NORTH SOUTH UNIVERSITY

Department of Electrical and Computer Engineering

Monocular Depth Estimation

Md. Arafat Islam 2111293642

Irfan Ali Sadab 2011054642

Instructor: Md. Ishan Arefin Hossain (IAH)

CSE 299: Junior Design

Section 16

Team: ENIGMA

APPROVAL

Md. Arafat Islam (ID-2111293642) and Irfan Ali Sadab (ID # 2021280042) from Electrical and Computer Engineering Department of North South University, have worked on the Junior Design Project titled “Monocular Depth Estimation” under the supervision of Ishan Arefin Hossain partial fulfillment of the requirement for the degree of Bachelor of Science in Engineering and has been accepted as satisfactory.

Supervisor’s Signature

.....

Ishan Arefin Hossain

Lecturer

Department of Electrical and Computer Engineering

North South University

Dhaka, Bangladesh.

Chairman’s Signature

.....

Dr. Rajesh Palit

Professor

Department of Electrical and Computer Engineering

North South University

Dhaka, Bangladesh.

DECLARATION

This is to declare that this project is our original work. No part of this work has been submitted elsewhere. partially or fully for the award of any other degree or diploma. All project related information will remain confidential and shall not be disclosed without the formal consent of the project supervisor. Relevant previous works presented in this report have been properly acknowledged and cited. The plagiarism policy, as stated by the supervisor, has been maintained.

Students' names & Signatures

1. Md. Arafat Islam

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

2. Irfan Ali Sadab

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

ACKNOWLEDGEMENTS

The authors would like to sincerely thank Ishan Arefin Hossain, Lecturer in the Department of Electrical and Computer Engineering at North South University, for overseeing their project and conducting their research.

University, Bangladesh, for his priceless assistance, exact direction, and counsel about the research, experimentation, and theoretical studies completed throughout the current project as well as in the writing of the current report.
The Department of Electrical and Computer Engineering at North South University in Bangladesh is also appreciated by the authors for helping to make the research possible. The writers also like to express their gratitude to their family members for their unwavering support and innumerable sacrifices.

Table of content

List of Figures & Table

Abstract:

Monocular depth estimation stands as a fundamental challenge in computer vision, with far-reaching implications across numerous domains, including robotics, augmented reality, autonomous systems, and medical imaging. This project proposal outlines a comprehensive and ambitious endeavor aimed at addressing this challenge through a multifaceted approach, integrating deep learning methodologies and state-of-the-art convolutional neural network (CNN) architectures.

The research seeks to develop a robust and versatile solution capable of accurately predicting depth maps from single images, transcending the limitations of traditional methods. Key to this endeavor is the strategic integration of cutting-edge CNN architectures such as U-Net, MobileNet, and ResNet, each offering unique advantages in terms of efficiency, accuracy, and computational complexity.

The model development phase entails the design and implementation of novel CNN architectures tailored specifically for monocular depth estimation. Drawing inspiration from recent advancements in deep learning, including U-Net's efficient encoder-decoder architecture, MobileNet's lightweight design, and ResNet's residual connections, the research explores innovative designs optimized for extracting intricate depth features from single images with unparalleled efficiency and accuracy.

Training the developed models constitutes a pivotal stage of the project, involving iterative refinement through the application of state-of-the-art optimization techniques. By leveraging large-scale datasets and sophisticated training protocols, the aim is to enhance the model's capacity to discern subtle depth cues and nuances inherent in complex visual scenes, thereby fostering superior performance and adaptability.

Anticipated outcomes of this research endeavor include the development of a highly efficient and accurate monocular depth estimation model poised to catalyze advancements in various fields reliant on precise depth perception from single images. By integrating CNN architectures such as U-Net, MobileNet, and ResNet, the project aims to unlock new possibilities for applications such as autonomous navigation, augmented reality visualization, object recognition, and scene understanding.

In conclusion, this project report outlines a comprehensive and ambitious research endeavor aimed at advancing the frontier of monocular depth estimation through innovative deep learning methodologies.

Introduction:

Depth estimation stands as a fundamental pillar in the realm of computer vision, serving as a crucial component for addressing complex tasks such as 3D reconstruction and spatial perception, essential for robotics grasping and autonomous vehicle navigation. Over the years, researchers have grappled with the challenge of deriving accurate depth information from images to enable various applications. Particularly for autonomous systems like robots and vehicles, generating precise and dense depth maps is imperative for tasks including 3D reconstruction, mapping, and localization.

Traditional depth estimation methods, such as structured light projection and LiDAR, have provided high precision but often face limitations in real-time processing, hindering their practical deployment in real-world scenarios. Additionally, these methods can be constrained by environmental factors such as sunlight interference and mechanical misalignment, posing challenges to their reliability and scalability.

In response to these limitations, monocular depth estimation has emerged as a promising alternative due to its simplicity in deployment configuration and minimal hardware requirements. With the advent of deep learning and the availability of labeled depth datasets, researchers have delved into regressing depth values from single scene images using well-trained models. Early approaches predominantly relied on convolutional neural networks (CNNs), while recent advancements have seen the adoption of larger network architectures such as U-Net, ResNet, and MobileNet, each offering unique advantages in terms of efficiency, accuracy, and computational complexity.

Despite the progress achieved, monocular depth estimation methods still grapple with the loss of scene geometry information inherent in the transition from 3D space to 2D images. This limitation hampers further performance improvements, necessitating the incorporation of geometry constraints into model training. By enforcing geometry constraints, not only can depth estimation accuracy be enhanced, but also better 3D scene structure reconstruction can be achieved.

Innovative model architectures such as U-Net, with its efficient encoder-decoder architecture, and ResNet, with its residual connections, have demonstrated significant promise in enhancing depth estimation accuracy. Additionally, models like MobileNet have addressed concerns regarding computational efficiency, making them suitable for deployment in resource-constrained environments. A novel approach involves the combination of ResNet and U-Net architectures, leveraging the strengths of both to achieve superior depth estimation performance.

The cornerstone of our approach lies in the utilization of these diverse model architectures to analyze image data and infer depth information from single images. Leveraging deep learning techniques, we intend to train a lightweight model capable of efficiently estimating depth while maintaining high accuracy. This model will serve as the backbone of our web-based depth estimation platform, enabling users to effortlessly obtain depth maps from their images with minimal computational overhead.

In addition to addressing the limitations of current approaches, our project aims to enhance accessibility and usability by developing intuitive user interfaces for both the web platform and mobile application. By prioritizing user experience and simplicity, we aim to democratize access to monocular depth estimation technology and empower a broader community of users to leverage its capabilities in diverse applications

Furthermore, the integration of these models into web-based platforms offers accessibility and usability advantages. Tools like Streamlit facilitate the development of user-friendly interfaces for both web and mobile applications, enabling users to effortlessly obtain depth maps from their images with minimal computational overhead. These platforms prioritize user experience and simplicity, democratizing access to monocular depth estimation technology and empowering a broader community to leverage its capabilities across diverse applications..

In summary, our project report seeks to democratize monocular depth estimation by providing a user-friendly web platform that empower individuals and organizations to harness the power of depth information in their applications and workflows. By combining cutting-edge deep learning techniques with a focus on accessibility and efficiency, we aim to accelerate the adoption of depth estimation technology and unlock its transformative potential across diverse domains.

Literature Review:

The document reviews various approaches to monocular depth estimation (MDE) using deep learning, including supervised learning (SL), unsupervised learning (UL), and semi-supervised learning (SSL). Dos Santos et al. addressed the challenge of creating denser ground truth depth (GTD) maps from sparse LIDAR measurements, which improved model performance118. Ranftl et al. tackled issues such as changing resolution and disparity values when preparing datasets from three-dimensional movies, achieving high precision with their MDE model119. Sheng et al. proposed a lightweight SL model with local–global optimization, using an autoencoder network for depth prediction120. Zhou et al. developed a UL approach that estimates depth maps and camera pose simultaneously, facing the challenge of camera transformation estimation between frames16.

Datasets used include the Make3D dataset for quantitative results of DL algorithms98, the DIODE dataset for diverse indoor and outdoor scenes103, and the Driving Stereo dataset focusing on new metrics for stereo matching evaluation105. The results varied across different methods and datasets, with SL methods generally achieving higher accuracy due to labeled GTD but at the cost of extensive data labeling, while UL methods performed better considering the time saved on labeling15. The document concludes that despite the challenges, DL techniques show great potential for MDE, with future research needed to improve model architectures and dataset diversity18. [1]

The review paper is based lightweight monocular depth estimation method using a Token-Sharing Transformer (TST) optimized for embedded devices. The proposed TST architecture combines the design concepts of hierarchy-focused and bottleneck-focused architectures to achieve high throughput without performance drop. The paper presents experimental results showing that TST outperforms existing methods in accuracy and throughput, especially on embedded devices like NVIDIA Jetson Nano and Jetson TX2. The proposed model utilizes global token sharing to obtain accurate depth predictions efficiently. The paper also includes ablation studies validating the effectiveness of the Token-Sharing Transformer in monocular depth estimation.

For training, they used ADAM optimizer with customized Cosine annealing warm restarts learning rate scheduler. Specifically, for optimizer, we use $\beta1 = 0.9$, $\beta2 = 0.99$, learning rate = 0.0003. For scheduler, they used T0 = 10, Tmult = 2, $\gamma = 0.5$. they train for 100 epochs and apply the learning rate scheduler for additional 100 epochs. For the data augmentation, random

horizontal flips, random brightness, contrast, gamma, hue, saturation, and value are used with 0.5 probabilities. Vertical CutDepth is also utilized with 0.25 probability. [2]

This paper reviews various approaches to monocular depth estimation (MDE) using deep learning, including supervised learning (SL), unsupervised learning (UL), and semi-supervised learning (SSL). Dos Santos et al. addressed the challenge of creating denser ground truth depth (GTD) maps from sparse LIDAR measurements, which improved model performance. Ranftl et al. tackled issues such as changing resolution and disparity values when preparing datasets from three-dimensional movies, achieving high precision with their MDE model. Sheng et al. proposed a lightweight SL model with local–global optimization, using an autoencoder network for depth prediction. Zhou et al. developed a UL approach that estimates depth maps and camera pose simultaneously, facing the challenge of camera transformation estimation between frames.[3]

The documents discuss various deep learning-based models for monocular depth estimation, comparing their methodologies, challenges, and results. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Variational Auto-Encoders (VAEs), and Generative Adversarial Networks (GANs) have been effective in addressing monocular depth estimation27. The models are trained using different datasets like KITTI, NYU Depth, Cityscapes, and Make3D, each with unique characteristics and ground truth acquisition methods20. Evaluation metrics such as RMSE, RMSE log, Abs Rel, Sq Rel, and Accuracies are used to assess performance20. Challenges in monocular depth estimation include dealing with dynamic objects, occlusions, and achieving real-time performance on embedded systems4. Some models use explainability masks to minimize the impact of dynamic objects on view reconstruction6. Others incorporate camera parameters to improve transferability across different cameras and scenarios9. The semi-supervised and unsupervised methods show varying degrees of accuracy, with semi-supervised methods generally achieving better results due to additional signals like LIDAR data8. The results on the KITTI dataset show that semi-supervised methods outperform unsupervised methods, with lower error rates and higher accuracy8. However, the accuracy of semi-supervised methods heavily relies on the availability of ground truth data8. Unsupervised methods, while not requiring ground truth for training, still lag behind in accuracy9. The document concludes by highlighting the potential for future research in improving accuracy, transferability, and real-time performance, as well as exploring the mechanisms of depth estimation9.[4]

The review paper proposes a novel approach for robot navigation using monocular depth estimation (MDE) and considering relative object positions, integrating a PID controller for

obstacle navigation. It discusses related works combining depth estimation with object detection and image segmentation, highlighting various methods and frameworks for distance estimation and object tracking. The paper also explores mapless robotic navigation methods and contributions, including precise distance estimation and sensitivity analysis on monocular depth estimation's impact on distance accuracy. Additionally, it presents a map-less obstacle avoidance algorithm utilizing a pre-trained MDE model and a PID controller for robot navigation. The system integrates depth estimation, object detection, and image segmentation for target identification and distance estimation. [5]

The documents discuss various methods of monocular depth estimation based on deep learning. The methods are categorized into supervised, unsupervised, and self-supervised approaches, each with different architectures and strategies for depth prediction from single images. Supervised methods like DORN use CNNs and are trained with ground truth depth information, achieving high accuracy but at the cost of computational complexity, making them less suitable for real-time applications like robotics. Fast Depth proposes an efficient coding and decoding network architecture to reduce computational complexity and delay. Unsupervised methods, such as Monodelph, do not require ground truth depth data. Instead, they use image reconstruction as a network constraint to train the depth estimation network. These methods can reduce the error in image reconstruction by training on the disparity of the image and wrapping it to the nearby view. Self-supervised methods, like GeoNet and MonoResMatch, use additional constraints or inputs such as stereo pairs, video sequences, or optical flow to improve depth prediction accuracy. These methods can handle occlusion robustly and use adaptive geometric consistency error to improve system stability. The existing challenges include handling complex scenes with occlusions, scene clutter, and object materials. Future research directions involve adaptive methods that can adjust to changing environments with minimal supervision and the design of efficient network architectures for high-density image depth estimation. The results from evaluations on KITTI and NYU datasets show that while supervised methods generally perform better, unsupervised and self-supervised methods are closing the performance gap. The evaluation criteria include absolute relative error, root mean square error, and accuracy. [6]

The research paper focuses on generating depth maps from images using a novel methodology involving DNN and Transformers. The experiments conducted on large datasets with various input types, such as images of different resolutions, successfully captured the farthest vehicle in the input image and generated corresponding depth maps. The future scope emphasizes improving algorithm efficiency, accuracy, and developing new tools and technologies utilizing depth data.

Depth map generation is an active research area with efforts to enhance algorithm precision, efficiency, and applications relying on depth data for understanding 3D scenes and object placements.[7]

The research paper discusses various methods for monocular depth estimation using deep learning techniques. It covers the use of residual networks, attention models, and loss functions like mean squared error and structural similarity for training depth estimation models. Different approaches such as semi-supervised and unsupervised learning are explored, including the use of stereo images, left-right consistency, and motion modeling for depth prediction. The paper also highlights the challenges faced in training models, such as overfitting and the need for robust depth predictions in dynamic scenes. Overall, the study focuses on improving the accuracy and efficiency of monocular depth estimation models.[8]

The research paper discusses various methodologies for enhancing satellite image segmentation using deep learning models. One approach combines UNet++ architecture with MobileNetV2 encoder to improve segmentation accuracy efficiently[P: 2]. The study emphasizes the importance of training and validation loss in machine learning models like neural networks [P:17]. Transfer learning is explored to boost segmentation performance in scenarios with limited annotated data [P :4]. A novel Self-Supervised Learning Based Instance Segmentation Metshod is proposed to address challenges in satellite image processing, leveraging self-supervised learning and instance segmentation techniques [P:4].[9]

The research paper focuses on satellite image segmentation using a combination of UNet++ architecture and MobileNetV2 encoder for applications like disaster management, environmental monitoring, and land cover classification. The proposed model integrates the hierarchical feature capturing ability of UNet++ with the computational efficiency of MobileNet, aiming to achieve accurate and efficient satellite image segmentation. Experiments conducted on a diverse dataset evaluate the model's segmentation accuracy, computational efficiency, and generalization capability, showing promising results for real-world applications in remote sensing and geospatial analysis[P: 2].[10]

The research paper focuses on the development and evaluation of URNet, a model for monocular depth estimation in autonomous driving systems. URNet incorporates residual blocks, attention blocks, and ASPP blocks into a modified UNet architecture to improve depth estimation accuracy. Experimental results show superior performance compared to existing methods, with lower error rates and higher precision values. The study utilizes the KITTI dataset for training and evaluation, demonstrating the effectiveness of URNet in collision avoidance for smart vehicles. Overall, the findings highlight the improved performance and confidence in the proposed algorithm for depth estimation in autonomous driving systems [P:9].[11]

Why this topic?

The motivation behind embarking on this project stems from a profound passion for technology, exploration, and the vast array of possibilities they offer. Initially, the objective is to lay a sturdy groundwork through a dedicated website, serving as a central hub for disseminating information, updates, and cultivating a community enthralled by the fusion of technology and exploration.

