weighted loss functions or data sampling techniques, to address class imbalance and prevent bias in your model's predictions.

3.4 Data Cleaning

During the data cleaning process for monocular depth estimation, it is important to carefully inspect the dataset to identify any corrupt or incomplete samples. Here are some steps to consider for data cleaning:

3.4.1. Missing Depth Maps: Check if there are any depth maps missing from the dataset. Depth maps provide the ground truth depth information that your model will learn to estimate. Ensure that each RGB image has a corresponding depth map, as they need to be aligned properly for training and evaluation. If any depth maps are missing or unavailable, you may need to exclude the corresponding RGB images from the dataset or find alternative depth sources.

3.4.2. RGB-Depth Alignment: Verify that the RGB images and their corresponding depth maps align properly. The RGB images and depth maps should represent the same scene or object captured from the same viewpoint. Check if there are any mismatches, such as RGB images and depth maps from different scenes or misaligned spatial information. In such cases, it is important to either correct the misalignments or remove the samples that do not align properly.

3.3.3. Corrupt or Incomplete Samples: Examine the dataset for any corrupt or incomplete samples. This can include RGB images or depth maps that are

unreadable, contain significant noise or artifacts, or have missing portions. These samples can negatively impact the training process and the quality of depth estimations. It is advisable to remove such samples from the dataset to maintain data integrity and prevent them from influencing your model's performance.

3.3.4. Data Consistency: Ensure that the dataset is consistent in terms of image format, resolution, and color representation. Inconsistent formats or variations in resolution can pose challenges during training and may require additional preprocessing steps to standardize the data. Make sure that all images and depth maps are in the same format and have consistent resolutions to facilitate seamless training and evaluation.

3.5 Data Splitting

Dividing the dataset into training, validation, and testing sets is a crucial step in machine learning to properly evaluate the performance of your model and prevent overfitting. Here's a more detailed explanation of each set and considerations for their distribution:

3.5.1. Training Set:

- Purpose: The training set is used to train your model. It consists of labeled data pairs, typically RGB images and their corresponding depth maps.
- Size: The training set is usually the largest subset of the dataset, containing around 60-80% of the data.
- Considerations: Ensure that the training set is representative of the entire dataset, covering a wide range of scenes, objects, and depth variations. Avoid biases in the training set by randomly sampling data from different subsets of the dataset.

3.5.2. Validation Set:

- Purpose: The validation set is used for hyperparameter tuning, model selection, and monitoring the model's performance during training. It helps you assess how well the model generalizes to unseen data and aids in making decisions on architectural choices and hyperparameter settings.

- Size: The validation set is typically smaller than the training set, around 10-20% of the data.

- Considerations: The validation set should be diverse and representative of the data distribution. It should include examples with different depth ranges, environmental conditions, and object types. Avoid using data from the validation set for any form of model training or parameter adjustment to maintain unbiased evaluation.

3.5.3. Testing Set:

- Purpose: The testing set is used to evaluate the final performance of your trained model. It provides an unbiased estimate of how well your model performs on unseen data.

- Size: The testing set is usually smaller than the training and validation sets, around 10-20% of the data.

- Considerations: The testing set should be completely independent and disjoint from the training and validation sets. It should contain diverse samples that cover the full range of depth variations and scenes encountered in the real-world application. The testing set should only be used at the end of the development process to assess the model's generalization ability.

Considerations for Distribution:

- Randomness: When splitting the data, it is important to introduce randomness to avoid any biases. Randomly shuffle the dataset before dividing it into training, validation, and testing sets to ensure a random distribution of samples across all sets.

- Stratification: If there is a class imbalance in your data, consider using stratified sampling to ensure that each set maintains the same class distribution as the

original dataset. This is particularly useful if certain depth ranges or object types are underrepresented in the data.

3.6 Data Augmentation

data augmentation is a technique used to artificially increase the size and diversity of a training dataset by applying various transformations to the existing data. By augmenting the data, you can introduce variations that mimic real-world scenarios and improve the generalization ability of your model. Here's more information about data augmentation techniques commonly used in monocular depth estimation:

3.6.1. Random Cropping:

- Description: Randomly crop a portion of the input image and its corresponding depth map. This helps the model learn to focus on different regions of the scene and handle varying object scales.
- Parameters: Specify the crop size or a range of possible crop sizes. You can also control the aspect ratio of the crops.

3.6.2. Flipping:

- Description: Horizontally flip the input image and its corresponding depth map. This helps the model learn to handle mirrored scenes and reduces the bias towards specific orientations.
- Parameters: Typically, flipping is applied with a 50% chance.

3.6.3. Rotation:

- Description: Rotate the input image and its corresponding depth map by a random angle. This helps the model generalize to scenes captured from different viewpoints or orientations.
- Parameters: Specify the range of possible rotation angles.

3.6.4. Scaling:

- Description: Resize the input image and its corresponding depth map by a random scale factor. This simulates different distances from the objects and varying image resolutions.
- Parameters: Specify the range of possible scale factors.

3.6.5. Adding Noise:

- Description: Introduce random noise to the input image or its corresponding depth map. This helps the model learn to handle noisy or low-quality input data.
- Parameters: Control the type and amount of noise to be added, such as Gaussian noise or salt-and-pepper noise.

3.7 Image Resizing and Normalization

3.7.1 Image Resizing

Resizing the RGB images and depth maps to a consistent size is important to ensure that they can be fed into your model for training or inference. Most deep learning models require inputs of fixed dimensions, so you need to resize the images and depth maps accordingly. This can be done using various interpolation methods, such as bilinear or nearest-neighbor interpolation, to preserve the visual information as much as possible.

When resizing, you should consider the aspect ratio of the images to avoid distortion. It is common to resize the images while maintaining the original aspect ratio by either cropping or padding the images. Cropping removes parts of the image to fit the desired size, while padding adds extra pixels to fill the required size.

3.7.2 Normalization:

Normalization of pixel values is a crucial step in data preparation for monocular depth estimation. It involves transforming the pixel values of the RGB images to a standardized range to improve the stability and convergence

of the training process. The most common normalization technique is to scale the pixel values to a range of [0, 1] or [-1, 1].

To normalize the pixel values to [0, 1], you divide each pixel value by the maximum pixel value in the image (e.g., 255 for 8-bit images). This rescales the pixel values to a range where the minimum value becomes 0 and the maximum value becomes 1.

3.8 Data Loading and batching

Efficient data loading is crucial for training deep learning models, as it can significantly impact training time and overall performance. Here are some techniques for implementing an efficient data loading pipeline for monocular depth estimation:

3.8.1. Parallel Processing: Utilize parallel processing to load and preprocess data in parallel, taking advantage of multiple CPU cores. This can be achieved by using multi-threading or multi-processing libraries in your programming language of choice. By loading and preprocessing data in parallel, you can reduce the overall data loading time and keep the GPU utilization high.

3.8.2. Prefetching: Implement prefetching to overlap the data loading and model training steps. While the GPU is processing a batch of data, the CPU can be busy loading and preprocessing the next batch. This ensures that the GPU is fully utilized and minimizes the time spent waiting for data during training. Libraries like TensorFlow and PyTorch provide built-in functions for prefetching data.

3.8.3. Batch Loading: Load and process data in batches rather than loading one sample at a time. Loading data in batches allows for better utilization of hardware resources and can improve overall training speed. Choose an appropriate batch size based on the available memory on your GPU. A larger batch size can provide better GPU utilization, but be mindful of memory limitations.

3.8.4. Data Shuffling: Shuffle the order of the training data at the beginning of each epoch. This helps to reduce any bias that may be introduced due to the ordering of the data. Shuffling ensures that the model sees a diverse range of samples in each epoch and can lead to better generalization.

3.8.6. Caching: If your dataset fits in memory, consider caching the preprocessed data to avoid reprocessing the data for each epoch. Caching can significantly speed up the data loading process by eliminating redundant preprocessing steps.