Looking ahead, the aspiration to delve into the realm of environmental conservation and contribute to initiatives aimed at preserving our planet serves as a driving force. The pressing challenges posed by climate change, habitat loss, and biodiversity decline present a compelling opportunity to innovate and address critical environmental issues. Additionally, the prospect of advancing autonomous vehicle and robot technologies to enhance efficiency and safety in various industries further fuels our ambition. The ultimate goal is to make significant strides in advancing sustainable practices, fostering a deeper understanding of our natural world, and pioneering groundbreaking advancements in autonomous systems.

In summary, this project is driven by an unwavering passion for exploration, a steadfast commitment to technological progress, and a resolute dedication to environmental stewardship. While the website serves as a foundational platform, the integration of efforts towards environmental conservation and advancements in autonomous technologies represents a pivotal step forward, propelling the project towards impactful contributions in addressing pressing global challenges and shaping the future of exploration and innovation.

Need:

1) **Autonomous Systems and Robotics: There is a burgeoning demand for precise depth perception in autonomous systems and robotics. Monocular depth estimation addresses this demand by enabling machines to comprehend the three-dimensional structure of their surroundings using only a single camera.**
2) **Real-world Applications: Various industries, including augmented reality, virtual reality, and self-driving cars, necessitate reliable depth information for enriched user experiences and heightened safety.**
3) **Computer Vision Applications: Depth understanding is indispensable in computer vision for tasks such as object recognition, scene understanding, and image segmentation.**
4) **Virtual Reality and Augmented Reality: In virtual reality (VR) and augmented reality (AR) applications, accurate depth perception is essential for creating immersive**

experiences and realistic spatial interactions. Monocular depth estimation enables the generation of depth maps from 2D images, facilitating the seamless integration of virtual and real-world elements.

5) **Medical Imaging: Depth estimation has significant applications in medical imaging, particularly in areas such as surgical navigation, organ segmentation, and pathology detection. Accurate depth information extracted from medical images can aid clinicians in diagnostic decision-making and treatment planning.**
6) **Environmental Monitoring: Depth estimation is valuable for environmental monitoring applications, including terrain mapping, forestry management, and disaster response. By accurately assessing the three-dimensional structure of natural landscapes, depth estimation techniques can support conservation efforts and facilitate rapid disaster assessment and relief operations.**

Opportunity:

1) **Low-cost Sensors: Monocular depth estimation presents an opportunity to diminish the dependency on expensive depth-sensing hardware. This paves the way for cost-effective solutions across various industries, rendering depth perception more accessible for a broader spectrum of applications.**
2) **Enhanced Accessibility in Robotics: By integrating monocular depth estimation into robotic systems, there is an opportunity to augment the autonomy and performance of robots in diverse environments. This can lead to breakthroughs in fields such as logistics, agriculture, and search and rescue operations.**
3) **Research and Innovation: The realm of monocular depth estimation offers fertile grounds for research and innovation.**
4) **Environmental Conservation: Monocular depth estimation can contribute to environmental conservation efforts by providing accurate 3D models of natural landscapes. These models can inform decision-making processes related to land management, biodiversity conservation, and ecosystem restoration.**
5) **Urban Planning and Infrastructure Development: Depth estimation techniques can assist urban planners and civil engineers in creating accurate 3D models of cities and infrastructure assets.**

Aim/Goal:

Our aim is to develop a highly accurate depth estimation model accessible through a user-friendly website interface, fostering engagement and interaction within a community passionate about the intersection of technology and environmental conservation. Additionally, our long-term goal is to explore opportunities for extending our project into the realm of autonomous vehicle and robot technologies, leveraging advancements in these fields to address environmental challenges and contribute to sustainable practices on a global scale.

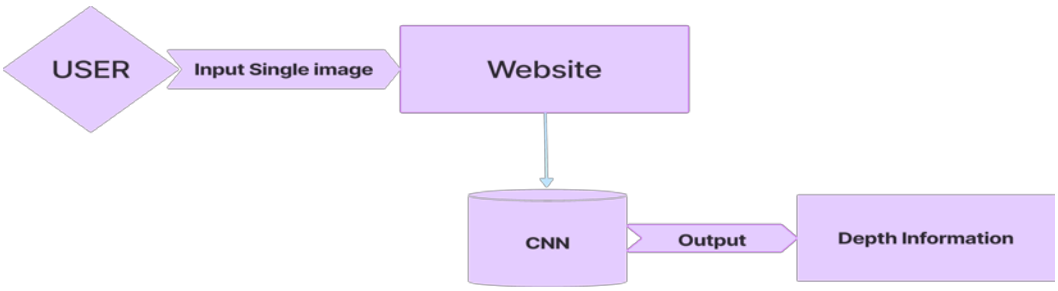

Figure:1

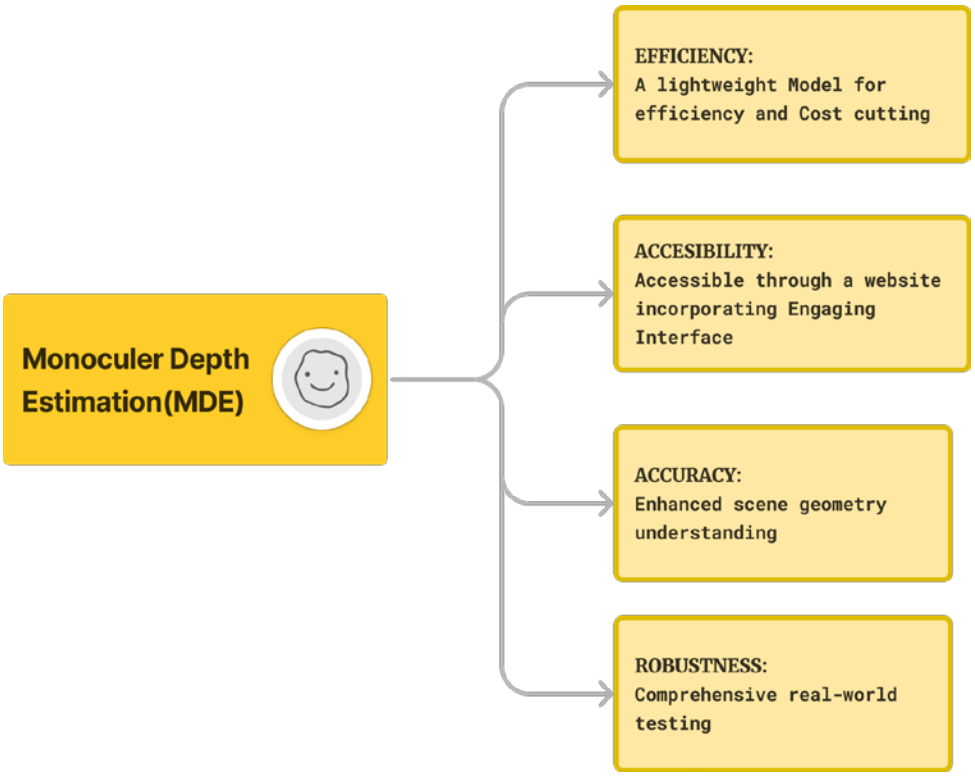

Figure:2

Specifications/Features:

1) **Input Image Processing: Our project accepts input images in various formats (e.g., JPEG, PNG) and sizes. Pre-processing techniques are employed to ensure uniformity and optimal model performance.**
2) **Model Selection: Users have the flexibility to choose from a selection of state-of-the-art depth estimation models, including U-Net, ResNet, MobileNet, or a combination such as ResUNet. Each model offers unique advantages in terms of accuracy, speed, and computational efficiency.**
3) **Depth Estimation: The selected model analyzes the input image and generates a predicted depth map. Leveraging deep learning techniques, the model extracts intricate features and captures spatial relationships to accurately estimate depth information from the 2D image data.**
4) **Comparison with Ground Truth: The predicted depth map is compared with the ground truth depth map, if available, to assess the accuracy of the model's predictions. This comparison provides valuable insights into the model's performance and highlights areas for improvement.**
5) **Mean Squared Error (MSE) Calculation: The Mean Squared Error (MSE) between the predicted depth map and the ground truth depth map is computed to quantitatively evaluate the model's performance. The MSE metric provides a measure of the average squared differences between the predicted and ground truth depth values, offering a standardized assessment of prediction accuracy.**
6) **Visualization: The project offers visualization tools to display the input image, predicted depth map, ground truth depth map (if available), and the corresponding MSE errors. Visual representations aid users in understanding the quality of depth estimation and identifying areas of discrepancy between the predicted and ground truth depth maps.**
7) **User Interface: The user interface is designed to be intuitive and user-friendly, allowing users to easily upload input images, select desired models, and visualize the depth estimation results and MSE errors. Clear and concise feedback is provided to enhance user experience and facilitate seamless interaction with the platform.**
8) **Scalability and Performance: The project is designed to be scalable, capable of handling large volumes of image data and accommodating future expansions and enhancements.**

Efficient implementation ensures optimal performance, enabling timely processing of input images and generation of accurate depth estimations.

Application of the Project in Real Life:

1) **Autonomous Vehicles: Our depth estimation project finds significant application in autonomous vehicles, enabling them to perceive the three-dimensional structure of their surroundings using onboard cameras. Accurate depth estimation facilitates obstacle detection, lane segmentation, and object recognition, contributing to safer and more efficient autonomous driving experiences.**
2) **Augmented Reality: In augmented reality (AR) applications, our project enhances the realism and immersion of virtual overlays by accurately estimating depth in real-world scenes. This enables virtual objects to interact seamlessly with the physical environment, enriching user experiences in gaming, education, architecture, and interior design.**
3) **Medical Imaging: Depth estimation plays a crucial role in medical imaging applications, aiding in surgical navigation, organ segmentation, and pathology detection. Our project can assist healthcare professionals in visualizing complex anatomical structures and identifying abnormalities with greater precision and accuracy.**
4) **Environmental Monitoring: Environmental monitoring efforts benefit from our depth estimation project by providing detailed 3D reconstructions of natural landscapes. This enables scientists and conservationists to assess terrain features, monitor vegetation health, and track changes in ecosystems over time, supporting biodiversity conservation and ecosystem management.**
5) **Robotics: In robotics, accurate depth estimation is essential for tasks such as object manipulation, navigation, and scene understanding. Our project can be integrated into autonomous robots operating in various environments, including manufacturing facilities, warehouses, and hazardous locations, enhancing their perception capabilities and enabling more sophisticated behaviors.**
6) **Construction and Architecture: Depth estimation technology can assist architects, engineers, and construction professionals in creating detailed 3D models of buildings and infrastructure projects. By accurately estimating depth from 2D images, our project streamlines the design process, facilitates spatial planning, and enables virtual walkthroughs of architectural designs.**
7) **Geospatial Mapping: Our depth estimation project contributes to geospatial mapping efforts by providing precise elevation data from aerial and satellite imagery. This**

information is invaluable for urban planning, land management, disaster response, and environmental conservation initiatives, supporting informed decision-making and sustainable development practices.

8) **Retail and E-commerce: Depth estimation technology enhances the online shopping experience by enabling virtual try-on for clothing and accessories. By accurately estimating depth from user-uploaded images or live video feeds, our project enables customers to visualize how products will look and fit in real-world settings, reducing the need for returns and improving customer satisfaction.**

In summary, the application of our depth estimation project spans across diverse industries and domains, revolutionizing various aspects of daily life and contributing to advancements in technology, science, and society.

Required Tools:

1) **Python (programming language):** Python is a versatile, high-level programming language known for its readability and simplicity. It is widely used in web development, data analysis, artificial intelligence, scientific computing, and automation.
2) **TensorFlow (machine learning library):** TensorFlow is an open-source machine learning library developed by Google. It provides a comprehensive ecosystem for building and deploying machine learning models, particularly deep learning applications.
3) **Keras (high-level neural networks API):** Keras is an open-source, high-level neural networks API, written in Python and capable of running on top of TensorFlow. It is designed to enable fast experimentation with deep neural networks.
4) **OpenCV (computer vision library):** OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It contains over 2500 optimized algorithms for tasks like image processing, object detection, and face recognition.
5) **NumPy (numerical computing library):** NumPy is a fundamental package for scientific computing with Python. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
6) **Pandas (data manipulation library):** Pandas is an open-source data manipulation and analysis library for Python. It provides data structures like DataFrames to store and manipulate large datasets easily, making it a key tool for data analysis and cleaning.

7) **Matplotlib (data visualization library):** Matplotlib is a widely used 2D plotting library for Python. It allows the creation of static, interactive, and animated visualizations in various formats, making it essential for data analysis and presentation.
8) **Streamlit:** Streamlit is an open-source app framework for machine learning and data science projects. It allows the rapid creation of interactive web applications directly from Python scripts, facilitating easy sharing and exploration of data and models.
9) **Google Colab:** Google Colab (Colaboratory) is a free, cloud-based Jupyter notebook environment provided by Google. It allows users to write and execute Python code in a web browser, with access to free computing resources, including GPUs, making it ideal for data analysis, machine learning, and collaborative research.

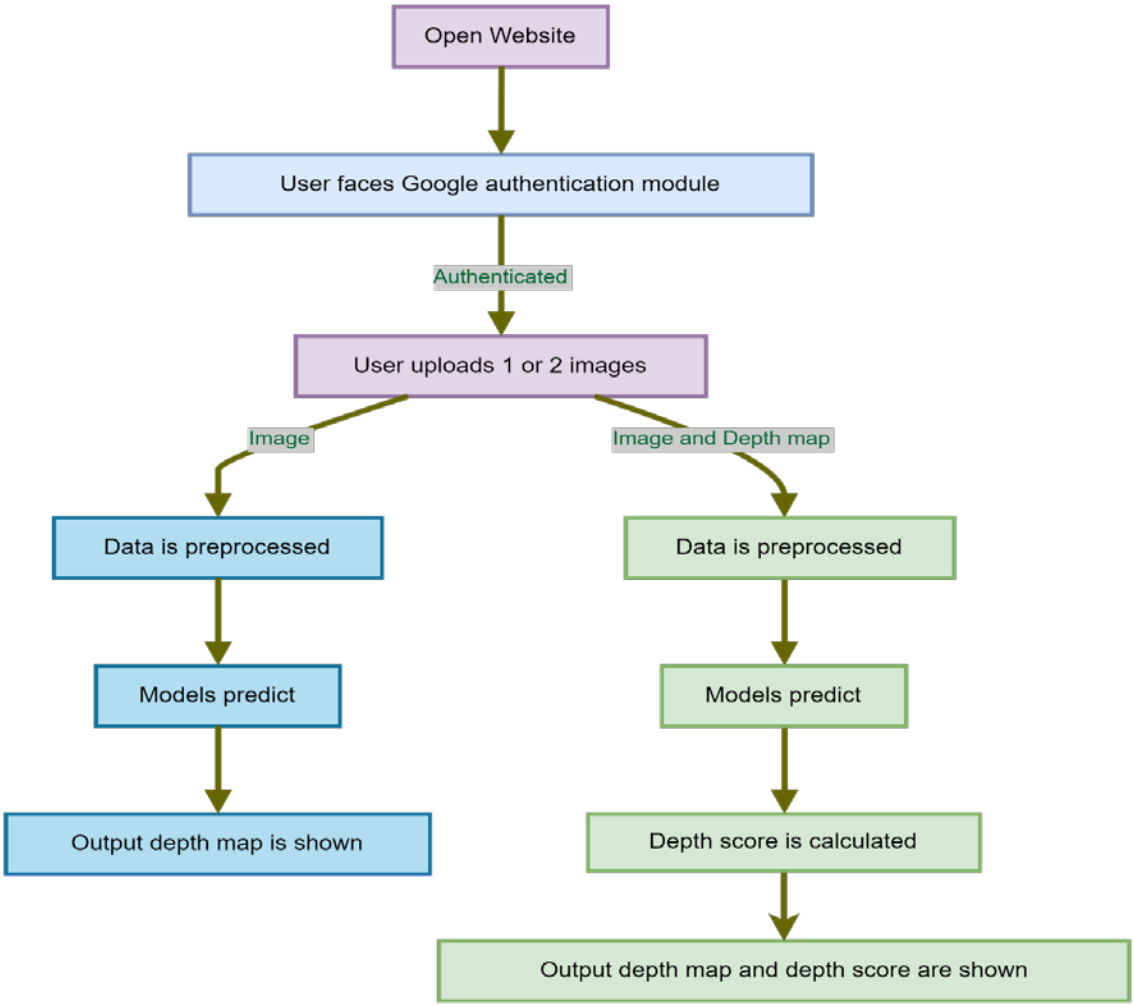

Figure:3

Project Plan:

The following was our project plan at the beginning:

1) **Research and Literature Review:** Begin with an in-depth study of existing techniques and algorithms for monocular depth estimation, including deep learning approaches such as CNNs and GANs.
2) **Data Collection and Preprocessing:** Gather a diverse dataset of images paired with corresponding depth maps. Preprocess the data, including resizing, normalization, and augmentation to enhance model generalization.
3) **Model Selection and Development:** Experiment with various architectures like Monocular Depth Estimation Networks and refine them based on performance metrics. Train the chosen model using suitable optimization techniques and loss functions.
4) **Evaluation and Validation:** Assess the model's performance. Validate the results qualitatively by visual inspection of predicted depth maps compared to ground truth.
5) **Fine-tuning and Optimization:** Fine-tune the model parameters and hyperparameters to enhance performance further. Optimize computational resources and memory usage for real-time applications if necessary.
6) **Integration and Deployment:** Integrate the trained model into a user-friendly application or deploy it as a service. Ensure compatibility with various platforms and provide documentation for ease of use.
7) **Testing and Iteration:** Conduct rigorous testing to identify potential issues or biases. Iterate on the model architecture and training process based on feedback and performance **evaluations.**

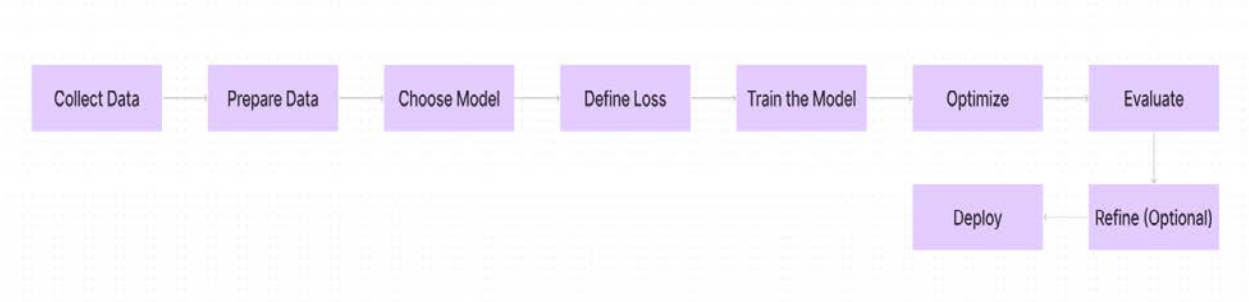

Figure:4

Methodology:

1. Data Preparation:

a. **Dataset:** The NYU Depth V2 dataset was used for this project. It consists of RGB images and corresponding depth maps captured using a Kinect camera in various indoor scenes.

b. **Data Loading:** The dataset was loaded using pandas, with the image and depth map file paths stored in separate columns of a dataframe.

c. **Preprocessing:**

i. **Resizing:** Both the RGB images and depth maps were resized to a fixed size of 224x224 pixels using OpenCV's resize function.

ii**. Normalization:** The RGB image pixel values were normalized to the range [0, 1] by dividing by the maximum pixel value. The depth map values were also normalized to the range [0, 1] by subtracting the minimum value and dividing by the maximum value.

iii. **Data Augmentation:** To increase the diversity of the training data, random horizontal flipping was applied to both the RGB images and depth maps during training.

iv. Train-Validation-Test Split: The dataset was split into training, validation, and test sets. The first 80% of the data was used for training, and the remaining 20% was further split into validation (10%) and test (10%) sets.

2. Model Architecture:

a. **Backbone Network:** The MobileNetV2 architecture, pre-trained on the ImageNet dataset, was used as the backbone network for feature extraction from the input RGB images.

```
def create_model(input_shape):
    # Load pre-trained EfficientNetB0
    base_model = EfficientNetB0(input_shape=input_shape, include_top=False, weights='imagenet')
```

Figure:5

b**. Encoder:**

1) A pretrained network (MobilenetV2,Densenet121,Efficientnet etc)served as the encoder, extracting features at different scales from the input RGB images.

2) Residual Block in Encoding Layer: In the context of combining ResNet and U-Net architectures, a residual block in the encoding layer can enhance the feature extraction process by leveraging skip connections. This allows the network to learn more complex features while maintaining the benefits of both architectures, leading to improved performance in tasks such as image segmentation.

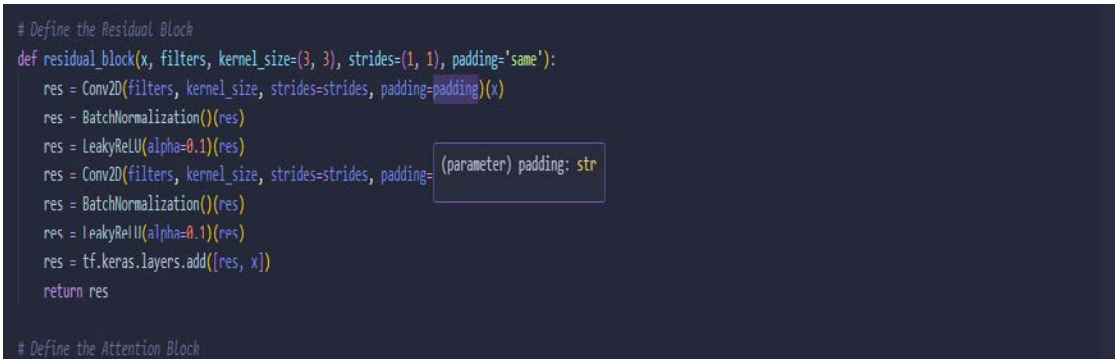

Figure6:

3) Dense Block in Encoding Layer: In the context of integrating DenseNet and U-Net architectures, a dense block in the encoding layer can significantly enhance the feature extraction process. Dense blocks, originally introduced in DenseNet, consist of multiple convolutional layers where each layer is connected to every other layer in a feed-forward fashion. This design improves information flow and gradient propagation throughout the network.

Figure:7

c. Decoder:

i. The decoder part of the network consisted of a series of upsampling layers that combined features from different scales of the encoder to generate the final depth map.

```
# upsampling layers
x = base_model.output
x = UpSampling2D()(x)
x = Concatenate()([base_model.get_layer('block6a_expand_activation').output, x])
x = DepthwiseConv2D((3, 3), padding='same')(x)
x = ReLU()(x)
x = Conv2D(1, (1, 1))(x)
x = UpSampling2D()(x)2365
x = Concatenate()([base_model.get_layer('block4a_expand_activation').output, x])
x = DepthwiseConv2D((3, 3), padding='same')(x)
x = ReLU()(x)
x = Conv2D(1, (1, 1))(x)
x = UpSampling2D()(x)
x = Concatenate()([base_model.get_layer('block3a_expand_activation').output, x])
x = DepthwiseConv2D((3, 3), padding='same')(x)
x = ReLU()(x)
x = Conv2D(1, (1, 1))(x)
x = UpSampling2D()(x)
x = Concatenate()([base_model.get_layer('block2a_expand_activation').output, x])
x = DepthwiseConv2D((3, 3), padding='same')(x)
x = ReLU()(x)
x = Conv2D(1, (1, 1))(x)
x = UpSampling2D()(x)
```

Figure:8

Figure:9

ii. **<u>Attention Block in Decoder</u>**: For res_unet and dense_unet model architecture we used an attention block in decoder section. the attention block is used in the decoder part of the U-Net architecture, specifically in the upsampling path. The attention block is used to refine the features being passed to the next layers by focusing on the relevant parts of the feature maps from the encoder. This helps in capturing fine details and improving the segmentation performance.

```
# Define the Attention Block
def attention_block(x, g, inter_channel_ratio=8):
    theta = Conv2D(filters=g.shape[-1]//inter_channel_ratio, kernel_size=1, strides=(1, 1), padding='same')(x)
    phi = Conv2D(filters=g.shape[-1]//inter_channel_ratio, kernel_size=1, strides=(1, 1), padding='same')(g)
    theta_phi = Activation('relu')(tf.keras.layers.add([theta, phi]))
    psi = Conv2D(filters=1, kernel_size=1, strides=(1, 1), padding='same')(theta_phi)
    psi = Activation('sigmoid')(psi)
    return tf.keras.layers.multiply([x, psi])
```

Figure:10

Figure:11

iii. The upsampling layers used transposed convolutions (also known as deconvolutions) to increase the spatial resolution of the feature maps.

iv. Concatenation layers were used to combine the upsampled features with the corresponding features from the encoder, allowing the decoder to leverage both low-level and high-level features for depth estimation.

v. Depthwise separable convolutions were used in the decoder to reduce the computational complexity while preserving the model's performance.

d. **Output Layer:** The final output layer of the network was a single-channel convolutional layer with a sigmoid activation function, producing a depth map with pixel values in the range [0, 1].

```
# Final output layer
output = Conv2D(1, (1, 1), padding='same', activation='sigmoid')(x)
```

Figure:12

3. Loss Function:

a. **BerHu Loss:** The BerHu loss function, a combination of the Huber loss and the Euclidean loss, was used for training the model.

b. **SSIM Loss:** SSIM loss measures the similarity between two images by focusing on structural information, luminance, and contrast, aligning more with human visual perception than pixel-wise metrics like MSE.

c. **MSE LOSS:** MSE (Mean Squared Error) loss measures the average squared difference between predicted and actual values, penalizing larger errors more heavily.

d.Binary Cross Entropy: Binary Cross Entropy (BCE) is a loss function used in binary classification tasks, measuring the difference between the predicted probabilities and the actual binary labels.

```
# Define the SSIM loss function
def ssim_loss(y_true, y_pred):
    return 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=1.0))

# Define the combined loss function
def combined_loss(y_true, y_pred):
    return binary_crossentropy(y_true, y_pred) + ssim_loss(y_true, y_pred)
```

Figure:13

```python
# Define the SSIM loss function
def ssim_loss(y_true, y_pred):
    return 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=1.0))

# Define the combined loss function
def combined_loss(y_true, y_pred):
    mse = MeanSquaredError()(y_true, y_pred)
    ssim = ssim_loss(y_true, y_pred)
    return mse + ssim
```

Figure:14

4. Training:

a. **Data Generators:** Custom data generators were implemented using Keras' Sequence class to efficiently load and preprocess the data during training and evaluation.

b**. Training Configuration:**

i. **Optimizer:** The Adam optimizer was used for training the model.

ii. **Batch Size:** A batch size of 4 was used for training.

iii. **Epochs:** The model was trained for 10 & 25(fpr res_unet & dense_unet) epochs.

c. **Training Process:** The model was trained on the training dataset using the defined BerHu loss function, combined SSIM and MSE total loss and the Adam optimizer with a learning rate with the custom data generator for efficient data loading and preprocessing.

```python
# Define model parameters
input_shape = (224, 224, 3)

# Build the model
model = build_resunet_with_attention(input_shape)

# Compile the model with Adam optimizer and custom loss function
model.compile(optimizer=Adam(learning_rate=0.0001), loss=combined_loss, metrics=[ssim_loss])

# Train the model
history = model.fit(train_generator, validation_data=val_generator, epochs=25)  # Increased epochs for better training

# Save the trained model
model.save('resunet_with_attention.h5')

# Evaluate the model on test data
test_loss, test_ssim = model.evaluate(test_generator)

print("Test Loss:", test_loss)
print("Test SSIM:", test_ssim)
```

Figure:15

5. Evaluation:

a. Metrics: The trained model's performance was evaluated on the validation dataset using the following metrics:

i. **Mean Absolute Error (MAE):** The average absolute difference between the predicted depth map and the ground truth depth map.

ii. **Root Mean Squared Error (RMSE):** The square root of the average squared difference between the predicted depth map and the ground truth depth map.

iii. Structural Similarity Index (SSIM): The similarity between objects in the image.

b**. Visualization:** The predicted depth maps were visualized and compared with the ground truth depth maps to qualitatively assess the model's performance.

Here's the updated deployment section with the addition of deploying the best model as a web app using **Streamlit:**

6. **Inference and Deployment:**

a. **Test Set Evaluation:** The trained model was evaluated on the test dataset to assess its performance on unseen data.

b. **Visualization:** The predicted depth maps on the test set were visualized and analyzed to assess the model's generalization capability.

c. **Model Selection:** Based on the evaluation metrics and qualitative analysis, the best-performing model was selected for deployment.

d. **Web Application Development:**

i. **Streamlit:** The best-performing model was deployed as a web application using the Streamlit framework, which allows for the creation of interactive and user-friendly web applications directly from Python code.

ii. **User Interface**: The web application provided a user interface where users could upload RGB images, and the application would predict and display the corresponding depth maps.

iii. **Depth Score:** In addition to displaying the predicted depth map, the application also calculated and displayed a depth score, which quantified the accuracy of the predicted depth map compared to the ground truth depth map (if available).

e. **Deployment:** The web application was deployed and made accessible to users, allowing them to upload RGB images and obtain predicted depth maps and depth scores in real-time.We have implemented google authentication using *google_auth_oauthlib.flow* .The user must authenticate with their google account to use the website.The deployment of the best-performing model as a web application using Streamlit enabled easy access and interaction with the depth estimation model. Users could simply upload RGB images, and the application would generate and display the predicted depth maps along with a depth score, providing a convenient and user-friendly way to utilize the depth estimation capabilities of the model.

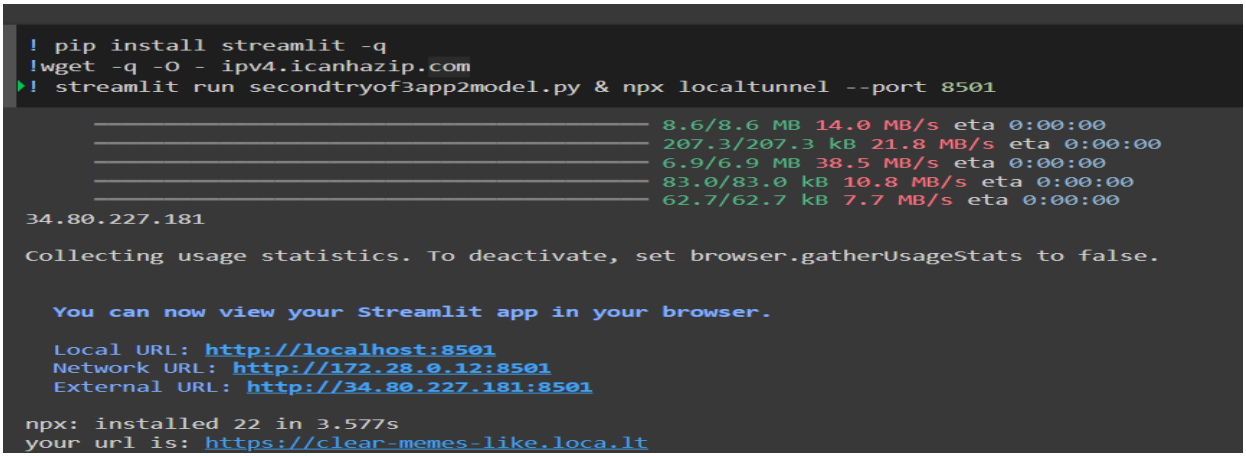

Figure16:

Design and Implementation:s

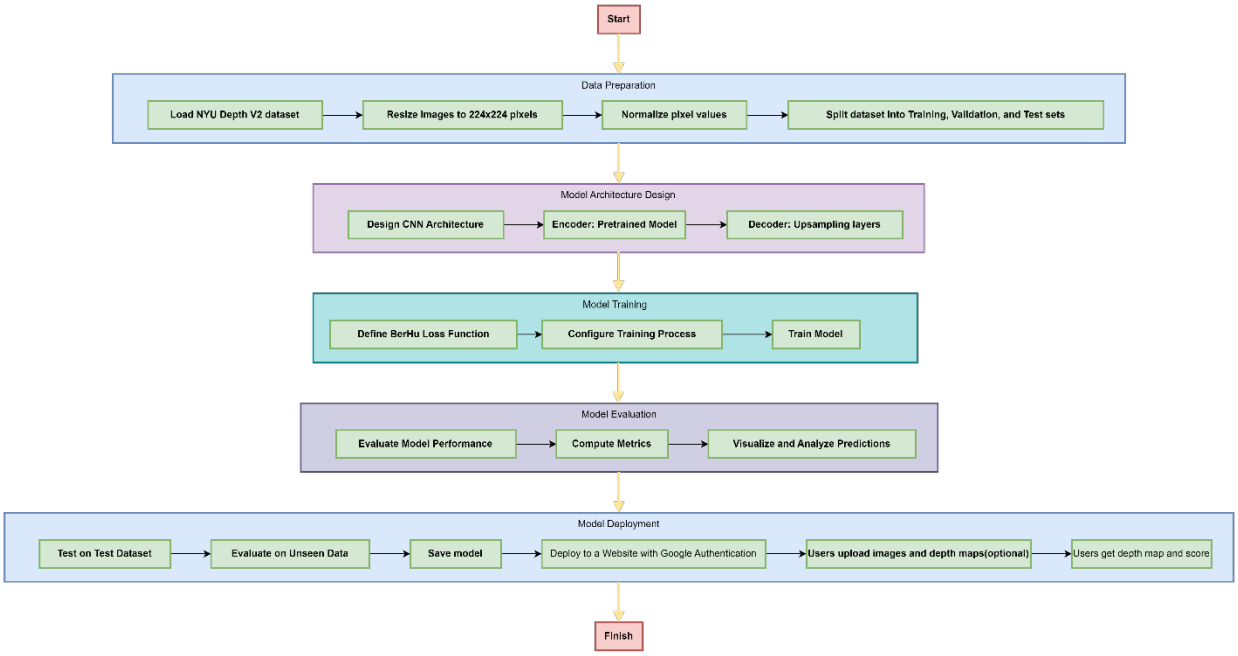

Figure:17

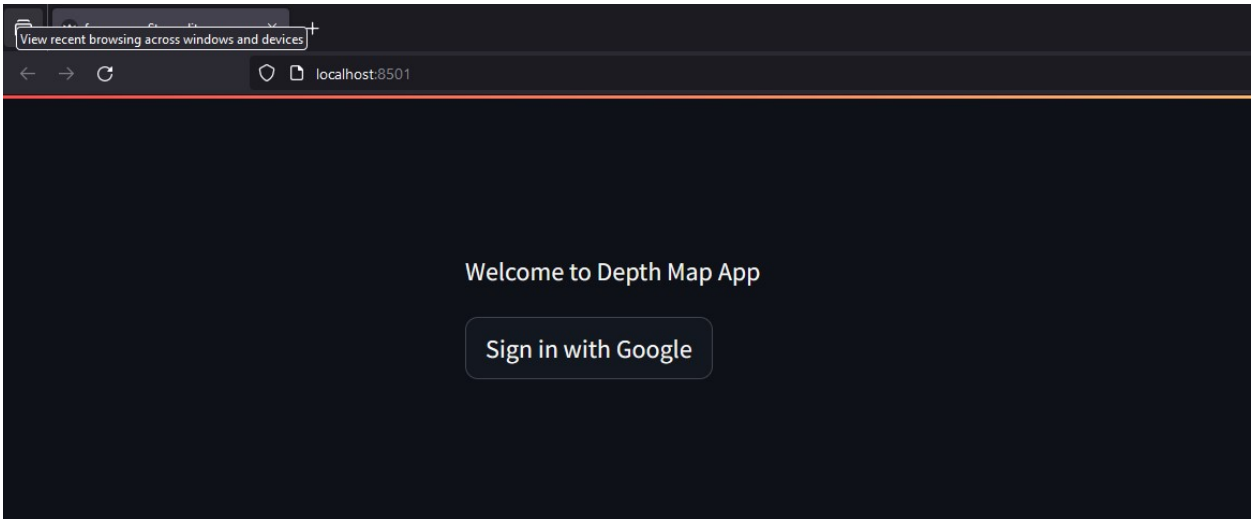

Figure:18

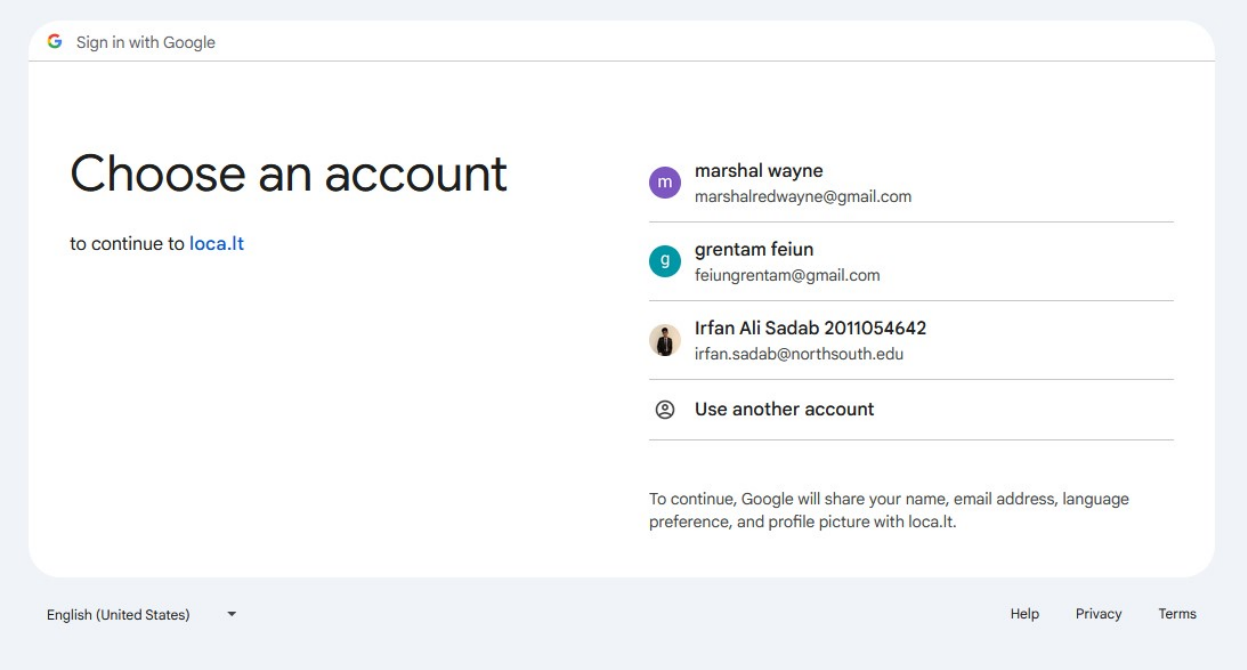

Figure:19

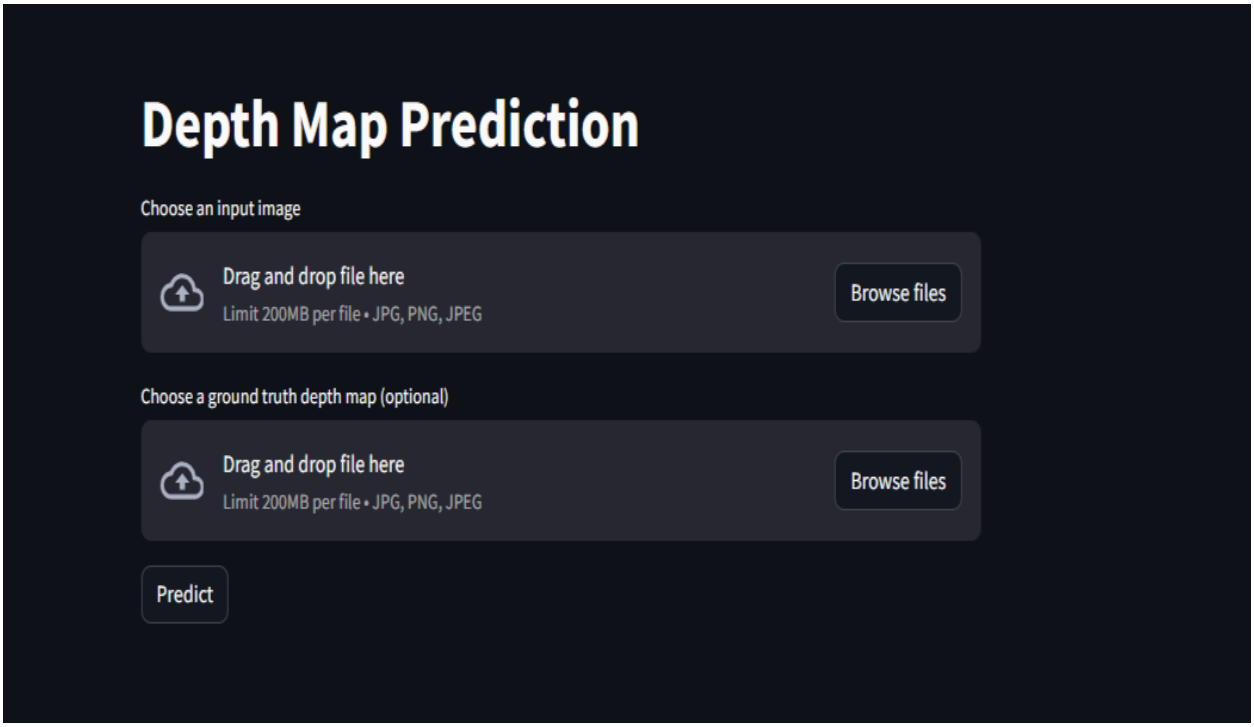

Figure:20

Figure:21

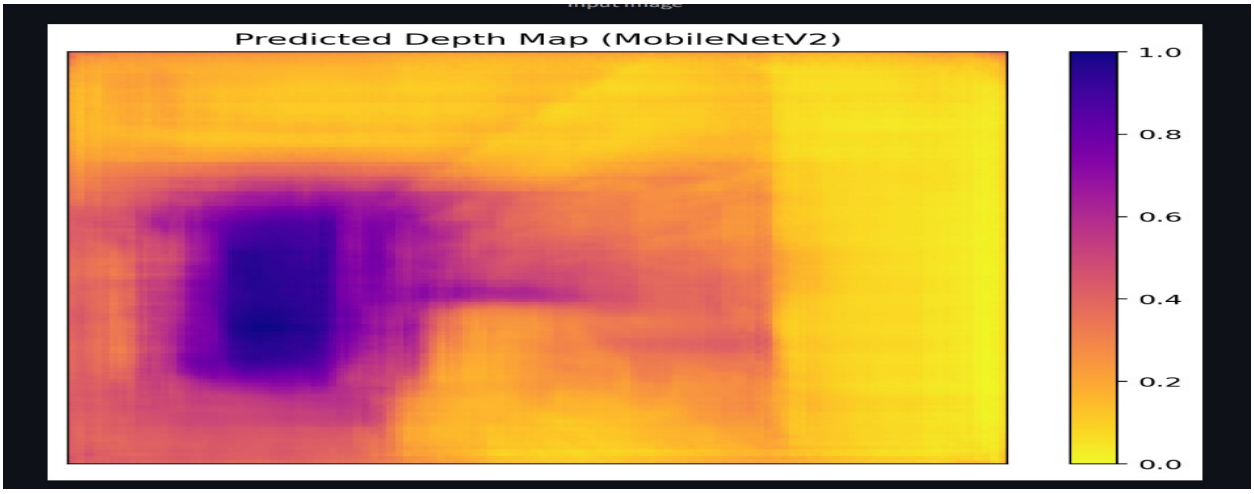

Figure:22

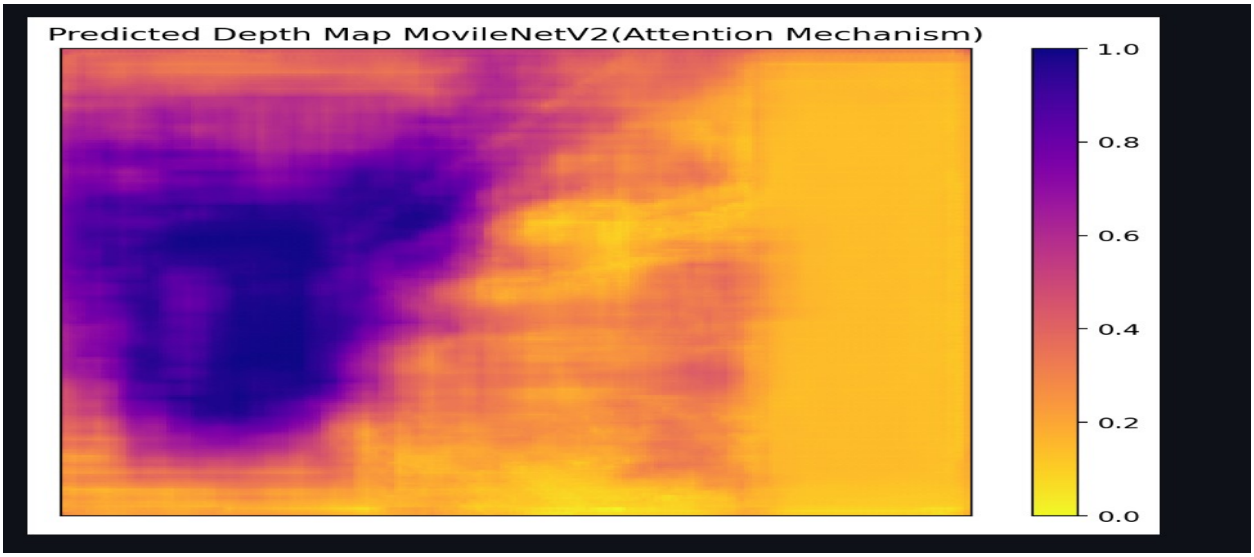

Figure:23

Figure:24

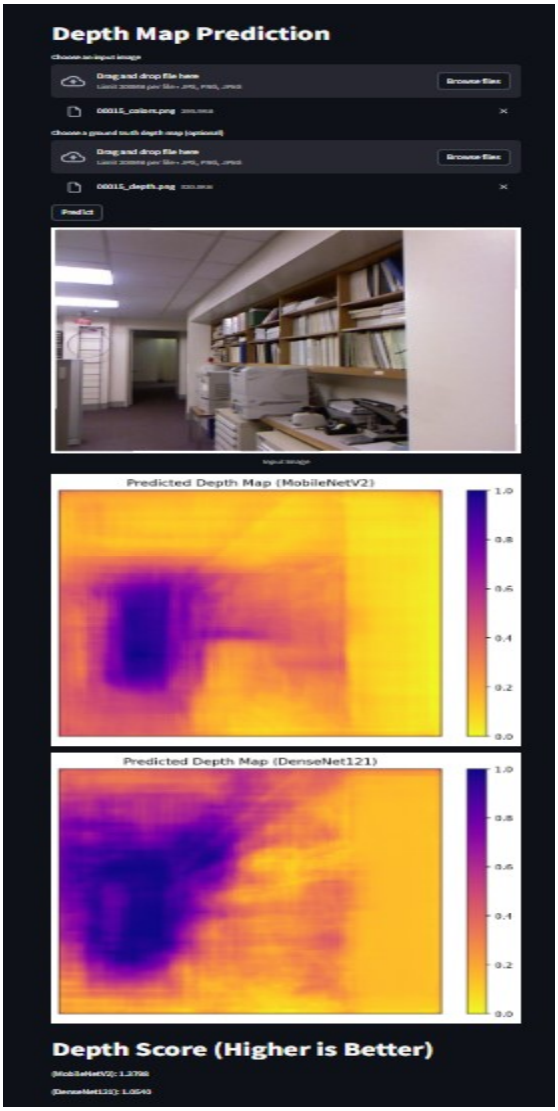

Figure:25

Results Analysis and Evaluation:

	Mobile NetV2	MobileNetV2(attention mechanism)	DenseNet121	DenseNet169	EfficientNet	U-Net	Res_Unet(with attention mechanism)	Dense_Unet(with attention mechanism)
Validation set Loss after the Final epoch	0.0585	0.0779	0.0972	0.179	1.4721	**0.3680**	**0.9368**	**0.9781**