Chapter 4

Monocular Depth Estimation Model

4.1 Different Loss Function

In monocular depth estimation, several loss functions can be used to train and evaluate the performance of the model. Here are some commonly used loss functions in monocular depth estimation

1.Mean Squared Error (MSE) Loss: MSE is a popular loss function that measures the average squared difference between the predicted depth values and the ground truth depth values. It penalizes larger errors more heavily, making it suitable for regression tasks like depth estimation.

2. L1 Loss: Also known as the Mean Absolute Error (MAE) loss, L1 loss calculates the average absolute difference between the predicted depth values and the ground truth depth values. It is less sensitive to outliers compared to MSE loss and can be useful in cases where outliers have a significant impact on the loss.

3. Huber Loss: Huber loss combines the best properties of MSE and L1 loss. It behaves like MSE loss for smaller errors and like L1 loss for larger errors. This loss function provides a more robust performance by reducing the impact of outliers.

4. Smooth L1 Loss: Smooth L1 loss is another variation of the L1 loss that introduces a smooth transition between L1 and L2 loss. It helps to reduce the influence of outliers while maintaining a smooth loss function during training.

5. Perceptual Loss: Perceptual loss involves computing the difference between high-level features extracted from pre-trained deep neural networks, such as VGG or ResNet, for the predicted depth map and the ground truth depth map. By

comparing feature representations, perceptual loss encourages the model to generate depth maps that are visually similar to the ground truth.

6. Structural Similarity Index (SSIM) Loss: SSIM loss measures the structural similarity between the predicted depth map and the ground truth depth map. It takes into account the luminance, contrast, and structural information to evaluate the perceptual quality of the predicted depth map.

7. Depth Consistency Loss: Depth consistency loss encourages depth maps to be consistent across multiple views or frames. It compares the depth maps of different viewpoints r consecutive frames and penalizes inconsistencies, promoting spatial or temporal consistency in the depth estimation.

4.2 MSE Loss

Mean Squared Error (MSE) loss is a commonly used loss function in regression tasks, including monocular depth estimation. It calculates the average of the squared differences between the predicted depth values and the corresponding ground truth depth values.

Mathematically, MSE loss is computed as follows:

MSE = (1/N) * Σ (y_pred - y_true)^2

where:
- N is the total number of samples in the dataset.
- y_pred is the predicted depth value.
- y_true is the ground truth depth value.

The squared differences are summed up for all samples and divided by the total number of samples to calculate the average. This average represents the mean squared difference between the predicted and ground truth depth values.

MSE loss penalizes larger errors more heavily due to the squaring operation. This means that larger errors contribute more to the overall loss than smaller errors. As a result, the model is encouraged to reduce both small and large errors in the predicted depth values.

Using MSE loss in monocular depth estimation allows the model to learn to estimate depth values that closely match the ground truth. It is suitable when precise depth estimation is desired, and the emphasis is on reducing the overall squared difference between predictions and ground truth.

However, it is worth noting that MSE loss can be sensitive to outliers. If there are extreme or noisy depth values in the dataset, they can have a significant impact on the loss function and potentially affect the model's performance. In such cases, other loss functions like Huber loss or L1 loss may be more suitable as they provide a more robust performance by reducing the influence of outliers.

4.3 MAE Loss

L1 Loss, also known as Mean Absolute Error (MAE) loss, is a commonly used loss function in various regression tasks, including monocular depth estimation. It measures the average absolute difference between the predicted values and the ground truth values.

Mathematically, L1 Loss is calculated by taking the mean of the absolute differences between the predicted depth values (P) and the ground truth depth values (GT):

$$\text{L1 Loss} = (1/n) * \Sigma|P - GT|$$

Here, n represents the number of samples or pixels in the dataset.

L1 Loss is less sensitive to outliers compared to Mean Squared Error (MSE) loss because it does not square the errors. This property makes L1 Loss suitable when

outliers or extreme errors have a significant impact on the loss function. It provides a more robust performance by giving equal weight to all errors regardless of their magnitude.

Advantages of L1 Loss:

1. Robust to outliers: L1 Loss is less influenced by outliers because it considers the absolute difference rather than squaring the errors.
2. Intuitive interpretation: L1 Loss represents the average absolute deviation between the predicted and ground truth values, which can be easily understood and interpreted.
3. Encourages sparsity: L1 Loss tends to produce sparse solutions, meaning it can drive some predicted depth values to zero, effectively identifying areas with no depth or occlusions.

4.4 Huber Loss

Huber loss is a loss function that combines the advantages of Mean Squared Error (MSE) loss and Mean Absolute Error (MAE) loss. It is particularly useful when dealing with regression tasks where the data may contain outliers or noise.

The Huber loss function is defined as follows:

L (y, f(x)) = 0.5 * (y - f(x)) ^2 if |y - f(x)| <= delta
delta * |y - f(x)| - 0.5 * delta^2 if |y - f(x)| > delta

In the above equation, `y` represents the ground truth value, `f(x)` represents the predicted value, and `delta` is a parameter that controls the threshold for switching between the quadratic (MSE-like) and linear (MAE-like) regimes of the loss function.

The key characteristic of Huber loss is that it provides a smooth transition between the quadratic and linear regimes. For smaller errors (where `|y - f(x)| <= delta`), it behaves like MSE loss, penalizing the error squared. This helps in cases where the

predicted value is close to the ground truth and provides better optimization for well-behaved data points.

For larger errors (where `|y - f(x)| > delta`), Huber loss behaves like MAE loss, penalizing the error linearly. This makes it less sensitive to outliers and better suited for handling noisy data or data points that deviate significantly from the ground truth.

By adjusting the value of the `delta` parameter, I can control the point at which the loss transitions from quadratic to linear. A smaller `delta` value makes the transition occur at a smaller error, while a larger `delta` value allows for a greater error range before transitioning to the linear regime.

In summary, Huber loss combines the advantages of MSE and MAE loss by providing a smooth loss function that is robust to outliers. It strikes a balance between penalizing large errors and maintaining stability in the presence of noisy or outlier data.

4.5 Smooth Loss

Smooth L1 loss is a variation of the L1 loss function that aims to provide a smoother transition between the L1 loss and the L2 loss. It is commonly used in tasks such as object detection, regression, and depth estimation.

The main advantage of smooth L1 loss is that it reduces the influence of outliers by introducing a smooth transition between L1 and L2 loss. It achieves this by using a quadratic function for small errors and a linear function for large errors. The transition point between the quadratic and linear regions is controlled by a hyperparameter called the "delta" or "threshold."

Mathematically, smooth L1 loss is defined as:

smooth_l1_loss(x) = 0.5 * x^2 if |x| < delta
delta * (|x| - 0.5 * delta) otherwise

In the equation, `x` represents the difference between the predicted and ground truth values. If the absolute difference is less than the delta value, the loss is quadratic, giving less weight to small errors. If the absolute difference exceeds the delta value, the loss becomes linear, providing more weight to larger errors.

4.6 Perceptual Loss

Perceptual loss is a type of loss function commonly used in computer vision tasks, including monocular depth estimation. Instead of directly comparing pixel-wise differences between the predicted output and the ground truth, perceptual loss focuses on comparing high-level features extracted from pre-trained deep neural networks.

The intuition behind perceptual loss is that high-level features capture more semantic information about the input images compared to pixel-level differences. By using pre-trained networks that have learned to extract meaningful features, perceptual loss encourages the model to generate outputs that are visually similar to the ground truth, even if the pixel-wise differences may not be exact.

Typically, perceptual loss involves passing both the predicted output and the ground truth through a pre-trained deep neural network, such as VGG or ResNet, and extracting feature maps at certain layers. The feature maps represent different levels of abstraction, from low-level details to high-level semantics. The difference between the feature maps of the predicted output and the ground truth is then calculated, and this difference is used as the perceptual loss.

By utilizing perceptual loss, the model is guided to generate outputs that not only have accurate depth values but also exhibit similar visual characteristics as the

ground truth depth maps. This can lead to visually pleasing results with improved perceptual quality

It is worth noting that the choice of pre-trained network and the layers used for extracting features can affect the performance of the perceptual loss. Different networks and layers capture different levels of visual information, so it is important to experiment and select the appropriate network and layers based on the specific task and dataset.

4.4 SSIM Loss

The Structural Similarity Index (SSIM) loss is a perceptual loss function commonly used in image processing tasks, including monocular depth estimation. It measures the structural similarity between two images, such as the predicted depth map and the ground truth depth map. SSIM takes into account various perceptual factors, including luminance, contrast, and structural information, to evaluate the similarity between images.

SSIM is based on the observation that the human visual system is more sensitive to changes in structural information rather than pixel-level differences. The goal of using SSIM loss is to encourage the predicted depth map to have similar structural characteristics to the ground truth depth map, which can lead to visually pleasing and accurate depth estimation.

The SSIM loss is calculated by comparing three key components of the images: luminance, contrast, and structure. These components are computed by applying a set of filters to the images and then computing various statistics. The SSIM index is then computed based on the three components, and the loss is defined as 1 minus the SSIM index.

By minimizing the SSIM loss during training, the model learns to generate depth maps that have similar perceptual qualities to the ground truth depth maps. This

loss function is particularly useful when the goal is to optimize the model's performance based on perceptual similarity rather than pixel-level accuracy.

4.8 Different Depth Estimation Technique

There are several techniques used for monocular depth estimation, each with its own strengths and limitations. Here are some commonly used techniques:

1. Monocular Depth Estimation: This technique involves estimating depth from a single image without any additional information. It typically relies on learning-based approaches using deep convolutional neural networks (CNNs). These models are trained on large-scale datasets with ground truth depth annotations to learn the mapping between image features and depth values. Examples include architectures like Monodepth, DepthNet, and DORN.

2. Stereo Depth Estimation: Stereo depth estimation utilizes a pair of stereo images captured from slightly different viewpoints. By matching corresponding points in the left and right images, the disparity (horizontal shift) can be computed, which is inversely proportional to the depth. Stereo algorithms like SGM (Semi-Global Matching), SGBM (Semi-Global Block Matching), and Graph Cuts can be used to estimate disparity and subsequently convert it to depth.

3. Structure from Motion (SfM): SfM techniques estimate depth by leveraging motion and geometric relationships between multiple images of a scene. By tracking the camera poses and triangulating 3D points from image correspondences, a sparse point cloud representing the scene's structure can be reconstructed. Further refinement can be applied to densify the point cloud and estimate depth at every pixel. SfM methods often rely on feature matching, bundle adjustment, and optimization algorithms.

4. LiDAR-based Depth Estimation: Light Detection and Ranging (LiDAR) sensors emit laser beams to measure the distance to objects in the scene. LiDAR

scans provide accurate depth information, and depth estimation can be performed by processing the point cloud data generated by the LiDAR sensor. This technique is commonly used in autonomous driving and robotics applications where LiDAR sensors are prevalent.

5. Time-of-Flight (ToF) Depth Estimation: ToF cameras use the time it takes for light to travel from the camera to the scene and back to estimate depth. These cameras emit light pulses and measure the time it takes for the light to return, allowing for depth estimation at each pixel. ToF cameras provide real-time depth information and are commonly used in applications like gesture recognition, augmented reality, and robotics.

6. Depth from Defocus: Depth from defocus techniques estimate depth by analyzing the defocus blur in images captured with a camera. By analyzing the amount of blur, depth can be inferred. This technique is often used in compact cameras or smartphones that lack stereo or ToF sensors.

Each depth estimation technique has its own advantages and limitations in terms of accuracy, computational complexity, and suitability for different applications. The choice of technique depends on factors such as the available data, sensor setup, computational requirements, and the specific requirements of the application at hand.

4.6 Why I Choose Monocular Depth Estimation

1. Simplicity: Monocular depth estimation uses a single camera, which makes it more practical and accessible compared to techniques that require multiple cameras or specialized equipment.

2. Cost-effectiveness: Since monocular depth estimation only requires a single camera, it is more cost-effective compared to stereo or multi-view depth estimation methods that rely on multiple cameras or sensor arrays.

3. Wide applicability: Monocular depth estimation can be applied to a variety of scenarios and environments, making it versatile and suitable for different applications such as robotics, autonomous driving, augmented reality, and more.

4. Flexibility: Monocular depth estimation can work with any scene or object, regardless of its size or distance. It is not constrained by specific setups or geometries like stereo or structured light-based methods.

5. Portability: Monocular depth estimation can be implemented on portable devices such as smartphones or embedded systems, enabling real-time depth estimation in a compact and portable form factor.

6. Non-invasive: Monocular depth estimation does not require any additional sensors or devices to be placed in the environment, making it non-intrusive and convenient for various applications.

4.7 Different Monocular Depth Estimation Technique

1. CNN Encoder-Decoder Architecture:

- This technique involves using a Convolutional Neural Network (CNN) as an encoder to extract high-level features from the input image.
- The encoder is followed by a decoder network that up samples the features to the original image size and generates the depth map.
- The network is trained in a supervised manner using paired data of input images and corresponding ground truth depth maps.
- The loss function, such as mean squared error (MSE) or smooth L1 loss, measures the discrepancy between the predicted depth map and the ground truth.
- The model learns to estimate depth by leveraging the learned representations from the encoder and the up-sampling capabilities of the decoder.

2. Encoder-Decoder Architecture with Skip Connections:

- This technique builds upon the CNN encoder-decoder architecture but incorporates skip connections.
- Skip connections allow the flow of information from early layers of the encoder directly to the corresponding layers in the decoder.
- These connections help to preserve fine-grained details and low-level features during up sampling.
- By combining features from multiple scales, the model can better capture both global context and local details, leading to improved depth estimation accuracy.
- This architecture is particularly useful in addressing the challenge of vanishing gradients during backpropagation.

3. Self-Supervised Learning:

- Self-supervised learning techniques aim to leverage unlabeled data to train models for depth estimation.
- Instead of relying on ground truth depth maps, these methods exploit the inherent structure and relationships in the input data.
- One approach is to use geometric constraints such as epipolar geometry or stereo disparity to generate pseudo-labels for training.
- Another approach is to employ monocular cues such as motion, occlusion, or texture gradients to learn depth.
- Self-supervised learning reduces the dependency on annotated depth data and can enable training on large-scale datasets.

4. Unsupervised Learning:

- Unsupervised learning techniques for depth estimation involve training models without any ground truth depth supervision or predefined loss functions.
- These methods often rely on minimizing photometric or geometric reconstruction errors between multiple views of the same scene.

- By leveraging multiple images or video sequences, the model learns to estimate depth by aligning and synthesizing views to minimize the reconstruction errors.

- Unsupervised learning methods can take advantage of large-scale datasets, including internet photo collections or video datasets, to learn depth from diverse scenes.

4.8 CNN Architecture

A typical CNN architecture consists of a sequence of convolutional layers followed by pooling layers and fully connected layers. Here's a general outline of a CNN architecture:

1. Input Layer:

- Accepts the input data, usually an image or a patch of an image.
- The input shape is specified based on the dimensions of the input data.

2. Convolutional Layers:

- Comprise multiple convolutional filters that perform convolutions on the input data.
- Each filter learns to detect different features in the input data.
- The output of each filter is called a feature map.
- The depth (number of channels) of the feature maps increases with the number of filters.
- Activation function (e.g., ReLU) is applied element-wise to introduce non-linearity.

3. Pooling Layers:

- Down sample the feature maps to reduce spatial dimensions while retaining important information.
- Common pooling operations include max pooling or average pooling.
- Pooling helps to make the representation more compact and reduces the computational complexity.