Table: 1

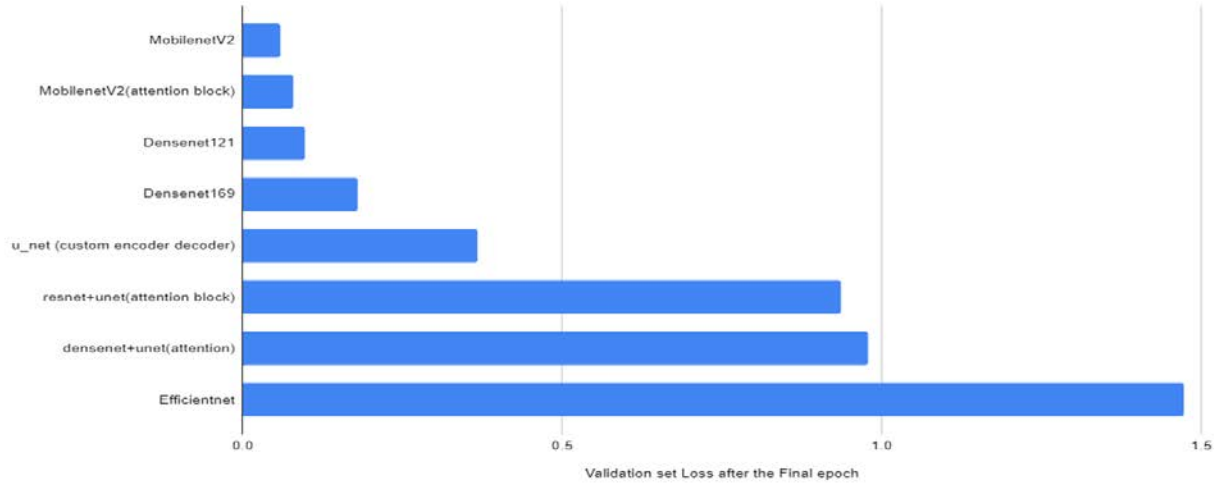

Figure:26

MoblienetV2:

The most accuracy we have got was with approach where we used MobileNetV2 as an encoder and Depthwise separable convolutions in the uplsampling layers. Depthwise separable convolutions in the decoder helps us to reduce the computational complexity while preserving the model's performance.a total of 4 upsampling layer (+1 additional layer) were used along with a final output layer.We have used 'berhu loss function in this approach'.The Validation loss after the final epoch we have got was 0.0585

```
Epoch 1/10
1000/1000 [==============================] - 3089s 3s/step - loss: 0.1082 - val_loss: 0.1864
Epoch 2/10
1000/1000 [==============================] - 1192s 1s/step - loss: 0.0828 - val_loss: 0.1872
Epoch 3/10
1000/1000 [==============================] - 1284s 1s/step - loss: 0.0732 - val_loss: 0.1019
Epoch 4/10
1000/1000 [==============================] - 1282s 1s/step - loss: 0.0653 - val_loss: 0.0975
Epoch 5/10
1000/1000 [==============================] - 1226s 1s/step - loss: 0.0598 - val_loss: 0.0800
Epoch 6/10
1000/1000 [==============================] - 1226s 1s/step - loss: 0.0539 - val_loss: 0.0884
Epoch 7/10
1000/1000 [==============================] - 1324s 1s/step - loss: 0.0512 - val_loss: 0.0700
Epoch 8/10
1000/1000 [==============================] - 1284s 1s/step - loss: 0.0467 - val_loss: 0.0654
Epoch 9/10
1000/1000 [==============================] - 1260s 1s/step - loss: 0.0440 - val_loss: 0.0582
Epoch 10/10
1000/1000 [==============================] - 1233s 1s/step - loss: 0.0407 - val_loss: 0.0585
```

Figure:27

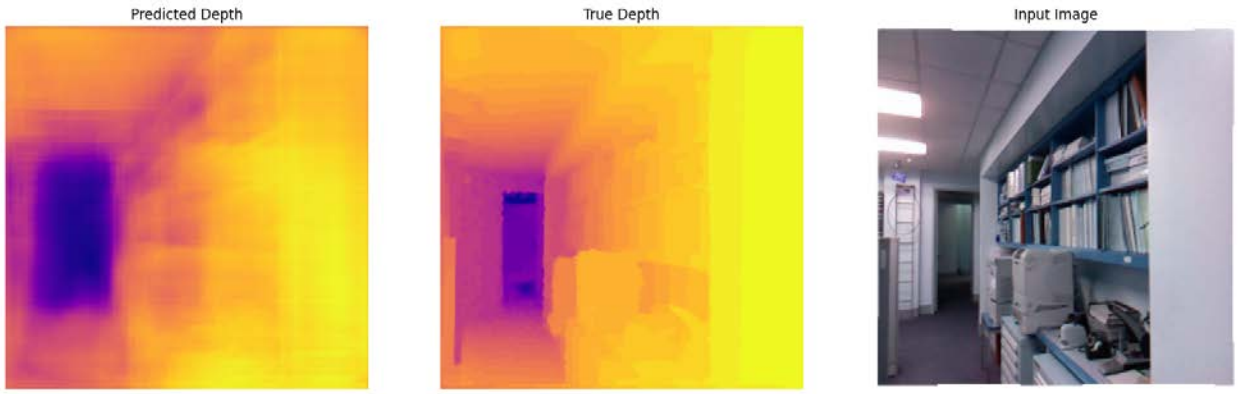

Figure:28

MoblienetV2(Attenttion mechanism):

Our next approach was to incorporate an attention mechanism into the previous approach. But in contrary to our expectation the the validation set loss value has gotten a little worse after the final epoch to 0.0779

```
Epoch 1/10
1000/1000 [==============================] - 6846s 7s/step - loss: 0.1214 - val_loss: 0.1700
Epoch 2/10
1000/1000 [==============================] - 1290s 1s/step - loss: 0.0976 - val_loss: 0.2133
Epoch 3/10
1000/1000 [==============================] - 1291s 1s/step - loss: 0.0898 - val_loss: 0.0964
Epoch 4/10
1000/1000 [==============================] - 1302s 1s/step - loss: 0.0861 - val_loss: 0.1141
Epoch 5/10
1000/1000 [==============================] - 1366s 1s/step - loss: 0.0817 - val_loss: 0.1149
Epoch 6/10
1000/1000 [==============================] - 1333s 1s/step - loss: 0.0820 - val_loss: 0.1429
Epoch 7/10
1000/1000 [==============================] - 1373s 1s/step - loss: 0.0801 - val_loss: 0.0856
Epoch 8/10
1000/1000 [==============================] - 1300s 1s/step - loss: 0.0751 - val_loss: 0.0830
Epoch 9/10
1000/1000 [==============================] - 1293s 1s/step - loss: 0.0754 - val_loss: 0.0951
Epoch 10/10
1000/1000 [==============================] - 1363s 1s/step - loss: 0.0759 - val_loss: 0.0779
```

Figure:28

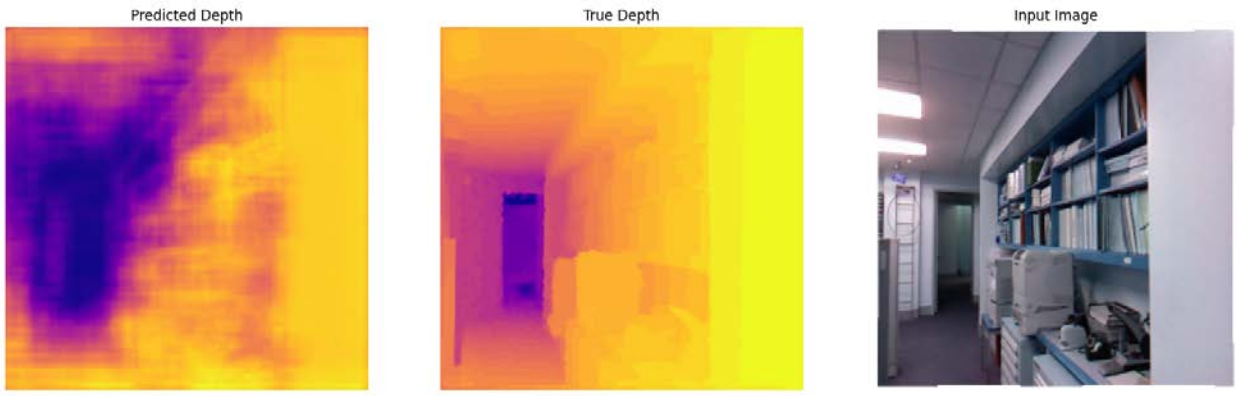

Figure:29

EfficientNet:

Then we have tested with EfficientNet as an encoder.The approach was pretty similar to that one of MobileNetV2,But this the time we used scale invariant logarithm as our loss function.the validation loss after the final epoch were way worse (1.472) and the performance on the test set was very poor compared to MoblieNet.

```
Epoch 1/10
1000/1000 [==============================] - 6798s 7s/step - loss: 1.6047 - val_loss: 1.6865
Epoch 2/10
1000/1000 [==============================] - 988s 988ms/step - loss: 1.5459 - val_loss: 1.6555
Epoch 3/10
1000/1000 [==============================] - 987s 987ms/step - loss: 1.5276 - val_loss: 1.5937
Epoch 4/10
1000/1000 [==============================] - 963s 963ms/step - loss: 1.5188 - val_loss: 1.5283
Epoch 5/10
1000/1000 [==============================] - 973s 973ms/step - loss: 1.5118 - val_loss: 1.5096
Epoch 6/10
1000/1000 [==============================] - 969s 969ms/step - loss: 1.5039 - val_loss: 1.5007
Epoch 7/10
1000/1000 [==============================] - 970s 970ms/step - loss: 1.4979 - val_loss: 1.4886
Epoch 8/10
1000/1000 [==============================] - 978s 978ms/step - loss: 1.4941 - val_loss: 1.4730
Epoch 9/10
1000/1000 [==============================] - 983s 983ms/step - loss: 1.4870 - val_loss: 1.4666
Epoch 10/10
1000/1000 [==============================] - 977s 977ms/step - loss: 1.4813 - val_loss: 1.4721
```

Figure:30

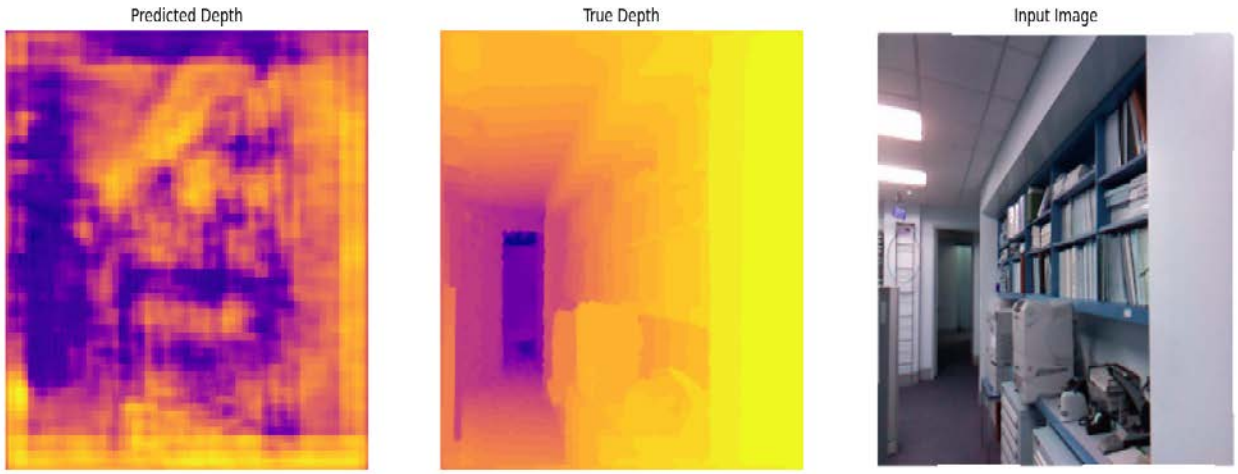

Figure:31

Denstenet121:

Then we went on to try with Densenet121.This model contains less layers than Densenet 169. We have trained this model in the similar approach as MoblineNetV2 .While the validation set loss was not too bad (0.0972) , but the model's output images had checkerboard effect, which makes the output depth maps look distorted.

```
Epoch 1/10
1000/1000 [==============================] - 4336s 4s/step - loss: 0.1222 - val_loss: 0.1031
Epoch 2/10
1000/1000 [==============================] - 929s 928ms/step - loss: 0.1070 - val_loss: 0.1115
Epoch 3/10
1000/1000 [==============================] - 925s 925ms/step - loss: 0.1038 - val_loss: 0.0999
Epoch 4/10
1000/1000 [==============================] - 920s 920ms/step - loss: 0.1035 - val_loss: 0.1057
Epoch 5/10
1000/1000 [==============================] - 923s 923ms/step - loss: 0.1009 - val_loss: 0.0996
Epoch 6/10
1000/1000 [==============================] - 925s 925ms/step - loss: 0.1007 - val_loss: 0.0971
Epoch 7/10
1000/1000 [==============================] - 926s 927ms/step - loss: 0.1005 - val_loss: 0.0962
Epoch 8/10
1000/1000 [==============================] - 883s 883ms/step - loss: 0.0997 - val_loss: 0.0953
Epoch 9/10
1000/1000 [==============================] - 910s 910ms/step - loss: 0.0990 - val_loss: 0.0973
Epoch 10/10
1000/1000 [==============================] - 910s 910ms/step - loss: 0.0991 - val_loss: 0.0972
```

Figure:32

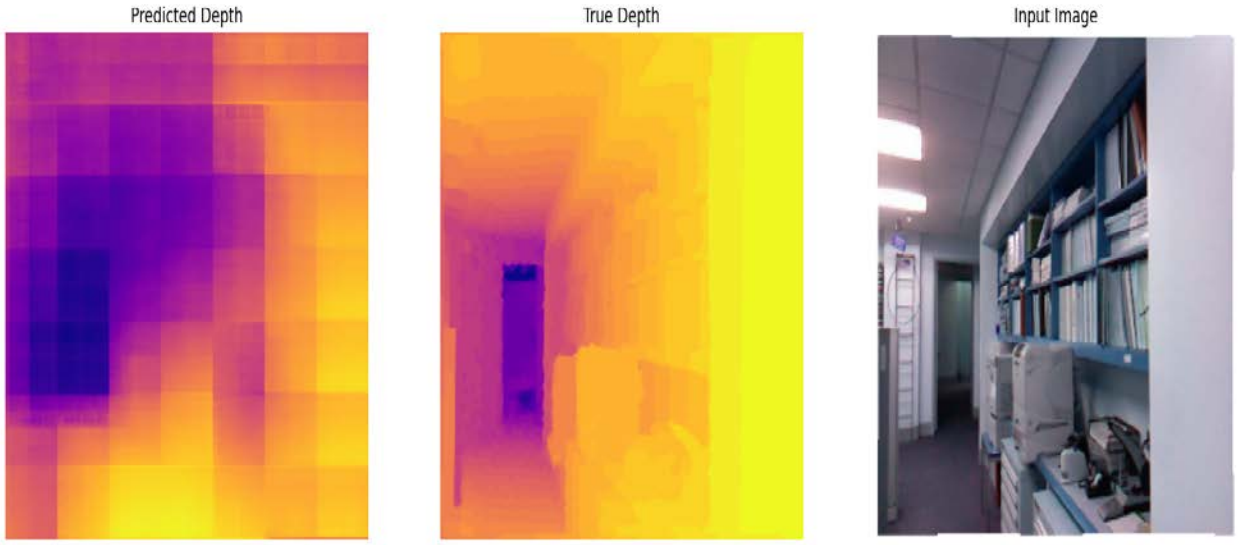

Figure:33

Densenet169(trained on 200 data):

After that we took a different approach to try DenseNet169.Unlike the previous resizing approaches, where we set both depth map and depth images to resize to be (224x224) This time we resized the images to (640x480)and the depth maps to (320x240) .Unfortunately the training process became very inefficient and resource hungry for that we had to train the model on only 200 data for 3 epochs.After the final epoch,the validation set loss were 0.179 .The predicted output was not bad given that the model was only trained with 200data .This indicates that if we train this model with more data it might generate better results.

```
Epoch 1/3
100/100 [==============================] - 480s 4s/step - loss: 0.1933 - accuracy_function: 0.7067 - val_loss: 0.2543 - val_accuracy_function: 0.6460
Epoch 2/3
100/100 [==============================] - 96s 963ms/step - loss: 0.1606 - accuracy_function: 0.7738 - val_loss: 0.1830 - val_accuracy_function: 0.7517
Epoch 3/3
100/100 [==============================] - 96s 963ms/step - loss: 0.1446 - accuracy_function: 0.8181 - val_loss: 0.1790 - val_accuracy_function: 0.7317
```

Figure:34

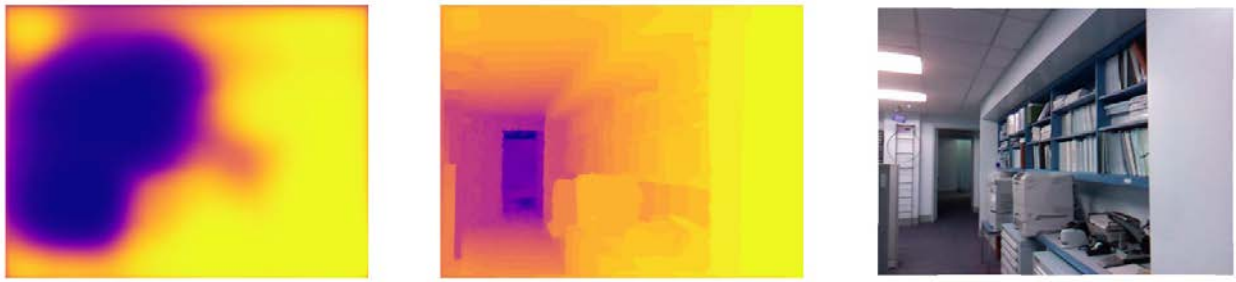

Figure:35

Finally we have tested our dataset with two pretrained models

1. **MiDaS model:** This is a deep learning model developed by Intel for estimating depth from a single image. It's used in computer vision tasks like robotics and self-driving cars. This models outputs very smooth and accurate depth maps. MiDas was trained on NYUV2,KITTY etc datasets.

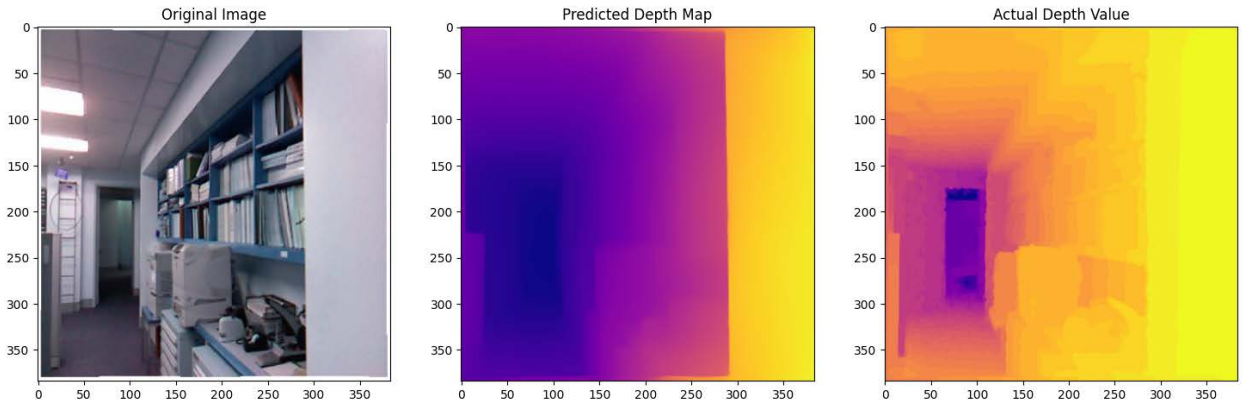

Figure:36

2. **Monodept2:** The model itself is built on a U-Net architecture, which is a type of convolutional neural network (CNN) well-suited for tasks like image segmentation and depth prediction. Monodepthv2 incorporates several key features to improve depth estimation accuracy:

   Reprojection Loss: This loss function helps the model learn from geometric constraints by comparing predicted depth with reconstructed depth based on camera motion.

   Multi-scale Sampling: This technique reduces visual artifacts in the final depth map by using features extracted at different resolutions within the U-Net.

   Auto-masking Loss: This loss helps the model ignore pixels that violate camera motion assumptions, leading to more robust depth predictions.While this model doesn't show as clean and accurate ,they are still satisfactory to some level.

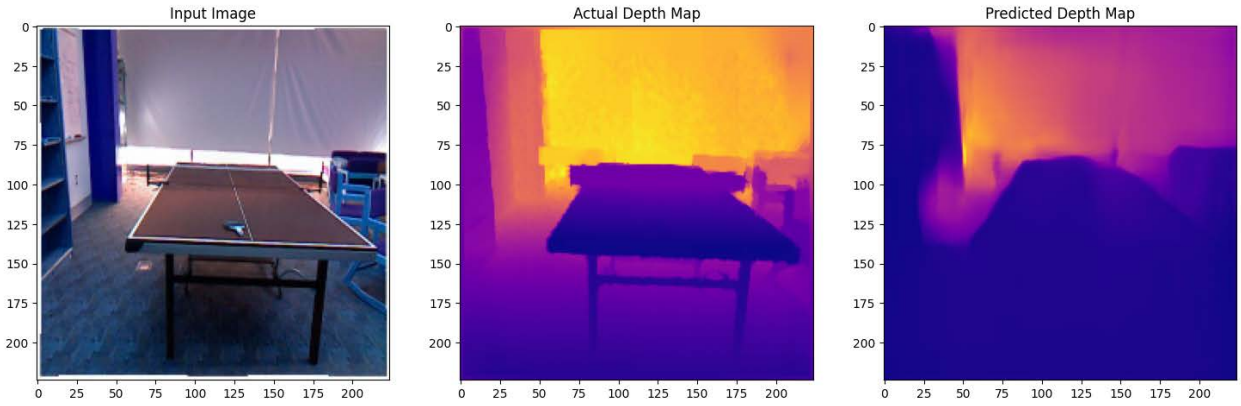

Figure:37

U-Net:

Importing Libraries:

1) **tensorflow: TensorFlow library for building and training deep learning models. Model, Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate, LeakyReLU, BatchNormalization: Various layers and functions from Keras to build the neural network.**
2) **Adam: Adam optimizer.**
3) **MeanSquaredError: Mean Squared Error loss function.**
4) **EarlyStopping, ReduceLROnPlateau: Callbacks for training.**
5) **matplotlib.pyplot: Library for plotting graphs.**
6) **numpy: Library for numerical operations.**

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate, LeakyReLU, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
import numpy as np
```

Figure:38

Define SSIM and Combined Loss Functions:

1) **ssim_loss: Custom loss function calculating the Structural Similarity Index (SSIM). SSIM measures the similarity between two images.**
2) **combined_loss: Combines Mean Squared Error (MSE) and SSIM loss for training, encouraging both pixel-wise accuracy and perceptual similarity.**

```
# Define the SSIM loss function
def ssim_loss(y_true, y_pred):
    return 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=1.0))

# Define the combined loss function
def combined_loss(y_true, y_pred):
    mse = MeanSquaredError()(y_true, y_pred)
    ssim = ssim_loss(y_true, y_pred)
    return mse + ssim
```

Figure:39

Building the U-Net Model with Batch Normalization

1) **Input Layer: Defines the input shape.**
2) **Encoding Layers: Series of convolutional, batch normalization, and LeakyReLU activation layers followed by max-pooling layers to downsample the input.**

3) **Bottleneck Layer: Convolutional layers in the middle of the network, after downsampling.**
4) **Decoding Layers: UpSampling layers to upsample the feature maps, concatenation with corresponding encoding layers, followed by convolutional, batch normalization, and LeakyReLU activation layers.**
5) **Output Layer: A convolutional layer with a sigmoid activation function to produce the final depth map.**

Figure:40

Model Compilation and Training

1) **input_shape: Specifies the shape of the input images.**
2) **depth_model: Builds the model using the defined architecture.**
3) **optimizer: Adam optimizer with a specified learning rate.**
4) **compile: Compiles the model with the optimizer and combined loss function.**
5) **fit: Trains the model using train_generator and validates it using val_generator for a specified number of epochs.**

Figure:41

Output Image:

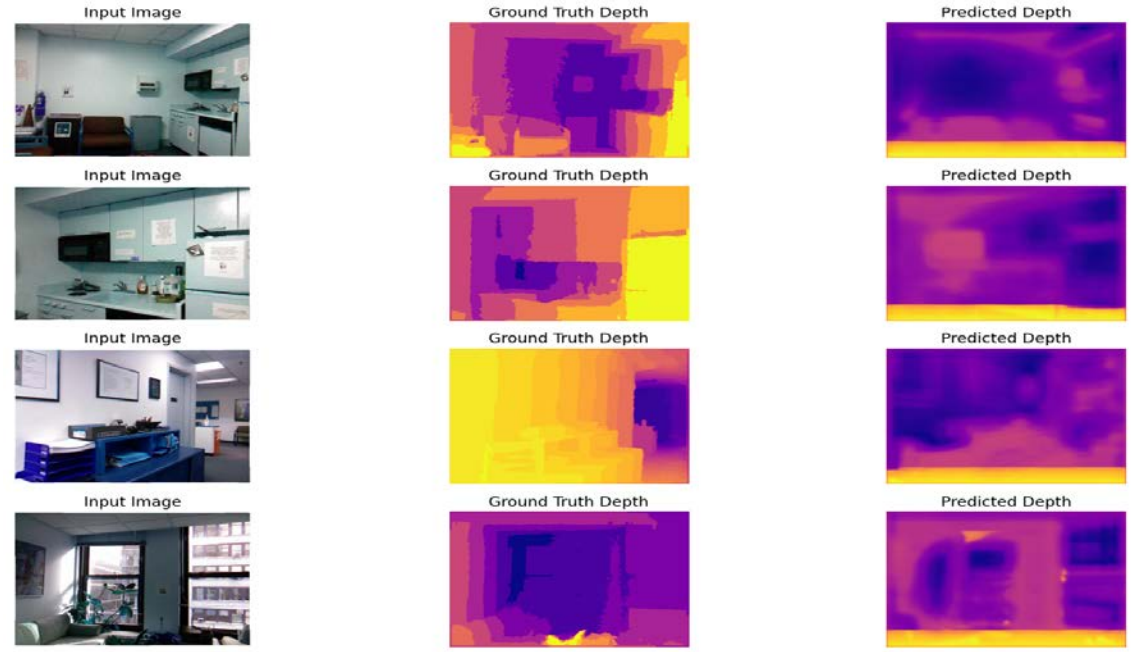

Figure:42

```
c:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\activations\leaky_relu.py:41: UserWarning: Argument `alpha` is deprecated. Use `ne
  warnings.warn(
Epoch 1/10
c:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:120: UserWarning: Your `PyDataset` class sho
  self._warn_if_super_not_called()
100/100 ━━━━━━━━━━━━━━━━━━━━ 1607s 16s/step - loss: 0.4450 - val_loss: 0.4206
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1577s 16s/step - loss: 0.3627 - val_loss: 0.3731
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1571s 16s/step - loss: 0.3659 - val_loss: 0.3808
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1562s 16s/step - loss: 0.3468 - val_loss: 0.3751
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1553s 16s/step - loss: 0.3445 - val_loss: 0.3709
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1570s 16s/step - loss: 0.3497 - val_loss: 0.3621
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1573s 16s/step - loss: 0.3422 - val_loss: 0.3763
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1572s 16s/step - loss: 0.3409 - val_loss: 0.3741
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1579s 16s/step - loss: 0.3445 - val_loss: 0.3650
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1575s 16s/step - loss: 0.3442 - val_loss: 0.3680
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using ir
```

Figure:43

Res_U-net (with Attention mechanism):

Importing Libraries:

1) **tensorflow: TensorFlow library for building and training deep learning models.**
2) **Model, Input, Conv2D, MaxPooling2D, concatenate, Conv2DTranspose, BatchNormalization, LeakyReLU, Activation: Various layers and functions from Keras to build the neural network.**
3) **Adam: Adam optimizer.**
4) **binary_crossentropy: Binary Cross-Entropy loss function.**
5) **matplotlib.pyplot: Library for plotting graphs.**

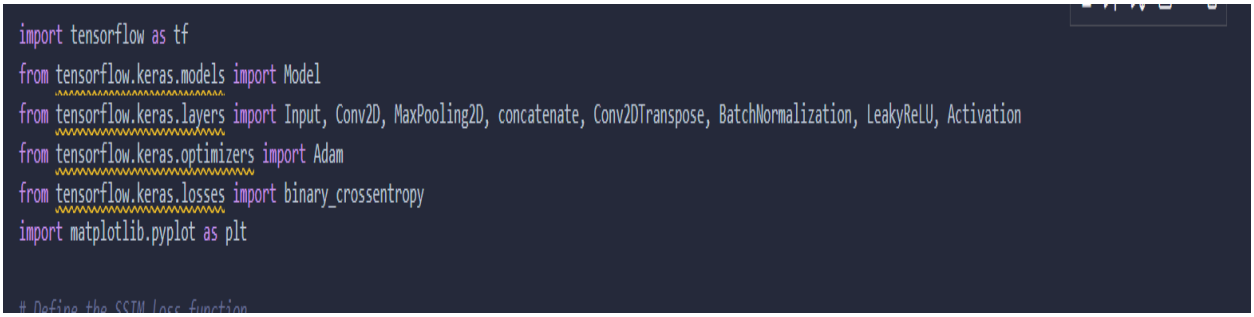

Figure:44

Define SSIM and Combined Loss Functions:

1) **ssim_loss: Custom loss function calculating the Structural Similarity Index (SSIM).**
2) **combined_loss: Combines Binary Cross-Entropy and SSIM loss for training.**

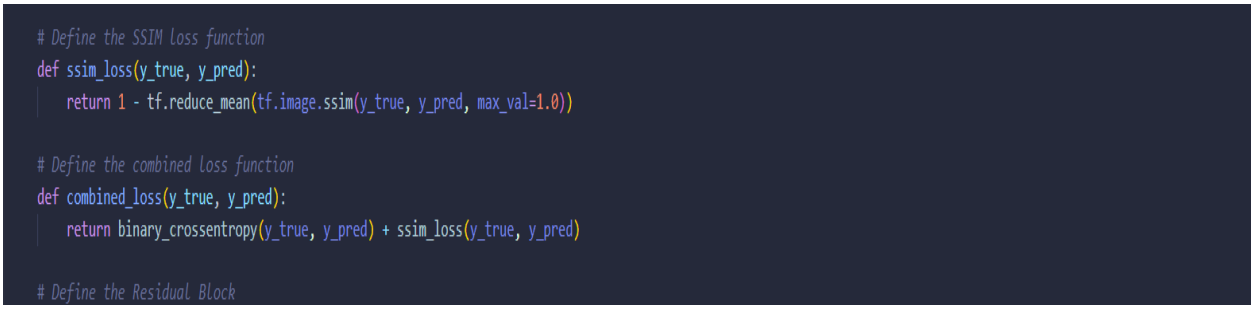

Figure:45

Define Residual and Attention Blocks:

1) **residual_block: Implements a residual block in the model architecture.**

```
# Define the Residual Block
def residual_block(x, filters, kernel_size=(3, 3), strides=(1, 1), padding='same'):
    res = Conv2D(filters, kernel_size, strides=strides, padding=padding)(x)
    res = BatchNormalization()(res)
    res = LeakyReLU(alpha=0.1)(res)
    res = Conv2D(filters, kernel_size, strides=strides, padding=
    res = BatchNormalization()(res)
    res = LeakyReLU(alpha=0.1)(res)
    res = tf.keras.layers.add([res, x])
    return res

# Define the Attention Block
```

(parameter) padding: str

Figure:46

Build ResUNet with Attention Block:

1) **build_resunet_with_attention: Constructs a U-Net model with residual blocks and attention mechanisms.**

```
# Define the Attention Block
def attention_block(x, g, inter_channel_ratio=8):
    theta = Conv2D(filters=g.shape[-1]//inter_channel_ratio, kernel_size=1, strides=(1, 1), padding='same')(x)
    phi = Conv2D(filters=g.shape[-1]//inter_channel_ratio, kernel_size=1, strides=(1, 1), padding='same')(g)
    theta_phi = Activation('relu')(tf.keras.layers.add([theta, phi]))
    psi = Conv2D(filters=1, kernel_size=1, strides=(1, 1), padding='same')(theta_phi)
    psi = Activation('sigmoid')(psi)
    return tf.keras.layers.multiply([x, psi])