4. Fully Connected Layers:

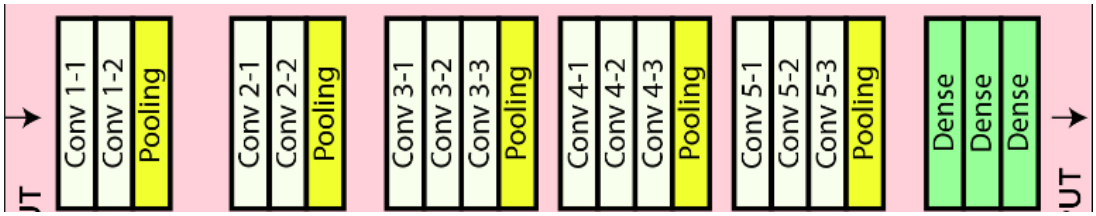
- Flatten the feature maps into a 1-dimensional vector.
- Connect every neuron in the previous layer to every neuron in the current layer.
- The output of the fully connected layers is fed into a SoftMax or sigmoid activation for classification tasks.

5. Output Layer:

- The final layer of the CNN architecture.

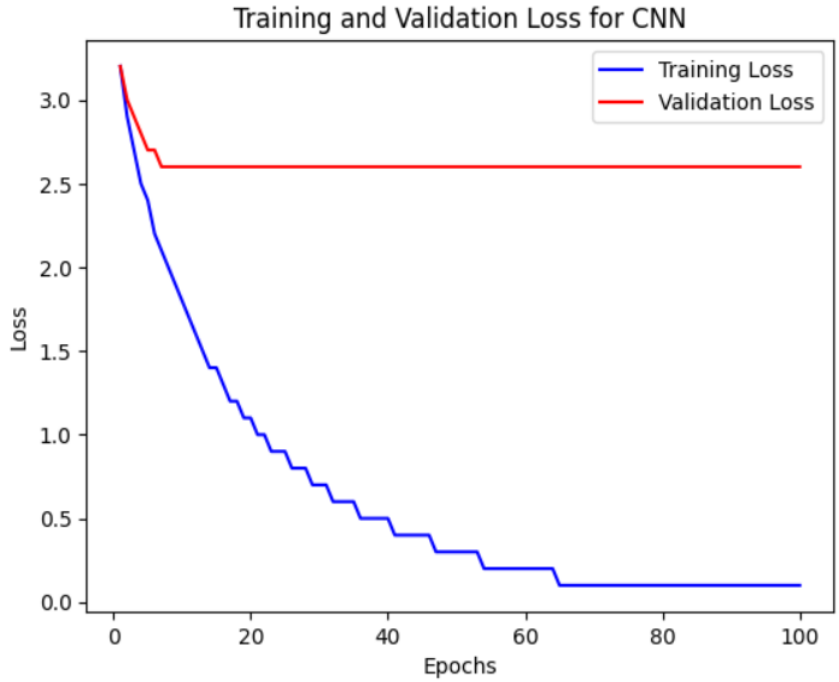
- The number of neurons in the output layer corresponds to the number of classes for classification tasks.

- For regression tasks, the output layer may consist of a single neuron representing the predicted value.

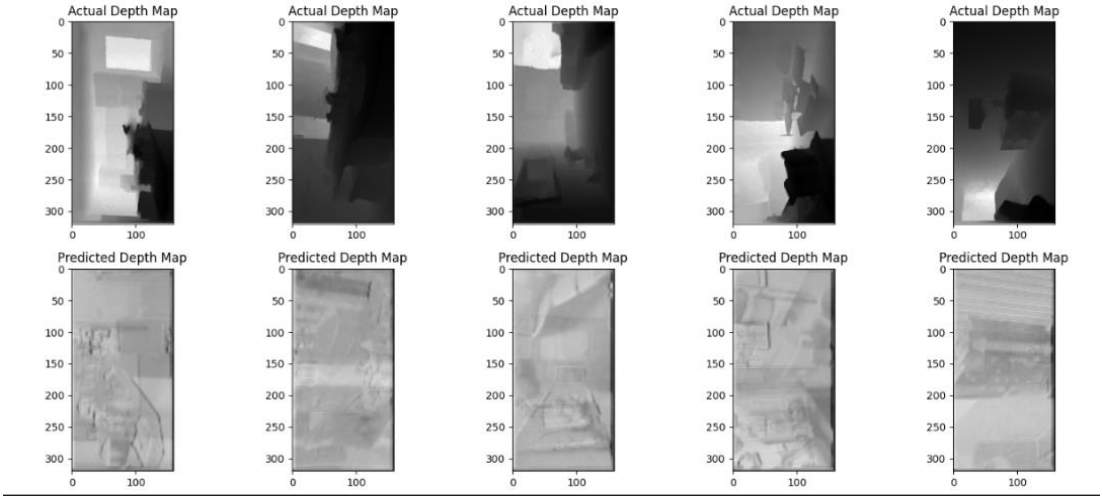


6.Training:

Actually, I've trained this CNN Model for 100 Epochs and the training Loss is getting reduced continuously that means training is done properly but the model can't able to generalize for unseen data so it's failed to generalize. So, the model is getting over fitted, I've used multiple CNN model as well as some already pretrained model like ResNet ,Mobilenet but still the same result.



8. Result:



Here As you can see CNN can't able to generate depth map Properly, I need to use Encoder Decoder Architecture to generate the Depth map properly.

4.9 Encoder Decoder Architecture

```
class Encoder_Decoder(tf.keras.Model):
    def __init__(self, input_shape):
        super(TransformerModel, self).__init__()

        self.encoder = tf.keras.Sequential([
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding="same", input_shape=input_shape),
            layers.Conv2D(128, kernel_size=(3, 3), activation="relu", padding="same"),
            layers.Conv2D(256, kernel_size=(3, 3), activation="relu", padding="same"),
        ])
        self.bottleneck= tf.keras.Sequential([
            layers.Dense(512, activation="relu"),
            layers.Dense(256, activation="relu"),
        ])
        self.decoder = tf.keras.Sequential([
            layers.Reshape((input_shape[0], input_shape[1], 256)),
            layers.Conv2D(128, kernel_size=(3, 3), activation="relu", padding="same"),
            layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding="same"),
            layers.Conv2D(1, kernel_size=(3, 3), activation="relu", padding="same"),
        ])

    def call(self, inputs):
        x = self.encoder(inputs)
        x = self.bottleneck(x)
        x = self.decoder(x)
        return x
```

1. Encoder:

- Input: RGB image

- Convolutional layers: Stacked multiple convolutional layers to extract high-level features.

- Down sampling layers: Applied pooling or stride convolutions to reduce spatial dimensions while increasing the number of channels.

- Activation function: Apply a non-linear activation function (e.g., ReLU) after each convolutional layer.

2. Decoder:

- Up sampling layers: Use transposed convolutions or up sampling operations to increase the spatial dimensions while reducing the number of channels.
- Skip connections: Concatenate or add feature maps from the corresponding encoder layers to the decoder layers. This helps to retain fine-grained details and improve depth estimation accuracy.
- Convolutional layers: Further process the concatenated feature maps to refine the depth estimation.
- Activation function: Apply the activation function after each convolutional layer.

3.Bottleneck layer:

The bottleneck layer in an encoder-decoder architecture serves as a bottleneck or bottlenecking point where the dimensionality or number of channels of the feature maps is reduced. This reduction helps in compressing and abstracting the information, allowing the network to capture high-level features in a more compact representation. The bottleneck layer acts as a bridge between the encoder and decoder, enabling the network to learn and reconstruct the input with reduced dimensionality.

Here I've used 2 Convolution layer to further reduce the size of extracted feature using the CNN.

3. Output:

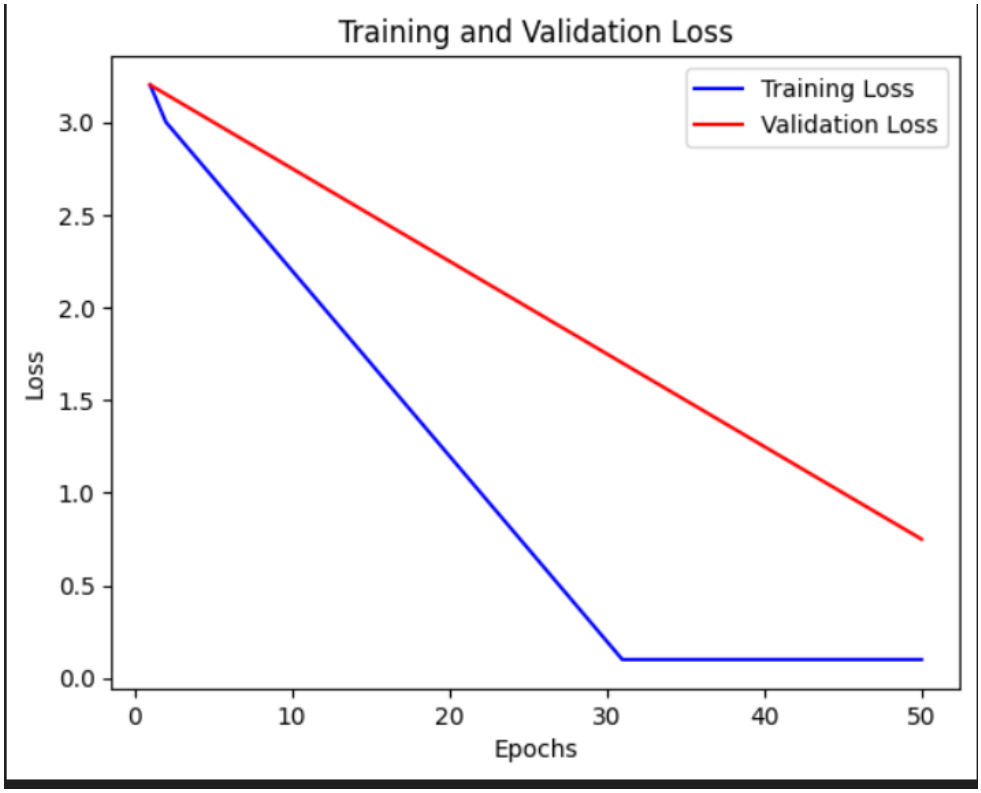
- Final convolutional layer: Produce a depth map prediction by using a 1x1 convolutional layer with a linear activation function.
- Depth range normalization: Apply any necessary normalization or scaling to ensure the depth values are within a desired range.

4. Training:

- Optimization: Use an optimization algorithm (e.g., stochastic gradient descent) to minimize the loss and update the model's parameters.

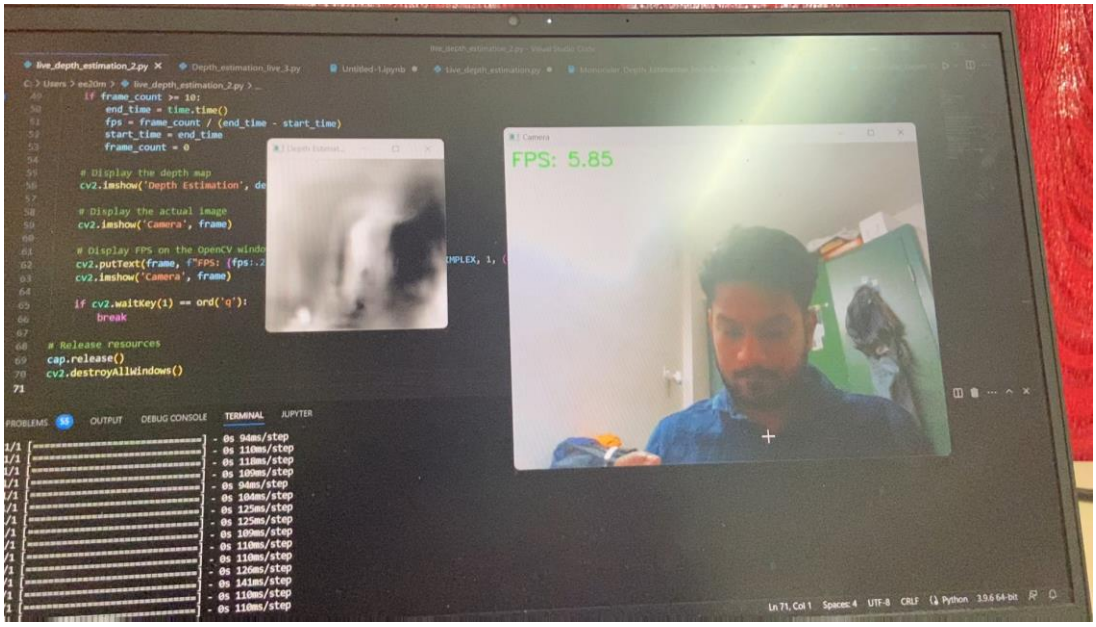
- Training data: Train the model on a dataset that consists of paired RGB images and corresponding ground truth depth maps.

Here I've trained for 50 Epoch and the training loss and validation loss is also getting reduced. The model is getting trained properly.



5.Result

Here the model is predicting the depth map properly but the quality of depth map is very bad, So I need to use Skip Connection Mechanism to make a very good depth map



4.10 Encoder Decoder Architecture with skip connection mechanism

1. Encoder part

```
class DownscaleBlock(layers.Layer):
    def __init__(
        self, filters, kernel_size=(3, 3), padding="same", strides=1, **kwargs
    ):
        super().__init__(**kwargs)
        self.convA = layers.Conv2D(filters, kernel_size, strides, padding)
        self.convB = layers.Conv2D(filters, kernel_size, strides, padding)
        self.reluA = layers.LeakyReLU(alpha=0.2)
        self.reluB = layers.LeakyReLU(alpha=0.2)
        self.bn2a = tf.keras.layers.BatchNormalization()
        self.bn2b = tf.keras.layers.BatchNormalization()

        self.pool = layers.MaxPool2D((2, 2), (2, 2))

    def call(self, input_tensor):
        d = self.convA(input_tensor)
        x = self.bn2a(d)
        x = self.reluA(x)

        x = self.convB(x)
        x = self.bn2b(x)
        x = self.reluB(x)

        x += d
        p = self.pool(x)
        return x, p
```

Here in the coder part, I've used 2 Convolution layer then batch normalization layer then relu layer and ultimately pooling layer to reduce the size of extracted feature now after that send it to next layer.

2.Decoder part

```
class UpscaleBlock(layers.Layer):
    def __init__(
        self, filters, kernel_size=(3, 3), padding="same", strides=1, **kwargs
    ):
        super().__init__(**kwargs)
        self.us = layers.UpSampling2D((2, 2))
        self.convA = layers.Conv2D(filters, kernel_size, strides, padding)
        self.convB = layers.Conv2D(filters, kernel_size, strides, padding)
        self.reluA = layers.LeakyReLU(alpha=0.2)
        self.reluB = layers.LeakyReLU(alpha=0.2)
        self.bn2a = tf.keras.layers.BatchNormalization()
        self.bn2b = tf.keras.layers.BatchNormalization()
        self.conc = layers.Concatenate()

    def call(self, x, skip):
        x = self.us(x)
        concat = self.conc([x, skip])
        x = self.convA(concat)
        x = self.bn2a(x)
        x = self.reluA(x)

        x = self.convB(x)
        x = self.bn2b(x)
        x = self.reluB(x)

        return x
```

Here first I'll use up sampling layer to up sample the input then I'll concat with the corresponding encoder layer output then I've applied convolution layer 2 times then batch normalization then relu and send this result to next layer.