# Build the ResUNet with Attention Block
```

Figure:47

```
# Build the ResUNet with Attention Block
def build_resunet_with_attention(input_shape):
    inputs = Input(input_shape)

    # Encoder
    conv1 = Conv2D(64, 3, padding='same')(inputs)
    conv1 = LeakyReLU(alpha=0.1)(conv1)
    conv1 = residual_block(conv1, 64)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, padding='same')(pool1)
    conv2 = LeakyReLU(alpha=0.1)(conv2)
    conv2 = residual_block(conv2, 128)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, padding='same')(pool2)
    conv3 = LeakyReLU(alpha=0.1)(conv3)
    conv3 = residual_block(conv3, 256)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    # Bridge
    conv4 = Conv2D(512, 3, padding='same')(pool3)
    conv4 = LeakyReLU(alpha=0.1)(conv4)
    conv4 = residual_block(conv4, 512)

    # Decoder with attention
    up5 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(conv4)
    up5 = concatenate([up5, conv3], axis=3)
    up5 = Conv2D(256, 3, padding='same')(up5)
    up5 = LeakyReLU(alpha=0.1)(up5)
    up5 = residual_block(up5, 256)
    up5 = attention_block(up5, conv3)

    up6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(up5)
    up6 = concatenate([up6, conv2], axis=3)
    up6 = Conv2D(128, 3, padding='same')(up6)
    up6 = LeakyReLU(alpha=0.1)(up6)
    up6 = residual_block(up6, 128)
    up6 = attention_block(up6, conv2)

    up7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(up6)
    up7 = concatenate([up7, conv1], axis=3)
    up7 = Conv2D(64, 3, padding='same')(up7)
    up7 = LeakyReLU(alpha=0.1)(up7)
    up7 = residual_block(up7, 64)
    up7 = attention_block(up7, conv1)

    # Output
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(up7)

    model = Model(inputs=[inputs], outputs=[outputs])
    return model
```

Figure:48

Model Compilation, Training, and Evaluation:

1) **model.compile: Compiles the model with Adam optimizer and the custom combined loss function.**
2) **model.fit: Trains the model using the specified data generators.**
3) **model.evaluate: Evaluates the trained model on the test data.**

```
# Define model parameters
input_shape = (224, 224, 3)

# Build the model
model = build_resunet_with_attention(input_shape)

# Compile the model with Adam optimizer and custom loss function
model.compile(optimizer=Adam(learning_rate=0.0001), loss=combined_loss, metrics=[ssim_loss])

# Train the model
history = model.fit(train_generator, validation_data=val_generator, epochs=25)  # Increased epochs for better training

# Save the trained model
model.save('resunet_with_attention.h5')

# Evaluate the model on test data
test_loss, test_ssim = model.evaluate(test_generator)

print("Test Loss:", test_loss)
print("Test SSIM:", test_ssim)
```

Figure:49

Output:

```
  warnings.warn(
Epoch 1/25
c:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:120: UserWarning: Your `PyDataset` class sho
  self._warn_if_super_not_called()
100/100 ━━━━━━━━━━━━━━━━━━━━ 308s 3s/step - loss: 1.1928 - ssim_loss: 0.5086 - val_loss: 1.0501 - val_ssim_loss: 0.3655
Epoch 2/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 289s 3s/step - loss: 1.0543 - ssim_loss: 0.3746 - val_loss: 1.0432 - val_ssim_loss: 0.3628
Epoch 3/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 286s 3s/step - loss: 1.0560 - ssim_loss: 0.3828 - val_loss: 1.0295 - val_ssim_loss: 0.3590
Epoch 4/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 285s 3s/step - loss: 1.0203 - ssim_loss: 0.3672 - val_loss: 1.0044 - val_ssim_loss: 0.3472
Epoch 5/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 282s 3s/step - loss: 0.9975 - ssim_loss: 0.3473 - val_loss: 0.9718 - val_ssim_loss: 0.3295
Epoch 6/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 281s 3s/step - loss: 1.0013 - ssim_loss: 0.3535 - val_loss: 0.9718 - val_ssim_loss: 0.3299
Epoch 7/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 279s 3s/step - loss: 0.9835 - ssim_loss: 0.3439 - val_loss: 0.9643 - val_ssim_loss: 0.3233
Epoch 8/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 279s 3s/step - loss: 0.9903 - ssim_loss: 0.3455 - val_loss: 0.9616 - val_ssim_loss: 0.3204
Epoch 9/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 284s 3s/step - loss: 0.9797 - ssim_loss: 0.3413 - val_loss: 0.9955 - val_ssim_loss: 0.3379
Epoch 10/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 283s 3s/step - loss: 0.9800 - ssim_loss: 0.3404 - val_loss: 0.9622 - val_ssim_loss: 0.3242
Epoch 11/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 282s 3s/step - loss: 0.9758 - ssim_loss: 0.3373 - val_loss: 0.9851 - val_ssim_loss: 0.3279
Epoch 12/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 283s 3s/step - loss: 0.9785 - ssim_loss: 0.3399 - val_loss: 0.9563 - val_ssim_loss: 0.3210
Epoch 13/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 281s 3s/step - loss: 0.9705 - ssim_loss: 0.3385 - val_loss: 0.9668 - val_ssim_loss: 0.3283
...
Epoch 24/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 279s 3s/step - loss: 0.9520 - ssim_loss: 0.3218 - val_loss: 0.9306 - val_ssim_loss: 0.2999
Epoch 25/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 280s 3s/step - loss: 0.9466 - ssim_loss: 0.3188 - val_loss: 0.9368 - val_ssim_loss: 0.3052
```

Figure:50

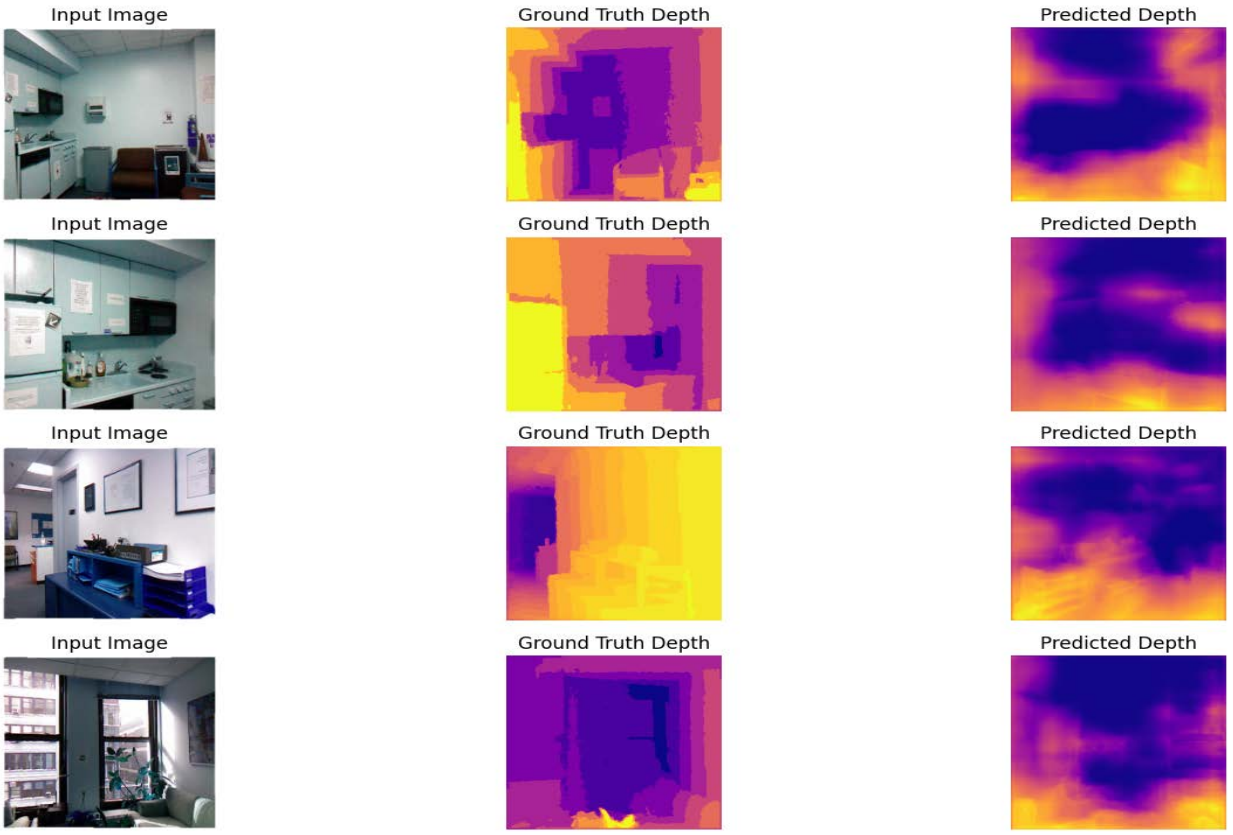

Figure:51

Dense_U-net (with Attention mechanism):

Importing Libraries:

1) **tensorflow: TensorFlow library for building and training deep learning models.**
2) **Model, Input, Conv2D, MaxPooling2D, concatenate, Conv2DTranspose, BatchNormalization, LeakyReLU, Activation: Various layers and functions from Keras to build the neural network.**
3) **Adam: Adam optimizer.**
4) **binary_crossentropy: Binary Cross-Entropy loss function.**
5) **matplotlib.pyplot: Library for plotting graphs**

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, concatenate, Conv2DTranspose, BatchNormalization, LeakyReLU, Activation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import binary_crossentropy
import matplotlib.pyplot as plt
```

Figure:52

Define SSIM and Combined Loss Functions:

1) **ssim_loss: Custom loss function calculating the Structural Similarity Index (SSIM).**
2) **combined_loss: Combines Binary Cross-Entropy and SSIM loss for training.**

```
# Define the SSIM loss function
def ssim_loss(y_true, y_pred):
    return 1 - tf.reduce_mean(tf.image.ssim(y_true, y_pred, max_val=1.0))

# Define the combined loss function
def combined_loss(y_true, y_pred):
    return binary_crossentropy(y_true, y_pred) + ssim_loss(y_true, y_pred)
```

Figure:53

Define Dense Block:

1) **dense_block: Implements a dense block in the model architecture.**

```
# Define the Dense Block
def dense_block(x, growth_rate, layers):
    for _ in range(layers):
        cb = Conv2D(4 * growth_rate, (1, 1), padding='same')(x)
        cb = BatchNormalization()(cb)
        cb = LeakyReLU(alpha=0.1)(cb)
        cb = Conv2D(growth_rate, (3, 3), padding='same')(cb)
        cb = BatchNormalization()(cb)
        cb = LeakyReLU(alpha=0.1)(cb)
        x = concatenate([x, cb], axis=-1)
    return x
```

Figure:54

Define Residual and Attention Blocks:

1) **residual_block: Implements a residual block in the model architecture.**

```
# Define the Attention Block
def attention_block(x, g, inter_channel_ratio=8):
    theta = Conv2D(filters=g.shape[-1]//inter_channel_ratio, kernel_size=1, strides=(1, 1), padding='same')(x)
    phi = Conv2D(filters=g.shape[-1]//inter_channel_ratio, kernel_size=1, strides=(1, 1), padding='same')(g)
    theta_phi = Activation('relu')(tf.keras.layers.add([theta, phi]))
    psi = Conv2D(filters=1, kernel_size=1, strides=(1, 1), padding='same')(theta_phi)
    psi = Activation('sigmoid')(psi)
    return tf.keras.layers.multiply([x, psi])
```

Figure:55

Build DenseUNet with Attention Block:

1) **build_denseunet_with_attention: Constructs a Dense U-Net model with dense blocks and attention mechanisms.**

```
# Define the Dense Block
def dense_block(x, growth_rate, layers):
    for _ in range(layers):
        cb = Conv2D(4 * growth_rate, (1, 1), padding='same')(x)
        cb = BatchNormalization()(cb)
        cb = LeakyReLU(alpha=0.1)(cb)
        cb = Conv2D(growth_rate, (3, 3), padding='same')(cb)
        cb = BatchNormalization()(cb)
        cb = LeakyReLU(alpha=0.1)(cb)
        x = concatenate([x, cb], axis=-1)
    return x
```

Figure:56

```python
# Build the DenseUNet with Attention Block
def build_denseunet_with_attention(input_shape, growth_rate=32, layers_per_block=4):
    inputs = Input(input_shape)

    # Encoder
    conv1 = Conv2D(64, 3, padding='same')(inputs)
    conv1 = LeakyReLU(alpha=0.1)(conv1)
    db1 = dense_block(conv1, growth_rate, layers_per_block)
    pool1 = MaxPooling2D(pool_size=(2, 2))(db1)

    conv2 = Conv2D(128, 3, padding='same')(pool1)
    conv2 = LeakyReLU(alpha=0.1)(conv2)
    db2 = dense_block(conv2, growth_rate, layers_per_block)
    pool2 = MaxPooling2D(pool_size=(2, 2))(db2)

    conv3 = Conv2D(256, 3, padding='same')(pool2)
    conv3 = LeakyReLU(alpha=0.1)(conv3)
    db3 = dense_block(conv3, growth_rate, layers_per_block)
    pool3 = MaxPooling2D(pool_size=(2, 2))(db3)

    # Bridge
    conv4 = Conv2D(512, 3, padding='same')(pool3)
    conv4 = LeakyReLU(alpha=0.1)(conv4)
    db4 = dense_block(conv4, growth_rate, layers_per_block)

    # Decoder with attention
    up5 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(db4)
    up5 = concatenate([up5, db3], axis=3)
    up5 = Conv2D(256, 3, padding='same')(up5)
    up5 = LeakyReLU(alpha=0.1)(up5)
    db5 = dense_block(up5, growth_rate, layers_per_block)
    db5 = attention_block(db5, db3)

    up6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(db5)
    up6 = concatenate([up6, db2], axis=3)
    up6 = Conv2D(128, 3, padding='same')(up6)
    up6 = LeakyReLU(alpha=0.1)(up6)
    db6 = dense_block(up6, growth_rate, layers_per_block)
    db6 = attention_block(db6, db2)

    up7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(db6)
    up7 = concatenate([up7, db1], axis=3)
    up7 = Conv2D(64, 3, padding='same')(up7)
    up7 = LeakyReLU(alpha=0.1)(up7)
    db7 = dense_block(up7, growth_rate, layers_per_block)
    db7 = attention_block(db7, db1)

    # Output
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(db7)

    model = Model(inputs=[inputs], outputs=[outputs])
    return model
```

Figure:57

Model Compilation, Training, and Evaluation:

1) **model.compile: Compiles the model with Adam optimizer and the custom combined loss function.**
2) **model.fit: Trains the model using the specified data generators.**
3) **model.evaluate: Evaluates the trained model on the test data.**