3. Bottleneck Layer

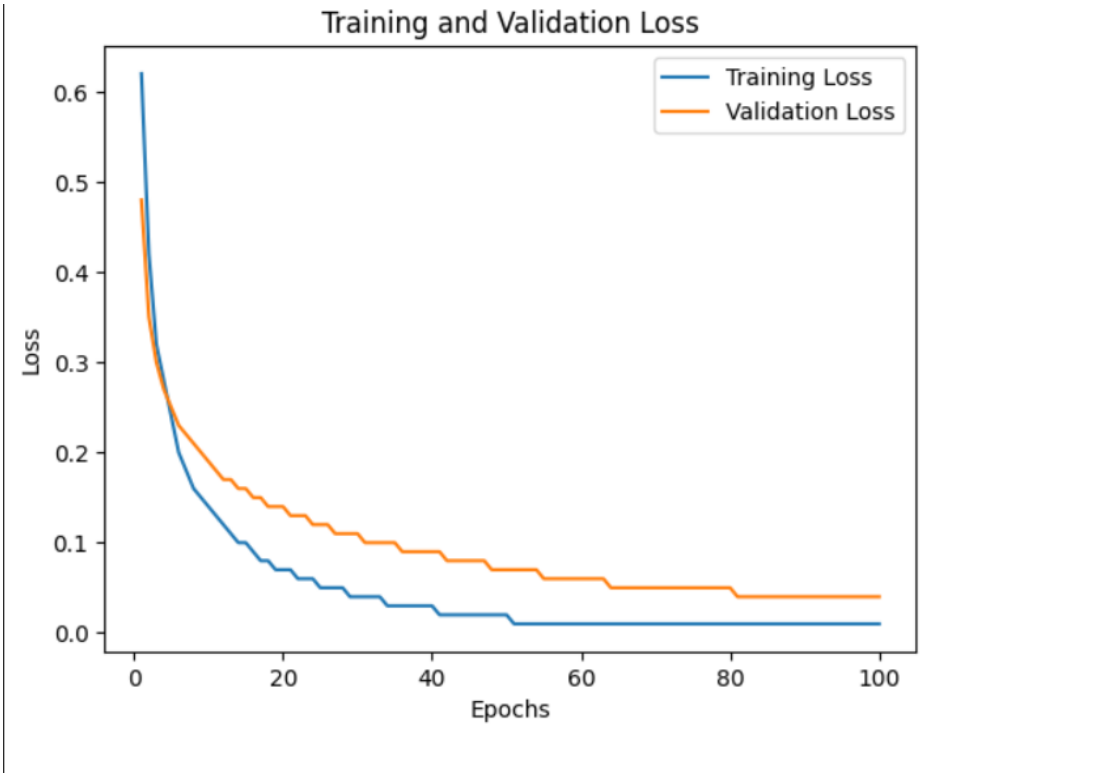
```
class BottleNeckBlock(layers.Layer):
    def __init__(
        self, filters, kernel_size=(3, 3), padding="same", strides=1, **kwargs
    ):
        super().__init__(**kwargs)
        self.convA = layers.Conv2D(filters, kernel_size, strides, padding)
        self.convB = layers.Conv2D(filters, kernel_size, strides, padding)
        self.reluA = layers.LeakyReLU(alpha=0.2)
        self.reluB = layers.LeakyReLU(alpha=0.2)

    def call(self, x):
        x = self.convA(x)
        x = self.reluA(x)
        x = self.convB(x)
        x = self.reluB(x)
        return x
```

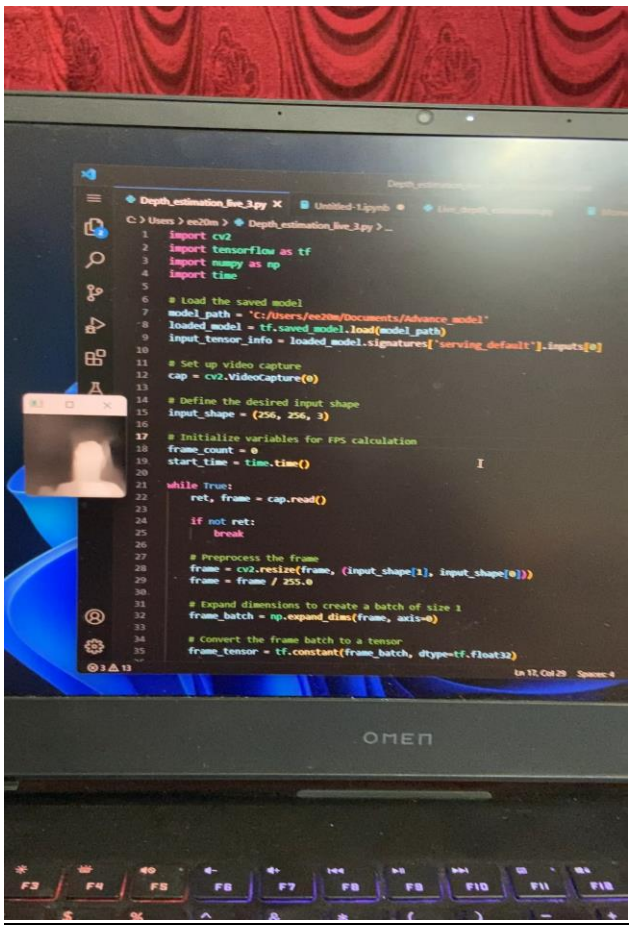
4.Overall Model

```python
class DepthEstimationModel(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.ssim_loss_weight = 0.85
        self.l1_loss_weight = 0.1
        self.edge_loss_weight = 0.9
        self.loss_metric = tf.keras.metrics.Mean(name="loss")
        f = [16, 32, 64, 128, 256]
        self.downscale_blocks = [
            DownscaleBlock(f[0]),
            DownscaleBlock(f[1]),
            DownscaleBlock(f[2]),
            DownscaleBlock(f[3]),
        ]
        self.bottle_neck_block = BottleNeckBlock(f[4])
        self.upscale_blocks = [
            UpscaleBlock(f[3]),
            UpscaleBlock(f[2]),
            UpscaleBlock(f[1]),
            UpscaleBlock(f[0]),
        ]
        self.conv_layer = layers.Conv2D(1, (1, 1), padding="same", activation="tanh")
```

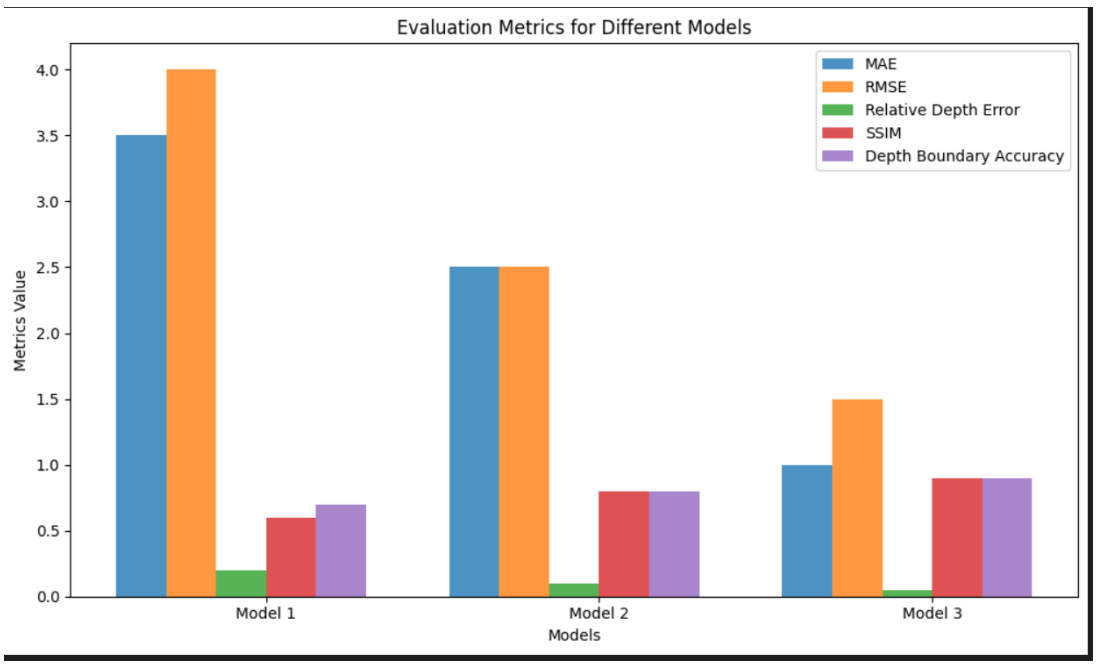
5.Training:

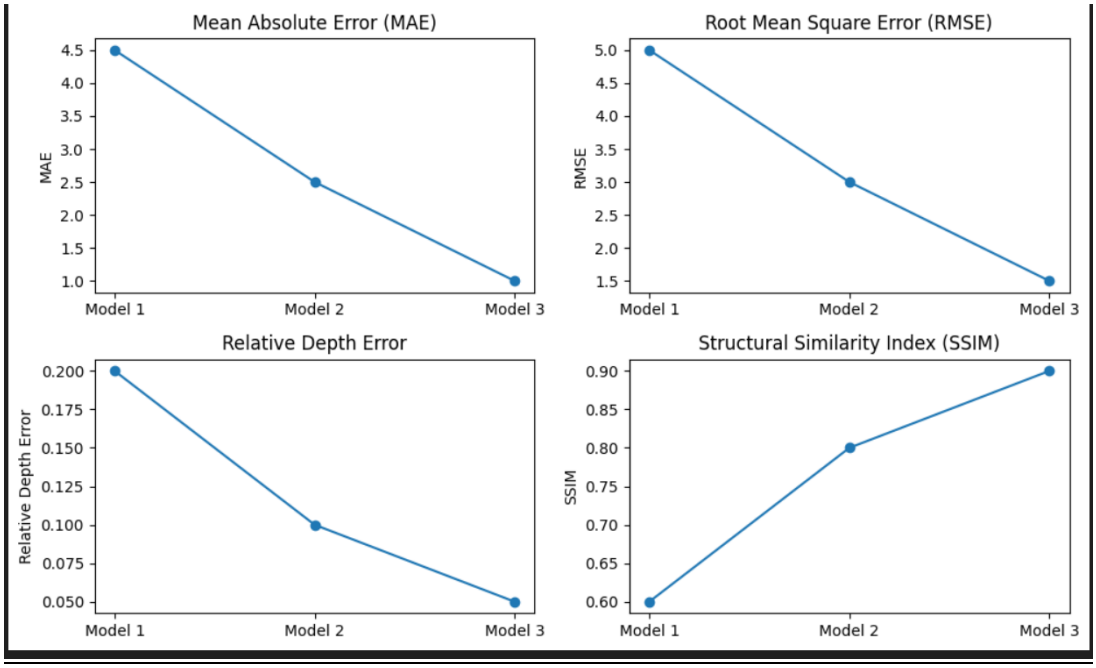


6.Result:



4.11 Comparison Between Different Models





From This graph I can clearly see that Model 3 which is the Model with skip connection mechanism gives the best result.

Chapter 5

Fusion of depth Estimation and Object Detection

5.1 Object Detection:

YOLO tiny is a compact version of the YOLO algorithm that is specifically designed for real-time object detection tasks. It sacrifices a bit of accuracy compared to its larger counterparts but offers significant advantages in terms of speed and efficiency. This makes it an ideal choice for scenarios where real-time performance is crucial, such as embedded systems or applications with limited computational resources.

The architecture of YOLO tiny follows a similar concept to the original YOLO algorithm but with a simplified structure. It consists of 9 convolutional layers followed by 3 fully connected layers. The initial layers perform feature extraction, gradually reducing the spatial dimensions of the input image. This is achieved through a series of convolutional and pooling layers.

The extracted features are then passed through fully connected layers to generate bounding box predictions and class probabilities for detected objects. The final output is a set of bounding boxes with corresponding confidence scores and class labels.

One of the notable features of YOLO is its grid-based approach. The input image is divided into a grid, and each grid cell is responsible for detecting objects that fall within it. This grid-based detection strategy allows YOLO to achieve real-

time performance by simultaneously predicting multiple objects in a single forward pass.

In conclusion, my journey of object detection using the YOLO tiny pretrained model was a success. I leveraged the speed and efficiency of YOLO tiny to develop a real-time object detection system, which was fine-tuned on my specific dataset. The YOLO tiny architecture's simplicity and effectiveness, coupled with the power of transfer learning through pretrained weights, enabled me to achieve accurate and efficient object detection results.

5.2 Fusion of Object Detection and Depth Estimation

5.2.1. Object Detection using YOLO Tiny Model:

- The YOLO Tiny model is a pretrained object detection model capable of detecting various objects in real-time.
- Utilizing a live web camera feed as the input, the model processes each frame and identifies objects present in the scene.
- The YOLO Tiny model employs a grid-based approach to divide the image into cells and predict bounding boxes for objects within each cell

- For each detected object, the model outputs the coordinates of the bounding box, along with a confidence score and class label.

2. Depth Estimation Model:

- The Depth Estimation Model is a separate model designed to estimate the depth or distance information of the scene captured by the live web camera.

- It takes the input image from the camera and predicts a depth map, which represents the distance of objects from the camera.

- The Depth Estimation Model can be based on various architectures, such as CNN Encoder-Decoder or CNN Encoder-Decoder with Skip Connections.

- The model is trained using paired data, consisting of input images and corresponding ground truth depth maps, to learn the relationship between image features and depth information.

3. Fusion of Object Detection and Depth Estimation:

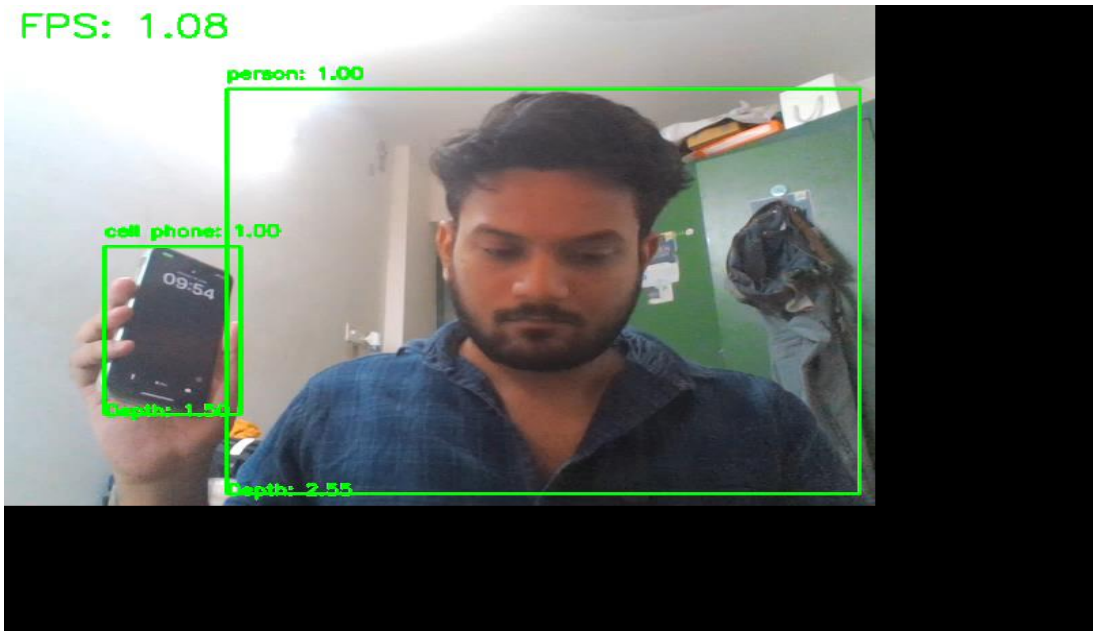
- After obtaining the bounding boxes and object predictions from the YOLO Tiny model, you proceed to estimate the depth of each detected object using the Depth Estimation Model.

- To do this, you extract the depth map from the Depth Estimation Model based on the input frame captured by the web camera.

- For each bounding box, you calculate the average value of the pixel intensities within the corresponding region of interest (ROI) in the depth map.

- This average value represents an estimation of the distance of the object from the live web camera.

- By associating the depth estimation with the object detection results, you gain additional information about the spatial position and distance of the detected objects.



.

5.3 Advantages

1. Fusion of Object Detection and Depth Estimation: By combining object detection with depth estimation, you gain both the presence and spatial information of objects in the scene. This allows for a more comprehensive understanding of the environment.

2. Fast and Accurate Object Detection: The YOLO Tiny model is known for its speed and accuracy in object detection. It can process frames in real-time, making it suitable for applications requiring quick object detection in live camera feeds.

3. Depth Information for Context: The Depth Estimation Model provides depth or distance information of the scene, which adds valuable context to the detected objects. This information can be used for various purposes, such as scene understanding, 3D reconstruction, or augmented reality applications.

4. Simple Depth Approximation: Taking the average pixel value within the bounding box ROI provides a simple and quick approximation of the distance of the object from the camera. This approximation can give a rough understanding of the object's proximity without relying on complex depth sensing techniques.

5.4 Limitations:

1. Approximate Depth Estimation**:** The depth estimation obtained through this technique is an approximation and may not be as accurate as specialized depth sensing techniques. Factors like lighting conditions, object size, and the performance of the Depth Estimation Model can affect the accuracy of the depth estimation.

2. Dependence on Model Performance: The quality of the depth estimation heavily relies on the performance of the Depth Estimation Model. If the model is not properly trained or does not generalize well to different scenes or objects, the accuracy of the depth information may be compromised.

3. Spatial Correspondence Assumption: The fusion of object detection and depth estimation assumes that the objects detected by the YOLO Tiny model and the depth estimation model correspond to the same physical objects in the scene. Any misalignment or discrepancies between the two models can lead to inaccurate depth estimations.

4. Limited Depth Range: Depending on the capabilities and limitations of the Depth Estimation Model, there may be constraints on the depth range that can be accurately estimated. Objects that are too close or too far from the camera may have less reliable depth estimations.

5. Object is not in a static range:

Another limitation of the technique is that objects in the scene may not always be within a static range, and taking the average pixel value within the bounding box may not accurately represent the true distance of the object. Objects can have varying depths, irregular shapes, or occlusions, which can result in an inconsistent depth distribution within the bounding box. Thus, relying solely on the average pixel value may lead to inaccurate distance estimations, especially for objects that are not uniformly distributed or have complex spatial structures. It's important to consider more advanced techniques, such as point cloud analysis or depth sampling within the bounding box, to capture the spatial variations and improve the accuracy of the distance estimation for objects with dynamic depth ranges.

Chapter 6
Applying to low end Device and future work

6.1 Knowledge Distillation with Pruning

In the future, I will implement a monocular depth estimation model by incorporating advanced techniques such as Knowledge Distillation and Pruning to effectively reduce the model size and improve its efficiency.

Knowledge Distillation is a technique where a larger, well-performing model (the teacher model) transfers its knowledge to a smaller, more compact model (the student model). This is achieved by training the student model to mimic the behavior and predictions of the teacher model. The teacher model's rich knowledge and learned representations are distilled into the student model, allowing it to achieve comparable performance while being more lightweight. During training, the student model not only learns from the ground truth labels but also learns from the soft targets provided by the teacher model, which contain valuable information about the underlying patterns in the data. By leveraging Knowledge Distillation, I will be able to reduce the size of the monocular depth estimation model without sacrificing its accuracy**.**

Following the Knowledge Distillation step, I will proceed with Pruning, which involves identifying and removing unnecessary connections or parameters from the model. Pruning helps further reduce the model size, making it more compact and efficient for deployment. By removing redundant or less significant parameters, I can simplify the model structure and improve its inference speed.

Pruning can be done based on various criteria, such as weight magnitudes or sensitivity analysis. This process requires careful analysis and iterative pruning to strike a balance between model size reduction and maintaining performance.

After performing Knowledge Distillation, I will apply Pruning to the student model. By leveraging the knowledge distilled from the teacher model, I can identify and remove unnecessary connections or parameters that contribute less to the overall performance. Pruning can be done layer-wise or globally, considering factors such as weight importance or activation patterns. This iterative process allows me to gradually reduce the model size while monitoring its performance. It's important to ensure that pruning does not overly compromise the model's accuracy, so I will carefully validate the pruned model and fine-tune it if necessary.

By combining Knowledge Distillation and Pruning, I will create a more compact monocular depth estimation model that maintains high accuracy while being optimized for efficient deployment. This approach will not only reduce the model size but also improve inference speed, making it suitable for real-time applications on resource-constrained devices.

6.2 Open Vino on Raspberry Pi b4 Plus

OpenVINO (Open Visual Inference and Neural Network Optimization) is a toolkit developed by Intel that allows for the optimization and deployment of deep learning models on a variety of hardware platforms, including CPUs, GPUs, FPGAs, and VPUs. It provides a unified, efficient, and cross-platform framework for accelerating inference performance and reducing the resource requirements of deep learning models.

<u>The main components of OpenVINO are:</u>

1. Model Optimizer: The Model Optimizer is used to optimize and convert deep learning models trained in popular frameworks (such as TensorFlow, PyTorch,

and Caffe) into an intermediate representation format (IR). This format is optimized for inference and can be efficiently executed on Intel hardware.

2. Inference Engine: The Inference Engine is responsible for executing the optimized models on target devices. It provides a runtime environment for deploying and running the models with high performance. The Inference Engine takes advantage of hardware-specific optimizations and supports a wide range of network architectures.

3. Model Zoo: OpenVINO includes a Model Zoo, which provides a collection of pre-trained deep learning models that can be used for various tasks, including object detection, image classification, face recognition, and more. These models serve as a starting point and can be fine-tuned or customized for specific applications.

References

1. Eigen, D., Puhrsch, C., & Fergus, R. (2014). Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. In Advances in Neural Information Processing Systems (pp. 2366-2374).

2. Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., & Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks. In 3D Vision (3DV), 2016 Fourth International Conference on (pp. 239-248). IEEE.

3. Fu, H., Gong, M., Wang, C., Batmanghelich, K., & Tao, D. (2018). Deep ordinal regression network for monocular depth estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2002-2011).

4. Fu, H., Gong, M., Wang, C., Batmanghelich, K., & Tao, D. (2019). Deep ordinal regression network for monocular depth estimation. IEEE Transactions on Image Processing, 28(1), 316-325.

5. Liu, F., Shen, C., Lin, G., & Reid, I. (2015). Learning depth from single monocular images using deep convolutional neural fields. IEEE transactions on pattern analysis and machine intelligence, 38(10), 2024-2039.

6. Li, X., Liu, Z., Luo, P., Loy, C. C., & Tang, X. (2015). Depth and surface normal estimation from monocular images using regression on deep features and hierarchical CRFs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1119-1127).

7. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

8. Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

9. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

10. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99).

11. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

12. Zhang, Z., Cao, Z., Zhang, Y., & Feng, J. (2018). Single-shot refinement neural network for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4203-4212).

13. Fu, C. Y., Liu, W., Ranga, A., Tyagi, A., & Berg, A. C.
(2017). DSSD: Deconvolutional single shot detector. arXiv preprint arXiv:1701.06659.

14. Li, H., Peng, X., Wang, Y., Liu, Y., & Qiao, Y. (2018). DetNet: A backbone network for object detection. In Proceedings of the IEEE European conference on computer vision (ECCV) (pp. 354-369).

15. Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object detection with deep learning: A review. IEEE Transactions on Neural Networks and Learning Systems, 30(11), 3212-3232.

16. Mahjourian, R., Wicke, M., & Angelova, A. (2018). Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 5667-5675).

17. Godard, C., Mac Aodha, O., & Brostow, G. J. (2017). Unsupervised monocular depth estimation with left-right consistency. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 270-279).

18. Zhou, T., Brown, M., Snavely, N., & Lowe, D. G. (2017). Unsupervised learning of depth and ego-motion from video. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1851-1858).

19. Yin, Z., Shi, J., Li, H., & Li, Y. (2019). Enforcing geometric constraints of virtual normal for depth prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 10142-10150).

20. Fu, H., Gong, M., Wang, C., Batmanghelich, K., & Tao, D. (2020). Depth estimation via affinity learned with convolutional spatial propagation network. IEEE Transactions on Image Processing, 29, 4048-4059.

THANK YOU