Figure:58

Output:

```
  warnings.warn(
Epoch 1/25
c:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:120: UserWarning: Your `PyDataset` class should call `super().__init__(**kwarg
  self._warn_if_super_not_called()
100/100 ━━━━━━━━━━━━━━━━━━━━ 698s 7s/step - loss: 1.1672 - ssim_loss: 0.4704 - val_loss: 1.0707 - val_ssim_loss: 0.3842
Epoch 2/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 632s 6s/step - loss: 1.0596 - ssim_loss: 0.3786 - val_loss: 1.0608 - val_ssim_loss: 0.3807
Epoch 3/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 631s 6s/step - loss: 1.0526 - ssim_loss: 0.3812 - val_loss: 1.0561 - val_ssim_loss: 0.3791
Epoch 4/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 630s 6s/step - loss: 1.0073 - ssim_loss: 0.3624 - val_loss: 1.0705 - val_ssim_loss: 0.3841
Epoch 5/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 630s 6s/step - loss: 0.9752 - ssim_loss: 0.3345 - val_loss: 1.0640 - val_ssim_loss: 0.3823
Epoch 6/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 630s 6s/step - loss: 0.9783 - ssim_loss: 0.3350 - val_loss: 1.0081 - val_ssim_loss: 0.3588
Epoch 7/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 633s 6s/step - loss: 0.9740 - ssim_loss: 0.3365 - val_loss: 0.9875 - val_ssim_loss: 0.3481
Epoch 8/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 630s 6s/step - loss: 0.9617 - ssim_loss: 0.3252 - val_loss: 0.9853 - val_ssim_loss: 0.3472
Epoch 9/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 630s 6s/step - loss: 0.9726 - ssim_loss: 0.3298 - val_loss: 0.9739 - val_ssim_loss: 0.3409
Epoch 10/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 632s 6s/step - loss: 0.9669 - ssim_loss: 0.3317 - val_loss: 0.9819 - val_ssim_loss: 0.3481
Epoch 11/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 631s 6s/step - loss: 0.9553 - ssim_loss: 0.3239 - val_loss: 0.9728 - val_ssim_loss: 0.3389
Epoch 12/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 631s 6s/step - loss: 0.9581 - ssim_loss: 0.3228 - val_loss: 0.9811 - val_ssim_loss: 0.3421
Epoch 13/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 633s 6s/step - loss: 0.9555 - ssim_loss: 0.3215 - val_loss: 0.9946 - val_ssim_loss: 0.3468
...
Epoch 24/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 645s 6s/step - loss: 0.9186 - ssim_loss: 0.3023 - val_loss: 0.9462 - val_ssim_loss: 0.3214
Epoch 25/25
100/100 ━━━━━━━━━━━━━━━━━━━━ 640s 6s/step - loss: 0.9174 - ssim_loss: 0.3038 - val_loss: 0.9781 - val_ssim_loss: 0.3467
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure:59

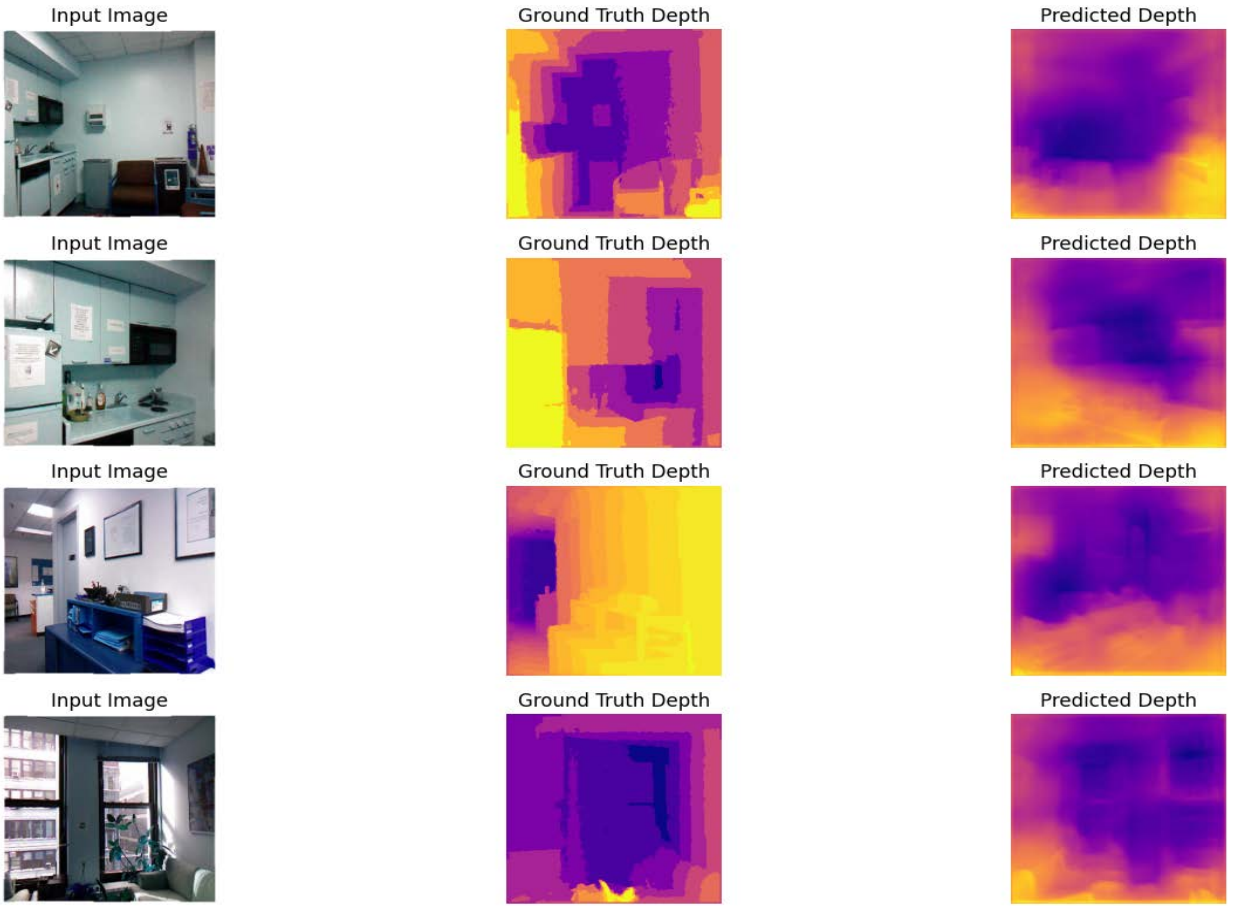

Figure:60

Future plan:

In the future, we plan to further advance our monocular depth estimation model and explore several avenues for improvement and real-world applications:

1. **Model Architecture Enhancements: We will investigate more advanced neural network architectures specifically tailored for monocular depth estimation. This includes exploring encoder-decoder models with different types of attention mechanisms, multi-scale feature fusion, and innovative ways to incorporate global context information. By leveraging the latest advancements in computer vision and deep learning, we aim to improve the model's accuracy and robustness in predicting high-quality depth maps.**

2. **Loss Function Exploration: While our current implementation utilizes the BerHu loss function, we plan to conduct extensive research on alternative loss functions and their impact on depth prediction performance. We will experiment with scale-invariant log losses, combinations of losses, and potentially develop custom loss functions tailored to our specific task and dataset. This exploration could lead to more accurate and consistent depth estimation across various scene types and lighting conditions.**

3. **Unsupervised and Semi-Supervised Learning: While our current approach relies on supervised learning with ground truth depth maps, we recognize the limitations and challenges of acquiring large-scale labeled datasets. To address this, we will explore unsupervised and semi-supervised learning techniques that leverage additional data sources, such as stereo images, motion cues, or synthetic data generation. This could enable us to leverage abundant unlabeled data and reduce the reliance on costly manual annotations.**

4. **Advanced Data Augmentation: To increase the diversity and robustness of our training data, we will investigate and implement advanced data augmentation techniques beyond simple horizontal flipping. This could include color jittering, random cropping, synthetic depth data generation, and other techniques to simulate real-world variations in imaging conditions, viewpoints, and scene compositions. Improved data augmentation could lead to better generalization and performance on diverse real-world scenarios.**

5. **Real-Time Performance Optimization: While the MobileNetV2 backbone provides a good balance between accuracy and efficiency, we aim to further optimize our model for real-time performance on resource-constrained devices, such as mobile phones or embedded systems. This could involve techniques like quantization, pruning, or specialized hardware acceleration using**

GPUs, TPUs, or dedicated AI accelerators. Real-time depth estimation could enable a wide range of applications in augmented reality, robotics, and autonomous navigation systems.

6. Integration into Practical Applications: We will explore integrating our trained depth estimation model into practical applications, such as augmented reality systems for indoor navigation, robotics for obstacle avoidance and path planning, and autonomous vehicle perception systems. By deploying our model in real-world scenarios, we can evaluate its performance, identify potential limitations, and gather valuable feedback to iteratively improve the model.

7. Domain Adaptation and Generalization: To ensure our model's robustness and generalization capabilities, we will investigate domain adaptation techniques to improve its performance on different datasets or real-world environments, which may have different characteristics compared to the training data. This could involve techniques like domain adversarial training, style transfer, or meta-learning approaches to enable seamless adaptation to new domains without extensive retraining.

8. Evaluation on Diverse Datasets: While our initial work focused on the NYU Depth V2 dataset, we plan to extensively evaluate our model's performance on diverse datasets covering a wide range of indoor and outdoor scenes, lighting conditions, and object types. This comprehensive evaluation will help identify potential limitations, biases, or failure modes, guiding future improvements and ensuring our model's robustness across various scenarios.

10. Explainable AI Techniques: To enhance the interpretability and trustworthiness of our depth estimation model, we will incorporate explainable AI techniques. This could involve methods like saliency maps, concept activation vectors, or counterfactual explanations to understand the model's decision-making process and identify potential biases or limitations. Explainable AI could help us refine the model, improve its transparency, and foster trust in its predictions, especially in safety-critical applications.

12. Uncertainty Estimation: We plan to incorporate uncertainty estimation techniques into our depth estimation model. By quantifying the model's uncertainty or confidence levels for each predicted depth value, we can provide more reliable and trustworthy results. This could involve techniques like Monte Carlo dropout, ensemble methods, or Bayesian deep learning approaches. Uncertainty estimation is particularly important in safety-critical applications, such as autonomous navigation or robotics, where reliable depth perception is crucial.

13. Multi-Task Learning: Instead of treating depth estimation as a standalone task, we will explore multi-task learning approaches that jointly learn depth estimation along with other related tasks, such as semantic segmentation, surface normal estimation, or object detection. By leveraging the shared representations and feature spaces across these tasks, we could improve the overall performance and robustness of our depth estimation model while benefiting from the mutual information and regularization provided by the auxiliary tasks.

14. Deployment on Edge Devices: While our initial focus has been on developing accurate depth estimation models, we recognize the importance of deploying these models efficiently on edge devices with limited computational resources. We will explore techniques for model compression, quantization, and hardware-aware optimization to enable efficient inference on devices such as smartphones, embedded systems, or internet-of-things (IoT) devices. This could unlock new applications in areas like augmented reality, robotics, and smart home systems.

15. Integration with Other Sensor Modalities: While our current work focuses on monocular depth estimation from RGB images, we plan to investigate the integration of our model with other sensor modalities, such as LiDAR, radar, or specialized depth sensors. By fusing complementary information from different sensor types, we could potentially improve the overall accuracy, robustness, and reliability of our depth perception system, especially in challenging scenarios or under varying environmental conditions.

Through these future plans, we aim to push the boundaries of monocular depth estimation, addressing current limitations, improving accuracy and robustness, and enabling practical applications that can benefit various industries and domains. Our goal is to contribute to the advancement of this field and deliver reliable and efficient depth perception solutions for a wide range of real-world scenarios.

Total cost: The project didn't cost us any money ,Since it was all done by using free resources and our pre-obtained resources.

Conclusion:

In this project, we developed a number of deep learning models for monocular depth estimation using the MobileNetV2 architecture and a custom decoder network. Our models were trained on the NYU Depth V2 dataset, and while not the best, still we achieved promising results in predicting depth maps from single RGB images.

While our current implementation demonstrates the feasibility and potential of monocular depth estimation, there are numerous avenues for future research and improvement. As outlined in our future plans, we aim to explore more advanced neural network architectures, alternative loss functions, and techniques to incorporate temporal information and unsupervised learning. Additionally, we plan to investigate advanced data augmentation strategies, domain adaptation methods, and explainable AI techniques to enhance the model's performance, generalization, and interpretability.

Furthermore, we recognize the importance of optimizing our model for real-time performance on resource-constrained devices, enabling practical applications in areas such as augmented reality, robotics, and autonomous navigation systems. To this end, we will explore techniques for model compression, quantization, and hardware-aware optimization.

Beyond improving the core depth estimation capabilities, we also plan to integrate our model with other sensor modalities, such as LiDAR or radar, to leverage complementary information and enhance the overall accuracy and robustness of our depth perception system. Additionally, we will investigate adversarial robustness techniques to ensure the reliability and consistency of our model's predictions in the presence of adversarial attacks or data manipulations.

As we continue to advance our research, we will actively collaborate with the broader scientific community, participate in academic conferences, and contribute to open-source repositories. By fostering knowledge sharing and collaboration, we can accelerate progress in the field of monocular depth estimation and related computer vision tasks.

Ultimately, our goal is to develop reliable and efficient depth perception solutions that can be seamlessly integrated into a wide range of real-world applications, such as autonomous vehicles, robotics, augmented reality, and smart home systems. Through our future research efforts, we aim to push the boundaries of what is possible with monocular depth estimation, enabling new opportunities for innovation and contributing to the advancement of computer vision and artificial intelligence.

Contribution:

Irfan Ali Sadab:

- **Data Preprocessing**
- **MobileNetV2**
- **MobileNetV2 (Attention mechanism)**
- **EfficientNet**
- **Densenet169**
- **Densenet121**
- **MiDas**
- **Monodepthv2**
- **Website Implementation using streamlit**
- **Google authentication**

Md. Arafat Islam:

- **Data Preprocessing: Iterate through train test folder and created demo dataset for initial work.**
- **U-net**
- **Res_U-net (with Attention mechanism)**
- **Dense_U-net (with attention mechanism)**
- **Model and loss function implementation on streamlit website.**

References:

[1] D. Rocha, M. Ángel, and P. Guisado, "'Estimation of Depth Maps from Mo- nocular Images using Deep Neural Networks' This work is licensed under Creative Commons Attribution -Non Commercial -Non Derivative," 2018. Accessed: Mar. 05, 2024. [Online]. Available: https://e-archivo.uc3m.es/rest/api/core/bitstreams/8ba91e3c-584c-4ffd-b3d4-abf91488be15/content?fbclid=IwAR1H5oadyPJImLp17gYuMIMaL1ubFL2p0s7YmJCVKlQD_RBMgN1ecBwWeOg

[2] D.-J. Lee et al., "Lightweight Monocular Depth Estimation via Token-Sharing Transformer," arXiv.org, Jun. 09, 2023. https://doi.org/10.48550/arXiv.2306.05682

[3] A. Masoumian, H. A. Rashwan, J. Cristiano, M. S. Asif, and D. Puig, "Monocular Depth Estimation Using Deep Learning: A Review," Sensors, vol. 22, no. 14, p. 5353, Jul. 2022, doi: https://doi.org/10.3390/s22145353.

[4] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, "Monocular Depth Estimation Based On Deep Learning: An Overview," Science China Technological Sciences, vol. 63, no. 9, pp. 1612–1627, Sep. 2020, doi: https://doi.org/10.1007/s11431-020-1582-8

[5] Machkour, Z., Ortiz-Arroyo, D. & Durdevic, P. Monocular Based Navigation System for Autonomous Ground Robots Using Multiple Deep Learning Models. Int J Comput Intell Syst 16, 79 (2023). https://doi.org/10.1007/s44196-023-00250-5

[6] R. Xiaogang, Y. Wenjing, H. Jing, G. Peiyuan and G. Wei, "Monocular Depth Estimation Based on Deep Learning:A Survey," 2020 Chinese Automation Congress (CAC), Shanghai, China, 2020, pp. 2436-2440, doi: 10.1109/CAC51589.2020.9327548

[7] Şafak Kılıç, İman Askerzade and Yılmaz Kaya, "Deep Learning Using MobileNet for Personal Recognizing" Conference: The International Conference on Advanced Engineering, Technology and Applications (ICAETA-2021) At: July 09-10, 2021, Istanbul, Turkey

[8] Duong, H.-T.; Chen, H.-M.; Chang, C.-C. URNet: An UNet-Based Model with Residual Mechanism for Monocular Depth Estimation. Electronics 2023, 12, 1450. doi:https://doi.org/10.3390/electronics12061450

[9] Babitha Lokula, Ramakrishna Tirumuri and L V Narasimha Prasad ," Satellite image segmentation using Unet++ and MobileNetV2 deep learning model"DOI: https://doi.org/10.21203/rs.3.rs-4144393/v1

[10] Uday Kulkarni1, Shreya B Devagiri1, Rohit B Devaranavadagi1, Sneha Pamali1, Nishanth R Negalli1 and Prabakaran V2," Depth Estimation using DNN Architecture and Vision-Based Transformers" ITM Web Conf.Volume 53, 2023,2nd International Conference on Data Science and Intelligent Applications (ICDSIA-2023) doi: https://doi.org/10.1051/itmconf/20235302010

[11] Jan, A.; Seo, S. Monocular Depth Estimation Using Res-UNet with an Attention Model. Appl. Sci. 2023, 13, 6319. https://doi.org/ 10.3390/app13106319